PROCEEDINGS OF

15th Modelica Conference

THE 15TH INTERNATIONAL MODELICA CONFERENCE

AACHEN

09.-11. OCTOBER 2023

RWTH AACHEN UNIVERSITY

Modelica Association

JÜLICH Forschungszentrum

SPONSORED BY

LTX

DASSAULT SYSTEMES

JuliaHub

Modelon

WOLFRAM

TLK ENERGY

BOSCH Invented for life

DLR

SCILS SYSTEMS & CONTROL INNOVATION LAB

CLAYTEX TECHNIA COMPANY

CHIASTEK

Maplesoft

OpenModelica

XRG Simulation GmbH

Proceedings of the 15th International Modelica Conference
Aachen, Germany, October 09-11, 2023

Editor
Prof. Dirk Müller, RWTH Aachen University
Prof. Antonello Monti, PhD, RWTH Aachen University
Prof. Andrea Benigni, PhD, Forschungszentrum Jülich GmbH

Published by
Modelica Association and Linköping University Electronic Press

Series: Linköping Electronic Conference Proceedings Nr. 204
ISBN: 978-91-8075-505-4
ISSN: 1650-3686
eISSN: 1650-3740
DOI: https://doi.org/10.3384/ecp204

Organized by

RWTH Aachen University
52062 Aachen
Germany

Forschungszentrum Jülich GmbH
52425 Jülich
Germany

In cooperation with
Modelica Association
c/o PELAB, Linköpings University
SE-581 83 Linköping
Sweden

Conference location
Das Liebig, Aachen

Conference website
https://2023.international.conference.modelica.org

PREFACE

The Modelica Conference is the main event for users, library developers, tool vendors and language designers to share their knowledge and learn about the latest scientific and industrial progress related to Modelica, the Functional Mockup Interface (FMI), System Structure & Parametrization (SSP), the Distributed Co-Simulation protocol (DCP) and the Functional Mock-up Interface for embedded systems (eFMI).

Since the start of the collaborative design work for Modelica in 1996, Modelica has matured from an idea among a small number of dedicated enthusiasts to a widely accepted standard language for the modeling and simulation of cyber-physical systems. The Modelica language was standardized by the non-profit organization Modelica Association which enabled Modelica models to be portable between a growing number of tools. Modelica is the language of choice for model-based systems engineering and is now used in many industries including automotive, energy and process, aerospace, and industrial equipment.

The Modelica Association has since grown to include several projects supporting modeling and simulation, creating a family of inter-related standards complementing each-other. FMI is an open standard that defines a container and an interface to exchange dynamic models using a single file (an FMU). SSP is a tool-independent standard to define complete systems consisting of one or more FMUs including its parameterization that can be transferred between simulation tools. DCP is a platform and standard for the integration of models or real-time systems into simulation environments. eFMI tooling enables the automatic transformation of higher-level acausal model representations (such as Modelica) to causal solutions suitable for integration in embedded systems.

Highlights of the conference include:

- 8 tutorials
- 2 keynotes
- 3 vendor presentation sessions with 9 presentations
- FMI User Meeting with 5 presentations
- Industrial User Presentation with 3 presentations
- 22 paper sessions with 73 presentations
- 1 poster session with 13 posters
- 13 sponsors
- 2 Best paper awards
  - HVAC and Control Templates for the Modelica Buildings Library (Antoine Gautier; Michael Wetter; Jianjun Hu)
  - Design proposal of a standardized Base Modelica language (Gerd Kurzbach; Oliver Lenord; Hans Olsson; Martin Sjölund; Henrik Tidefelt)
- Best poster award
  - Automatic Optimization of Energy Supply Systems in Buildings and City Quarters based on Modelica Models (Torsten Schwan; David Feige; Leonhard Wenzel; Charlotte Voelckner; Martin Leuschke)
- Best library award: ClaRa (Ales Vojacek; Johannes Brunnemann; Tim Hanke; Thomas Marx-Schubach; Jörg Eiden)

WELCOME

Welcome to the Modelica Conference 2023 in Aachen!

For many years, Modelica has been an important tool at my institute for the calculation of interesting details in technical systems up to the analysis of complex energy systems. The equation-based approach, especially, allows the early integration of Modelica into the education of students. In this way, students can already develop their own models and combine them with existing models from libraries during their studies and in their final theses. This strengthens students' simulation skills and makes an important contribution to ongoing research projects.

Dynamic simulation has also established itself as a development tool in many companies. Simulation-based decisions can be made at a very early stage of product development, and initial optimizations can be implemented even before a first prototype is created. The object-oriented structure of Modelica makes it possible to develop increasingly detailed and more accurate models based on the initial approaches as the depth of development increases and by calibration using the first experimental data. And so, over the product development phases, a concept model becomes a digital twin that can be used in a variety of ways even after development.

Together with my colleagues Antonello Monti and Andrea Benigni and I am pleased to welcome you as pioneers, developers and users of Modelica in Aachen. The variety of application topics at this conference once again demonstrates the flexibility of Modelica and I am excited to see what we will learn together during the conference. I would like to take this opportunity to especially thank all the sponsors, the Modelica Association and the organizing team, led by Dominik Hering!

With best regards

Prof. Dirk Müller
Conference Chair

CONFERENCE BOARD

Prof. Dirk Müller, RWTH Aachen University, Aachen, Germany
Prof. Antonello Monti, RWTH Aachen University, Aachen, Germany
Prof. Andrea Benigni, Forschungszentrum Jülich GmbH, Jülich, Germany
Prof. Martin Otter, DLR, Oberpfaffenhofen, Germany
Prof. Francesco Casella, Politecnico di Milano, Milano, Italy
Dr. Martin Sjölund, Linköping University, Linköping, Sweden
Dr. Michael Tiller, Realis Simulation, Michigan, USA

REVIEWERS

Prof. Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany
Dr. John Batteh, Modelon, Ann Arbor, Michigan, USA
Dr. Albert Benveniste, INRIA, Rennes, France
Christian Bertsch, Robert Bosch GmbH, Stuttgart, Germany
Volker Beuter, VI-grade GmbH, Darmstadt, Germany
Dr. Robert Braun, Linköping University, Linköping, Sweden
Leonardo Carreras Rodriguez, ACS, E.ON Energy Research Center, RWTH Aachen, Aachen, Germany
Prof. Francesco Casella, Politecnico di Milano, Milano, Italy
Dr. Yan Chen, Pacific Northwest National Lab. Portland, Oregon, USA
Christoph Clauss, Dresden, Germany
Dr. Johan de Kleer, Palo Alto Research Center, Inc., Palo Alto, California, USA
Dr. Atiyah Elsheikh, Mathemodica.com
Dr. Olaf Enge-Rosenblatt, Fraunhofer IIS EAS, Dresden, Germany
Prof. Gianni Ferretti, Politecnico di Milano, Milano, Italy
Leo Gall, LTX Simulation GmbH, München, Germany
Sriram Karthik Gurumurthy, ACS, E.ON Energy Research Center, RWTH Aachen, Aachen, Germany
Prof. Anton Haumer, OTH Regensburg, Regensburg, Germany
Tobias Heins, ACS, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Dr. Dan Henriksson, Dassault Systèmes AB, Lund, Sweden
Dominik Hering, EBC, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Dr. Yutaka Hirano, Woven Planet Holdings, Inc., Tokyo, Japan
David Jansen, EBC, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Martin Kremer, EBC, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Larissa Kühn, EBC, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Dr. Alessandro Maccarini, Aalborg University, Aalborg, Denmark
Marwa Maghnie, EBC, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Laura Maier, EBC, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Martin Moraga, ACS, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Marc Moritz, ACS, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Maximilian Mork, Forschungszentrum Jülich GmbH, Jülich, Germany
Prof. Thierry S Nouidui, The United African University of Tanzania, Dar es Salaam, Tanzania
Dr. Hans Olsson, Dassault Systèmes AB, Lund, Sweden
Prof. Martin Otter, DLR, Oberpfaffenhofen, Germany
Dr. Kaustubh Phalak, Ingersoll Rand, Davidson, North Carolina, USA
Dr. Adrian Pop, Linköping University, Linköping, Sweden
Marcus Karel Richardson, The Boeing Company, Boeing, United States of America

Clemens Schlegel, Schlegel Simulation GmbH, Freising, Germany
Dr. Elena Shmoylova, Maplesoft, Waterloo, Ontario, Canada
Dr. Michael Sielemann, Modelon, München, Germany
Dr. Martin Sjölund, Linköping University, Linköping, Sweden
Prof. Dr.-Ing. Arne Speerforck, Hamburg University of Technology, Hamburg
Felix Stegemerten, EBC, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Jan Stock, Forschungszentrum Jülich GmbH, Jülich, Germany
Dr.-Ing. Wilhelm Tegethoff, TLK-Thermo GmbH, Braunschweig, Germany
Dr. Bernhard Thiele, DLR, Oberpfaffenhofen, Germany
Dr. Michael Tiller, Realis Simulation, Michigan, USA
Dr. Jakub Tobolar, DLR, Oberpfaffenhofen, Germany
Dr. Hubertus Tummescheit, Modelon, Hartford, Connecticut, USA
Eziama Ubachukwu, Forschungszentrum Jülich GmbH, Jülich, Germany
Dr. Alfonso Urquia, Universidad Nacional de Educacion a Distancia, Madrid, Spain
Dr.-Ing. Christian Vering, EBC, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Volker Waurich, TU Dresden, Dresden, Germany
Dr. Michael Wetter, Lawrence Berkeley National Laboratory, Berkeley, California, USA
Stefan Wischhusen, XRG Simulation GmbH, Hamburg, Germany
Fabian Wüllhorst, EBC, E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
Dr. Dirk Zimmer, DLR, Oberpfaffenhofen, Germany

ORGANIZING COMMITTEE

Julia Breuer
Sina Dibos
Kai Droste
Nina Franck
Sriram Karthik Gurumurthy

Vanessa Hendricks
Dominik Hering
Rahul Karuvingal
Martin Kremer
Larissa Kühn
Marwa Maghnie

Dirk Müller
Felix Stegemerten
Rita Streblow
Christian Vering
Fabian Wüllhorst
Moritz Zuschlag

KEYNOTE SPEAKERS

Dr. Dirk Zimmer
Team leader for aircraft energy system in the Institute of System Dynamics and Control (DLR e.V.)

Plenary Session 1:
Dealing with complex models and how to use the idealization of physics to our advantage.

Abstract

The complexity of models can be assessed in different ways. We can look at the complexity in terms of computational time it takes for simulation. We can also look at the complexity in terms of the underlying program size. There is a trade-off between these two forms and it is often unclear where the optimum is. How we perform this trade-off is determined by the way we choose to idealize the underlying physical system. To become better modelers, we will hence revisit the familiar schemes of idealization and then investigate a new approach that is favorable for many applications and offers new opportunities for code generation.

Dr. Bruno Lüdemann
Leiter F&E, Energiesysteme und Simulation (ROM-Technik GmbH & Co. KG)

Plenary Session 2:
Dealing with complex models and how to use the idealization of physics to our advantage.

Abstract

In the last three decades, simulations have been used more and more as a matter of course to answer a wide variety of questions for the planning and construction of buildings and technical building systems. ROM Technik, as a large construction company for technical equipment of buildings, has been using these possibilities intensively for its own projects for over 30 years and is actively involved in the development and integration of simulation tools into the overall construction process. From the lecturer’s many years of practice as research assistant and at ROM R&D, examples will be shown to illustrate the growing importance and acceptance of simulations in practice and to present the deeper anchoring in the digital construction process that is currently being strived for the future.

Contents

Session 6-A: Thermodynamic and energy systems applications 5 **589**

Session 6-B: Multi-engineering modeling and simulation with free and commercial Modelica libraries **617**

Session 6-C: Other industrial applications, such as electric drives, power systems, aerospace, etc. 3 **643**

Session 6-D: Automotive applications 1 **675**

Poster Presentation **703**

MARCO: An Experimental High-Performance Compiler for Large-Scale Modelica Models

Giovanni Agosta[1] Francesco Casella[1] Daniele Cattaneo[1] Stefano Cherubin[2] Alberto Leva[1]
Michele Scuttari[1] Federico Terraneo[1]

[1]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy,
{name.surname}@polimi.it
[2]NTNU - Norwegian University of Science and Technology, Norway, stefano.cherubin@ntnu.no

Abstract

This paper introduces MARCO, a research compiler aimed at the efficient generation of efficient simulation code from a large-scale Modelica model. MARCO's design goals, requirements, and specifications are discussed in the paper, as well as the software architecture, the current development status, and a future development roadmap. The results of two test cases demonstrate MARCO's capability to handle non-trivial Modelica models with over 10 million equations very efficiently.

Keywords: Modelica, compiler construction, large scale models

1 Introduction

The Modelica Language, first introduced in 1997, has become a widespread standard in the field of system-level modelling and simulation, and is now supported by many different tools, both commercial and open source. For many years, the focus of Modelica tools was mainly to support the modelling of individual systems, such as a robot, a power plant, an air conditioning system, a heat pump, an aircraft, and so on. Such models are built by connecting heterogeneous sub-systems belonging to different physical domains (e.g., mechanical, thermal, electrical) and often require the efficient and robust solution of non-trivial systems of non-linear equations. However, their scale usually is quite limited, from a few hundred equations up to one or two hundred thousand equations. According to how flattening is described in the language specification (Modelica Association 2021), such models are transformed into a system of scalar equations involving scalar variables.

This approach has served the Modelica community well for about 25 years, but suffers from severe performance limitations in two cases. One is the case of models containing large array equations, e.g., stemming from the 2D or 3D discretisation of distributed-parameters systems. The other is the case of systems-of-systems with many repeated instances of the same basic components. In both cases, the typical workflow of today's Modelica tools turns out to be inefficient, particularly as regards the time required to generate the executable simulation code from the original Modelica source code, and also as regards the size of the generated code, which contains many repetitions of essentially the same lines of code. This issue was highlighted eight years ago in (Casella 2015), Section 2.6, but until now, no industrial-grade solutions have been developed to overcome this problem.

This issue hampers the use of Modelica for effectively modelling systems-of-systems and large-scale, smart, distributed systems of all kinds (e.g., smart grids, smart neighbourhoods, and IoT systems in general). The Modelica language perfectly suits the task, as it can conveniently describe structured multi-domain cyber-physical systems. Still, tools are not up to the task when the size and complexity grow towards the one million equation threshold, and beyond.

For example, the French Electrical Transmission System Operator RTE decided several years ago to use Modelica to model and simulate national and continental-wide power transmission systems. However, limitations in existing Modelica technology were such that they could use Modelica to generate the code of individual components, but then had to write their own simulation engine in the Dynaωo software (Masoom et al. 2021) to build and simulate the systems of their interest at the required scale, within the time frames allotted for real-time monitoring and management of the French power grid.

To overcome these inefficiencies, four years ago some of the authors of this paper started a research line with three main goals:

1. Compile Modelica code into the simulation code with a low runtime footprint in terms of both memory and execution time, running on a range of different machines, from the workstations typically used by engineers to run their simulations, to embedded devices where Modelica models can be deployed as part of control systems.

2. Exploit arrays of variables, equations, and models as first-class citizens to drastically cut code generation time and generated code size and improve simulation run time. (Schuchart et al. 2015) (Otter and Elmqvist 2017)

DOI
10.3384/ecp20413

3. Skip the traditional C-code generation step in favour of generating LLVM-IR code that could be directly turned into highly optimised machine code.

A very preliminary experimental development was reported in (Agosta et al. 2019), together with a first tentative roadmap.

Based on that first experience, which could be classified as TRL-2 and whose results seemed promising, the development of the MARCO (Modelica Advanced Research COmpiler) compiler was started and has since then grown into a full-fledged research project involving several master's and PhD students, as well as more senior faculty with Computer Science and with Automation expertise.

The purpose of this paper is thus to present to the Modelica community the current state of the art of this project, which has now reached TRL-4, together with an updated roadmap for future work. Some interesting results obtained on non-trivial case studies will also be reported.

The paper is structured as follows: in Section 2, the MARCO compiler's objectives, requirements, and specifications are stated. Section 3 briefly describes the compiler architecture and its current development status. Section 4.3 reports the results of two non-trivial case studies inspired by real-life applications. Section 6 concludes the paper with some final remarks.

2 Goals, Requirements, Specifications

The main goal of MARCO is to experiment with methodologies and algorithms to generate the fastest possible executable code from large Modelica models, and to do it quickly and efficiently. The medium-term objective is to handle models with one million to ten million differential-algebraic equations (DAEs), eventually reaching 100 million in the long term, although this may require more fundamental breakthroughs.

At the time of writing, MARCO is not primarily meant to be a production-grade compiler. As such, it does not aim at covering the complete range of models that can be written in Modelica. The idea is to *demonstrate* the capability of generating fast executable code fast on a subset of large-scale system models that can be written in the Modelica language, that are however relevant for industrial application domains. This sub-set could then be progressively enlarged as time passes, possibly – but not necessarily – covering the full range of models that can be written using the Modelica language. At some point, MARCO could turn into industrial-grade software, or alternatively be used as a research prototype for implementing such software; it is currently too early to say that.

This project is specifically interested in monitoring and improving metrics related to the quality of the tool, on top of the quality of the result. Thus, MARCO aims at optimising the time-to-solution, which is to be intended as the sum of the time required to generate an executable simulation, plus the proper simulation time of a Modelica model.

Efficient handling of arrays of models and equations is an essential feature to fulfil this goal. Arrays should be handled as first-class citizens throughout the entire toolchain, avoiding expanding them into their scalar constituents, unless strictly required, thus shortening the structural analysis time and the executable code generation time.

To generate efficient runtime simulation code, the mathematical structure of the problem should be preserved as much as possible throughout the toolchain, and exploited during its latest stages to allow the generation of more efficient machine code.

One important point when handling very large systems with over a million variables and equations concerns handling the simulation results. The default behaviour of Modelica compilers is to save all the variable values at each reporting time step, possibly skipping protected components. However, for such large-sized models, this approach easily leads to massive, multi-GB-sized simulation results files, which are unnecessarily cumbersome and largely useless since most of those variables have little or no specific interest for end-users that generally on a relatively small subset of relevant output variables.

The idea is then not only to avoid wasting CPU time and disk space to store all the simulation results but actually to structure efficient simulation code around the fact that only some variables (which can be declared, e.g., as top-level outputs or listed in a custom annotation) are interesting for the end user. For example, if a certain variable is only of interest as an intermediate computation step towards the computation of the state derivatives, the generated code could only store it in some CPU registers so that not only the time to save it to disk is saved, but also the time to shuffle it back and forth from the CPU cache to RAM. In some cases, the computation of certain variables could even be skipped outright.

Along the same line, Modelica compilers usually generate code that allows changing parameter values at runtime without re-building the simulation executable from scratch. This is of course essential if the build time is comparable or even larger than the simulation run time, as it often is with current Modelica tools. However, this has a price in terms of less efficient simulation code because of additional indirection and memory access, as well as leaving less room for extreme machine-code optimisations.

Also, during typical simulation-based studies (including parameter optimisation), only a relatively small number of parameters are subject to change; these are a tiny fraction of the complete set of parameters for million-equations models, which could count tens of thousands or more parameters. Since the goal of MARCO is to generate code which is as fast as possible and to do it as fast as possible, all parameters that are determined by binding equations should be constant-evaluated at compile time. Although it should remain possible to make some exceptions to support parameter-sweeping or parameter-optimisation studies, the (few) parameters *not* to be constant-evaluated should be declared explicitly.

3 Current Status of MARCO

In this section, we provide an overview of the current status of MARCO with respect to the objectives we are aiming to achieve. First, we focus on the software architecture chosen, which reflects the state-of-the-art in compiler design and construction. Then we follow with a discussion of the features currently supported by the compiler, both in terms of Modelica language features, and in terms of the set of runtime solvers currently supported for the simulation code generation.

3.1 Software Architecture

MARCO is written using the C++ language, and it is based on the technologies developed within the LLVM project (Lattner and Adve 2004). The LLVM project is a collection of modular and reusable compiler technologies. Its most important part are the LLVM core libraries (often simply referred as LLVM), which provide a modern target-independent optimizer and code generator for an increasing amount of processor architectures. LLVM is a mature project heavily used both in the industry and in compiler research, and in MARCO we rely on it to implement the back-end of the compiler, which therefore outputs machine code directly instead of C code. Is is worth noting that using LLVM and its intermediate representation (LLVM-IR) enables the reuse of the backend optimisations provided by LLVM itself and the possibility of targeting different architectures – i.e. ARM-based embedded systems and not just PCs based on Intel processors – without the need to implement any additional transformations.

The front-end of MARCO is based on MLIR, also part of the LLVM project, which represents a novel approach to building reusable and extensible compiler infrastructures (Lattner, Amini, et al. 2021). Before MLIR, compiler front-ends were often built from the ground up, because different language features call for different internal data structures for the code – or, in other words, different intermediate representations. However, while these intermediate representations may differ from each other, there is a set of common abstract tasks performed on such representations that does not depend on the semantic of each operation in the code. MLIR provides a construction set, so-to-speak, where the compiler developer only has to declaratively define the set of domain-specific intermediate representations they need – called *dialects* – based on simple concepts like *operations*, *types*, and *attributes*. Additionally, MLIR provides built-in dialects for semantics that history has shown to be common amongst multiple programming languages. The implementation of new dialects and the combination of existing ones contribute to the definition of Multiple Layers of Intermediate Representations of the code, from which the MLIR acronym stems. In summary, MLIR allows to build compilers with less human effort, providing a library of primitives that previously needed to be rewritten from scratch for each different language.

From a high-level point of view, MARCO is composed of multiple libraries organized as a pipeline. This pipeline is overall similar to the familiar one known from the literature (Cellier and Kofman 2006) and already employed in other state-of-the-art Modelica compilers like the OpenModelica Compiler. In our compiler design, however, all the steps required for the causalization of the model are performed through successive transformations of a new MLIR dialect explicitly devised for the Modelica language. In addition, at the end of the pipeline, the causalized model in MLIR dialect form is translated into LLVM-IR code, the intermediate representation (IR) used by LLVM. Then, we exploit LLVM to translate such code into an object file, which is then linked with the MARCO runtime library to obtain the executable simulation. The translation to LLVM-IR exploits the existing MLIR built-in dialects and transformations to the maximum possible extent, greatly reducing the workload required for its implementation.

The MARCO runtime library is also written in C++, and serves two purposes: the first is to provide the implementations for functions that are inconvenient to be represented directly using LLVM-IR; the second is to actually drive the simulation process, by leveraging other functions which are instead emitted during the compilation process, which typically provide information about the compiled model. It is worth noticing how, even if not strictly necessary for the generation of the simulation, the MARCO runtime library enables faster development and testing, together with the possibility of using more complex solutions – like multithreading – that would otherwise be way more difficult to handle.

3.2 Arrays & Flattening

The first step of the process that transforms a Modelica model into executable simulation code (Fritzson 2014) is the so-called *flattening* (Modelica Association 2021). During flattening, the models with their variables, parameters, and equations are instantiated according to the rules that govern name lookup, inheritance, and modular composition of Modelica models. This first step results in a set of variable and parameter declarations, a set of record type definitions, and a set of hybrid DAEs.

The fundamental requirement for preserving arrays as first-class citizens is to carry out the flattening without expanding array variables into their scalar constituents and without unrolling array or for-loop equations into their scalar requirements.

Given the complexity of the Modelica language, this first step is rather involved and would require substantial development effort. Luckily, recent advances in the development of the OpenModelica Compiler (OMC), namely the new OMC front-end (Pop et al. 2019) provide this functionality out of the box. The new OMC front-end provides more than adequate performance also for very large models, as long as they are built by instantiating large arrays of a comparably limited number of classes, which is typically the case in many systems-of-systems and smart grid applications. In this case, most of the flattening effort can be performed once for an array of components that may count hundreds or thousands of elements, thus slashing the flattening time dramatically.

Additionally, the new OMC front-end can also process models that contain a large number of individual instances of the same class with the same modifier structure, automatically collecting them in arrays before proceeding with the rest of the flattening process. This allows to process models of systems such as power grids (Bartolini, Casella, and Guironnet 2019), gas networks (De Pascali et al. 2022), or district heating networks (Long et al. 2021), that can be built automatically by translating graph-based system descriptions into Modelica system models, eventually transforming them into array-based models.

Recent advances in the definition of Base Modelica, formerly known as Flat Modelica (*MCP-0031: Base Modelica and MLS modularization* 2023), were used to interface the OMC new frontend and the MARCO compiler, with some extensions to support array-preserving model descriptions. Specifically, declarations of arrays of models are turned into the corresponding declarations of arrays of variables, where the variables of the model array become array variables; accordingly, the equations of the model array become array equations, declared via for-loops covering the entire array index range, see (Casella 2023) for some concrete examples.

MARCO thus accepts array-preserving Base Modelica textual models as inputs. Using a textual interface may be somewhat less efficient than directly accessing the OMC frontend internal data structures. On the other hand, relying on a high-level, reasonably stable, human-readable interface, which is presumably going to become a Modelica Association standard eventually, seems to be the best option for long-term development, without running the risk of relying on low-level on features that may change or become obsolete in the future.

This decision allowed to focus the development of MARCO on the current bottlenecks of the typical Modelica simulation workflow for very large models, namely the structural analysis, the code generation, and the runtime execution.

3.3 Supported Modelica Features

Therefore, MARCO relies on the OMC's new front-end for flattening, which is a complete, efficient implementation of the Modelica Language Specification, fully supporting the Modelica Standard Library. From this point of view, MARCO can accept models built with the most sophisticated features of Modelica, such as replaceable classes, conditional components, overconstrained connectors, stream variables, etc., which will be converted into flat hybrid DAE systems, possibly involving multi-dimensional arrays of variables.

Current limitations in the range of the models that MARCO can turn into efficient simulation code thus only regard the *mathematical* structure of the model rather than its *object-oriented* structure.

MARCO only supports continuous-time variables and equations at the time of this writing. Although we acknowledge that this limitation is particularly severe for practical applications, on the other hand, MARCO already enables us to demonstrate the scaling capabilities of the compiler with respect to the model's size. Support of discrete variables, event-handling, if-equations and when-equations is planned for the near future.

Thanks to recently developed array-based extensions of matching and sorting algorithms (Fioravanti et al. 2023), MARCO can very efficiently handle the causalization of array-based DAEs, including multi-dimensional arrays. The output of this phase is the matching of continuous slices of these arrays with corresponding for-loop equations and the ordering of their solution in Block Lower Triangular (BLT) form.

In fact, one interesting result proven in (Fioravanti et al. 2023) is that the optimal matching problem in the case of arrays (where optimal means that the arrays slices and corresponding array-equations should have the maximum possible size) is in general an NP-complete problem. Heuristics were then developed to efficiently handle roto-translation of index – i.e., cases where the equations in for-loop involve exchanging indexes and adding fixed offsets, as shown in Listing 1.

Listing 1. Example of array equations with fixed offset

```
  Real x[N, M];
  Real y{N, M];
equation
  for i in 2:N-1 loop
    for j in 1:M loop
      x[i,j] = y[j, i + 1] + x[i - 1, j];
    end for;
  end for;
```

MARCO supports arbitrary Modelica functions, possibly with inlining, with the exception of external functions, whose support is planned for the future. It can also differentiate functions using AD techniques (Neidinger 2010) whenever needed for Jacobian computations.

The support for records is currently being implemented, including the support of operator records, which is necessary for power system models using Complex numbers, a potentially very interesting application, in the near future.

At the time of writing, index reduction, dummy derivatives and state variable changes are not yet supported. Although this lack also represents a severe limitation for a Modelica compiler, there are some interesting application fields – e.g., modelling the thermal dynamics of buildings and district heating systems, as well as electro-mechanical modelling of power transmission and distribution systems – where these features are not needed.

3.4 Runtime Solvers

Regarding the runtime solvers, the initial goal of MARCO is to demonstrate its potential in two categories of application scenarios.

The first one involves non-stiff models that may be simulated with explicit ODE integration methods. This is also useful for co-simulation or real-time simulation, possibly running on embedded hardware using FMI or e-FMI. In this case, fixed-time-step explicit Euler's method is used. More sophisticated higher-order explicit integration methods such as Runge-Kutta could be implemented, but they do not represent a priority as long as MARCO remains a technology demonstrator rather than a full-fledged production-level tool.

In this context, it may be necessary to solve algebraic loops at each time step corresponding to strong components in the BLT. Currently, MARCO is restricted to small linear systems that can be solved efficiently in closed-form by symbolic manipulation. The integration of sparse linear (KLU) and nonlinear (Kinsol) solvers with symbolic Jacobian code generation is planned for the near future.

The second scenario instead involves systems which are stiff or involve large algebraic systems of equations. In this case, the design choice was to rely on the open-source DAE solver IDA from the Sundials tool suite (Gardner et al. 2022), which provides the efficient solution of large, sparse DAE models using BDF algorithms, with adaptive step size and error control.

IDA optionally requires the (sparse) Jacobians of the DAE formulation of the system with respect to all the variables and to the state derivatives to solve the implicit BDF formula, and it goes without saying that an overall efficient implementation requires computing such Jacobians analytically, to reduce the time spent computing Jacobians and also to avoid unnecessary iterations of the BDF solver caused by poor Jacobians. MARCO is thus endowed with automatic differentiation algorithms and generates efficient code to compute the Jacobians required by IDA.

To reduce the size of the implicit system that IDA needs to solve at each iteration of the solution of the BDF formula, the results of the causalization algorithm are exploited: instead of passing to the IDA solver the complete DAE system $F(x,\dot{x},v,t) = 0)$, where x is the vector of state variables, v the vector of all algebraic variables, and t the time variable, a reduced system of equations $F_r(x,\dot{x},w,t)$ is passed instead, where w is the vector of the algebraic variables that are unknowns of implicit systems; the other algebraic variables are computed by sequences of assignments that correspond to the explicit solutions of equations that have 1×1 blocks on the BLT diagonal (Scuttari et al. 2023). In other words, IDA is directly used to solve the linear and nonlinear implicit algebraic equations and the (stiff) differential equations, while the results of the causalization steps are used to compute all the other variables explicitly in the generated code.

While a single thread currently carries out the sequential part of the residual computation, the subsequent computation of the residuals matched to $\dot{x}$ and w and of the Jacobian element is carried out by parallel threads, since they can be computed independently. In the future, also the sequential part could be parallelised.

Initial equations are also solved using IDA, which acts as an interface to the underlying sparse Kinsol solver. Currently, MARCO can only handle square non-singular initialisation problems, where the number of initial equations matches the number of differentiated variables plus the number of `fixed = false` parameters. The solution of under and over-determined initialisation problems, which is closely related to index reduction and dummy derivatives, is currently not yet supported.

Last but not least, MARCO only outputs to the CSV result file the top-level output variables of the model. An extension of the array-aware matching and sorting algorithm along the lines of (Manzoni and Casella 2011) could identify the system equations and variables that are strictly needed to compute the state derivatives and the top-level system outputs, allowing to skip the computation of all other algebraic variables defined in the model. This feature, which is planned for the near future, will further optimise the simulation time.

4 Case Studies

In this section, the results of two case studies are reported to demonstrate the current capabilities of the MARCO compiler. These case studies are motivated by real-life applications; they are simple enough to be contained in a few dozen lines of code (see the Appendix) but are nevertheless definitely non-trivial to handle, in particular as regards the need for matching slices of the arrays to subsets of for-loop equations involving them. Also, both case studies are easily scalable via parameters to test the tool's performance with the increasing model size.

4.1 3D Thermal Model of a Microchip

Modern microprocessors feature higher and higher power density, requiring more and more advanced fluid-based cooling systems. These models combine 0D and 1D cooling system models, which are conveniently represented in Modelica, with 3D thermal models of the microchip body and heat sink body, which need high spatial definition to identify potentially harmful hot spots.

This is currently achieved by co-simulation set-ups (Terraneo et al. 2022), where the microchip thermal dynamics are simulated by a separate dedicated simulation tool. However, it would be quite convenient to embed a detailed 3D thermal model of the microchip directly within the Modelica model, avoiding the complication and inconvenience of the co-simulation setup.

This first case study thus demonstrates the capability of MARCO to handle high spatial resolution 3D thermal models of solid bodies. The 3D thermal model is built in a fully object-oriented way, by first defining an elementary 0D Volume model, with a lumped thermal capacitance in the middle and 6 thermal conductances and 6 thermal ports in the directions east, west, north, south, up, and down. The microchip thermal model is assembled by instantiating a 3D $N \times M \times P$ array of such 0D models and connecting them via for loops.

This approach leads to a very compact Modelica source code with just three for loops, one for each orthogonal spatial direction. The alternative would be to write the discretised 3D heat equations directly in the body of the model, but that requires handling the inner volumes, the face volumes, the edge volumes, and the corner volumes differently, leading to a much longer and error-prone code with many more for loops, and a much higher likelihood of making some mistakes with the loop indices when writing the equations for the various cases.

Furthermore, with the full object-oriented approach, one can connect heat sources or other thermal objects of arbitrary shape to *portions* of the outer faces of the 3D microchip model, e.g. representing specific active semiconductor areas on the microchip surface; unconnected outer faces are automatically considered as thermally insulated, thanks to the default connection equations `Q_flow = 0` generated by the compiler for unconnected thermal ports. Handling the non-trivial geometry of such active areas without using connection equations becomes extremely complicated and counter-intuitive, as one would also need to write specific for-loop equations for each rectangular *thermally insulated* region.

In the present case study, for simplicity, half of the lower face of the chip (corresponding to the active semiconductor area) was connected to a uniformly distributed 2D heat source, while the other half was left unconnected, and thus insulated. More elaborate setups could be conceived, e.g. representing active cores on the chip surface.

The upper face of the microchip is instead connected to a 2D fixed temperature source, corresponding to the surface of the heat sink block. A more realistic model could include a full thermal model of the heat sink and its cooling system.

This object-oriented model contains a huge number $N_c = O(NMP)$ of connection equations; each face-to-face connection of two adjacent 0D blocks generates a small system of linear equations, corresponding to the series connection of the two half-conductances of the adjacent 0D blocks in that direction. However, these systems can be easily solved in closed form, corresponding to the well-known formula for the conductance of series-connected conductors, allowing to explicitly compute the heat flow between the capacitances of adjacent 0D blocks in each spatial direction *without even computing the temperature at the block boundaries*. Additionally, thanks to the array-preserving nature of MARCO, this symbolic solution needs to be carried out only once during code generation, so it takes a negligible amount of time.

For the sake of this simple case study, only eight output temperatures were computed and saved, namely the temperatures at the four corners of the upper and lower faces of the microchip. This enabled the comparison of the simulation results with those obtained with OpenModelica.

The thermal microchip model was simulated both by explicit Euler's algorithm and by IDA, in a test case of increasing size, up to $M = N = 96$, $P = 32$, which leads to a model with over 4 million DAEs and about 250,000 state variables. The simulation starts at thermal equilibrium with zero thermal power input and simulates the response of the system to a step increase of the thermal flux applied to half of the bottom face of the microchip.

4.2 Heat Exchanger Network with Methanol

Another interesting field of application that can easily lead to large-scale models is thermo-fluid systems, as found in large industrial plants, district heating systems, and smart distributed energy systems in general. When modelling such systems, non-trivial fluid property models are often needed and computed using functions.

The second test case tries to capture the main features of these applications in a simple and scalable test model. The model contains a 2D $N_u \times N_h$ array of heat exchangers, which are arranged in N_u sequential rows, each containing N_h parallel heat exchangers, whose outlet flows are then mixed before being distributed to the next row. Each heat exchanger is then modelled with N_v finite volumes. The mass flow rates and heat flows of each heat exchanger are time-varying, and set up in a way that guarantees that no two heat exchangers ever operate at the same temperature. Overall, the number of variables and DAEs of the system model is $O(N_u N_h N_v)$.

The compressibility of the fluid inside the heat exchangers is neglected for simplicity, leading to trivial mass balance equations. On the other hand, an accurate model of the relationship between the temperature and the specific energy and enthalpy was developed using Modelica functions, using results from (Craven and de Reuck 1986).

4.3 Experimental Results

The results of the simulations of medium-size models were successfully compared with the simulation results obtained with the OpenModelica tool, using the same solution algorithm, and were found to agree with the results produced by MARCO with high accuracy. Then, MARCO was used to simulate the much larger instances of the test cases mentioned in the previous two sub-sections, which are beyond the current capabilities of the OpenModelica compiler.

All tests were conducted on a server with an i9-12900KF Intel processor and 96 GB of RAM, running Linux Ubuntu 20.04 LTS. At the time of this writing, the results obtained with IDA, although correct, are still not efficient as expected, so the results summarised in Table 1 and shown in Figures 1 and 2 only refer to explicit Euler's method. Results with IDA are expected to be available for the final revision of this paper.

A direct comparison of the performance of MARCO against other Modelica tools is beyond the scope of this

Table 1. Compilation and simulation times for largest simulated models.

	Equations	Steps	Compilation time [s]	Simulation time [s]
Heat exchangers network	14776202	40000	10.13	33574.76
3D thermal chip	4465160	60000	86.24	605.47

paper. It is worth mentioning, though, that the authors are not aware of any other Modelica tool which is currently able to handle models with 15 million equations or at least to do so with compile times of a few tens of seconds on low-cost hardware (a 1,500 € gaming machine).

5 Roadmap

The results presented in the previous section demonstrate that the MARCO compiler can handle non-trivial, array-based, very large models with very short compile times and good runtime performance, on a scale of model sizes currently inaccessible to mainstream Modelica tools.

On the other hand, the class of models that can be currently handled is minimal. Future development work is thus planned in different directions.

Record and operator record handling is currently being addressed and could be completed in time for the final version of this paper. Combined with the support of hybrid systems, this could make MARCO capable of handling large-scale power system models (Bartolini, Casella, and Guironnet 2019), potentially allowing it to replace the parts of Dynaωo (Masoom et al. 2021) that currently take care of assembling the whole system model starting from the generated C code of individual components.

The addition of external function handling could also allow tackling models of advanced microchip cooling systems, including detailed 3D thermal dynamics end using refrigerant models from the ExternalMedia library (Casella and Richter 2008).

FMI export and embedded code generation are other promising areas of development.

Finally, handling index reduction, dummy derivatives, and under/overconstrained initialisation problems could prove to be very hard. One option for index reduction, as already planned in (Agosta et al. 2019), is to solve these problems on a fully scalarised set of equations, adding the extra differentiated scalar equations to the system that are needed to make it index-1. All other equations would still be handled in an array-preserving way, still leading to a substantial performance advantage compared to the traditional flat-scalar equation tools.

In the long term, we plan to leverage support from both the Modelica and LLVM communities. To this end, we plan to release MARCO as an open-source project once record handling and hybrid system support are available, providing the capability to address a sufficiently large number of real-world large-scale problems.

6 Conclusions

This paper introduced the MARCO compiler, which is currently developed at the Dipartimento di Elettronica, Informazione e Bioingegneria of Politecnico di Milano in collaboration with Edinburgh Napier University. MARCO aims at demonstrating algorithms and methodologies to compile large-scale Modelica models efficiently, producing fast binary code. It accepts flat, array-preserving Base Modelica code as input, produced by the new front-end of the OpenModelica compiler, and causalizes it using novel array-preserving matching and sorting algorithms. Executable code is generated using the LLVM framework:

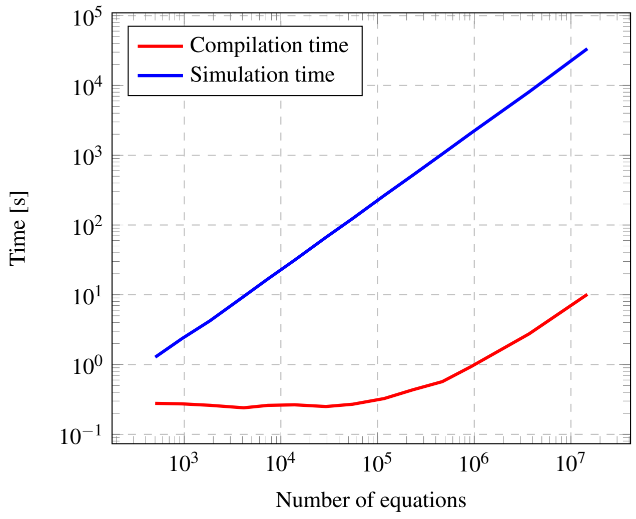


Figure 1. Heat exchangers network model

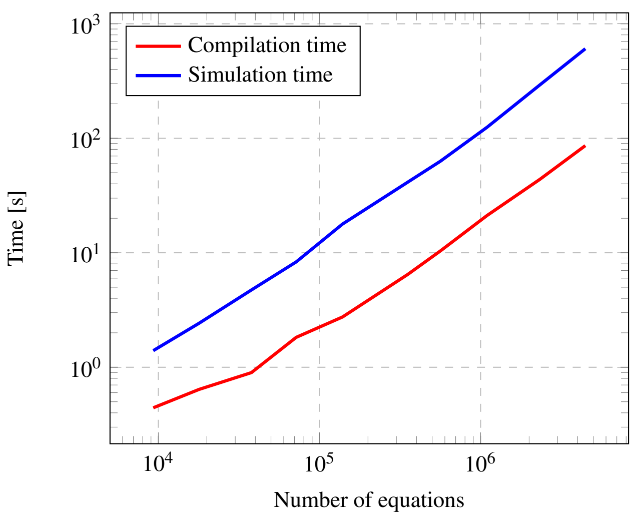


Figure 2. 3D thermal chip model

an LLVM-IR is first produced by the compiler and then directly turned into efficient, architecture-optimised executable code.

Thanks to the complete support of the Modelica language provided by the OpenModelica front-end, MARCO is capable of handling Modelica code using all the advanced object-oriented features of the language. Its current limitations regard the mathematical structure of the flat system, which currently needs to be an index-1, purely continuous-time dynamical system, possibly involving Modelica functions.

The generated code can simulate dynamical systems using explicit Euler's algorithm or by using the IDA DAE solver, in which case the code to compute symbolic Jacobians is also generated.

The current capabilities of the MARCO compiler were demonstrated on two large-scale test cases: 3D object-oriented thermal models of a microchip with up to 4 million equations, and equation-based models of networks of heat exchangers with a detailed function-based fluid model, with up to 15 million equations. In both cases, the compilation time is at most a few tens of seconds and is one or more orders of magnitude less than the run time. To the authors' knowledge, no other Modelica tool can handle Modelica models at this scale.

Future developments of MARCO in the short term regard the implementation of operator records and event handling, at which point the release of MARCO as open-source software is planned. These two additional features will enable the compilation and simulation of national- and continental-scale power system models such as those of the ScalableTestGrids library (Bartolini, Casella, and Guironnet 2019).

Medium- and long-term developments include supporting external functions, code generation for embedded hardware, FMI export, index reduction, and under/over-constrained initialisation problems.

Acknowledgements

The authors are grateful to former MSc students Massimo Fioravanti and Nicola Camillucci, who contributed to the development of the MARCO code base during the last three years.

References

Agosta, Giovanni et al. (2019-03). “Towards a High Performance Modelica Compilers”. In: *Proc. 13th International Modelica Conference*. Ed. by Anton Haumer. Regensburg, Germany, pp. 313–320. DOI: 10.3384/ecp19157313.

Bartolini, Andrea, Francesco Casella, and Adrien Guironnet (2019-03). “Towards Pan-European Power Grid Modelling in Modelica: Design Principles and a Prototype for a Reference Power System Library”. In: *Proc. 13th International Modelica Conference*. Ed. by Anton Haumer. Regensburg, Germany, pp. 627–636. DOI: 10.3384/ecp19157627.

Casella, Francesco (2015-09). “Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives”. In: *Proceedings 11th International Modelica Conference*. Ed. by Peter Fritzson and Hilding Elmqvist. The Modelica Association. Versailles, France, pp. 459–468. ISBN: 978-91-7685-955-1. DOI: 10.3384/ecp15118459.

Casella, Francesco (2023). *Simulation of large-scale Modelica models with array-preserving technology: early results and perspectives*. URL: https://tinyurl.com/OMWorkshopKeynote2023 (visited on 2023-05-24).

Casella, Francesco and Christoph C. Richter (2008-03). “ExternalMedia: a Library for Easy Re-Use of External Fluid Property Code in Modelica”. In: *Proceedings 6th International Modelica Conference*. Ed. by Bernhard Bachmann. Modelica Association. Bielefeld, Germany, pp. 157–161. URL: http://www.modelica.org/events/modelica2008/Proceedings/sessions/session2b1.pdf.

Cellier, F. E. and E. Kofman (2006). *Continuous System Simulation*. Springer-Verlag.

Craven, R. J. B. and K. M. de Reuck (1986). “Ideal-Gas and Saturation Properties of Methanol”. In: *International Journal of Thermophysics* 7.3, pp. 541–552.

De Pascali, Matteo Luigi et al. (2022-06). “Flexible object-oriented modelling for the control of large gas networks.” In: *Proc. 11th IFAC Symposium on Control of Power and Energy Systems, IFAC PapersOnLine*. Vol. 55. 9. Online, pp. 315–320. DOI: 10.1016/j.ifacol.2022.07.055.

Fioravanti, Massimo et al. (2023-07). “Array-Aware Matching: Taming the Complexity of Large-Scale Simulation Models”. In: *ACM Trans. Math. Softw.* Just Accepted. ISSN: 0098-3500. DOI: 10.1145/3611661. URL: https://doi.org/10.1145/3611661.

Fritzson, P. (2014). *Principles of Object Oriented Modeling and Simulation with Modelica 3.3*. Wiley IEEE Press.

Gardner, David J et al. (2022). “Enabling new flexibility in the SUNDIALS suite of nonlinear and differential/algebraic equation solvers”. In: *ACM Transactions on Mathematical Software (TOMS)*. DOI: 10.1145/3539801.

Lattner, Chris and Vikram Adve (2004). “LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation”. In: *Proc. Int'l Symp. on Code Generation and Optimization*. Palo Alto, California. ISBN: 0-7695-2102-9.

Lattner, Chris, Mehdi Amini, et al. (2021). “MLIR: Scaling Compiler Infrastructure for Domain Specific Computation”. In: *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 2–14. DOI: 10.1109/CGO51591.2021.9370308.

Long, Nicholas et al. (2021-09). “Modeling district heating and cooling systems with URBANopt, GeoJSON to Modelica Translator, and the Modelica Buildings Library,” in: *Proceedings of the 17th IBPSA Conference*. Bruges, Belgium, pp. 2187–2194. DOI: 10.26868/25222708.2021.30943.

Manzoni, Vincenzo and Francesco Casella (2011-03). “Minimal Equation Sets for Output Computation in Object-Oriented Models”. In: *Proceedings 8th International Modelica Conference*. Ed. by C. Clauss. Modelica Association. Dresden, Germany, pp. 784–790. ISBN: 978-91-7393-096-3. DOI: 10.3384/ecp11063784. URL: http://www.ep.liu.se/ecp/063/088/ecp11063088.pdf.

Masoom, Alireza et al. (2021-08). “Modelica-based simulation of electromagnetic transients using Dynaωo: Current status and perspectives”. In: *Electric Power Systems Research* 197, p. 107340. DOI: 10.1016/j.epsr.2021.107340.

MCP-0031: Base Modelica and MLS modularization (2023). URL: https://github.com/modelica/ModelicaSpecification/tree/MCP/0031/RationaleMCP/0031 (visited on 2023-05-24).

Modelica Association (2021-02). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.5*. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.5/MLS.html.

Neidinger, Richard D (2010). "Introduction to automatic differentiation and MATLAB object-oriented programming". In: *SIAM review* 52.3, pp. 545–563.

Otter, Martin and Hilding Elmqvist (2017). "Transformation of Differential Algebraic Array Equations to Index One Form". In: *Proceedings of the 12th International Modelica Conference*. Linköping Electronic Conference Proceedings. Linköping University Electronic Press. URL: https://elib.dlr.de/117431/.

Pop, Adrian et al. (2019-03). "A New OpenModelica Compiler High Performance Frontend". In: *Proc. 13th International Modelica Conference*. Ed. by Anton Haumer. Regensburg, Germany, pp. 689–698. DOI: 10.3384/ecp19157689.

Schuchart, Joseph et al. (2015). "Exploiting repeated structures and vectorization in modelica". In: *Proc. of the 11th Int. Modelica Conference, Versailles. www. ep. liu. se/ecp/118/028/ecp15118265. pdf*.

Scuttari, Michele et al. (2023). "Clever DAE: Compiler Optimizations for Digital Twins at Scale". In: *2nd Annual Compiler Frontiers Workshop*. ACM Press. DOI: 10.1145/3587135.3589945.

Terraneo, Federico et al. (2022). "3D-ICE 3.0: Efficient Nonlinear MPSoC Thermal Simulation With Pluggable Heat Sink Models". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.4, pp. 1062–1075. DOI: 10.1109/TCAD.2021.3074613.

TestCases (2023). URL: https://casella.faculty.polimi.it/transfer/MARCO_Tests.zip (visited on 2023-05-24).

Appendix: Source code of the test cases

The code of the two test cases is reported in this Appendix for the reader's convenience. The code has been edited for conciseness, the full packages can be downloaded from (*TestCases* 2023).

Listing 2. Source code of the 3D thermal model of a microchip

```
package ThermalChipOO
 package Interfaces
  connector HeatPort
   Real T;
   flow Real Q;
  end HeatPort;
 end Interfaces;

 package Models
  model Volume
   parameter Real lambda = 148;
   parameter Real rho = 2329;
   parameter Real c = 700;
   parameter Real Tstart = 273.15 + 40;
   parameter Real C, Gx, Gy, Gz;
   Interfaces.HeatPort upper, lower;
   Interfaces.HeatPort left, right;
   Interfaces.HeatPort top, bottom;
   Interfaces.HeatPort center;
   Real T(start = Tstart, fixed = true);
  equation
   C*der(T) = upper.Q + lower.Q +
              left.Q + right.Q +
              top.Q + bottom.Q + center.Q;
   upper.Q  = Gx*(upper.T  - T);
   lower.Q  = Gx*(lower.T  - T);
   left.Q   = Gy*(left.T   - T);
   right.Q  = Gy*(right.T  - T);
   top.Q    = Gz*(top.T    - T);
   bottom.Q = Gz*(bottom.T - T);
   center.T = T;
  end Volume;

  model TemperatureSource
   Interfaces.HeatPort port;
   Real T = 298.15;
  equation
   port.T = T;
  end TemperatureSource;

  model PowerSource
   Interfaces.HeatPort port;
   input Real Q;
  equation
   port.Q = -Q;
  end PowerSource;

  partial model BaseThermalChip
   parameter Integer N;
   parameter Integer M;
   parameter Integer P;
   parameter Real L = 12e-3;
   parameter Real W = 12e-3;
   parameter Real H = 4e-3;
   parameter Real lambda = 148;
   parameter Real rho = 2329;
   parameter Real c = 700;
   parameter Real Tstart = 273.15 + 40;
   parameter Real l = L/N;
   parameter Real w = W/M;
   parameter Real h = H/P;
   parameter Real Tt = 273.15 + 40;
   parameter Real C = rho*c*l*w*h;
   parameter Real Gx = lambda*w*h/l;
   parameter Real Gy = lambda*l*h/w;
   parameter Real Gz = lambda*l*w/h;

   Volume vol[N,M,P](
    each T(start = Tstart, fixed = true),
    each C = C, each Gx = 2*Gx,
    each Gy = 2*Gy, each Gz = 2*Gz);

   TemperatureSource Tsource[N,M]
    (each T = Tt);

   output Real Tct1 = vol[1,1,1].T;
   output Real Tct2 = vol[1,N,1].T;
   output Real Tct3 = vol[N,N,1].T;
   output Real Tct4 = vol[N,1,1].T;
   output Real Tcb1 = vol[1,1,P].T;
   output Real Tcb2 = vol[1,N,P].T;
   output Real Tcb3 = vol[N,N,P].T;
   output Real Tcb4 = vol[N,1,P].T;
  equation
   for i in 1:N loop
    for j in 1:M loop
     connect(vol[i,j,1].top,
             Tsource[i,j].port);
     for k in 1:P-1 loop
      connect(vol[i,j,k].bottom,
              vol[i,j,k+1].top);
     end for;
    end for;
   end for;
   for i in 1:N loop
    for k in 1:P loop
     for j in 1:M-1 loop
      connect(vol[i,j,k].right,
              vol[i,j+1,k].left);
     end for;
    end for;
   end for;
   for j in 1:M loop
    for k in 1:P loop
     for i in 1:N-1 loop
      connect(vol[i,j,k].lower,
              vol[i+1,j,k].upper);
     end for;
    end for;
   end for;
  end BaseThermalChip;

  model ThermalChipSimpleBoundary
   extends BaseThermalChip;
   parameter Real Ptot = 100;
   parameter Real Pv = Ptot/(N*M/2);
   PowerSource Qsource[N,div(M,2)]
    (each Q = Pv);
  equation
   connect(Qsource.port,
           vol[:,1:div(M,2),P].center);
  end ThermalChipSimpleBoundary;
 end Models;
end ThermalChipOO;
```

Listing 3. Source code of the heat exchanger network

```
package MethanolHeatExchangersDAE
 package Models
  model MethanolHeatExchangers
   parameter Integer Nu = 3;
   parameter Integer Nh = 4;
   parameter Integer Nv = 6;
   parameter Real w_nom = 1;
   parameter Real Q_nom = 500e3;
   parameter Real f_w = 1/30;
   parameter Real f_Q = 1/100;
   parameter Real T0 = 493.15;
   parameter Real V = 1;
   parameter Real beta = 0.01;
   parameter Real UA_nom = 10000;
   parameter Real alpha = 0.8;
   parameter Real Cw = 10000;
   parameter Real p_nom = 20e5;
   parameter Real V_v =
    V*(1-beta)/(Nu*Nh*Nv);
   parameter Real V_m = V*beta/Nu;
   parameter Real C_wv = Cw/(Nu*Nh*Nv);
   constant Real pi = 3.14159265359;
   Real w, w_h;
   Real Q[Nh], Q_c[Nu,Nh,Nv];
   Real T[Nu,Nh,Nv+1];
   Real h[Nu,Nh,Nv+1], h_m[Nu];
   Real T_tilde[Nu,Nh,Nv]
    (each start = T0, each fixed = true);
   Real T_w[Nu,Nh,Nv]
    (each start = T0, each fixed = true);
   output Real T_m[Nu]
    (each start = T0, each fixed = true);
   Real rho[Nu,Nh,Nv], rho_m[Nu];
   Real cv[Nu,Nh,Nv],  cv_m[Nu];
  equation
   w = w_nom*(1 + 0.2*sin(2*pi*f_w*time));
   w_h = w / Nh;
   for j in 1:Nh loop
    Q[j] = Q_nom/(Nu*Nh)*
      (1 + sin(2*pi*f_Q*time + 2*pi*j/Nh));
   end for;
   for j in 1:Nh loop
    T[1,j,1] = T0;
   end for;
   for i in 2:Nu loop
    for j in 1:Nh loop
     T[i,j,1] = T_m[i-1];
    end for;
   end for;
   T_tilde = T[:,:,2:Nv + 1];
   for i in 1:Nu loop
    V_m*rho_m[i]*cv_m[i]*der(T_m[i]) =
      w_h*sum(h[i,j,Nv+1] for j in 1:Nh) -
      w*h_m[i];
    for j in 1:Nh loop
     for k in 1:Nv loop
      (V_v*rho[i,j,k]*cv[i,j,k])*
          der(T_tilde[i,j,k]) =
        w_h*(h[i,j,k] - h[i,j,k+1]) +
          Q_c[i,j,k];
      C_wv*der(T_w[i, j, k]) =
        Q[j]/Nv - Q_c[i,j,k];
      Q_c[i,j,k] = UA_nom/(Nu*Nh*Nv)*
        (w/w_nom)^alpha*(T_w[i,j,k] - T_tilde[i,
            j,k]);
     end for;
    end for;
   end for;
   for i in 1:Nu loop
    rho_m[i] = p_nom/Methanol.R*T_m[i];
    h_m[i] = Methanol.h_T(T_m[i]);
    cv_m[i] = Methanol.cv_T(T_m[i]);
    for j in 1:Nh loop
     for k in 1:Nv loop
      rho[i,j,k] = p_nom /
        (Methanol.R*T_tilde[i,j,k]);
      cv[i,j,k] =
        Methanol.cv_T(T_tilde[i,j,k]);
     end for;
     for k in 1:Nv+1 loop
      h[i,j,k] = Methanol.h_T(T[i,j,k]);
     end for;
    end for;
   end for;
  end MethanolHeatExchangers;

  package Methanol
   constant Real R = 8.314462/32.04e-3;
   constant Real Tc = 512.64;
   constant Real f[8] =
    {3.90086, 10.9929, 18.3371, -16.3663,
     -6.22334, 2.80358, 1.07783, 0.96967};
   constant Real g[8] =
    {0.0, 4.12575, 3.26973, 3.77492,
     2.93574, 8.23747, 10.3312, 0.53326};

    function cp_T
     input Types.Temperature T;
     output Types.SpecificHeatCapacity cp;
    protected
     Types.PerUnit tau;
     Types.PerUnit u[8];
     Types.PerUnit x;
    algorithm
     tau := Tc / T;
     u := g * tau;
     x := f[1];
     for i in 2:8 loop
       x := x + f[i]*u[i]^2*exp(u[i])/
         (exp(u[i]) - 1)^2;
     end for;
     cp := x*R;
    end cp_T;

   function cv_T
    input Types.Temperature T;
    output Types.SpecificHeatCapacity cv;
   algorithm
    cv := cp_T(T) - R;
   end cv_T;

   function h_T
    input Types.Temperature T;
    output Types.SpecificEnthalpy h;
   protected
    Types.PerUnit tau;
    Types.PerUnit u[8];
    Types.PerUnit x;
   algorithm
    tau := Tc / T;
    u := g * tau;
    x := f[1]/tau;
    for i in 2:8 loop
     x := x + f[i]*g[i]/(exp(u[i]) - 1);
    end for;
    h := R*T*tau*x - 1361.810*tau/Tc;
   end h_T;
  end Methanol;
 end Models;
end MethanolHeatExchangersDAE;
```

Control development and sizing analysis for a 5th generation district heating and cooling network using Modelica

Ettore Zanetti David Blum Michael Wetter

Building Technology & Urban Systems Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA

Abstract

5th generation district heating and cooling system (5GDHC) are a relatively new concept. They use a single district loop near ambient temperature to provide heating and cooling. This paper improves on the modelling and control of a 5GDHC system called the reservoir network. The study updates the sewage heat exchanger plant model to more realistically represent seasonal changes, uses refined pump models with variable efficiency, introduces a ground coupled district pipe model to consider the inertia of the district network and implements a new control strategy for geothermal storage and sewage heat exchanger. The new approach reduced operating costs, mainly due to pumping cost for storage, sewage heat exchanger plant and distribution pump, while increasing the overall robustness of the approach in different sizing conditions. Thanks to the new controller, the pumping consumption was reduced by 21% with respect to the original baseline. Furthermore, the new control makes the system take better advantage of design changes, when reducing borehole field size and increasing the sewage heat exchanger size, the pumping energy savings become 29% with respect to the original baseline. Lastly, borehole field temperature stability was analyzed through 40 years of simulation.

Keywords: 5th generation district heating and cooling, geothermal borehole field, supervisory controller,sewage waste heat,Modelica

1 Introduction

The rapid pace of urbanization has transformed the world's population distribution, with an increasing number of people residing in urban areas. Currently, 55% of the global population lives in cities, and this figure is projected to rise to 68% by the year 2050 Nations et al. (2012). This urbanization presents both challenges and opportunities. Regarding energy consumption and sustainability, one advantage of urbanization is the potential for implementing centralized heating systems, which offer numerous benefits such as increased overall efficiency and reduced emissions Lake, Rezaie, and Beyerlein (2017). Historically, separate centralized plants were built for heating and cooling purposes. For heating, four generations of district plants have been developed over the last century, each aiming to improve efficiency and integrate more sustainable heat sources, especially renewable Lund et al. (2014). The first generation involved steam plants, while the fourth generation operates at temperatures between 60 and 70 °C, with a focus on incorporating renewable and waste heat sources like solar energy while reducing the primary energy consumption and operating costs of the district Averfalk and Werner (2020). On the other hand, cooling districts traditionally relied on large chillers with evaporative towers, operating at temperatures ranging from 7 to 18 ^{o}C. In recent years, the growing need for cooling, driven by global warming and rising temperatures, as well as opportunities for heat recovery, have given rise to a new concept: the simultaneous provision of heating and cooling referred to as 5th generation combined district heating and cooling networks (5GDHC). Various approaches were proposed in the literature to achieve this goal, including cold district heating Pellegrini and Bianchini (2018), bi-directional low-temperature networks Bünning et al. (2018), anergy networks Sulzer (2011), natural temperature district heating, and the ambient network Calixto, Cozzini, and Manzolini (2021). Among these different concepts, this paper will focus on a type of ambient network, called the reservoir network as presented by Sommer et al. (2020), which works by distributing water in a single loop at ambient temperature maintaining the temperature between a predefined interval (i.e. 6 - 17 ^{o}C). The single loop improves hydronic balancing among network participants and ambient temperature facilitates integration of waste heat sources. One crucial aspect highlighted by Sommer et al. (2020) is the significant impact of pumping energy in a network with a lower temperature range. Their research demonstrates that a variable flow approach, which keeps the temperature within a specific interval, can drastically reduce pump consumption.

This paper improves on the reservoir network concept by focusing on the flow rate control in specific components of the system, namely a borehole field storage and a sewage water heat exchanger plant. Although the temperature range in this type of network may be relatively small, it has a considerable impact on pumping energy Maccarini et al. (2023) and is closely correlated with the current demand of the district. Therefore, to improve the performance of the reservoir network and further reduce pump energy consumption, a better rule-based controller was designed. The control output is the mass flow rate of each agent and it is calculated accounting for current de-

DOI
10.3384/ecp20423

mand and temperature level of the agent and district loop. Furthermore, a sensitivity analysis was carried out for the borehole field and sewage water plant sizes on the performance of the system. The idea is to show the potential of reducing the number of boreholes, which lead to lower capital costs, and increasing the waste heat plant capacity, which lead to greater waste heat utilization, with the new control. The analysis and new control implementation is carried out using the Modelica Buildings Library Wetter et al. (2014), which enables modeling of network mass flows, temperatures, and control logic important for analyzing this type of system. Our model extends from the original model used in Sommer et al. (2020).

2 Methodology

This section presents the case study, the modeling assumptions and the changes to the overall Modelica model used with respect to Sommer et al. (2020). Component level models come from the Modelica Buildings Library Wetter et al. (2014). Dymola 2023x was used to run the simulations on Linux with a Radau solver and tolerance set at 1E-6.

2.1 Case Study Description

The case study expands the Modelica model presented in Sommer et al. (2020). The network consists of a single hydronic loop, where the various agents, consisting of prosumers, storage and plants, take water from the reservoir loop and inject it back into the same loop. This ensures decoupling of the differential pressure fluctuations between agent pumps and the main reservoir loop. The reservoir loop includes a borehole field and sewage heat-exchanger, which can be considered the storage and plant of the district able to compensate for the load. Three representative buildings, a residential, an office and a hospital represent the prosumers. The term prosumer is used because the building Energy Transfer Stations (ETS) include a heat pump that can draw thermal energy from the network and a heat exchanger for direct cooling that provides thermal energy back to the network. A schematic view of the network topology and associated controls are shown in Figure 1.

2.1.1 Load Profiles

The loads are pre-calculated as hourly profiles and based on Swiss archetypes Murray, Niffeler, et al. (2019), Kristina Orehounig,Matthias Sulzer (2019), and Murray, Marquant, et al. (2020) and scaled up to provide demand profiles for a typical Swiss district. The ETS in each building will instantaneously compensate for the load while keeping a ΔT of 4 K between district water supply and return. The space heating demand in the residential building is 2.40 $GWh/year$, corresponding to around 60,000 m^2 considering average Swiss space occupation and consumption SIA et al. (2015), Staub, Rütter, et al. (2014), and Wohnfläche (2017). The heating demand of the office building is 0.19 GWh/y or 8 % of the heating demand of the residential building. This consumption ratio corresponds to typical values in Switzerland. For the residential building and the office building, the ratio of annual heating to cooling demand is 7.8 and 2.1, respectively. This ratio is in line with the expected increase in cooling demand scenario for Switzerland Settembrini et al. (2017). The hospital has a heating demand of 0.97 $GWh/year$ and a cooling demand of 0.23 $GWh/year$, with a ratio of heating to cooling of 4.3. In comparison to the residential or office demand profiles, the main difference of the hospital is the large share of domestic hot water. In total for all prosumers, the heating demand is 3.55 $GWh/year$ and the cooling demand is 0.62 $GWh/year$. The overall ratio of heating to cooling demand for all prosumers is 5.7. Figure 2 presents a summary of the yearly loads for cooling, heating and domestic hot water of the three buildings.

2.1.2 Heat Pumps and Cooling Heat Exchanger

Each prosumer utilizes two heat pumps: one for space heating and another for domestic hot water. Regarding space heating, the condenser outlet temperature is set at 38 ^oC when demand is at its design value, and it is reset linearly to 28 ^oC when there is no demand. For domestic hot water, the set point temperature is 63 ^oC. The heat pumps are based on the *Fluid.HeatPumps.Carnot_TCon* model, which includes an ideal internal control system that enables the heat pump to track the setpoint temperature leaving the condenser. The heat pump coefficient of performance (COP) is calculated using a Carnot effectiveness of $\eta_{carnot,ref}$ = 0.5. For space cooling, direct cooling is provided by a heat exchanger that instantly provides the scheduled cooling demand.

2.1.3 Mass Flow Rates and Pressure Drops

The mass flow rates on the network side of the heat pumps and heat exchangers are controlled to keep the nominal temperature difference of $\Delta T = 4K$. The pressure drops at nominal mass flow rate for the distribution network between prosumers and plants are assumed to be 50 kPa or 250 Pa/m, with a nominal flow rate of 97.3 kg/s and a pipe diameter of 18 cm. The pressure drops for the sewage heat exchanger (HX) plant are 50 kPa at design flow rate 11.46 kg/s as for the baseline case study in Sommer et al. (2020), while in this study we consider also a scenario where we consider three heat exchangers in parallel instead of one, leading 34 kg/s. For the borehole fields the nominal pressure drop for each bore is 30 kPa and the nominal flow rate depends on the number of boreholes used in the simulation. In Sommer et al. (2020) a total of 350 boreholes were considered, in this study we also added two simulations that consider 250 boreholes and the flow rate is also adjusted considering 0.3 kg/s in each probe for a total of 105 kg/s in one case and 75 kg/s in the other.

2.1.4 Plant and Storage Models

The sewage heat exchanger plant model is based on the *Fluid.HeatExchangers.ConstantEffectiveness* model.

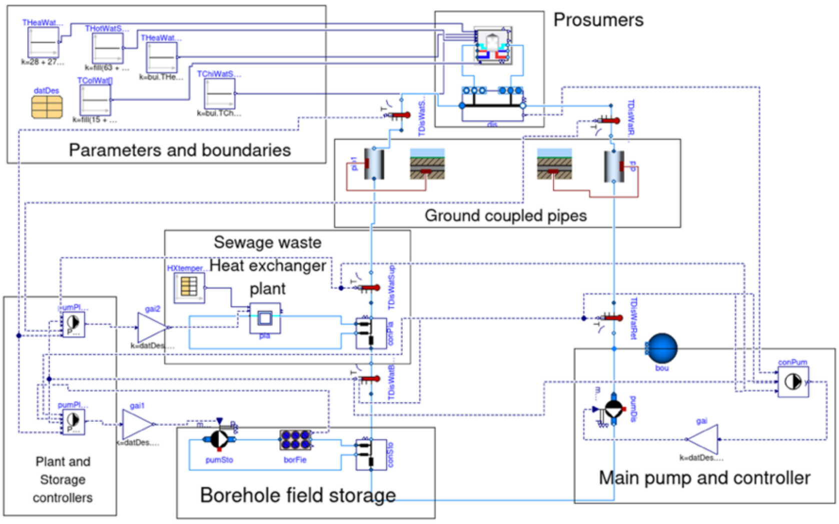


Figure 1. Network scheme represented in Modelica. Each box is a main component of the district model. Solid lines represent physical and digital connections, while dashed lines represent control inputs and outputs

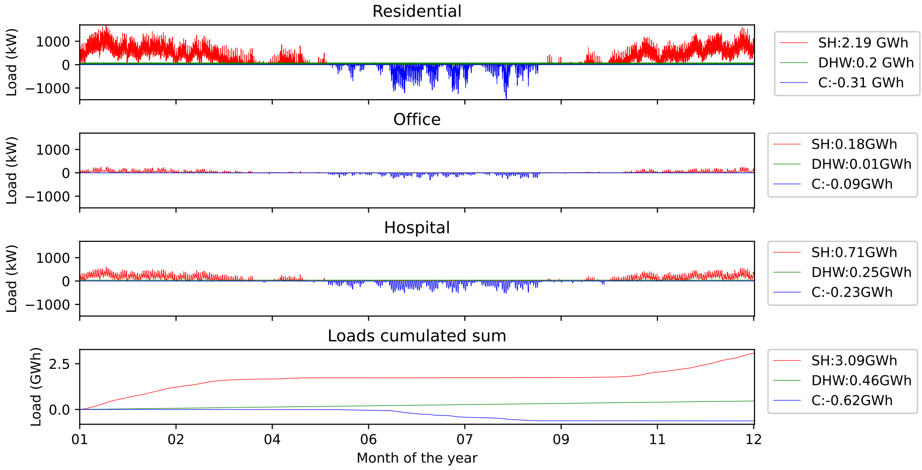


Figure 2. Demand profiles of the residential area (1), office area (2), hospital (3) and total cumulated sum (4). The legend indicates demand for space heating (SH), domestic hot water (DHW) and space cooling (SC).

This is a constant effectiveness model with $\varepsilon = 0.91$. The storage is a borehole field with U-tube probes of 250 m depth. The starting ground temperature is assumed to be 9.4 ^oC in the top 10 m and increases by 0.02 $^oC/m$ up to 14.2 ^oC at the bottom of the borehole. The boreholes are modelled using the model *Experimental.DHC.Plants.Reservoir.BoreField*. This model is based on the following key assumptions: The soil's thermal properties, such as conductivity and diffusivity, remain constant, homogeneous, and isotropic. Similarly, the ground and pipe material exhibit constant, homogeneous, and isotropic values for conductivity, capacitance, and density. Before the simulation begins, there is no heat extraction or injection. All boreholes in the field have uniform dimensions, including the pipe dimensions. Inside the boreholes, heat transfer occurs solely in the radial direction with no advection.

2.1.5 Circulation Pumps

The circulation pumps provide enough head to overcome the pressure losses occurring in the network. To estimate the electricity consumption, the model *Fluid.Movers.FlowControlled_m_flow* model was used. The motor and hydraulic efficiency nominal values are $\eta_m = 0.8$ and $\eta_h = 0.6$. Furthermore, the motor efficiency η_m changes according to U.S. Department of Energy (2014), while the hydraulic efficiency η_h changes according to Fu, Blum, and Wetter (2022).

2.1.6 Ground Coupling

Thermal ground coupling of the distribution pipes adds heat capacity to distribution network and ground heat exchange that were not present in the previous study. This allows to have a more accurate estimation of thermal losses and of the inertia of the distribution network, which were absent in the previous work. The new controller introduced in this study for the borehole field storage, described in Section 2.1.7, can turn off the borehole field pump. When the pump is off, the heat capacity of the borefield is decoupled from the district network. As a consequence, if the district network has no storage capacity modelled, its temperature changes instantaneously, e.g., the rate of change in temperature is fast and non-physical. Therefore, we modelled the heat transfer between the pipes and the ground, as shown in Figure 4. The model represents a radial 1D discretization of the conduction heat transfer between the pipe wall and the undisturbed ground. The pipe is assumed to be made of uninsulated plastic with a thickness of 1 *cm* and a heat conductivity of $u = 0.2W/mK$. Omitting the pipe thickness in the conduction calculation would lead to an overestimation of the heat transfer between the distribution network and the ground. For the discretization, a capacitance-resistance approach was used, dividing the radial direction into five volumes. The ground temperature was assumed to reach equilibrium with the undisturbed ground temperature T_g after 0.5 *m* and the *BoundaryConditions.GroundTemperature.UndisturbedSoilTemperature* model from the Buildings Library is used, which is based on Smith (1996). Soil data comes from the ASHRAE climatic constants to calculate subsurface temperature. The pipe is placed 1 *m* below ground. Furthermore, a discretization is also carried out in the axial direction where the 500 *m* of distribution pipes are divided in 100 *m* segments between supply, return of the plant and storage and each prosumer supply, each pipe is discretized with 10 volumes to approximate the water outlet temperature after exchanging heat with the ground. The yearly temperature variation for the ground is shown in Figure 3.

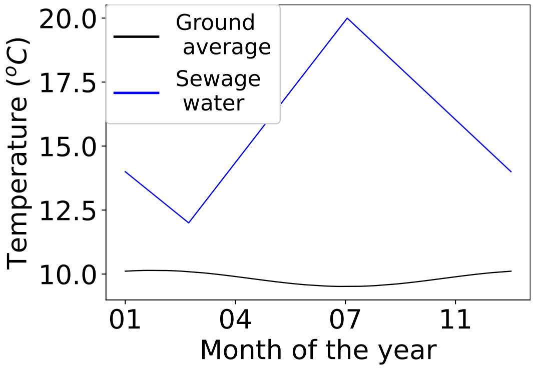


Figure 3. Effluent sewage water and undisturbed ground temperature profiles used for simulations.

2.1.7 Network Controllers

The main distribution controller for variable flow operation was developed in Sommer et al. (2020) and works as follows: The controller will reduce the network water flow rate until the water temperature at the outlet of the different prosumers becomes too close to user-provided upper or lower bounds. Let $TMixMin$ and $TMixMax$ be the minimum and maximum measured outlet temperatures from each prosumer, let $TMin$ and $TMax$ be the lower and upper bounds for the mixing temperatures and $dTslo$ a tuning parameter, that can be seen as the slope along which the main pump controller curve is defined for partial load. If $TMixMin - TMin > dTslo = 2K$ or $TMax - TMixMax > dTslo = 2K$ then the pump speed is set to the minimum speed *yPumMin*. Otherwise, it is linearly increased to the full speed until $TMin = TMixMin$ or $TMax = TMixMax$, where the pump will work at nominal capacity. This calculation is done for the lower and upper bound and the actual pump speed is the larger of the two pump signals. This control logic is implemented in the model *Experimental.DHC.Networks.Controls.MainPump*. In Figure 5 the logic is represented visually.

In this study, we developed a rule-based controller for the sewage heat exchanger plant and the borehole field storage control, each of them has a separate instance of the controller. The controller takes as input the average source temperature (i.e. sewage water or average borehole field temperature), the inlet and outlet temperature of the agent, and the supply to the first prosumer and return temperature from the last. These supply and return temperatures are used to estimate the net need of the district for heating or cooling by looking at their difference. Then, the agent outlet is used as the measured input to a proportional controller that controls the mass flow rate through the agent pump. This controller's setpoint is the source temperature adjusted with a negative shift for heating and a positive shift for cooling to account for a pinch point temperature difference between source temperature and outlet temperature. Lastly, an on/off controller with hysteresis based on the difference between inlet temperature to the agent and shifted source temperature is used to determine when to turn on and off the agent circulation pump and avoid frequent switching behavior.

2.1.8 Baseline Case Study Hypotheses and Changes

In Sommer et al. (2020), the main assumptions for the network side are:

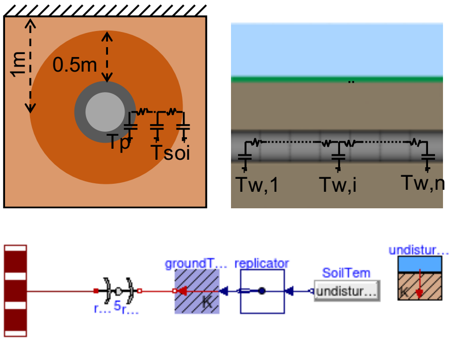


Figure 4. Representative diagram of the radial heat transfer between the distribution pipe T_p and the ground T_{soi} (left). Axial discretization of the pipe water volumes $T_{w,i}$ (right). Below is a diagram view of the ground coupling model.

1. The pumps use a constant motor and hydraulic efficiency of $\eta_m = 0.7$ and $\eta_h = 0.7$.
2. The annual energy balance of the storage has to be zero. The sewage heat exchanger plant will provide the net heating and cooling demand.
3. The water temperature in the reservoir loop must always be between 6 and 17 ^oC. This ensures that direct cooling is possible and that with a nominal $\Delta T = 4K$ the heat pumps can draw heat without danger of freezing.
4. The sewage water temperature is a constant value at 17 ^oC for the whole year.
5. The distribution pipes are simplified as adiabatic and hence do not exchange heat with the ground.
6. The sewage heat exchanger circulation plant is always working at nominal flow rate. The storage circulation pump uses the same control signal as the main network circulation pump according to the logic explained in Section 2.1.7.

In this study, these assumptions are modified in the following ways:

1. Since an objective of this study is to reduce pump consumption through better control logic, we updated the pump models to account for variable efficiency at part loads, as described in Section 2.1.5.
2. In this case the yearly energy balance of the storage does not need to be zero, however, it has to reach a reasonable steady state condition after a certain period of , for this study 40 years were considered. The borehole average temperature difference between the initial condition should be within an acceptable range of around 1 ^oC, for example the minimum temperature of the borehole has to be above freezing point to avoid potential damage to the borehole filling.

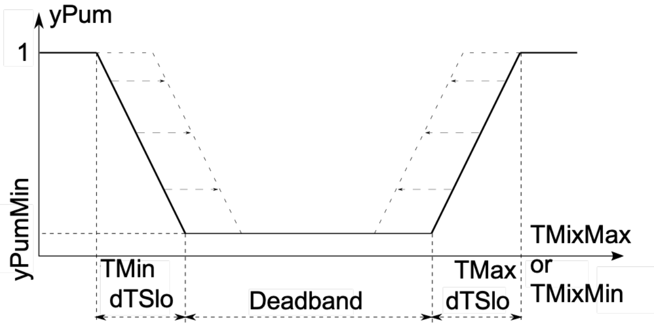


Figure 5. Distribution pump controller. On the x-axis there is the district temperature, while on the y-axis there is the main pump control signal as a function of the minimum and maximum prosumer outlet temperatures $TMixMin$ and $TMixMax$ as explained in Subsection 2.1.7. The dotted lines show the effect of shifting $TMax$ or $TMin$.

3. Instead of keeping a constant upper limit of 17 ^oC, the temperature limit can be increased if no prosumer requires cooling. This can be done by checking the cooling pump signal. In the case no prosumer requires cooling, the 17 ^oC temperature limit is increased to 19 ^oC. This is to avoid that during periods with low heating demand, such as Spring and Autumn in this study, the sewage heat exchanger plant will bring the district temperature close to 17 ^oC causing the distribution pump controller to rise the flow rate according to the logic in Figure 5, which only causes an increase in electricity consumption with no benefit.
4. The distribution network pipes now have a ground thermal coupling with an approach similar to the one presented in Maccarini et al. (2023) and described in Section 2.1.6.
5. Instead of considering a constant sewage temperature value for the whole year, a variable temperature profile is derived from Schmid (2008) under the hypothesis of placing the sewage heat exchanger at the effluent water of waste water treatment plant. The yearly temperature variation for the sewage water is shown in Figure 3.
6. The new rule based controller for the sewage heat exchanger and borehole field is used as described in Section 2.1.7.

2.1.9 Simulation Scenarios

A total of six scenarios were considered in this study described as follows and summarized in Table 1:

1. $bs_{nbor350mpla11}$: considered the baseline scenario, since the model used is identical to the one used in Sommer et al. (2020) apart from the sewage temperature profile, the distribution pipe coupling and the pump efficiency modelling.
2. $bs_{nbor250mpla11}$: the model is similar to $bs_{nbor350mpla11}$, however, the borehole number is downsized to 250. The reasoning is to carry out a small sensitivity analysis and see how the baseline controller behaves when the storage capacity is reduced by around 30%, and so is the investment cost. Looking at the technical report Oakridge national laboratory (2018), the drilling cost of a borehole can be assumed to be between 30 and 50 $\$/m$, considering that each borehole is 250 m deep, this would amount to around \$1M saved with respect to baseline.
3. $bs_{nbor250mpla34}$: the model is similar to $bs_{nbor250mpla11}$, however, the nominal flow rate of the sewage plant is increased to 34.5 kg/s. The reason for this sensitivity analysis is to give more

room to the sewage plant heating capacity while keeping the overall investment cost equal or lower. The cost increase of the heat exchanger can be calculated according to the cost per area needed Hewitt and Pugh (2007). We can calculate the area starting from the effectiveness value ε model, under the following assumptions:

- The sewage heat exchanger is a plate heat exchanger with enough plates to be approximated at counter flow so that $\varepsilon = NTU/(1+NTU)$
- The global heat transfer coefficient $U = 2000 W/(m^2K)$, which is an average value for such heat exchangers
- The minimum fluid heat capacity rates of for the two cases are $C_{min} = 50 kJ/(Ks)$ and $C_{min} = 150 kJ/(Ks)$

Under these assumptions the area is

$$A = \frac{NTU\, C_{min}}{U}. \qquad (1)$$

Doing this calculation for our case leads to a total area of 315 m^2 when $m_{flow} = 11.46 kg/s$ and 945 m^2 when $m_{flow} = 34.45 kg/s$. This leads to average investment cost of \$20,000 and \$60,000. This exercise does not account for the increase in price for the sewage heat exchanger pump and pipes, which are likely smaller components. The increase in cost for the sewage heat exchanger is smaller than the decrease in cost for the reduced number of boreholes, making this scenario cheaper than the baseline.

4. $nc_{nbor350mpla11}$: similar to $bs_{nbor350mpla11}$, but the sewage heat exchanger plant and the borehole field storage circulation pumps are controlled with the new rule based controller. Furthermore, the relaxation logic for the upper temperature bound is used in the main distribution pump controller and the parameter $dTslo = 1.5K$, slightly increasing the main distribution pump controller dead band.

5. $nc_{nbor250mpla11}$: similar to $nc_{nbor350mpla11}$, but the size of the borehole field is also downsized to 250.

6. $nc_{nbor250mpla34}$: similar to $nc_{nbor250mpla11}$, but the nominal flow rate of the sewage heat exchanger plant increased to 34.45 kg/s.

3 Results

3.1 Borehole Field Temperature Drift

In Subsection 2.1.8 we state the requirement that the borehole field energy balance does not need to be zero at the beginning, however the average borehole field temperature needs to reach a reasonable equilibrium point. In Figure 6, the evolution of the average ground temperature at the interface with the borehole for the scenarios $nc_{nbor250mpla11}$ and $nc_{nbor250mpla34}$ are shown. The temperature reaches a new equilibrium point after around 35 years of simulation, as it can be seen from Table 2. For scenario $nc_{nbor250mpla11}$ the temperature difference after 40 years of simulation is 3.2 K, while for scenario $nc_{nbor250mpla34}$ it is 1.6 K. This is to be expected since increasing the size of the sewage plant heat exchanger satisfies the heating demand during winter, reducing the depletion of the borehole field. This is also reflected in the total energy cost, which in scenario $nc_{nbor250mpla11}$ increases by 20%, while in scenario $nc_{nbor250mpla34}$ it increases by only 0.8%. The larger increase in total energy consumption for the first scenario is due to a lower average district temperature during winter, affecting the heat pump COP and leading to an increase in average mass flow rate for the main distribution pump to maintain the minimum temperature constraint of 6 ^oC. However, as mentioned in Subsection 2.1.1, the cooling demand is expected to increase in Zurich, further reducing the negative temperature shift of the borehole field. Furthermore, with such a large time span, the district could also expand or differentiate its demand due to more prosumers connecting, which increases the uncertainty. A sensitivity analysis on such a long period of time would be a separate study. The current Modelica models are computationally efficient enough to carry out such a study since the current implementation of the model takes less than 5 h to run on a single thread Lenovo workstation with a Xeon(R) W-2245 CPU @ 3.90GHz for a 40 year simulation.

Table 1. Summary of simulation scenarios considered. Thermal coupling, variable sewage heat exchanger temperature and variable efficiency pump are included in all scenarios.

Scenarios	Controller	nbor	Sewage HX
$bs_{nbor350mpla11}$	Default	350	m_flow =11.47
$bs_{nbor250mpla11}$	Default	250	m_flow =11.47
$bs_{nbor250mpla34}$	Default	250	m_flow =34.45
$nc_{nbor350mpla11}$	New	350	m_flow =11.47
$nc_{nbor250mpla11}$	New	250	m_flow =11.47
$nc_{nbor250mpla34}$	New	250	m_flow =34.45

Table 2. Summary of borehole field average temperature evolution T_N and total electrical energy consumption Eel_N at a given year, where N is the year number.

Scenarios	T_1 ^oC	T_{35} ^oC	T_{40} ^oC	Eel_1 $MWh/year$	Eel_{40} $MWh/year$
$nc_{nbor250mpla11}$	11.75	8.69	8.59	867	1037
$nc_{nbor250mpla34}$	11.90	10.28	10.25	829	836

3.2 District Energy and Temperature Profile Analysis

This section presents the results from the scenarios introduced in Table 1. Figure 7 presents the yearly cumula-

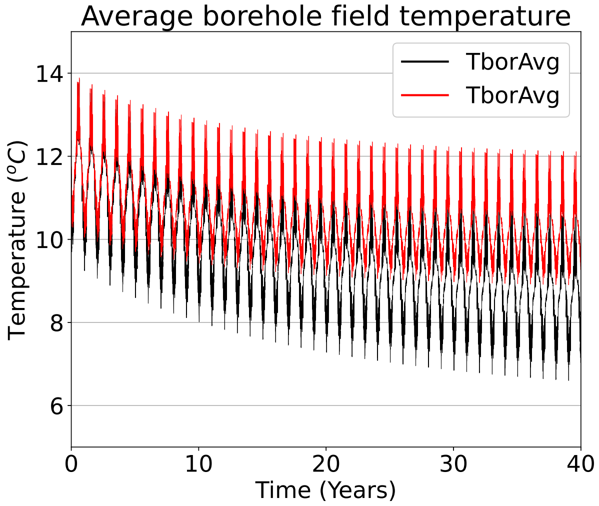


Figure 6. The average borehole temperature is plotted on the y-axis for scenarios *newcon_nbor*250*mpla*11,black, and *newcon_nbor*250*mpla*34,red, while on the x-axis the time in years is shown up to 40 years. This plot shows the temperature drift of the ground in the borehole.

tive energy and temperature profiles for each scenario for the first year of operation. Plot a) shows the yearly total thermal demand of the buildings, which is net heating of almost 3 $GWh/year$. Plot b) shows the losses through the distribution pipes for each scenario. This chart indicates that the distribution losses are generally between 2% and 4% of the overall thermal demand. Furthermore, the new controller (*nc*) scenarios have lower distribution losses with respect to the baseline $bs_{nbor350mpla11}$, by 25% in scenarios $nc_{nbor350mpla11}$ and $nc_{nbor250mpla11}$ by 7% in scenario $nc_{nbor250mpla34}$. This is due to an average lower temperature during the summer months for the (*nc*) scenarios, as shown by the flatter slope of the cumulative energy curve during this period. Instead, for the baseline (*bs*) scenarios, the losses remain the same for $bs_{nbor250mpla11}$ and increase by 40% for scenario $bs_{nbor250mpla34}$ with respect to $bs_{nbor350mpla11}$. The reason for this increase in losses is due to the increase of average temperature of the district for $bs_{nbor250mpla34}$ with respect to the ground temperature.

Plots c) and d) of Figure 7 show the cumulative energy flows from the borehole field and sewage heat exchanger to the network in each scenario. Looking at the baseline (*bs*) vs. new controller (*nc*) scenarios, the cumulative energy supply for borehole field and sewage HX are very close for the initial winter season. The reason is that, in winter, only heating is present as shown in Figure 2, meaning that both in *bs* and *nc* scenarios, the sewage HX pump will run most of the time. In the summer, the situation changes because the *bs* scenarios continuously run the sewage heat exchanger pump at nominal capacity, while the *nc* scenarios only turn it on when the domestic hot water demand surpasses the cooling demand. In the (*bs*) scenarios, since the average temperature of the sewage is higher than the network limit for cooling and the sewage plant pump is continuously running, this causes the storage to have to overcompensate to keep the network temperature lower than in the case of just serving the building cooling load. This phenomenon is exacerbated in the scenarios with reduced number of boreholes and increased mass flow rate in the sewage heat exchanger. On the other hand, the *nc* scenarios can turn off the sewage plant production when the demand is cooling dominated, and it benefits from any increased sewage plant capacity by supplying more heat during heating dominated periods. This not only improves performance, but makes the overall operation more robust to cases of adding more waste heat capacity, or changing demand due to connecting new prosumers or climate change.

Plots e) and f) report the daily maximum and minimum network temperatures and their limits over the year. Starting from the *bs* scenarios, the default $bs_{nbor350mpla11}$ is able to satisfy the temperature constraints in heating and cooling seasons, while $bs_{nbor250mpla11}$ and $bs_{nbor250mpla34}$ violate the constraints. The reason is that by increasing the size of the sewage heat exchanger plant and reducing the number of boreholes, it becomes impossible for the borehole field to compensate for cooling demand and heat injected by the sewage water plant into the reservoir loop, as described previously. Among the *nc* scenarios, scenario $nc_{nbor250mpla11}$ slightly violates the constraint in the worst months of winter, and both $nc_{nbor250mpla11}$ and $nc_{nbor250mpla34}$ slightly violate the constraint in the summer. Furthermore, the plot shows the *nc* scenarios using the flexible upper boundary as a function of current district demand. This relaxes the upper limit, making the district circulation pump controller dead band larger, and ultimately slowing down the pump, according to the logic presented in Figure 5. The reason for the upper bound being increased during the summer is due to times when no cooling demand is present, but domestic hot water demand is.

3.3 Summary Results and KPI Analysis

This section shows a KPI analysis on the performance of the district for the various scenarios described in Table 1. Figure 9 reports the overall electrical consumption of the district and the circulation pumps of the network. The top chart shows that, in general, the heat pump and prosumer pumps make up 87.5% of the overall electricity demand of the district, while the remaining 12.5% is due to the main circulation, sewage heat exchanger plant and borehole field pumps. Therefore, the new controller *nc* scenarios show only moderate total electricity savings with respect to the baseline. However, it is interesting to notice the increase in electricity consumption for the baseline *bs* scenarios *bs_nbor*250*mpal*11 and *bs_nbor*250*mpal*34 with respect to the other scenarios. In these two scenarios, the temperature constraints are often violated as shown in Figure 7, causing the main distribution pump to

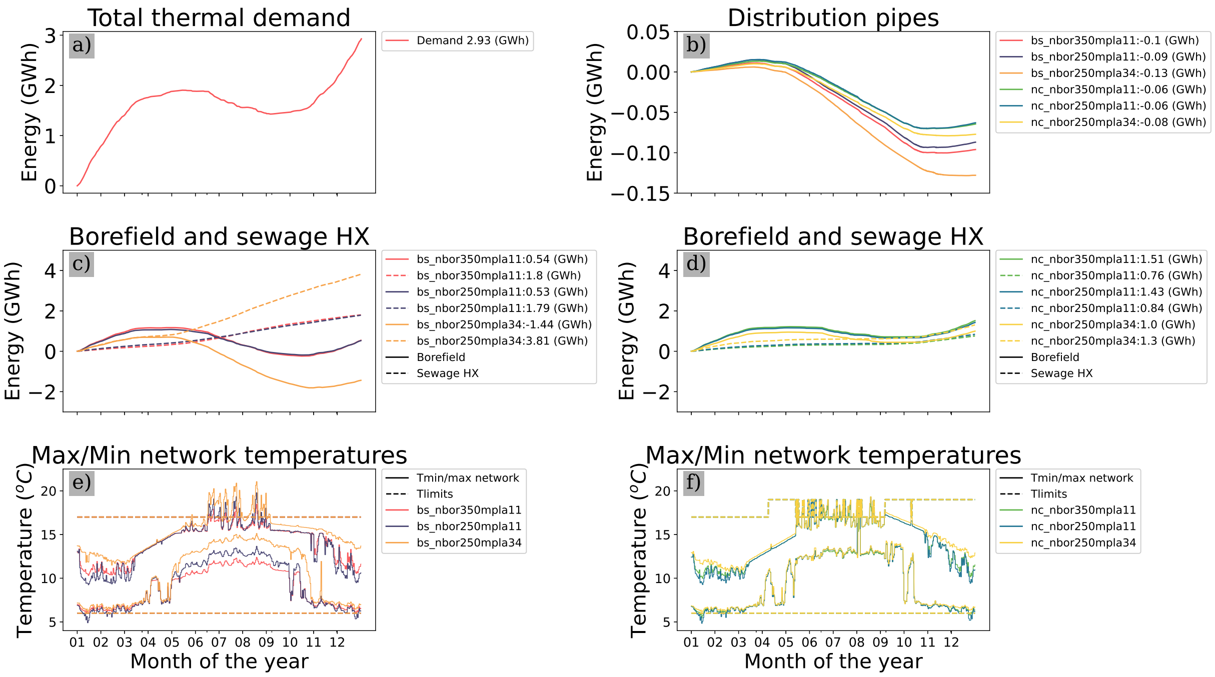


Figure 7. Visualization of energy flows and temperature distributions across the district network. For all the figures the x-axis is time shown for the first year of simulation from January (1) to December (12). a) chart shows the overall cumulative thermal demand of the district buildings, b) chart shows the cumulative energy flow from the distribution pipes to the ground c) and d) show the cumulative energy flows from the sewage plant (dashed lines) and borefield (solid lines) to the reservoir loop for the baseline *bs* (left) and new controller *nc* (right) scenarios e) and f) show the district loop daily minimum and maximum temperatures for the different scenarios and the temperature limits of the main distribution pump control.

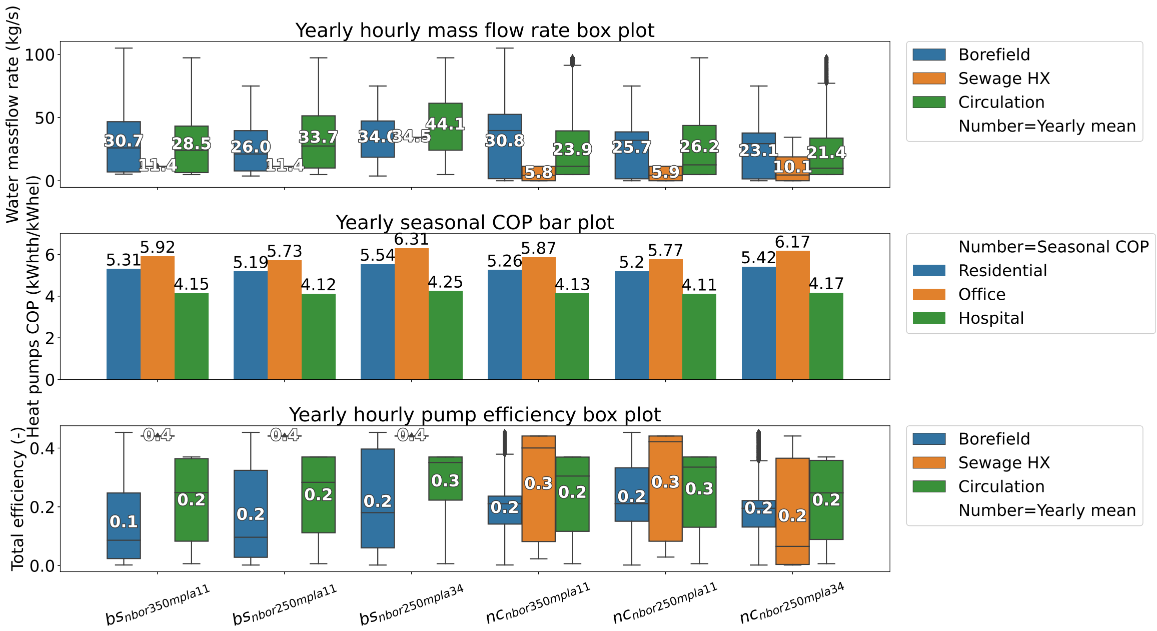


Figure 8. The x-axis shows the different scenarios presented in Table 1. The y-axis shows the water mass flow rate for each agent (top), the COP for the each prosumer (middle), and pump efficiency for the network pumps (bottom). The numbers for each plot corresponds to the yearly average value of the mass flow rate and efficiency (only when flow rate is 10% higher than nominal value to avoid quick transients) and the Seasonal COP for the COP plot.

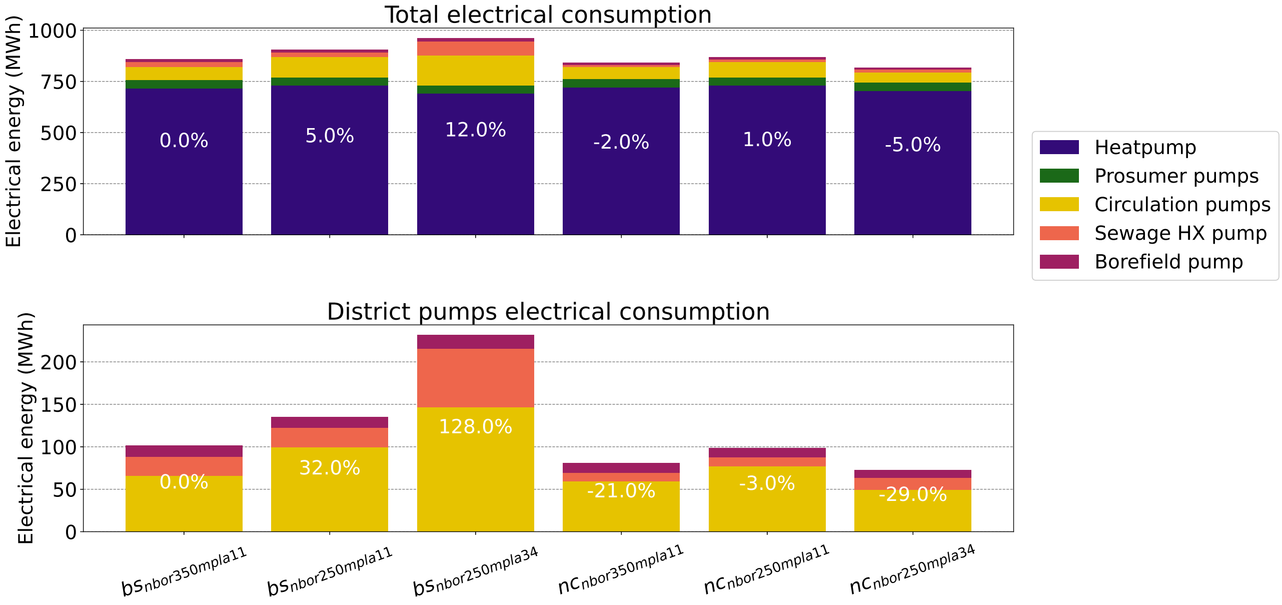


Figure 9. The x-axis shows the different scenarios presented in Table 1. The y-axis shows the total yearly electrical consumption by the whole network (top) and only distribution, plant, and stroage pumps (bottom). The % number in white corresponds to the difference with respect to the baseline scenario $bs\_bor350mpla11$.

run more often at nominal capacity as explained in Subsection 2.1.7. The situation is further exacerbated in scenario $bs_{nbor250mpla34}$. This indicates that the nc scenarios are more robust towards sizing changes, which allow for more sizing choices of different components (i.e. plant and storage), which is critical for opportunities to reduce capital costs. For example in this study, Section 2.1.9 described how reducing the borefield size could save approximately $1M dollars in capital cost. As shown with scenario $nc\_bor250mpla11$, the new control enables this without an increase in operating costs from electricity consumption. Furthermore, scenario $nc\_bor250mpla34$, with only slightly higher capital cost of around $100k dollars than $nc\_bor250mpla11$ to pay for higher sewage plant capacity, though still cheaper than the cases with full 350 boreholes, further reduces the overall electricity consumption thanks to increase in additional heating energy provided by the sewage heat exchanger. Lastly, if we consider the absolute savings for $nc\_bor250mpla34$ compared to $bs\_bor350mpla11$, they equate to 40 MWh/year of electrical energy, $12,000/year assuming an average electricity price of $0.3/$kWh$, and 1.3 $tonCO2/year$, assuming 330 $kgCO2/kWh$. Furthermore, the bottom chart of Figure 9 shows the $nc\_bor250mpla34$ scenario reduces the total pump consumption by 29% compared to $bs\_bor350mpla11$.

Figure 8 presents a summary of the hydraulic and thermal performance of the district in the different scenarios, where the top chart represents the yearly hourly mass flow rate distribution for the different network agents, the middle chart shows the yearly seasonal COP for the three prosumers, and the bottom chart shows the yearly average circulation pump efficiency, only when the flow rate is 10% higher than the nominal value to avoid fast transients. Looking at the top and bottom chart together, it is clear that the nc scenarios reduce the agent mass flow rates, thanks to the better control strategy that is able to maintain the average district temperature further from the upper and lower boundaries, running the main circulation pump at partial load according to the logic in 2.1.7, and using the storage and plant in more effective ways depending on the current demand and loop temperature levels. However, the partial load utilization in these scenarios increases the variability in the agents pump efficiency as shown in the bottom chart. Furthermore, looking at the pump efficiency box plot, we can see that using more realistic efficiency curves lead to an average efficiency of the pumps that is relatively low. There is certainly room to improve the pump sizing coupled with control that uses the pumps at partial load.

Lastly, looking at the middle chart, it can be seen that the increase in size of the sewage heat exchanger helps increase the seasonal COP in scenarios $bs_{nbor250mpla34}$ and $nc_{nbor250mpla34}$ with respect to the other scenarios, since it increases the average temperature of the district during winter, as shown in Figure 7 bottom plot. However, in $bs_{nbor250mpla34}$, this causes a great penalty in the summer since the sewage water heat exchanger pump is always running, while in the $nc_{nbor250mpla34}$, the pump is mostly turned off during the summer period.

4 Conclusions

This study extended Sommer et al. (2020) reservoir network with an updated sewage heat exchanger plant model to more realistically represent seasonal changes, new pump models with variable efficiency, ground-coupled district pipe model to consider the inertia of the district network, which is important for control stability, and a new control strategy for the distribution network, sewage heat exchanger, and borefield pumps. The updated model was used to carry out a sensitivity analysis on the size of the borehole field and sewage heat exchanger, using the baseline and the new controller.

The analysis shows the robustness and performance enhancement of the new control approach *nc* over the baseline *bs*. The new *nc* approach leads to a $4800 dollars increase in operational costs when reducing the size of the borehole field by 30%, saving $1M in investment cost, reducing the overall life cycle cost. Furthermore, when additionally increasing the capacity of the sewage heat exchanger, the new control better exploits the additional waste heat capacity, as shown in scenario $nc_{nbor250mpla34}$, where the overall investment cost is reduced compared to the baseline by around $0.9M, and operational costs are reduced by $12,000 per year thanks to the electrical energy saved. This sensitivity analysis shows the importance of coupling design, sizing and control to reduce first and life cycle costs. Future studies will include a more extensive sensitivity analysis and the introduction of control and design optimization to explore the untapped potential of the reservoir loop system and the model updates.

5 Acknowledgements

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Industrial Efficiency and Decarbonization Office and Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02- 05CH11231.

6 Data Availability

Results can be reproduced running *Experimental.DHC.Examples.Combined.SeriesVariableFlowUpdate* in branch commit a64685e3ac of repository modelica-buildings.

References

Averfalk, Helge and Sven Werner (2020). “Economic benefits of fourth generation district heating”. In: *Energy* 193, p. 116727.

Bünning, Felix et al. (2018). “Bidirectional low temperature district energy systems with agent-based control: Performance comparison and operation optimization”. In: *Applied Energy* 209, pp. 502–515.

Calixto, Selva, Marco Cozzini, and Giampaolo Manzolini (2021). “Modelling of an Existing Neutral Temperature District Heating Network: Detailed and Approximate Approaches”. In: *Energies* 14.2. ISSN: 1996-1073. URL: https://www.mdpi.com/1996-1073/14/2/379.

Fu, Hongxiang, David Blum, and Michael Wetter (2022). “Fan and Pump Efficiency in Modelica based on the Euler Number”. In: *Modelica Conferences*, pp. 19–25.

Hewitt, Geoff F and Simon J Pugh (2007). “Approximate design and costing methods for heat exchangers”. In: *Heat transfer engineering* 28.2, pp. 76–86.

Kristina Orehounig,Matthias Sulzer (2019). *Technical regulation for the building stock*. Tech. rep. Empa. URL: https://www.nfp70.ch/en/eFR3itStLevW3JiB/project/technical-regulation-for-the-building-stock.

Lake, Andrew, Behnaz Rezaie, and Steven Beyerlein (2017). “Review of district heating and cooling systems for a sustainable future”. In: *Renewable and Sustainable Energy Reviews* 67, pp. 417–425.

Lund, H et al. (2014). “Integrating smart thermal grids into future sustainable energy systems”. In: *Energy* 68, pp. 1–11.

Maccarini, Alessandro et al. (2023). “Influence of building heat distribution temperatures on the energy performance and sizing of 5th generation district heating and cooling networks”. In: *Energy* 275, p. 127457.

Murray, Portia, Julien Marquant, et al. (2020). “Optimal transformation strategies for buildings, neighbourhoods and districts to reach CO2 emission reduction targets”. In: *Energy and Buildings* 207, p. 109569.

Murray, Portia, Mathias Niffeler, et al. (2019). “Optimal retrofitting measures for residential buildings at large scale: A multi-objective approach”. In: *Proceedings of the International Building Simulation Conference, Rome, Italy*, pp. 2–4.

Nations, United et al. (2012). “World urbanization prospects: the 2014 revision”. In: *CD-ROM Edition*.

Oakridge national laboratory (2018). *ORNL Technical report TM 2018/756*. Tech. rep. ORNL.

Pellegrini, Marco and Augusto Bianchini (2018). “The Innovative Concept of Cold District Heating Networks: A Literature Review”. In: *Energies* 11.1. ISSN: 1996-1073. URL: https://www.mdpi.com/1996-1073/11/1/236.

Schmid, Felix (2008). “Sewage water: interesting heat source for heat pumps and chillers”. In: *Proceedings of the 9th International IEA Heat Pump Conference, Zürich, Switzerland*, pp. 20–22.

Settembrini, G et al. (2017). *ClimaBau–Planen angesichts des Klimawandels*. Tech. rep.

SIA, SIA et al. (2015). *Raumnutzungsdaten für die Energie-und Gebäudetechnik*. Tech. rep.

Smith, Daniel W (1996). “Cold regions utilities monograph”. In: American Society of Civil Engineers.

Sommer, Tobias et al. (2020). “The reservoir network: A new network topology for district heating and cooling”. In: *Energy* 199, p. 117418.

Staub, Peter, Heinz Rütter, et al. (2014). *“Die” volkswirtschaftliche Bedeutung der Immobilienwirtschaft der Schweiz*. HEV.

Sulzer, M (2011). “Effizienzsteigerung mit Anergienetzen: Potentiale–Konzepte–Beispiele”. In: *NRETIS Switzerland*.

U.S. Department of Energy (2014). *DETERMINING ELECTRIC MOTOR LOAD AND EFFICIENCY*. Tech. rep. DOE. URL: https://www.energy.gov/sites/prod/files/2014/04/f15/10097517.pdf.

Wetter, Michael et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270.

Wohnfläche, Wer braucht wieviel (2017). *statistik. info 2017/04*. Tech. rep.

Simulation Model as a Service (SMaaS): A concept for integrated deployment, execution and tracking of system simulation models

Philipp Emanuel Stelzig[1] Benjamin Rodenberg[1]

[1]simercator GmbH, Germany, {philipp.stelzig,benjamin.rodenberg}@simercator.com

Abstract

System simulation is dealing with increasingly multiphysical and cyber-physical systems that involve multiple engineering domains. In development and production, system manufacturers often rely on supplier parts and their digital representations. To deal with this inherently collaborative setting in a more efficient way we propose a concept of *Simulation Model as a Service* (SMaaS) developed at simercator. In this article, we apply established workflows from software engineering to system simulation to create more efficient workflows, discuss the compliance with technical, economic, and regulatory requirements, and present a software for digital supply chain management that implements SMaaS.

Keywords: System simulation, FMI, Modelica, traceability, as a service, continuous integration, microservice

1 Introduction

System simulation is an integral part in the development of mechatronic and cyber-physical systems. These are increasingly multiphysical systems that involve multiple engineering domains and – to an increasing extent – also algorithms from fields like machine learning. This makes system simulation inherently collaborative: System manufacturers rely on specialized suppliers to deliver subsystems, *e.g.* battery packs for electric vehicles or HVAC systems for cruise ships. In addition to the physical hardware, system manufacturers also require corresponding digital assets from the suppliers like geometry data, documentation, and simulation models.

Today, system simulation experts have access to a rich set of tools that allow to model and solve system simulation models. However, little attention has been given to the actual *process* of exchanging simulation models between the various parties that are involved. Consequently, there are few technical means supporting this *supply chain of simulation models*. In this article, we introduce a concept developed by simercator that we call *Simulation Model as a Service* (SMaaS). Our proposed workflow transfers established best-practices from software engineering to system simulation to obtain and manage 3rd party simulation models and is designed to comply with technical, economic, and regulartory requirements. Finally, we present with *simercator hub* an actual implementation of a digital supply chain for simulation models implementing SMaaS.

1.1 State of the art: Modeling and file-based model exchange

System simulation knows excellent modeling languages and software tools for modeling and simulation of complex systems. In a multidisciplinary or multiphysics context, Modelica (Modelica Association 2023) is very popular and numerous softwares implement the language. The Functional Mock-up Interface (FMI, see Modelica Association (2022)) has become a de-facto standard for exchanging models between tools of different vendors: See the long list of tools[1] that now support FMI. The list of tools reaches well beyond the system simulation domain and includes also tools for models based on partial differential equations (PDEs) from solid mechanics using the finite element method (FEM) or computational fluid-dynamics problems using finite volume solvers. A simulation model according to the FMI is a ZIP-file called *Functional Mock-up Unit* (FMU) that contains – as defined in the standard – a computational model implementing the FMI and model information as structured text specifying the model inputs, outputs, and additional meta-information. Thus, the FMI standard allows to share and distribute simulation models by exchanging FMU files.

However, the file-based exchange of simulation models is not standardized and the result of a bilateral, often personal contact between model owner and model user. If model owner and model user are representatives of different companies a legal framework such as non-disclosure agreements (NDA) is often required. This bilateral approach works for a small number of models and model users. However, as a manual process it is slow, error-prone, cannot be opened to a broader audience, and is costly. If these complications prove unmanageable, simulation experts have to fall back to making their own simulation models for 3rd party components, often merely relying on datasheets.

Circuit simulation is a special case: Virtually all commercial circuit simulation tools derive from the same open sourced SPICE simulator (Vladirmirescu 2011). Other than in mechanics-centered system simulation, this made it possible to exchange at least elementary components or simple SPICE netlists (plain text). However, complex circuit models developed in commercial tools can usually not

[1] List of supporting tools https://fmi-standard.org/tools/

DOI 10.3384/ecp20433

be imported in tools from different vendors, especially if the models are encrypted. There are model exchange platforms like Ultra Librarian.[2] Most semiconductor companies offer SPICE models for elementary components for download, often without access restrictions. Netlists for complex products are typically not openly available.

For the exchange of CAD models for supplied parts, geometry models are mostly reduced to the necessary outer features. The resulting CAD files are often made available as downloadable objects, either on company websites or via specialized CAD portals like PARTcommunity[3] or GrabCAD.[4] While sometimes no special access restrictions are imposed, legal restrictions usually apply.

Technical means to deliver, track, and maintain digital assets of real hardware products are currently in their infancies. Therefore, major efforts are being made in research projects and industrial associations. The Industrial Digital Twin Association (IDTA)[5] has formulated the *Asset Administration Shell* (AAS, see Federal Ministry for Economic Affairs and Climate Action (2022)) as a super-standard that can capture a wide variety of digital assets and even integrate with existing standards by means of *sub-models*, like with the simulation sub-model and the FMI (Industrial Digital Twin Organization 2022). The industry association Catena-X aims at establishing “a trustworthy, collaborative, open and secure data ecosystem”[6] and is developing an open source implementation.[7]

1.2 SaaS trend in simulation

In recent years, the simulation software industry is steadily moving from local software installations on users' computers to cloud services using *Software as a Service* (SaaS). This impacts the collaboration and exchange of simulation models, because either the entire modeling and simulation process is executed through the web browser in the cloud, or modeling is done locally and only the execution of computationally heavy simulations is deferred to a cloud infrastructure. One major advantage here is the good availability and on-demand allocation of computational resources. Most major simulation software vendors offer SaaS solutions. New developments tend to be cloud-native anyway, like *e.g.* SimScale[8] for PDE-based simulation or Modelon Impact[9] with its cloud-native modeling environment for Modelica.

Some simulation software vendors now allow sharing of easy-to-use and ready-to-use, *i.e.* executable simulation models as interactive web applications. This trend can also be viewed in the broader context of the so-called *Democratization of CAE* (Taylor et al. 2015). Interactive web applications can be generated *e.g.* with Modelon Impact's App Mode[10], or Siemens Simcenter Webapp Server.[11]

1.3 Related work

In the automotive industry, the research project SET Level proposed a “Credible Simulation Process” (SET Level 2021) to ensure simulation quality when integrating simulation models from different stakeholders. The maritime industry developed the *Open Simulation Platform* (OSP)[12] as an open source project[13] providing a co-simulation solution tailored to the needs of the maritime industry to “create a maritime industry ecosystem for co-simulation of 'black-box' simulation models”.[14] In the context of the OSP, the open source project *FMU-Proxy*[15] has been developed to enable single FMUs to be co-simulated via network, see Hatledal, Styve, et al. (2019) and Hatledal, Zhang, Styve, et al. (2019). FMU-Proxy enables model owners to share an original FMU by means of a proxy of the FMU, meaning the proxy FMU looks identical to the original FMU from the perspective of the user, but internally it features remote procedure calls (RPC) to a server, which evaluates the original FMU. In Hatledal, Zhang, Styve, et al. (2019, chapters 3, 4.1) one possible use-case of FMU-proxy is illustrated: Model owners can list original FMUs delivered as proxy FMUs to authorized model users through a “discovery service”. Similar to FMU-Proxy, UniFMU by Legaard et al. (2021) also uses a FMU as a communication interface and network communication, but as a means to enable running computational models in languages or tools that do not support the FMI. In Schranz et al. (2021), UniFMU is extended in that users can encapsulate the computational model hidden behind UniFMU in a Docker[16] container with all dependencies included, in order to improve portability; remote execution is not supported with UniFMU as of Schranz et al. (2021). In the defense sector, exchange of simulation models and physically distributed cooperative simulations (co-simulation) are natural requirements. For example in combat simulations, when simulated actions have to be synchronized amongst involved ships, aircraft, and vehicles. NATO has developed the concept of Modeling and Simulation as a Service (MSaaS) and created reference architectures regarding possible realizations (Siegfried, Lloyd, and TVD Berg 2018; Hannay and Tom van den Berg 2017). Note that MSaaS is different from the SMaaS that we propose, in that MSaaS describes a distributed execution framework for simulation, while SMaaS is a concept for providing building blocks to assemble simulations from.

[2] https://www.ultralibrarian.com/

[3] https://b2b.partcommunity.com

[4] https://grabcad.com/

[5] https://industrialdigitaltwin.org/

[6] https://catena-x.net/en/vision-goals

[7] https://github.com/eclipse-tractusx

[8] https://www.simscale.com/

[9] https://modelon.com/modelon-impact/

[10] https://help.modelon.com/latest/release_notes/impact_2023_2/

[11] https://plm.sw.siemens.com/en-US/simcenter/systems-simulation/webapp-server/

[12] https://opensimulationplatform.com/

[13] https://github.com/open-simulation-platform

[14] https://open-simulation-platform.github.io/

[15] https://github.com/NTNU-IHB/FMU-proxy

[16] https://www.docker.com/

Trauer et al. (2022) outline strategies to facilitate the exchange of digital twins, including simulation models, by means of trust and quality indicators.

1.4 Outline

In section 2, we discuss the differences of any formalized process to exchange simulation models compared to exchange processes for other digital assets. In section 3 we review various obstacles (technical, economic, legal, regulatory) that occur in such a process and present the resulting requirements in section 4. In section 5 we then present well-established best-practices from software engineering that are applicable to an exchange process for simulation models. In section 6 the SMaaS concept is described. Finally, simercator hub as an actual implementation of this concept is outlined as a showase in section 7, followed by conclusions and suggestions for future work in section 8.

2 Simulation models need dedicated supply chains

In the industries, system manufacturers require simulation models for supplier parts. Other than in academia, where research and simulation is mostly done as a collaborative effort with openly accessible knowledge, in the industries the involved organizations prefer to protect the internal workings of simulation models and at most provide simulation models as ready-to-use black boxes. If model user and owner work in different companies, the motivation for this is to secure a real or perceived value. But also if the model user and owner are part of the same company and merely work in different departments, distributing ready-to-use executables, in particular as FMUs, is common practice.

We will consider any computational model that may take part in a dynamic system simulation and that produces numerical outputs from given inputs as a simulation model. Examples are traditional system simulation models (event-based DAE systems), FEA or CFD simulations (PDEs), control algorithms, and data driven or machine learning models. In the world of system simulation the system under consideration is mostly built from individual submodels. Some typical examples are: 1) active power and aging models for batteries are analyzed in a renewable energy system simulation; 2) a finite element strength simulation for a damper component contributes to an overall simulation of the dynamics of a vehicle suspension; 3) a neural network inference model acts as a control algorithm for object detection in an autonomous vehicle simulation.

All the aforementioned simulation models are mostly files or a collection of files. Hence, bringing simulation models from a model owner to a model user means physically copying files to a location where the model user can access and import them into a system simulation tool. Like with other file-based digital data, this immediately leads to a number of issues: Who owns the digital data? What is the receiving party allowed to do with it? In section 3 we will address these and other questions in detail.

Finally, all involved parties need to agree on an exchange format. If all parties are already using exactly the same simulation software tool, then simulation models can be imported and further processed without major issues. If different tools are in use, the Functional Mock-up Interface (FMI) provides a tool-agnostic simulation model format. If this is not feasible, the parties first need to agree on a common software with a specific version and platform before collaboration can happen.

Here, we want to focus on the following two questions:

1. How are simulation models different compared to other digital assets?
2. Why does the FMI as a data format not satisfy all the resulting requirements?

Executable code Simulation models are executable. In order to run, various software dependencies on the user's system have to be satisfied. If dependencies, *e.g.* runtime dynamic libraries, are not met, the simulation model might not run at all or, even worse, produce wrong results.

Runtime behavior Simulation models have a runtime behavior. They produce simulation output as the result of time-varying input provided to the model and numerically solving a computational model (see the mathematical problem classes above). Inputs are generally only known at runtime and not at the time of model creation or model distribution. Here, CAD models clearly differ, because they are static and do not have a runtime behavior; the information that they convey to a model user is entirely known to the model owner at the time of creation.

Validity range Simulation models can only accurately represent reality for a limited range of inputs. However, one cannot expect a user to generally be able to judge the modeling error while the model owner has expert knowledge that helps to quantify the modeling error. Therefore, a model owner needs to know the inputs a model user is going to provide to the model, especially if the model owner has a liability for the correctness of the results. Naturally, also a model user needs to know for which inputs the model produces valid outputs. This knowledge is required both at modeling time, *i.e.* to ensure that an externally supplied model is used in a meaningful context only, as well as after simulation time, *i.e.* to ensure that a model produced meaningful results.

Maintainability Simulation models may require updates and bugfixes. If an error is discovered, in particular by the actual user, the model owner needs the ability to update the simulation model, or, to shut it down. If the number of users is small and it is known where the model has been deployed in the past, then doing this manually in a bilateral fashion is possible. However, if the number of users is large, or the model distribution is unclear, or it is used in highly critical applications, a manual update process is not feasible or satisfactory.

As a consequence, simulation models need a dedicated solution for the distribution from model owner to model user. It must ensure dependency management, usage control, traceability, and update mechanisms. These features are not part of the FMI and, from our perspective, are outside its scope. As we will show in sections 6 and 7, with a particular client/server architecture that we call *Simulation Model as a Service*, it is possible to build a supply chain for simulation models that considers the aspects above.

3 Obstacles

Today, a model user has to overcome many obstacles before a simulation model can actually be run. These obstacles are of technical, economic, legal, or regulatory nature.

3.1 Technical: Dependency management

The FMI is a widely used data format for the exchange of simulation models. A major issue of the FMI is the lack of dependency management. Both source code FMUs and compiled FMUs require dependencies to be present in the execution environment. If they are not met or wrong versions are provided, FMUs cannot run or might produce wrong results. Model users typically cannot resolve dependencies without help from the model owner. The FMI standard states that “FMUs must reduce their dependency on operating system services” (Modelica Association 2022, section 1.3) and that “tool dependencies must be documented” (Modelica Association 2022, section 2.5.3). However, beyond the requirement for written documentation in the FMU’s `documentation` folder (Modelica Association 2022, section 2.5.1.1), the FMI does not provide machine-readable dependency management to support an automatic and user-friendly process.

3.2 Economic: Real and perceived value

Economic value is attributed to simulation models by both model owners and users. For users, the economic value lies mainly in working time savings, if the alternative for them means creating their own model. This allows us to roughly approximate the value for users. For owners, this is more difficult: Simulation models, at least if they represent a real product sold by the model owner, have little market value on their own, because they are merely descriptive and can only generate revenue in combination with the real product they represent. In this case, the economic value of a simulation model is mainly the added value for the buyer. However, the costs to create a simulation model are usually very well known or can be estimated as the personnel costs spent on modeling, costs for the software tools, and, if applicable, the cost for model validation. Finally, there is a negative economic value, *i.e.* the risk of economic damage that might result from the loss of intellectual property, or from damage claims for a flawed simulation model.

There is no good literature available on this subject. At simercator we have experienced that managers, decision makers, and non-experts in simulation tend to make the following mistakes:

Mistake 1 Cost for model creation equals the market value of the models.

Mistake 2 The information *contained* in a simulation model is equal to the knowledge that *went into creating* the model.

As a consequence, managers and decision-makers tend to be over-reluctant when sharing simulation models, because they overestimate the risk of distributing models. Hence, there appears to be a significant discrepancy between actual and perceived value.

3.3 Legal: Intellectual property and liability

Simulation models contain intellectual property and the models themselves are subject to copyright and property law. Model owners need to ensure that distributing simulation models does not affect any rights. Export control laws also apply.

Simulation softwares, including Modelica tools, offer the possibility to encrypt simulation models. However, the processing of encrypted models is usually vendor-specific, and, in some cases, even version-specific. The FMI’s intellectual property protection relies on the binary compilation. To lesser extent, source code FMUs can be obfuscated, but by nature the internals of the model remain exposed. With simulation models, where phenomenological or reduced order models are used, it is quite often already hard to reverse-engineer product features from a descriptive model. Restricting the distribution, *i.e.* the *copying* of files that contain simulation models is not possible through technical means, but one can restrict the execution, *e.g.* by requiring runtime licenses or decryption keys.

Another big legal concern is liability for correctness. Mathematically, it is not possible to quantify *a priori* the modeling error of a simulation model, *i.e.* the accurate representation of physical reality through the model for *any input* that a user provides. The same applies to control algorithms and becomes even more pressing when the model itself evolves when the user provides new input during model usage, *e.g.* with machine learning models.

As of today, we do not know of effective mechanisms for owners to track usage after delivering the model to the user. Usage control is limited to restricting ranges of input variables and other mechanisms that are compiled into a simulation model. However, any control mechanism built into a model at compile time is based on previously known or anticipated model usage and model behavior. Consequently, without knowledge about actual usage, the model owner cannot improve on control mechanisms or discover unexpected runtime behavior. Likewise, a user cannot be warned and potential damage can only be analyzed *a posteriori* and only if the user keeps record of specific model versions together with the input and ideally output values.

3.4 Regulatory: Traceability, explainability

The recent years have witnessed the widespread adoption of computational methods from the fields of machine learning (ML), statistical methods or artificial intelligence (AI). Applications range from the prediction of human behavior for marketing purposes, artificial generation of text or graphics, chatbots, or even crime prediction, to industrial applications like object detection in quality control or control algorithms. Due to the "learning" nature and the dimensional complexity, their operational behavior is hard to predict with mathematical means and might evolve over time. Lawmakers all over the world will most likely enforce regulation on their usage soon. The European Union (EU) has taken a leading role. In 2018, it has already issued the *Ethics Guidelines for Trustworthy AI* (European Commission 2018). In 2021, the European Commission has published a proposal for the *EU Artificial Intelligence Act* (EU AI Act) (European Commission 2021b).

AI and ML are already used as computational models within system simulation, mostly as control algorithms. But also the other way round: System simulation models themselves can be used as part of AI-based or ML-based control algorithms to simulate the physical reality that is being controlled, *i.e.* as *simulation digital twins* (Boschert, Heinrich, and Rosen 2018).

The European Commission (2018, Chapter II.1.4) already demanded *traceability* and *explainability* from AI systems. The EU AI Act proposal is more specific: "[H]igh-risk AI systems" (European Commission 2021a, Annex I, Annex III) are required to implement "automatic recording of events ('logs')" (European Commission 2021b, Articles 12, 20) as well as "[c]orrective actions" (European Commission 2021b, Article 21) in order "to bring [the high-risk AI system] into conformity, to withdraw it or to recall it, as appropriate" (European Commission 2021b, Article 21). High-risk applications for AI systems according to European Commission (2021b, Section 5.2.3) and European Commission (2021a, Annex III) comprise "[m]anagement and operation of critical infrastructure [...] operation of road traffic and the supply of water, gas, heating and electricity".

Hence, it might very well be possible that model owners who provide certain computational models will legally have to ensure automated mechanisms to control distribution, maintainability (corrective action), and usage control (record-keeping and explainability).

4 Requirements

From section 3 we can now derive requirements for a realization of a supply chain for simulation models (section 2). The most important are:

Model owner A model owner must be able to

(RO1) control model distribution,
(RO2) enforce control, logging, and monitoring of model inputs and outputs,
(RO3) enforce a location for storage and execution,
(RO4) update or shut down distributed models,
(RO5) comply with data privacy laws.

Model user A model user must be able to

(RU1) obtain models ready-to-use and fitting the technical and organizational (data traffic) needs,
(RU2) know if his model usage produced valid outputs, both at modeling time and after simulation time,
(RU3) know which data is processed and communicated to the model owner, both *a priori* and *a posteriori*,
(RU4) retain full control over his own simulation models (no forced updates),
(RU5) rely on the fact that data logging does not reveal any of his own models or data.

Simulation model supply chain A supply chain for simulation models must allow for

(RC1) parallel model delivery and execution,
(RC2) integration with other digital supply chains,
(RC3) integration with simulation tools and standards.

5 Learning from software engineering

Deployment, monitoring, and maintenance of ready-to-use software have been a core challenge in software engineering for decades. Therefore, we seek best practices and inspiration in software engineering for a supply chain for simulation models.

5.1 Dependency management, software deployment, and maintenance

Modern operating systems use package managers and specific package formats to install software from given repositories, like *e.g.* Ubuntu's official package repository providing `.deb` Debian packages.[17] Package managers that take care of dependency resolution exist for all major operating systems: `apt`[18] or `rpm`[19] (Linux), `brew`[20] (MacOS), and `winget`[21] (Windows). There are also platforms like Google Play[22] or Apple's App Store[23] for mobile applications as well as dockerhub[24] for microservices in mostly cloud applications.

While FMI's role could be intepreted as a package format, there is – to the best of our knowledge – neither a *package manager with dependency management for simulation models* nor a *package repository for system simulation models*. (This statement does not apply to circuit simulation models, see section 1.1.)

[17] `https://packages.ubuntu.com/`
[18] `https://wiki.debian.org/AptCLI`
[19] `https://rpm.org/`
[20] `https://brew.sh/`
[21] `https://learn.microsoft.com/en-us/windows/package-manager/`
[22] `https://play.google.com`
[23] `https://apps.apple.com`
[24] `https://hub.docker.com/`

5.2 Services for source code development and exchange

In the software world, almost all developers rely on web-based services for exchanging software. Services like GitHub,[25] GitLab,[26] or Bitbucket[27] are used for development, collaboration with others, building, testing, and other tasks. These platforms support their users with features like version management, release management, issue tracking, continuous integration, automated testing, vulnerability scanning, user management, usage analytics, rights management, and collaborative development. The code repositories are available as open to the public (open source), or access is restricted to specific users in private repositories or on private instances of these platforms.

For model *development*, simulation experts often use the same services as software developers. In particular, version control systems like Git[28] and collaborative environments like GitHub or GitLab. Note that simulation modeling environments like *e.g.* Modelica tools take the role that integrated development environments (IDE) have in software development. Some simulation modeling environments like *e.g.* Dymola[29] already integrate version control.

5.3 Quality measures

In software engineering, developers expect several quality indicators from a software library: Update cycles, the existence of automated test and build toolchains, reliable tracking and handling of issues, but also social indicators like the number of contributors, regular commits, or response times for reported issues. Also, a large portion of software projects use the semantic versioning scheme `<major>.<minor>.<bugfix>`.[30]

In recent years, such measures are being adopted in model development, too. For example, the Modelica libraries IBPSA (Wetter, Treeck, et al. 2019), Buildings (Wetter, Zuo, et al. 2014) and AixLib (Müller et al. 2016) use issue tracking, automated test and build toolchains, as well as semantic versioning based releases.

6 SMaaS concept and architecture

The Simulation Model as a Service (SMaaS) concept developed at simercator proposes a simulation model repository software combined with a scalable, microservice-based computing backend. The core idea is to offer a simulation model not merely as a downloadable and executable file to the model user. Instead, model owners can offer their simulation model as an executable service, where the owner retains full control over model execution, distrubtion and usage. The SMaaS architecture is depictseded in the following Figure 1 and a concrete implementation is presented in the following section 7.

SMaaS incorporates different realizations of execution services. An integral part of SMaaS is a special execution service which we refer to as *simulation model streaming*.

Streaming *Simulation model streaming* describes original models being executed in the computing backend, while the user only uses a model *doppelgaenger* that is distributed as a downloadable file. The *doppelgaenger* model can be offered in different formats (*e.g.* FMI) to allow the import into various tools. Every time the model *doppelgaenger* is executed, the requested action (*e.g.* FMI function calls like `fmi2DoStep`) is deferred to the computing backend via a web-based RPC call, effectively “streaming” input data and results back and forth between *doppelgaenger* and original model.

This idea is similar to FMU-proxy by Hatledal, Zhang, Styve, et al. (2019). However, FMU-proxy uses a TCP/IP-based socket-to-socket communication where one proxy FMU (the equivalent to our *doppelgaenger*) communicates with one FMU-proxy server instance. To achieve a scalable execution service and to be able to run multiple instances of the same model in parallel when using SMaaS, we use microservices in the computing backend and a dedicated communication backbone for routing incoming requests to dedicated containers. For every request from a *doppelgaenger*, we instantiate a dedicated container from a container image repository. We then create a copy of the original simulation model in the container to perform the computations. In this fashion, the dependencies only need to be resolved once with a suitable container image, whereas the *doppelgaenger* delivered to the user does not require any special dependencies.

As a consequence, one can deliver simulation models as *doppelgaengers* in a format that the original simulation model does not even support. This comes at the price of matching model execution requests to corresponding evaluations of the original model during communication. This principle has also been used in FMU-proxy to “import FMI 1.0 models in software that otherwise only supports FMI 2.0”,[31] and furthermore in UniFMU by Legaard et al. (2021). The advantage of resolving dependencies of an original simulation model in a microservice has also been exploited by UniFMU (Schranz et al. 2021).

Also, streaming inputs back and forth allows us to implement observer and maintenance mechanisms in the SMaaS concept to satisfy requirements such as control of usage, distribution, as well as ensuring traceability and reproduceability as the basis for explainability.

Browser-based Being able to execute original simulation models via web-based RPC calls on a computing backend makes it possible to offer access over the web

[25] `https://github.com`

[26] `https://gitlab.com`

[27] `https://bitbucket.org/`

[28] `https://git-scm.com/`

[29] `https://www.3ds.com/products-services/catia/products/dymola/model-design-tools/`

[30] `https://semver.org/`

[31] `https://github.com/NTNU-IHB/FMU-proxy/blob/master/README.md`

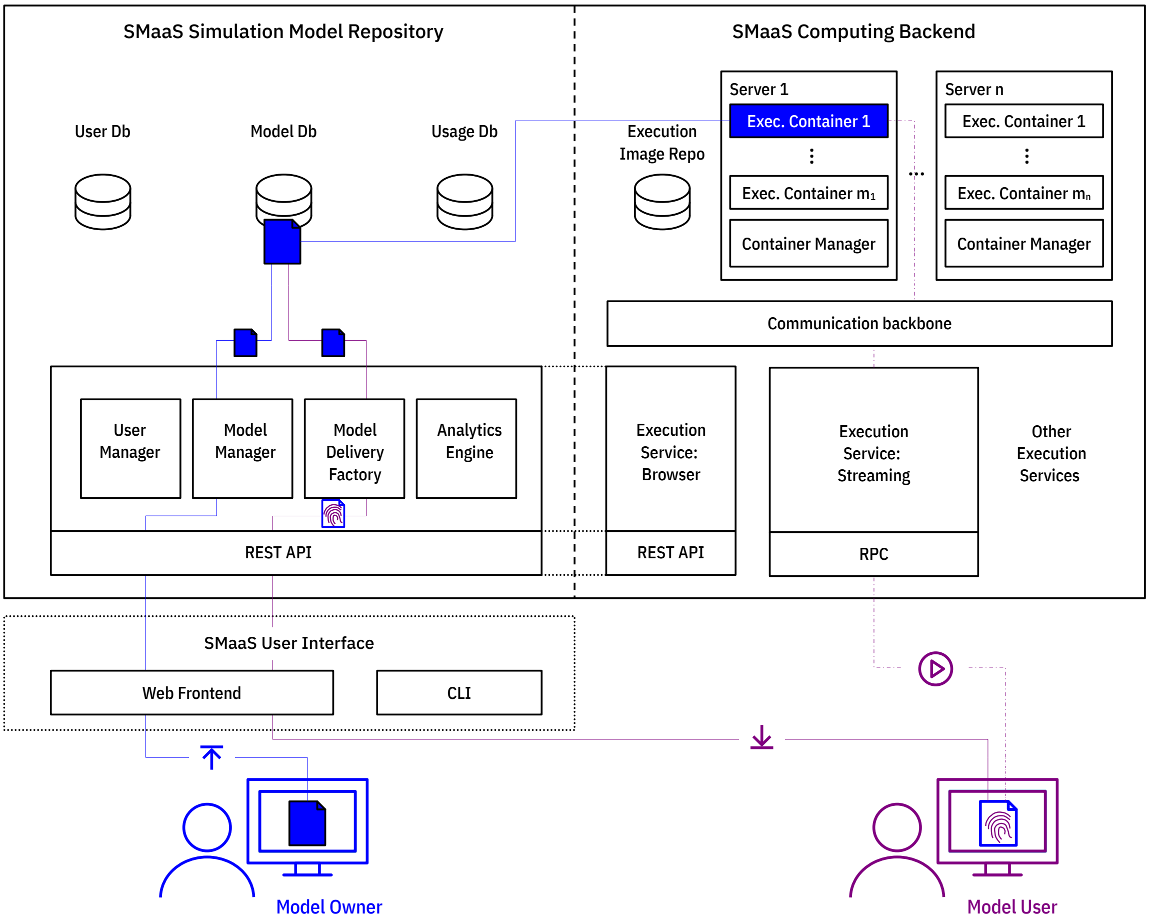


Figure 1. SMaaS concept architecture with illustration of model delivery and simulation using streaming execution

browser. Hence, within the SMaaS concept it is possible to implement a web-based frontend as an execution service, where model users can trigger model evaluations from a browser. This is particularly useful to perform simple runs in an explorative fashion, and also for users who do not have access to a simulation tool or lack the required knowledge. Hence, model owners can allow browser-based execution for a broader user audience without having to develop a dedicated web application.

Individualized model delivery Not in all applications simulation model streaming is an option. For instance, IT guidelines of a company could forbid communication to external services, or the target platform could have no (sufficiently reliable) network connection, *e.g.* a production machine or an autonomous vehicle. In these cases the only option is to deliver a copy of the original model to the model user – including all the drawbacks mentioned before. However, with the simulation model repository infrastructure it is possible to individualize copies of the simulation model and to augment them with additional control functionality. We call this *Simulation Model on a Leash* (SMoaL). It is ongoing research at simercator and might be subject to a future publication.

Finally, the SMaaS concept is not specific to the FMI, but can be adopted to future (*e.g.* microservice-based) exchange standards for computational models. For instance the Open Neural Network Exchange[32] format (ONNX).

Evaluation With the exception of RO5 and RU1, the fulfillment of the requirements from section 4 is inherently possible if SMaaS is implemented with the streaming execution service. Depending on the actual implementation, also RO5 and RU1 can be satisfied. In order to fulfill RU2 one can expose a subset of the model owner's monitoring mechanisms implemented for RO2 to the model user. We summarize the fulfillment of requirements in Table 1.

Table 1. Requirement analysis: SMaaS with streaming execution service. (* depends on implementation and deployment)

RO1	RO2	RO3	RO4	RO5	RU1	RU2	RU3	RU4	RU5	RC1	RC2	RC3
✓	✓	✓	✓	✓*	✓*	✓	✓	✓	✓	✓	✓	✓

7 Realization: simercator hub

At simercator we develop simercator hub[33] as a product that implements the SMaaS concept introduced above.[34]

[32] https://onnx.ai/

[33] https://simercator.com/product/

[34] Since the implementation is proprietary and part of commercial activity, we cannot not provide implementation details

7.1 Implementation and features

simercator hub offers a web frontend for user management, uploading or connecting to a model, and the possibility to share models with others. Simulation model streaming is implemented and FMU models in FMI 2.0 (both model exchange and co-simulation) are supported as original models that are then distributed via *doppelgaengers*. Note that the FMU *doppelgaenger* can be compiled and executed on platforms different from the ones the original FMU is supporting.

Our recent product iteration implements browser-based model evaluation, where a web-interface is auto-generated from a model's inputs and outputs and other meta-information. Python-native simulation models are supported as well, *e.g.* data-driven models, and the proprietary Python module `simercator` helps model owners to wrap their simulation models and define inputs and outputs in a single `main.py` file, so that simercator does not intrude the actual model implementation. Built-in visualization features allow the model owner to provide interactive 2D and 3D plots to the user through the web frontend.

We also feature a basic semantic versioning system that allows to enable and disable specific model versions. Furthermore, we provide integrated model usage data acquisition and basic analytics in an integrated dashboard.

simercator hub is designed as a licensable sofware with different commercial license options: 1) host a private instance of simercator hub, *e.g.* on company owned server infrastructure, or 2) work on an instance hosted and managed by simercator with the option to have a client-specific dedicated instance.

7.2 Showcase: FEM model as a service into vehicle system simulation

We now provide a fictitious showcase how simercator hub can help to achieve a significant speed-up in setting up and executing a system simulation, as illustrated in Figure 2. Here, a system engineer wants to perform a vehicle dynamics simulation for a vehicle climbing a curb at a comparatively high speed ($\approx 10\frac{\text{km}}{\text{h}}$). The simulation is used to predict potential damage to a hard rubber absorption buffer and effects on passenger comfort. To assess damage, the stresses in the absorption buffer need to be computed using FEM analysis (FEA), while the analysis of passenger comfort requires us to include a suitable buffer model into the overall vehicle system simulation. The system modeling is done in a Modelica tool (OpenModelica[35] (Fritzson et al. 2020)). In the following example, we will refer to the system engineer owning the vehicle system dynamics model as “Alice” and to the simulation engineer owning the buffer FEM model as “Bob”.

Typical workflow today Today, Alice would typically use a surrogate model for the absorption buffer in her system simulation (nonlinear spring). The system simulation result for the buffer compression is then handed over to Bob to carry out the damage assessment using the FEM model. This is slow and error-prone, because the construction of a meaningful surrogate model requires a known force-compression-dataset. Alternatively, Bob can provide a FMU that contains a FEM model of the absorption buffer to Alice. However, the solver of choice (Calculix[36] (Dhondt 2004)), does not provide an FMI interface that could be used here.

Simulation model exchange using SMaaS As a preparation, Bob needs to prepare a Python script that wraps the Calculix solver of the (for simplicity) elastostatic FEA. The script calls the actual FEA as an external process. Then PythonFMU[37] (Hatledal, Zhang, and Collonval 2020) is used to turn the script into a co-simulation FMU (FMI 2.0). The FMU accepts the enforced displacement on the buffer surface in x, y, and z direction as inputs. It computes and outputs the reaction force in corresponding directions x, y, z and also returns the maximum von Mises stress from the buffer's bulge.

Figure 3 illustrates the showcase workflow with simercator hub. After logging in, Bob uploads the FMU from above to the simercator hub instance, together with text-based meta-information (or as a JSON file). The simercator hub instance provides pre-built docker images with a collection of pre-installed open source solvers, including one with Calculix that Bob uses. After that, Bob creates an account for Alice on simercator hub and authorizes Alice to view and download the FEA model as a *doppelgaenger*.

Alice now logs in to the simercator hub instance to browse models that she has access to. She can also assess whether the model fits her needs from the metadata. Alice uses the download option to retrieve the model *doppelgaenger* of the model provided by Bob. It only contains a communication library, the same `modelDescription.xml` of the original FMU, and

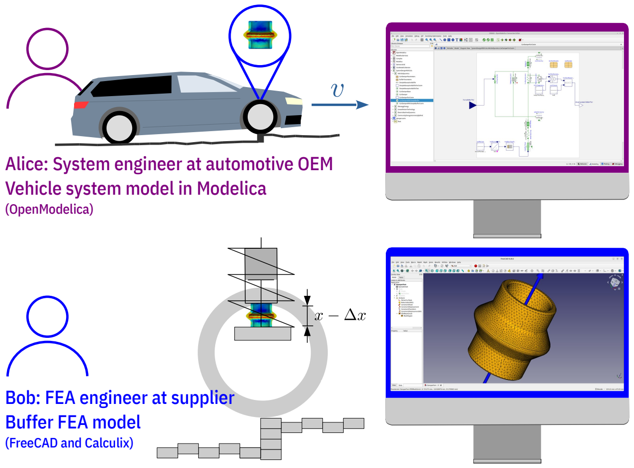

v
Alice: System engineer at automotive OEM
Vehicle system model in Modelica
(OpenModelica)
x − Δx
Bob: FEA engineer at supplier
Buffer FEA model
(FreeCAD and Calculix)

Figure 2. Showcase for a SMaaS scenario

[35] https://openmodelica.org/

[36] http://www.calculix.de/

[37] https://github.com/NTNU-IHB/PythonFMU

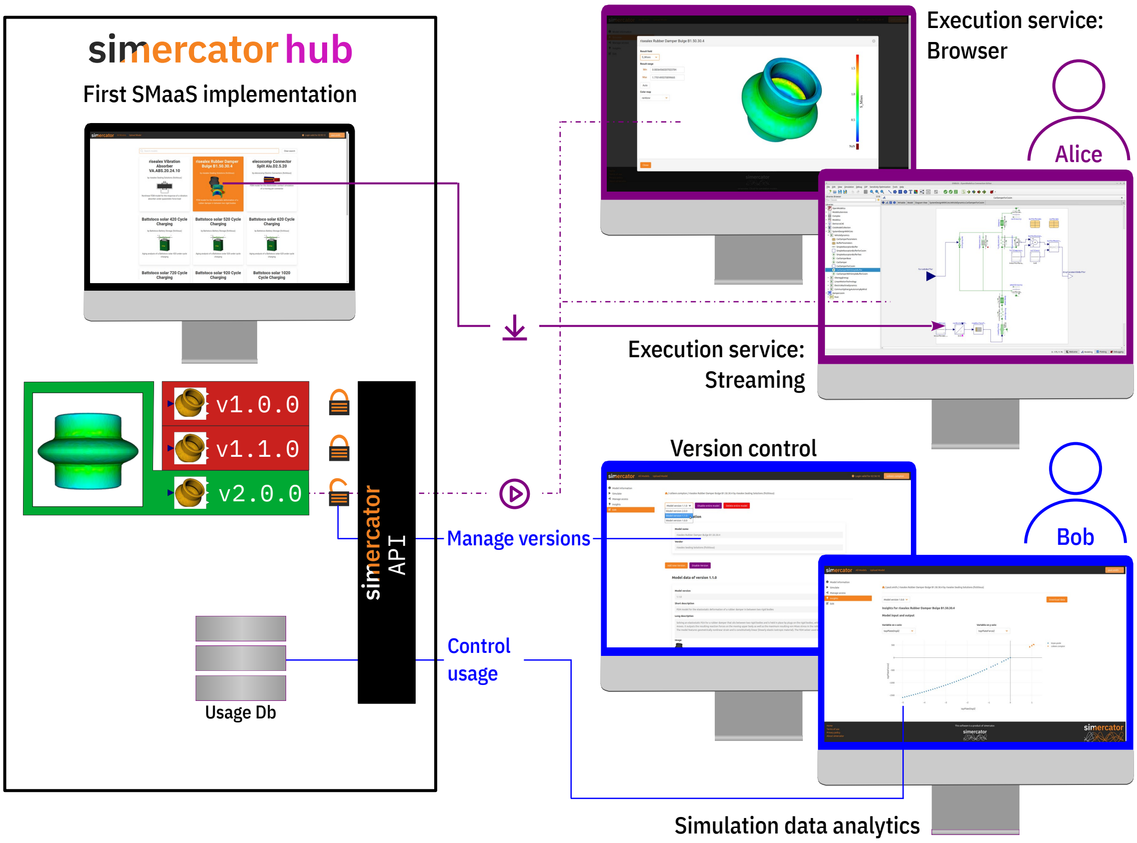

simercator hub
First SMaaS implementation
v1.0.0
v1.1.0
v2.0.0
simercator API
Usage Db
Manage versions
Control usage
Execution service: Browser
Alice
Execution service: Streaming
Version control
Bob
Simulation data analytics

Figure 3. simercator hub implementation of SMaaS. Screenshots of use in showcase.

an individually generated token that ensures that only authorized users can execute this *doppelgaenger*. Because it contains the same `modelDescription.xml`, to Alice it looks just like the original model.

Alice wants to use the vertical displacement that originates from her suspension model as an input for the buffer *doppelgaenger*. Then, she wants to use the vertical reaction force computed by the buffer *doppelgaenger* as an input for her suspension model. Therefore, Alice exports the suspension model with corresponding input and output connections into a FMU for co-simulation. Then she combines both the suspension FMU and the FMU with the *doppelgaenger* into one weakly coupled co-simulation. When the co-simulation is run, the *doppelgaenger* model for the buffer establishes communication with the simercator hub instance via the streaming process explained above. Bob does not know about the surrounding system simulation. Alice's simulation model is entire invisible to Bob. Likewise, Alice can only access the outputs of Bob's model and no internal details are exposed, such as the FE mesh or material properties.

The workflow described above takes Alice a few minutes (log-in, download, co-simulation setup). Also, Bob can now make this simulation model available to multiple experts via simercator hub's user management.

With simercator hub, also an alternative approach is possible: Using the browser-based evaluation, Alice can query the response forces for various displacements from the browser. This allows her to create a surrogate model with a nonlinear spring from data of the FEM model.

8 Conclusion and future work

We described why the exchange of simulation and computational models between system manufacturers and suppliers of mechatronic and cyber-physical systems needs a dedicated solution ("digital supply chain"). Then we outlined that today's file-transfer based exchange cannot satisfy natural technical and possible future regulatory requirements for the operation and maintenance of rolled out models. With SMaaS we have formulated a concept and an architecture that can provide a suitable solution for the exchange of simulation models and integrates with existing simulation tools and standards like FMI. Finally, we have presented our product simercator hub that implements SMaaS, illustrated how it can satisfy the identified requirements, and demonstrated SMaaS and simercator hub within a collaborative system simulation use case.

A key point for SMaaS will be whether and how organizations will accept communication to the outside when it comes to simulation, as some form of communication and collaboration is required by all implementation variants of SMaaS. Recent adoptions of package managers and software as a service (SaaS) solutions have shown that companies are willing to implement this if the gains are sufficient. This has been the case for business applications and can now be observed with the rise of SaaS solutions for simulation software. Additional efforts have to be undertaken to improve dependency management for FMI-based models or other standards that may arise in the future. Also, it remains to be seen how simulation models will be affected by future regulation. From a technical perspective, additional research is needed on performance speedup in network-based co-simulation. Finally, adoption of SMaaS or other *package repository like* solutions also depends on economic considerations with potential adopters. SMaaS implementations like simercator hub can provide infrastructure for digital supply chains. Whether companies will continue to share simulation models purely request-driven like today or use it as a digital service to differentiate from competitors will affect the rate of adoption.

References

Boschert, Stefan, Christoph Heinrich, and Roland Rosen (2018). “Next generation digital twin”. In: *Conference: TMCE 2018*. Vol. 2018. Las Palmas de Gran Canaria, Spain, pp. 7–11.

Dhondt, Guido (2004). *The finite element method for three-dimensional thermomechanical applications*. John Wiley & Sons.

European Commission (2018). *Ethics Guidlines for Trustworthy AI*. Tech. rep. European Union. URL: https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai.

European Commission (2021a). *Annexes to the Proposal for a regulation of the European Parliament and of the Council laying down harmonized rules on artificial intelligence (Artificial Intelligence Act) and ammending certain union legislative acts.* Tech. rep. European Union. URL: https://ec.europa.eu/newsroom/dae/redirection/document/75789.

European Commission (2021b). *Proposal for a regulation of the European Parliament and of the Council laying down harmonized rules on artificial intelligence (Artificial Intelligence Act) and ammending certain union legislative acts.* Tech. rep. European Union. URL: https://ec.europa.eu/newsroom/dae/redirection/document/75788.

Federal Ministry for Economic Affairs and Climate Action (2022). *Part 1 – The exchange of information between partners in the value chain of Industrie 4.0 (Version 3.0RC02)*. Tech. rep. Federal Ministry for Economic Affairs and Climate Action. URL: https://industrialdigitaltwin.org/wp-content/uploads/2022/06/DetailsOfTheAssetAdministrationShell_Part1_V3.0RC02_Final1.pdf.

Fritzson, Peter et al. (2020). “The OpenModelica integrated environment for modeling, simulation, and model-based development”. In: *Modeling, Identification and Control* 41.4, pp. 241–295.

Hannay, Jo Erskine and Tom van den Berg (2017). “The NATO MSG-136 reference architecture for M&S as a service”. In: *Proc. NATO Modelling and Simulation Group Symp. on M&S Technologies and Standards for Enabling Alliance Interoperability and Pervasive M&S Applications (STO-MP-MSG-149)*.

Hatledal, Lars Ivar, Arne Styve, et al. (2019). “A Language and Platform Independent Co-Simulation Framework Based on the Functional Mock-Up Interface”. In: *IEEE Access* 7, pp. 109328–109339. DOI: 10.1109/ACCESS.2019.2933275.

Hatledal, Lars Ivar, Houxiang Zhang, and Frederic Collonval (2020). “Enabling Python Driven Co-Simulation Models With PythonFMU.” In: *Proceedings of the 34th International ECMS-Conference on Modelling and Simulation-ECMS 2020*. ECMS European Council for Modelling and Simulation, pp. 235–239.

Hatledal, Lars Ivar, Houxiang Zhang, Arne Styve, et al. (2019). “FMU-proxy: A Framework for Distributed Access to Functional Mock-up Units.” In: Proceedings of the 13th International Modelica Conference (March 4–6, 2019). Ed. by Anton Haumer. Regensburg: Linköping University Electronic Press, pp. 157–008.

Industrial Digital Twin Organization (2022). *IDTA 02005-1-0. Provision of Simulation Models*. Tech. rep. Industrial Digital Twin Organization. URL: https://industrialdigitaltwin.org/wp-content/uploads/2023/01/IDTA-02005-1-0_Submodel_ProvisionOfSimulationModels.pdf.

Legaard, Christian M. et al. (2021). “A Universal Mechanism for Implementing Functional Mock-up Units”. In: *11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. SIMULTECH 2021. Virtual Event, pp. 121–129.

Modelica Association (2022). *Functional Mock-up Interface for Model Exchange and Co-Simulation Version 3.0*. Tech. rep. Linköping: Modelica Association. URL: https://fmi-standard.org/docs/3.0/.

Modelica Association (2023). *Modelica Specification, Version 3.6*. Tech. rep. Linköping: Modelica Association. URL: https://modelica.org/documents/MLS.pdf.

Müller, Dirk et al. (2016). “AixLib-An open-source modelica library within the IEA-EBC annex 60 framework”. In: *BauSIM 2016*, pp. 3–9. URL: https://github.com/RWTH-EBC/AixLib.

Schranz, Thomas et al. (2021). “Portable runtime environments for Python-based FMUs: Adding Docker support to UniFMU”. In: *Proceedings of the 14th International Modelica Conference*. Linköping University Electronic Press, pp. 419–424.

SET Level (2021). *Credible Simulation Process*. Tech. rep. SET Level Research Project. URL: https://setlevel.de/assets/forschungsergebnisse/Credible-Simulation-Process.pdf.

Siegfried, R, J Lloyd, and TVD Berg (2018). “A new reality: Modelling & Simulation as a Service”. In: *Journal of Cyber Security and Information Systems* 6.3, pp. 18–29.

Taylor, Simon et al. (2015-07). “Grand challenges for modeling and simulation: Simulation everywhere - From cyberinfrastructure to clouds to citizens”. In: *SIMULATION* 91, pp. 648–665. DOI: 10.1177/0037549715590594.

Trauer, J. et al. (2022). “A Digital Twin Trust Framework for Industrial Application”. In: *Proceedings of the Design Society* 2, pp. 293–302. DOI: 10.1017/pds.2022.31.

Vladirmirescu, Andrei (2011). “Shaping the History of SPICE”. In: *IEEE Solid-State Circuits Magazine* 3.2, pp. 36–39. DOI: 10.1109/MSSC.2011.942105.

Wetter, Michael, Christoph van Treeck, et al. (2019). “IBPSA Project 1: BIM/GIS and Modelica framework for building and community energy system design and operation–ongoing developments, lessons learned and challenges”. In: *Iop conference series: Earth and environmental science*. Vol. 323. 1. IOP Publishing, p. 012114. URL: https://github.com/ibpsa/modelica-ibpsa.

Wetter, Michael, Wangda Zuo, et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506. URL: https://github.com/lbl-srg/modelica-buildings.

MoPyRegtest: A Python package for continuous integration-friendly regression testing of Modelica libraries

Philipp Emanuel Stelzig[1]

[1]simercator GmbH, Germany, `philipp.stelzig@simercator.com`

Abstract

Regression testing is a commonly used strategy in continuous integration workflows to ensure reproduceability of outputs. It is widely used in software engineering and model development, including Modelica. In this article we introduce the open source regression testing framework MoPyRegtest written in Python. Its primary focus is to provide Modelica library developers with a simple regression testing tool that features test automation and can be integrated with continuous integration toolchains, in particular for open source developments. In order to simulate the Modelica models for testing analysis, we provide an interface to Modelica simulation tools that have a scripting interface, like *e.g.* `.mos` files. Our current implementation works with OpenModelica. We outline the design and functionality of MoPyRegtest and show its potential usefulness for open source development of Modelica models and libraries.

Keywords: Modelica, regression testing, Python, open source, continuous integration

1 Introduction

The development of Modelica libraries and models requires test and validation. It gives indications to model developers as to the quality and robustness of their developments. Also users request a variety of quality indicators before they choose a certain library. In particular test and test automation, besides a number of other indicators like reputation, community size, update cycles, or response times for support and handling of issues, license conditions and many more. Testing benefits both Modelica library developers and users.

For a user, testing can give an immediate feedback whether a certain Modelica model or library will run on the intended target environment, produce the results its developers intended and which parts of the library have been systematically studied for quality.

For developers, testing carries additional benefits. Unit testing means that debugging does not have to be done with large monolithic models, but rather with smaller individual elements, which helps in locating and isolating issues. Test automation means that developers can quickly detect effects of changes by running test suites, instead of discovering effects manually or, worse, having users discover bugs after releases. Especially with Modelica models, developers implicitely always do tests by running simulations and eventually judging the results as satisfactory. Since Modelica libraries shall feature examples with certain expected results, library developers usually have already created natural candidates to be turned into regression tests. Testing can also open up entirely different development methodologies, like *e.g. test-driven development* (TDD) (Beck (2003)). Indeed, if libraries or library elements are intended to model a certain product's or device's physical behavior rather than containing only generic modeling blocks, it is the *test data*, *i.e.* measurement data, that is available first. A model has to be created such that it correctly predicts the reference data. Hence, TDD can be a valid approach for Modelica library development, too. A TDD approach to simulation has been studied in Onggo and Karatas (2016).

Many test criteria employed during testing of Modelica libraries are basically the same as in software development, ranging from static code analysis, reproduceability to integration testing. The one we will focus on here is *reproduceability* of results through the technique of *regression testing*. Wong et al. (1997) summarizes that "[t] he purpose of regression testing is to ensure that changes made to software [...] have not adversely affected features of the software that should not change". Regression testing is an established practice in many Modelica library developments and a number of tools have been proposed or developed. Basically, regression testing for Modelica library development means evaluating whether simulating a certain library element produces a result that is sufficiently close – in a suitable metric – to a reference data set provided by the developer. As we shall see in section 2, a dedicated, lightweight regression test solution for mostly open source library development can be useful. Especially, if it focuses on test automation and integration into *continuous integration* (CI) or *continuous delivery* (CD) toolchains. This is why have developed *MoPyRegtest*[1] (from Modelica Python Regression testing) from a mere helper into an open source solution that can be run with open source tools only.

The outline of this article is as follows. In section 2 we give a broad overview over the many tools for testing Modelica models and their use in open source Modelica library developments. Section 3 summarizes requirements for a regression testing solution derived from the devel-

[1] `https://github.com/pstelzig/MoPyRegtest`

DOI
10.3384/ecp20443

opers' needs as identified in the previous section. Then, we outline the design of MoPyRegtest and its functionalities. Section 4 covers the currently built-in mathematical metrics that the user can choose from when comparing simulation and reference results. It also shows how we implemented the possibility to provide user-defined metrics. We present a showcase for how to use MoPyRegtest in section 5. Section 6 states conclusions as well as open points, and gives an outlook on future work.

2 State of the art

We outline the state of the art regarding testing and tools for Modelica library development starting with a rough analysis over actual usage of testing solutions. We then derive indicators as to how testing is primarily used today, and where we see that MoPyRegtest can be beneficial to library developers.

2.1 Testing in open source Modelica libraries

In order to have some quantitative indication on the usage of testing solutions for Modelica library development, we did some analysis on the repository collection "Modelica 3rd-party libraries" curated by Dietmar Winkler on GitHub.[2] This collection only covers open source libraries. Hence, from its analysis we cannot derive any insights on testing solutions as they are used in a commercial context. We only considered libraries that had at least one commit since the beginning of 2019. In these repositories we looked for

(C1) any automated tests in the repository's source folder (syntax checking or build/run automation or regression),

(C2) whether there are CI-pipelines set up defining automated tests, *e.g.* in a `.github` or `.gitlab` folder,

(C3) whether there is an automated regression test.

This approach is entirely manual and as such errorprone. To our knowledge, there is no universally adopted practice yet on where to put tests in Modelica libraries and how to formulate or execute them. We found that tests are sometimes hidden deeper inside folder structures and that documentation does not always reveal their existence. As we shall see, there is also some variety when it comes to testing tools used, and sometimes ad hoc testing solutions are implemented that are not obvious to discover. Also, not all developers use GitHub Actions[3] or GitLab CI/CD.[4] Moreover, the nonexistence of test automation, *i.e.* the automated execution of automated tests, does not imply the nonexistence of automated tests themselves. Hence, we might have overlooked or not correctly recognized tests. At this point, we emphasize once more that the numbers in the following Table 1 can at best be interpreted as an indication. We advise against using these numbers in follow-up work. Since we expect at least some corrections to this preliminary and manual analysis, rather than putting the detailed listing here, we put it in an openly accessible repository on GitHub.[5]

Table 1. Estimated relative occurrence of tests in repositories in the GitHub collection "Modelica 3rd-party libraries" as of 14 August 2023. These numbers might not be accurate.

$\exists$ commit since	#(repos)	(C1)	(C2)	(C3)
2019	75	$\approx 28\%$	$\approx 18\%$	$\approx 14\%$
2021	60	$\approx 32\%$	$\approx 20\%$	$\approx 15\%$
2022	45	$\approx 40\%$	$\approx 24\%$	$\approx 20\%$

2.2 Insights

Despite the likely uncertainty in the numbers, we think that one can at least observe tendencies from Table 1:

1. Repositories with more recent activity use more automated tests.
2. Developers prefer other testing strategies first before turning to regression tests.

2.3 Testing tools

In the repositories we looked into, two popular testing tools were BuildingsPy[6,7] for regression testing, and moparser for syntax checking.[8]

BuildingsPy has been developed as part of the Buildings library (Wetter et al. 2014). It supports unit testing and regression testing, but can also orchestrate simulation runs using Dymola[9], OPTIMICA[10] or OpenModelica[11]. The documentation also shows it can visualize simulation results, including regression test results.

moparser is a binary executable by MapleSoft. It is available at MapleSoft's homepage for various platforms. The Modelica Association includes it as the "MapleSim Standalone Modelica Parser" in its Modelica tools list. Several projects use moparser to validate correctness of Modelica syntax as an automated test.

There are also some sophisticated ad hoc regression testing solutions developed for specific projects using various languages. We have found a regression test implementation written in Python as part of the Modelica-Arduino

[2] https://github.com/modelica-3rdparty

[3] https://docs.github.com/en/actions

[4] https://docs.gitlab.com/ee/ci/

[5] https://github.com/pstelzig/modelica-oss-lib-testing-analysis

[6] https://github.com/lbl-srg/BuildingsPy

[7] https://simulationresearch.lbl.gov/modelica/buildingspy/development.html

[8] https://modelica.org/tools.html

[9] https://www.3ds.com/products-services/catia/products/dymola/model-design-tools/

[10] https://help.modelon.com/latest/reference/oct/

[11] https://openmodelica.org/

project.[12] The ModPowerSystems library[13] uses a testing solution developed as part of a bigger utility suite at the RWTH Aachen,[14] also in Python. A shell script solution is used for regression test automation in the PNlib project.[15]

As far as we understood, BuildingsPy, the solution in Modelica-Arduino, and various commercial tools use a sort of maximum deviation metric that checks whether the actual simulation result stays within a "funnel" (BuildingsPy) or a "band" (Modelica-Arduino) around the original result. Mathematically speaking, the "band" amounts to the $\|\cdot\|_{L^\infty}$ norm (in the sense of the space of essentially bounded functions; rather than $\|\cdot\|_{C^0}$ because Modelica simulation result data is usually not time-continuous) applied to the difference of the time-varying functions defined through the reference data and the actual simulation result. Since the timestamps in the reference data and the actual result do generally not coincide, some sort of interpolation technique has to be employed. BuildingsPy uses the pyfunnel[16] module for this; the funnel computation is more than just an $\|\cdot\|_{L^\infty}$-norm, but takes into account also the difference of the timestamps. It regards the reference data as a point cloud $(t_0,y_0),(t_1,y_1),\ldots$ and builds a tolerance area around each (t_i,y_i) datapoint of the reference data. Then, it checks whether the simulated data points fall into these tolerance areas. This requires that timestamps in reference and actual data need to be close, too. As the pyfunnel documentation states, this can make sense to enforce control events to occur at similar times.

Apart from the solutions we found, there are of course other powerful testing solutions available for Modelica library development.

In the open source world, besides the BuildingsPy library, there is the OpenModelicaLibraryTesting.[17] OpenModelica (Fritzson et al. 2020) has extensive library coverage including regression testing for the libraries featured in its package manager.

The tool CSV Result Compare[18] is well known in the Modelica community. Its main purpose is comparing result timeseries files in the `.csv` format. It can also compare `.csv` files recursively by walking through directory trees. It does not allow for test formulation or execution. But it can of course be used to perform result comparison as part of regression testing. Not unlike pyfunnel, it constructs rectangular or ellipsoidal tolerance areas around each datapoint $(t_0,y_0),(t_1,y_1),\ldots$ of the reference result, and then constructs a tube around these tolerance areas defined through an upper and a lower hull curve. csv-compare is implemented in C#, which can cause additional efforts in Linux-based CI toolchains due to its dependencies on .NET or mono.[19]

PySimulator by Pfeiffer et al. (2012) is a simulation and analysis environment with a graphical user interface that can use various different simulators through a plugin infrastructure. Therefore, Asghar et al. (2015) study the use of PySimulator[20] for regression analysis, in particular across different simulation tools, and outline the implementation of a dedicted testing plugin for PySimulator. This plugin uses a simple comparison metric, but features automatic reporting and parallelization of test execution. To our knowledge, PySimulator is no longer actively maintained on GitHub and the last release dates back to 2016. It uses Python 2 which is no longer supported since 2020. Its much broader scope and its plugin dependencies make it difficult to revive for regression testing only.

Commercial tools for regression testing are available from a number of tool vendors. They often come with sophisticated visualization functionality, sometimes directly integrated into Modelica simulation tools, sometimes as standalone products. Generally, except for the case where regression tests are executed by so-called *runners* in a privately managed runtime environment, it is not possible to use commercial, license-bound tools in open source CI toolchains. The term runner refers to an application that executes tests or build tasks for CI applications on resources outside of the CI applications' own infrastructure, and propagates results back to the CI application.

3 Design and functionality

We first sum up the design criteria that guided us in the development of MoPyRegtest. MoPyRegtest shall

(DG1) be a pure testing library,
(DG2) be self-contained,
(DG3) allow for simple formulation of test automation,
(DG4) allow for simple automatic execution with popular CI toolchains,
(DG5) be platform independent,
(DG6) use a popular programming language,
(DG7) allow for user-defined comparison metrics,
(DG8) allow for use with different Modelica tools,
(DG9) integrate with other test automation tools,
(DG10) be usable within open source projects.

3.1 Rationale

Our first design choice was to implement MoPyRegtest entirely in Python. Python is easy to learn, already in use for regression tests in Modelica (BuildingsPy), it is platform independent, easy to automize, and integrates well

[12] https://github.com/modelica-3rdparty/Modelica-Arduino

[13] https://github.com/ModPowerSystems/ModPowerSystems

[14] https://git.rwth-aachen.de/acs/public/simulation/python-for-modelica

[15] https://github.com/AMIT-FHBielefeld/PNlib

[16] https://github.com/lbl-srg/funnel

[17] https://github.com/OpenModelica/OpenModelicaLibraryTesting

[18] https://github.com/modelica-tools/csv-compare

[19] https://github.com/modelica-tools/csv-compare/blob/master/README.md

[20] https://github.com/PySimulator/PySimulator

with popular CI toolchains like GitHub Actions or GitLab CI/CD. Furthermore, the official Python distribution includes the `unittest`[21] framework by default. Hence, it is available to every Python user, it supports test automation, reporting as well as test discovery. For this reason, we use `unittest` as the basis for MoPyRegtest.

Any regression test for Modelica libraries requires a software tool that translates Modelica code into executable simulations that can be run by the testing tool. Since MoPyRegtest shall be usable in open source projects, the natural choice is to use OpenModelica, more precisely the OpenModelica compiler `omc`. However, we want to be able to run MoPyRegtest with other solvers, too. Therefore, we do not call OpenModelica natively from within Python, *e.g.* through OMPython[22] (Ganeson et al. 2012). Instead, we use a file interface and create Modelica script files `.mos` from templates, which we pass to `omc` in order to run simulations. This approach allows for integration with any other simulation tool that supports a file-based scripting interface. Also, it does not introduce any source code dependencies on 3rd party APIs, which could easily break builds or be incompatible for different API versions.

Regression requires the comparison between a reference result and a simulation result produced by a library model. MoPyRegtest does not aim at creating superior or new comparison metrics. Instead of a hard-coded comparison metric, for MoPyRegtest we chose to implement a built-in selection of metrics and give users also the option to define their own metrics. In this fashion, one could even reproduce proven algorithms from other regression or comparison tools like pyfunnel or csv-compare. Calling an external tool for comparison is also possible. For details on the implementation see section 4.

3.2 Architecture and functionality

Conceptually the architecture is very simple and illustrated in Figure 1.

Defining a test in MoPyRegtest is very similar to one in Python's `unittest` module, see Listing 3.1.

In the simplest case, the regression test is a file starting with the prefix `test_<...>.py` to allow for test discovery. It must contain a child class inheriting from `unittest.Testcase`. Inside this class, every single regression test is defined as a method called `test_<...>`, which in its body instantiates a `mopyregtest.RegresstionTest` object like in Listing 3.1. This object is given information on

- where to find the Modelica package to test (`package_folder`),
- which model to test (`model_in_package`),
- where to put the results (`result_folder`),
- [optional] which simulation binary to use from `PATH` (`tool`, default=`"omc"`)
- [optional] which Modelica Standard Library version to use (`modelica_version`, default=`"default"`),
- [optional] a list of Modelica library dependencies loaded before test execution (`dependencies`, default=`None`).

The `mopyregtest.RegresstionTest` object then calls its `compare_result` method with information on

- where to find the reference result as a `.csv` file (`reference_result`),
- [optional] a tolerance threshold which the distance between each individual variable of reference and actual result may not exceed in order to pass (`tol`, default=10^{-7}),
- [optional] a list of variable names for which the comparison metric shall be evaluated (`validated_cols`, default="all variables common in both data sets"),
- [optional] which comparison metric to use (`metric`, default=$\|\cdot\|_\infty$ vector norm on the difference of variable values),
- [optional] which method to use to fill in missing values for timestamps which are not present in either reference or actual result (`fill_in_method`, default=`"ffill"` from `pandas.DataFrame.fillna`).

Remark.

1. In the current implementation, the Modelica STL is treated differently from other dependencies. It is always required by MoPyRegtest. This is just a design choice, because in practice most Modelica models use the STL in one way or the other.

2. The default tolerance has been chosen small for two reasons. First, MoPyRegtest's original scope was to ensure reproduceability during refactorings. Second, choosing a small default value makes it is unlikely that results are judged as being close by accident.

3. The user has to specify the variables to be compared through the `validated_cols` parameter. Theoretically, this could be extended to passing a text file containing the respective variable names, *e.g.* like the `comparisonSignals.txt`[23] proposed for Modelica STL regression testing.

Note that a single file can contain more that one test case like it is common with `unittest`. If a test case definition like in Listing 3.1 is put in a file like `test_mymodel.py`, then all of its test methods are executed by running

```
$ python3 test_mymodel.py
```

[21] https://docs.python.org/3/library/unittest.html

[22] https://github.com/OpenModelica/OMPython

[23] https://github.com/modelica/ModelicaStandardLibrary/files/4270977/SetupForMSLRegressionTesting_2014-01-13.pdf

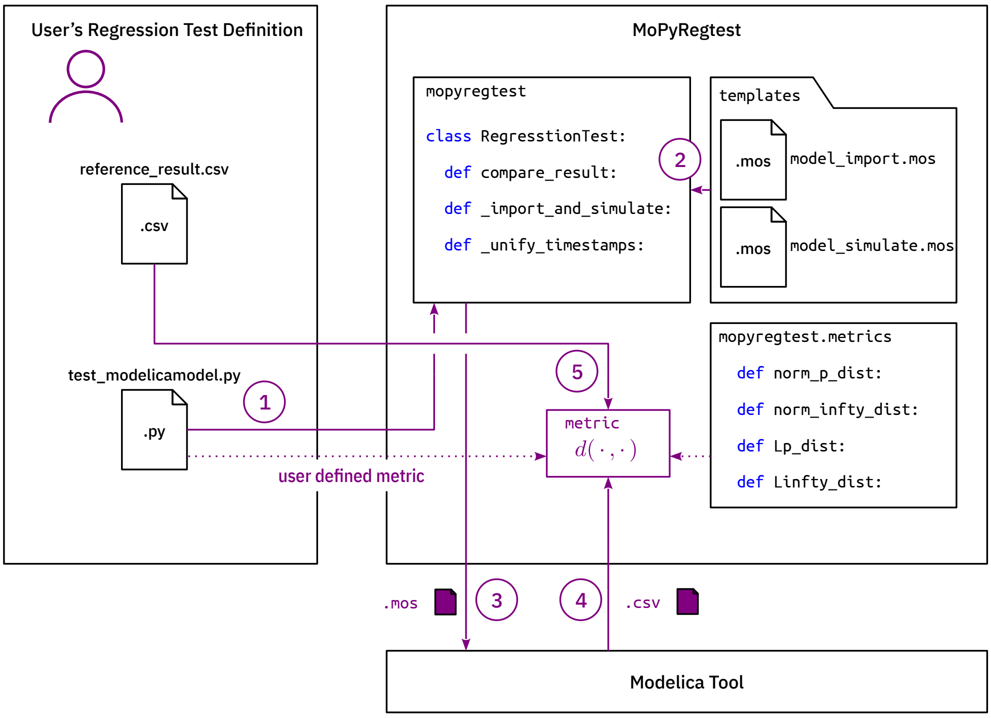


Figure 1. MoPyRegtest architecture with order of test execution steps

Or, if the file is put into a folder structure like required by `unittest` for test discovery,[24] from the structure's root folder one can run

```
$ python3 -m unittest
```

to discover and execute all `unittest` cases. See section 5 for a complete example, which is also contained in the MoPyRegtest implementation.

The call to `tester.cleanup()` in Listing 3.1 is optional. The examples in MoPyRegtest do not call the `cleanup` method, because automatic deletion must always be handled with extreme care.

4 Comparison metrics for regression analysis

Generally speaking, choosing a metric to measure the closeness, or rather the distance of a simulation result from a reference data set, is problem specific. Therefore, we designed MoPyRegtest to give the user full control over the metric which is used in a regression test. The current MoPyRegtest implementation also features a set of predefined metrics.

4.1 Motivation

In section 2 we have outlined strategies that in some way or the other ("funnel" or "band") require the *values* of reference and actual simulation result to be close. This might not always fit. Both data sets are in fact time-discrete representations of functions depending on time, which are known to us only through their values at the timestamps in the respective datasets. In our case, `.csv` files for reference result and the actual simulation run.

Mathematically speaking, there is a wide variety of norms and metrics to measure "closeness" for such time-dependent functions. Two functions may be close in one metric, but not in another one.

A classic example is Gibb's phenomenon in Fourier series approximations for discontinuous functions,[25] most prominently the approximation of a Heaviside step-function $h : [0,2\pi] \mapsto \mathbb{R}$ (step at π) with partial sums of its Fourier series. It will overshoot at the jump discontinuity. However, with the basis functions $\{t \mapsto e^{int} : n \in \mathbb{Z}\}$ forming an orthonormal basis of $L^2([0,2\pi])$ equipped with the canonical norm $\|f\|_{L^2([0,2\pi])} := \left(\int_0^{2\pi} |f(t)|^2 \mathrm{d}t\right)^{\frac{1}{2}}$, the partial sums of the Fourier expansion will converge in that norm (and for a suitable subsequence also pointwise *almost everywhere*). Hence, in this case it is more meaningful to use the $\|\cdot\|_{L^2([0,2\pi])}$ norm as a measure for closeness instead of the common "band" notion.

Another example are events for state-discrete variables, which might occur at slightly different times in reference and actual simulation result. See section 2 and how pyfunnel and csv-compare address this issue.

In order to give users full flexibility in choosing the right metric for the regression test formulation, we allow users to define their own metrics (subsection 4.2) or choosing from a set of predefined metrics (subsection 4.3).

[24] https://docs.python.org/3/library/unittest.html#unittest-test-discovery

[25] https://en.wikipedia.org/wiki/Gibbs_phenomenon

Listing 3.1. Test case definition with MoPyRegtest and pre-defined comparison metric

```
import unittest
import mopyregtest

class TestUserDefinedMetrics(unittest.TestCase):
  def test_modelicamodel(self):
    tester = mopyregtest.RegressionTest(
      package_folder="/path/to/mylibrary",
      model_in_package="MyModel",
      result_folder="/path/to/result/folder/MyModel",
      tool="omc", modelica_version="4.0.0", dependencies=None)

    tester.compare_result(
      reference_result=str("/path/to/reference/result/MyModel_res.csv"),
      metric=mopyregtest.metrics.Lp_dist,
      validated_cols=["myvar1", "myvar2"], tol=1e-8, fill_in_method="interpolate")

    tester.cleanup()

    return


if __name__ == ’__main__’:
  unittest.main()
```

4.2 Implementation

In our situation, both the reference data and the actual simulation data that a user compares in a regression test are given as `.csv` files. They contain columns of data for individual variables at time-discrete timestamps. The timestamps are identical for all variables within one `.csv` file. In general, the timestamps in different `.csv` files do not coincide. A `.csv` file from a simulation run[26] might look like in Table 2.

As it can be seen, timestamps might have multiple occurrences, depending on whether events occurred at that time (in one or more variables).

Metric definition We require a user-defined metric to be a function d that

- takes two `numpy.ndarray`[27] arrays of *identical* shape $(N_{\text{tstamps}}, 2)$, say r_{ref} and r_{act} with
- both r_{ref} and r_{act} having the timestamps as the first column and the values *of one result variable* in the second column, and then
- returns a nonnegative real number $d(r_{\text{ref}}, r_{\text{act}})$.

Then, a user can simply define metrics by passing a function handle, or even *in situ* using lambda functions. In software development, a lambda function refers to an anonymous, *i.e.* an unnamed function that can be defined *ad hoc* and in place. For example,

```
metric=lambda r_ref, r_act: numpy.linalg.
    norm(r_ref[:, 1] - r_act[:, 1], ord=1)
```

which amounts to taking the $\|\cdot\|_1$ vector norm in $\mathbb{R}^{N_{\text{tstamps}}}$ on the difference in values for all timestamps. Then, the distance according to this metric is computed for every variable (defined through the `validated_cols` parameter in `RegressionTest.compare_result`).

Note that in this fashion it is entirely up to the user if he wants to employ absolute or relative error measures in a comparison metric. One could also write

```
metric=lambda r_ref, r_act: mopyregtest.
    metrics.Lp_dist(r_ref, r_act)/
    mopyregtest.metrics.Lp_norm(r_ref)
```

to compute a relative error in the L^2-norm (Lebesgue space norm) weighted by the L^2-norm of the reference result. Here, $p = 2$ is the default value in `Lp_dist` and `Lp_norm`.

Despite being very convenient for the user, we require the data r_{ref} and r_{act} to have identical shape and, in order for computations to make sense, have identical timestamps. Which is generally not the case. For instance, when data sampling rates in reference result and actual result are different.

Timestamp unification One possibility would be to leave it to the user to provide meaningful interpolation for data at missing timestamps. To make it easier for the user, we have implemented the method `RegressionTest._unify_timestamps`. This function is always called in `RegressionTest.compare_result` before the metric is evaluated.

It takes both the reference result r_{ref} and the actual result r_{act} with their timestamps $\mathscr{T}_{\text{ref}} = [t_{\text{ref},0}, \ldots, t_{\text{ref},N_{\text{ref}}}]$ and $\mathscr{T}_{\text{act}} = [t_{\text{act},0}, \ldots, t_{\text{act},N_{\text{act}}}]$ and creates a union $\mathscr{T}_{\text{unified}}$ that

[26] https://github.com/pstelzig/MoPyRegtest/blob/master/examples/test_user_defined_metrics/references/SineNoisy_res.csv

[27] https://numpy.org/

Table 2. Example for a `.csv` result from a Modelica tool run.

time	sine.y	uniformNoise.y	y	uniformNoise.state[1]
0	0	0.289372473723095	2.89372473723095E-05	363258270
0	0	0.289372473723095	2.89372473723095E-05	363258270
0	0	0.289372473723095	2.89372473723095E-05	-2054081690
0.02	0.125333233564304	0.289372473723095	0.125362170811677	-2054081690
0.04	0.248689887164855	0.289372473723095	0.248718824412227	-2054081690
0.05	0.309016994374947	0.289372473723095	0.30904593162232	-2054081690
0.05	0.309016994374947	0.837498257278269	0.309100744200675	14228464
0.06	0.368124552684678	0.837498257278269	0.368208302510406	14228464

- contains every timestamp from the union of both $\mathscr{T}_{\text{ref}}$ and $\mathscr{T}_{\text{act}}$ interpreted as sets (*i.e. no* multiplicities) and
- repeats each timestamp as often as the maximum of its occurrences in $\mathscr{T}_{\text{ref}}$ and $\mathscr{T}_{\text{act}}$.

For both r_{ref} and r_{act}, data rows are repeatedly added for every timestamp from $\mathscr{T}_{\text{unified}}$ until in the such extended r_{ref} and r_{act} each timestamp's multiplicity matches the one in $\mathscr{T}_{\text{unified}}$. The newly added values are initalized with NaN. Then, both r_{ref} and r_{act} are sorted along the timestamp axis, using a stable sorting algorithm that preserves the original order of timestamps.

This leaves the question of interpolating the such added NaN values. To this end, we simply use the strategies offered by `pandas.DataFrame.fillna`[28] and `pandas.DataFrame.interpolate`.[29] The user can choose between these strategies, see the options in subsection 3.2.

Remark. A valid question is whether, instead of unifying the timestamps, it would be easier to define the metric $d(\cdot,\cdot)$ for results r_{ref} and r_{act} of different shapes $(N_{\text{ref}}, 2)$ and $(N_{\text{act}}, 2)$, respectively. And then, if needed, require interpolation as part of the metric implementation. *E.g.* for integral based metrics, timestamp unification is not needed. Only at the numerical integration points both r_{ref} and r_{act} need to be evaluated. Also, there is some risk as to the interpolation error introduced by the timestamp unification. We opted for the timestamp unification for three reasons:

1. It is more convenient for the user and allows for shorter metric definitions.
2. Users can still use any interpolation they want inside the metric implementation.
3. It has the benefit of being able to write out both r_{ref} and r_{act} into a single `.csv` result for visual comparison.

In the future, we might make the now always executed call to `RegressionTest._unify_timestamps` optional, putting such users who want back in control of handling different timestamps in their metric definitions themselves.

4.3 Built-in metrics

MoPyRegtest comes with the following built-in metrics in the module `mopyregtest.metrics`. With some slight abuse of notation:

`mopyregtest.metrics.norm_p_dist`

$$d(r_{\text{ref}}, r_{\text{act}}) := \Big\| r_{\text{ref}}[:,1] - r_{\text{act}}[:,1] \Big\|_p$$

for some $p \in \{1, 2, \ldots\}$ where $\|v\|_p = (\sum_{i=0}^{N-1} |v_i|^p)^{\frac{1}{p}}$ is the canonical p-norm in $\mathbb{R}^N$.

`mopyregtest.metrics.norm_infty_dist`

$$d(r_{\text{ref}}, r_{\text{act}}) := \Big\| r_{\text{ref}}[:,1] - r_{\text{act}}[:,1] \Big\|_\infty$$

where $\|v\|_\infty = \max\{|v_0|, \ldots, |v_{N-1}|\}$ is the canonical maximum norm in $\mathbb{R}^N$.

`mopyregtest.metrics.Lp_dist`

$$\begin{aligned} & d(r_{\text{ref}}, r_{\text{act}}) \\ & := \Bigg(\sum_{i=0}^{N_{\text{tstamps}}-1} (t_{i+1} - t_i) \cdot \Big| r_{\text{ref}}[i,1] - r_{\text{act}}[i,1] \Big|^p \Bigg)^{\frac{1}{p}} \end{aligned}$$

for some $p \in \{1, 2, \ldots\}$ where

$$[t_0, \ldots, t_{N_{\text{tstamps}}}] = r_{\text{ref}}[:,0] = r_{\text{act}}[:,0]$$

are the unified timestamps of r_{ref} and r_{act}. This is the common L^p-norm $\|f\|_{L^p} = \left(\int_{t_0}^{t_{N_{\text{tStamps}}}} |f|^p \mathrm{d}t \right)^{\frac{1}{p}}$ (Lebesgue space norm) when viewing r_{ref} and r_{act} as piecewise continuous functions.

`mopyregtest.metrics.Linfty_dist` When again viewing r_{ref} and r_{act} as piecewise continuous functions, the L^∞-norm (norm of essentially bounded functions) reduces to the canonical $\|\cdot\|_\infty$ maximum norm on the function's values. Hence, it returns the same value as `mopyregtest.metrics.norm_infty_dist`. This metric has been included for notational consistency.

[28] `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html`

[29] `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.interpolate.html`

Remark. The values of p are chosen as integers other than the usual real $p \in [1, \infty)$ because we use `numpy.linalg.norm` which requires the order p to be of type `int` rather than `float`.

5 Showcase

As a showcase we present the example `test_user_defined_metrics` that is included MoPyRegtest's sources.[30]

5.1 Test definition

In this example we want to formulate a regression test that validates during library development, whether a certain Modelica library model stays close to a reference data set. For the showcase, the reference data set is $[0,1] \ni t \mapsto \sin(2\pi t) + 10^{-4} \cdot \text{noise}(t)$. A Modelica model[31] has been created and run with OpenModelica to create the `.csv` reference result.[32] This model uses `Modelica.Blocks.Sources.Sine` and `Modelica.Blocks.Noise.UniformNoise`. The Modelica library model to be tested is the original `Modelica.Blocks.Sources.Sine` itself (without the noise).

The test has the folder structure

```
examples
└─ test_user_defined_metrics
   ├─ __init__.py
   ├─ test_user_defined_metrics.py
   └─ references
      └─ SineNoisy_res.csv
```

The `__init__.py` turns the folder `test_user_defined_metrics` into a Python package for test discovery. That is, if `python3 -m unittest` would be called from the parent directory `examples`, all `unittest` test definitions in `test_user_defined_metrics` would be executed. The entire test definition in shown in Listing 5.1.

5.2 Test automation

The test is automated "for free" because `unittest` features automated test execution and test discovery. The output is shown in Listing 5.2.

5.3 Automated test execution

Modern continuous integration toolchains like GitHub Actions or GitLab CI/CD allow the automated execution of tests triggered by certain events. With GitHub Actions for instance, one can automate test execution of Python code on `push` events to a GitHub repository. In that case, the respective user-defined job, say `python-test.yml`, in the repository's `.github/workflows/` folder is executed. The GitHub Actions documentation[33] explains how.

In our case, the execution of a regression test definition using MoPyRegtest, *e.g.* the one above, requires a suitable Modelica simulation tool. There is the option to install the OpenModelica compiler `omc` and the Modelica Standard Library as a step in the job definition, as well as the other dependencies of MoPyRegtest. This however would require significant computational resources and consume valuable usage time for GitHub Actions. The same goes for other continuous integration pipelines.

Another option is to execute the tests based on a Docker[34] image. Here, one could use the OpenModelica docker image tagged `v1.21.0-minimal` from dockerhub[35] (or respective newer versions) with the preinstalled `omc`. For instance, a GitHub Action that executes the same test definition as in Listing 5.1, but runs it in a docker container based on the OpenModelica `v1.21.0-minimal` image is shown in Listing 5.3.

In this fashion, one can easily implement test automation in an open source Modelica library development on GitHub. All that is needed are test definitions in MoPyRegtest like in Listing 5.1 and a GitHub Action like in Listing 5.3 that executes the test definitions. Either automatically, *e.g.* following push events, or manually triggered. Both developers and users can then review the test results in the repository's Actions tab.

6 Conclusions

We have outlined why regression testing is important in Modelica library development. Then we gave a rough overview over how test and test automation is being used with open source Modelica library development and identified some potential trends. We have also identified concrete tools used for regression testing in the open source Modelica library community. We then formulated the rationale why a continuous integration-friendly testing solution like MoPyRegtest could be of value for the community and described its design and functionality. We highlighted in detail how we implemented the possibility for users to define their own comparison metrics for regression tests. Then we presented a showcase that is included in MoPyRegtest.

MoPyRegtest is work in progress and still under development. It has not been investigated yet how it could integrate with Modelica simulation software other than Open-

[30] https://github.com/pstelzig/MoPyRegtest/tree/master/examples/test_user_defined_metrics

[31] https://github.com/pstelzig/MoPyRegtest/blob/master/examples/test_user_defined_metrics/SineNoisy.mo

[32] https://github.com/pstelzig/MoPyRegtest/blob/master/examples/test_user_defined_metrics/references/SineNoisy_res.csv

[33] https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

[34] https://www.docker.com/

[35] https://hub.docker.com/r/openmodelica/openmodelica/tags

Listing 5.1. Showcase `test_user_defined_metrics.py`

```
# Preparing the dependencies ###################################################
import unittest
import pathlib
import mopyregtest
import functools

# Define the test ##############################################################
# Example here for a Ubuntu environment with OpenModelica

class TestUserDefinedMetrics(unittest.TestCase):

    # Testing user defined metrics on a Modelica simulation result against a noisy
        reference result
    def test_Sine(self):
        tester = mopyregtest.RegressionTest(
            package_folder=pathlib.Path.home() / \
                               ".openmodelica/libraries/Modelica 4.0.0+maint.om/",
            model_in_package="Modelica.Blocks.Sources.Sine",
            result_folder=pathlib.Path(__file__).absolute().parent / \
                              "Modelica.Blocks.Sources.Sine",
            modelica_version="4.0.0",
            dependencies=None)

        # Comparing results
        tester.compare_result(
            reference_result=str(pathlib.Path(__file__).absolute().parent / \
                "references/SineNoisy_res.csv"),
            metric=functools.partial(mopyregtest.metrics.Lp_dist, p=2),
            validated_cols=["y"], tol=2e-3, fill_in_method="interpolate")

        return


if __name__ == '__main__':
    unittest.main()
```

Listing 5.2. Output of `test_user_defined_metrics.py`

```
$ python3 -m unittest

Testing model Modelica.Blocks.Sources.Sine
Simulating model Modelica.Blocks.Sources.Sine using the simulation tools: omc
Using simulation tool omc
Comparing simulation result /home/user/mopyregtest/examples/test_user_defined_metrics/
    Modelica.Blocks.Sources.Sine/Modelica.Blocks.Sources.Sine_res.csv and reference /home/
    user/mopyregtest/examples/test_user_defined_metrics/references/SineNoisy_res.csv
Comparing column "y"
.
----------------------------------------------------------------------
Ran 1 test in 2.679s

OK
```

Listing 5.3. GitHub Action to execute `test_user_defined_metrics.py` as part of MoPyRegtest's GitHub repo

```
name : Example job for Modelica library regression testing
on: [workflow_dispatch]
jobs:
  examples-test:
    runs-on: ubuntu-latest
    container: openmodelica/openmodelica:v1.21.0-minimal
    steps:
      - name: Install dependencies
        run: |
          apt-get -qq update
          apt-get -qq --no-install-recommends install python3 python3-pip git
          pip install numpy pandas
      - name: Install Modelica STL 4.0.0
        run: |
          echo "installPackage(Modelica, \"4.0.0+maint.om\", exactMatch=true);" >
              installModelicaStl.mos && omc installModelicaStl.mos
      - name: Install MoPyRegtest with tag v0.2.1
        run: |
          git clone https://github.com/pstelzig/MoPyRegtest.git mopyregtest
          cd mopyregtest
          git checkout v0.2.1
          pip3 install --user .
      - name: Run examples
        run: |
          cd mopyregtest/examples/test_user_defined_metrics
          python3 test_user_defined_metrics.py
```

Modelica. Also, it has no inherent reporting functionality except what is provided by Python's `unittest`. Furthermore, reference results need to be given as `.csv` files, whereas in the Modelica community result files are usually `.mat`, making reference result files bigger than they need to be. The timestamp unification has proven reliable in our use so far, but other interpolation techniques, as outlined in the respective remark in section 4, could perform better in certain scenarios. Multiple tests within a single test class are executed sequentially at the moment, despite being independent. Execution time could be saved by running tests in parallel. Finally, we have shown the feasibility of integrating MoPyRegtest with popular continuous integration toolchains like GitHub Actions.

Acknowledgements

MoPyRegtest is developed by Philipp Emanuel Stelzig as a private project. At simercator it is used for automating regression tests of inhouse Modelica libraries. The author would like to thank the reviewers for their helpful comments regarding the state of the art overview.

References

Asghar, Adeel et al. (2015). "Automatic regression testing of simulation models and concept for simulation of connected FMUs in PySimulator". In: *11th International Modelica Conference*. 118. Linköping University Electronic Press, pp. 671–679.

Beck, Kent (2003). *Test-driven development: by example*. Addison-Wesley Professional.

Fritzson, Peter et al. (2020). "The OpenModelica integrated environment for modeling, simulation, and model-based development". In: *Modeling, Identification and Control* 41.4, pp. 241–295.

Ganeson, Anand Kalaiarasi et al. (2012). "An OpenModelica Python Interface and its use in PySimulator". In: *9th International Modelica Conference*. Linköping University Electronic Press, pp. 537–548.

Onggo, Bhakti Stephan and Mumtaz Karatas (2016). "Test-driven simulation modelling: A case study using agent-based maritime search-operation simulation". In: *European Journal of Operational Research* 254.2, pp. 517–531.

Pfeiffer, Andreas et al. (2012). "PySimulator – A simulation and analysis environment in Python with plugin infrastructure". In: *9th International Modelica Conference*. Linköping University Electronic Press, pp. 523–536.

Wetter, Michael et al. (2014). "Modelica Buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Wong, W Eric et al. (1997). "A study of effective regression testing in practice". In: *PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering*. IEEE, pp. 264–274.

Object-Oriented Modelling of Flexible Cables based on Absolute Nodal Coordinate Formulation

Jianchen Wu[1] Baokun Zhang[1] Hualong Zhao[1] Zhihui Liu[1] Yujie Guo[2] Ji Ding[1] Fanli Zhou[1]

[1]Suzhou Tongyuan Software & Control Tech. Co. Ltd, Suzhou, China,
{wujianchen, zhangbk, zhaohl, liuzh, dingj, zhoufl}@tongyuan.cc
[2]College of Aerospace Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, China,
yujieguo@nuaa.edu.cn

Abstract

Cable-pulley systems consist of several segments of cables, winches, and pulleys, which are used in a wide range of engineering applications such as lifting equipment and pulley systems. However, its dynamics simulation has been a tough issue in the Modelica community. The absolute nodal coordinate formulation (ANCF) uses global displacements and slopes at nodes to describe the geometry of the deformed body, which allows the derivation of constant mass matrices and zero-valued quadratic velocity dependent centrifugal and Coriolis forces. In the last two decades this method shown its powerful capacity to model flexible multi-body systems. This paper presents an object-oriented approach to model cable-pulley system, where flexible cables are discretized using ANCF cable elements. It is compatible with the Modelica Multibody Library by using a unified frame interface and enables coupled analysis of cables and rigid bodies. The paper provides a rich set of application examples showing the ease and efficiency of the Modelica-based component drag-and-drop modelling way for modelling cable-pulley systems.

Keywords: cable, pulley, absolute nodal coordinate formulation, Modelica, MWORKS

1 Introduction

Flexible multi-body systems are defined as complex dynamic systems consisting of rigid and flexible bodies connected in different ways (Shabana 1997). It focuses on the coupling between the body's deformation and its large-scale spatial motion. Many approaches have been proposed based on different engineering background, such as the floating frame of reference approach (Likins 1967), the incremental finite element method (Shabana 1996) and the absolute nodal coordinate formulation (Shabana 1996) which will be used in this paper. Small deformations superimposed on large rigid body displacements have led to the well-known floating frame of reference formulation. It is used by various commercial dynamics analysis software but is usually considered unsuitable when the object undergoes large deformations and rotational motions. The absolute nodal coordinates method can solve these challenging problems, especially when the object under study is a flexible object floating in space, such as a thin film of a solar sail or a tethered net used to capture space debris (Liu 2013; Shan 2020).

Drive systems consisting of cables and pulleys are widely used in lifting equipment. The dynamics modelling of cables has been extensively investigated in recent years. The simplest way is reduced to linear springs with length-dependent stiffnesses, neglecting the cable weight and inertia forces (Rouvinen 2005). Such massless cables are certainly excellent in simulation efficiency, but when lateral vibrations and bending deformations become critical, the method becomes inadequate. Accurate modelling of the cable dynamic requires the use of non-linear finite element methods. Absolute nodal coordinate formulation has been used to great effect in modelling the dynamics of cables (Berzeri 2000).

Another area of interest is the simulation of the contact behavior of cables and pulleys. Modelling of contact forces is not always necessary, for example in (Aufaure 1993) Aufaure proposes a cable pulley element based on the assumption of complete elasticity, where the supporting pulley can slide frictionlessly along the cable. It brings the benefit of computational efficiency but does not reflect the actual situation as well. The most common way of establishing the contact force between the cable and the pulley is to use the penalty method. It derives from the simplest phenomenon that no penetration occurs between objects in contact (Lugrís 2011). In the penalty method the contact force is related to the penetration depth. Different contact models are proposed, for example normal contact forces can be modeled as the spring damping model, the Hertz's model and the non-linear damping model, and tangential friction forces can be modelled as the Coulomb friction model, the Hollars model (Botta 2017) and the bristle contact model. An alternative approach to modelling the cable pulley contact is to use the unilateral constraints and linear complementarity problem approach for numerical treatment (Pfeiffer 1996). Using this method, the normal and tangential contact forces are related to the unilateral constraint as Lagrange multipliers.

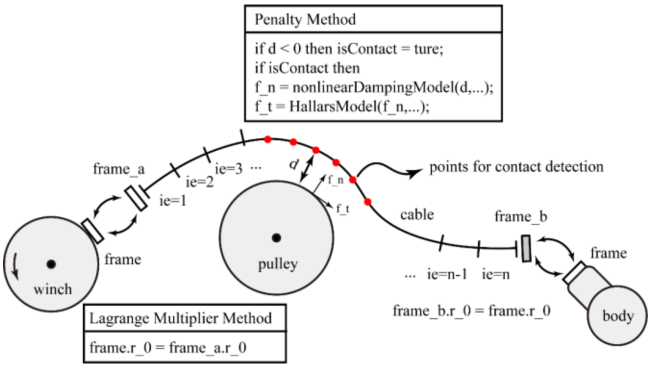
Modelling flexible bodies with the Modelica language is not novel and related work can be found in (Ferretti 2005; Heckmann 2006). In previous work the flexible bodies modelling has generally fallen into two categories. The

DOI
10.3384/ecp20453

first is based on the finite element method (FEM) using the native Modelica language to discretize the flexible body. Considering the complexity of meshing, this approach can only deal with simple geometries such as beams and plates. The second method is based on the floating coordinate method, which models the flexible body as a superposition of several eigenmodes. It relies on the modal files calculated in advance by structural analysis software and can model flexible bodies of arbitrary geometry. Therefore, it has a relatively wide range of applications. It's worth noting that we can only find two commercial cable pulley libraries online. One is released by DLR, and another is created by MapleSim. The theory behind them seems inaccessible since no publications for these libraries can be found. The innovation of this paper is to propose an object-oriented approach to model flexible cables, where the cable can undergo large deformations and rotational motions. The compatibility of flexible cables with the Modelica Multibody Library is achieved with a unified frame interface.

The paper is organized as follows. In section 2 supporting theories is presented. Firstly, the dynamic equations of ANCF cable elements are presented in subsection 2.1, followed by an explanation in subsection 2.2 of how the cable component is compatible with the Modelica Multibody Library by a unified frame interface. In subsection 2.3 the penalty method is given for modelling cable-pulley contact. In subsection 2.4 the constraint equations for sliding joints are given. Implementation details with the Modelica language are given in section 3. Extensive cases are presented in section 4. Finally in section 5 the main research results are summarized and future related works are looked at.

2 Fundamentals



Penalty Method

```
if d < 0 then isContact = ture;
if isContact then
f_n = nonlinearDampingModel(d,...);
f_t = HallarsModel(f_n,...);
```

Lagrange Multiplier Method

frame.r_0 = frame_a.r_0

frame_b.r_0 = frame.r_0

Figure 1. Diagram of cable-pulley composition.

The pulley-cable model is constructed in three main steps, as shown in Figure 1:

1. Division and assembly of cable elements.
2. Connection of flexible cables to rigid bodies by applying positional constraints through the Lagrange multiplier method.
3. Construction contact between cables and pulleys by means of the penalty method.

2.1 ANCF Cable Element

In the absolute nodal coordinate formulation, the cable element is defined in the inertial coordinate system and the nodal coordinates are described using the global displacements and slopes, which can be expressed as

$$\boldsymbol{e} = \left[e_1 \; e_2 \; e_3 \; e_4 \; e_5 \; e_6 \; e_7 \; e_8 \; e_9 \; e_{10} \; e_{11} \; e_{12} \right]^T \tag{1}$$

where $e_1 = r_1\big|_{x=0}$, $e_2 = r_2\big|_{x=0}$, $e_3 = r_3\big|_{x=0}$, $e_7 = r_1\big|_{x=l}$, $e_8 = r_2\big|_{x=l}$, $e_9 = r_3\big|_{x=l}$ are the global displacements and $e_4 = \frac{\partial r_1}{\partial x}\Big|_{x=0}$, $e_5 = \frac{\partial r_2}{\partial x}\Big|_{x=0}$, $e_6 = \frac{\partial r_3}{\partial x}\Big|_{x=0}$, $e_{10} = \frac{\partial r_1}{\partial x}\Big|_{x=l}$, $e_{11} = \frac{\partial r_2}{\partial x}\Big|_{x=l}$, $e_{12} = \frac{\partial r_3}{\partial x}\Big|_{x=l}$ are the global slopes at the element nodes.

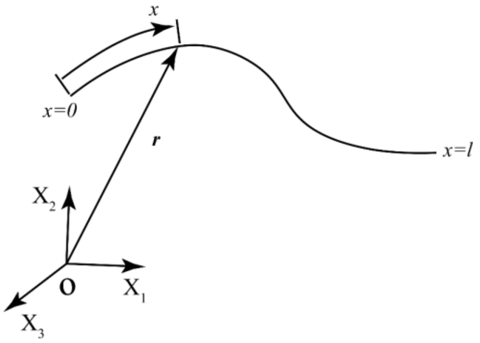


Figure 2. ANCF cable element.

The global position vector $\boldsymbol{r}$ of an arbitrary point on the neutral axis of a cable element, as shown in Figure 2, can be obtained by interpolating the nodal coordinate vector with the following expression:

$$\boldsymbol{r} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} = \boldsymbol{Se} \tag{2}$$

where $\boldsymbol{S}$ is the shape function matrix, which can be written as

$$\boldsymbol{S} = \begin{bmatrix} s_1 & 0 & 0 & s_2 & 0 & 0 & s_3 & 0 & 0 & s_4 & 0 & 0 \\ 0 & s_1 & 0 & 0 & s_2 & 0 & 0 & s_3 & 0 & 0 & s_4 & 0 \\ 0 & 0 & s_1 & 0 & 0 & s_2 & 0 & 0 & s_3 & 0 & 0 & s_4 \end{bmatrix} \tag{3}$$

where the functions s_i are defined as

$$s_1 = 1 - 3\xi^2 + 2\xi^3,\ s_2 = \xi - 2\xi^2 + \xi^3,\ s_3 = 3\xi^2 - 2\xi^3,$$
$$s_4 = \xi^3 - \xi^2$$

and $\xi = x/l$. Applying the principle of virtual work gives the dynamic equations of the cable element in matrix form as

$$\boldsymbol{M}\ddot{e} + \boldsymbol{Q}_k = \boldsymbol{Q}_e \tag{4}$$

where $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{Q}_k$ is the elastic force vector, and $\boldsymbol{Q}_e$ is the external force vector, respectively, with the following expressions:

$$\boldsymbol{M} = \rho A \int_0^l \boldsymbol{S}^T \boldsymbol{S} dx \tag{5}$$

$$\begin{aligned}\boldsymbol{Q}_k = & \int_0^l EI \frac{1}{g^2}\left(g\frac{\partial f}{\partial \boldsymbol{e}} - f\frac{\partial g}{\partial \boldsymbol{e}} \right) dx + \\ & \int_0^l EA3\left(\boldsymbol{r}_x^T \boldsymbol{r}_x\right)^{1/2} \left(\boldsymbol{r}_x^T \frac{\partial \boldsymbol{r}_x}{\partial \boldsymbol{e}} \right) dx\end{aligned} \tag{6}$$

where $g = |\boldsymbol{r}_x|^3$, $f = |\boldsymbol{r}_x \times \boldsymbol{r}_{xx}|$ .Where $\boldsymbol{Q}_e$ can be divided into point and distributed forces, with the following expressions:

$$\begin{aligned}\boldsymbol{Q}_e &= \boldsymbol{Q}_{cl} + \boldsymbol{Q}_{dl} \\ &= \boldsymbol{S}^T\left(x_p\right)\boldsymbol{F}^p + \int_0^l \boldsymbol{S}^T \boldsymbol{F}^d(x) dx\end{aligned} \tag{7}$$

where x_p is the point of action of the concentrated force, $\boldsymbol{F}^p$ is the concentrated force vector and $\boldsymbol{F}^d(x)$ is the position-dependent distributed force vector.

2.2 Rigid-Flexible Modeling

Using a single cable component does not really solve the problems encountered in engineering, as most driven systems consist of rigid and flexible bodies. ANCF elements can be combined with rigid or flexible bodies modelled in natural coordinates to couple the remaining components of the system. Here we assume that the cable is sufficiently flexible that it cannot transmit moments at the point where the cable is connected to the rigid body. This assumption is reasonable, especially when considering that the bending stiffness of the cable is relatively small. We use the Lagrange multiplier method to impose only position constraints at the connection of the cable to other bodies, modifying Equation (4) as follows.

$$\mathbf{M}\ddot{\mathbf{q}} + \boldsymbol{\Phi}_{\boldsymbol{q}}^T \boldsymbol{\lambda} = \boldsymbol{Q}_e - \boldsymbol{Q}_k - \boldsymbol{Q}_d \tag{8}$$

$$\boldsymbol{\Phi}(\boldsymbol{q}) = \boldsymbol{0} \tag{9}$$

where $\mathbf{q}$ is a generalized coordinate vector containing the nodal coordinates of the flexible body and the natural coordinates of the remaining bodies, $\boldsymbol{\Phi}$ is an algebraic constraint vector, $\boldsymbol{\Phi}_{\boldsymbol{q}}$ its Jacobian matrix and $\boldsymbol{\lambda}$ is the Lagrange multipliers vector. Considering the real-world energy dissipation, a damping term $\boldsymbol{Q}_d$ is added to the right-hand side of the equation, which can be described by the Rayleigh damping model as follows:

$$\boldsymbol{Q}_d = \left(\alpha \boldsymbol{M} + \beta \boldsymbol{K}\right)\dot{\boldsymbol{q}} \tag{10}$$

where α and β are two factors.

2.3 Pulley and Winch Modeling

Pulleys and winches are essential components of cable drive systems. Simulating the interaction behavior between the cable and the pulley or winch can be easily achieved using the penalty method in contact dynamics. The normal contact force between the pulley and the cable is modelled as a non-linear damping model and is described by the following equation:

$$f_n = k\delta^n + d\dot{\delta}^n \delta \tag{11}$$

where k is the contact stiffness, d is the contact damping, δ is the penetration depth between two objects in contact, and n is a factor related to the shape and the material of the objects in contact.

The tangential friction between the pulley and the cable is described using the Hollars model with the following equation:

$$f_t = \left[\min\left\{ \frac{v_t}{v_{t_0}}, 1 \right\} \left(\mu_k + \frac{2\left(\mu_s - \mu_k\right)}{1 + \left(\frac{v_t}{v_{t0}}\right)^2} \right) \right] f_n \tag{12}$$

where v_t is the relative velocity of the two colliding objects, v_{t_0} is the velocity threshold for the change from static to kinetic friction, f_n is the normal contact force, and μ_s and μ_k is the coefficient of static and kinetic friction, respectively.

There is no special treatment of contact detection in this paper. An exhaustive method is used where points on the cable are simply selected at equal intervals. The number of contact detection points should be chosen according to the practical situation. The material points on the cable are described in the inertial coordinate system, so they need to be converted to the body coordinate system of the pulley first, and then determine whether contact occurs according to the distance from the detection point to the pulley's axis of rotation.

2.4 Sliding Joint Modeling

The sliding joint used in the absolute nodal coordinate formulation was proposed in (Sugiyama 2003) by Sugiyama et al. It solves the problem of contact detection which cannot be avoided when using the penalty method. In engineering, sliding cables for river crossings and tail hooks on naval aircraft, for example, can be simplified as sliding joints. A frictionless sliding joint can be described by following constraint equations:

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{r}^i\left(\boldsymbol{x}^i\right) - \boldsymbol{r}^j\left(\boldsymbol{x}^j\right) \\ \dfrac{\partial \boldsymbol{r}^j\left(\boldsymbol{x}^j\right)}{\partial x_1^j} \cdot \boldsymbol{\lambda} \end{bmatrix} = \boldsymbol{0} \tag{13}$$

where, $\boldsymbol{x}^j$ denotes a point on the cable, $\boldsymbol{x}^i$ denotes a point on a rigid or flexible body connected to the cable and $\boldsymbol{\lambda}$ is the Lagrange multipliers of dimension 3. Derivation of the first three constraint equations with respect to time yield constraint equations of index 2, which can be written as

$$\boldsymbol{C} = \begin{bmatrix} \dfrac{\partial \boldsymbol{r}^i\left(\boldsymbol{x}^i\right)}{\partial t} - \dfrac{\partial \boldsymbol{r}^j\left(\boldsymbol{x}^j\right)}{\partial t} - \dfrac{\partial \boldsymbol{r}^j\left(\boldsymbol{x}^j\right)}{\partial x_1^j}\dot{s} \\ \dfrac{\partial \boldsymbol{r}^j\left(\boldsymbol{x}^j\right)}{\partial x_1^j} \cdot \boldsymbol{\lambda} \end{bmatrix} = \boldsymbol{0} \tag{14}$$

where s is the arc length coordinate of the sliding point on the cable, which is time varying.

3 Modelica Implementation

In this section the element division and assembly using the Modelica language will be introduced. Some of the programming details considered important are explained below. Firstly, since the ANCF element mass matrix is a constant matrix, it only needs to be calculated once at program runtime. In this paper it is achieved by adding `annotation (Evaluate = true)` to the definition of the mass matrix. Secondly, a Gauss–Legendre quadrature is used to obtain both matrixes and force vectors. It's a standard technique in the finite element method. Five and three Gaussian integration points per element are used for axial force and bending force calculations, respectively. Benefiting from the excellent symbolic processing capabilities of the Modelica language, function is used to calculate the elastic force vector, using nodal displacements as input variables.

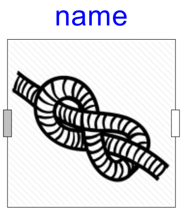


Figure 3. Cable component icon.

The icon for a cable component is shown in Figure 3, where the ends of cable are represented using the frame interfaces in the Modelica Multibody Library. The ends' positions should equal to the frames' positions. It can be achieved by imposing position constraints using the Lagrange multiplier method. The codes are as follows:

Listing 1. Code example using Lagrange multipliers

```
v = der(e);
a = der(v);
M * a + transpose(Phi_q) * lamda = Qg - Qe
- Qd;
e[1] = frame_a.r_0[1];
e[2] = frame_a.r_0[2];
e[3] = frame_a.r_0[3];
e[n - 5] = frame_b.r_0[1];
e[n - 4] = frame_b.r_0[2];
e[n - 3] = frame_b.r_0[3];
frame_a.t = {0, 0, 0};
frame_b.t = {0, 0, 0};
frame_a.f = Modelica.Mechanics.
MultiBody.Frames.resolve2(frame_a.R,
{-lamda[1], -lamda[2], -lamda[3]});
frame_b.f = Modelica.Mechanics.
MultiBody.Frames.resolve2(frame_b.R,
{-lamda[4], -lamda[5], -lamda[6]});
```

where `e` is the generalized nodal coordinates vector, and `n` is the number of degrees of freedom. Where `frame_a` and `frame_b` are the frame interfaces in Figure 3. In order to balance the number of equations, the multipliers vector `lamda` is assigned to the flow variable force `f` in `frame_a` and `frame_b`. The mechanism behind this is determined by the physical meaning of the Lagrange multipliers.

The following two examples are used to verify the correctness of the cable component. Example 1 is set up as a flexible cable in a gravity-free environment with a fixed left end and a concentrated moment applied at the right end. The magnitude of the moment is set to $\lambda\pi EI/L$, where $\lambda = 1$, E is the modulus of elasticity, I is the moment of inertia of the cross section and L is the length of the cable. According to (Gerstmayr 2008), the cable is finally stabilized into a semicircle, as shown in Figure 4, under the action of this moment.

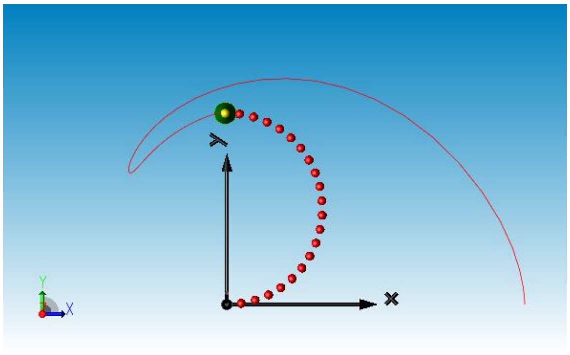


Figure 4. Example 1: Flexible cable bent into a semicircle.

Example 2 is set up as a cable with one end hinged and one end free, the initial configuration is along the positive X-axis, the initial angular velocity is along the positive Y-axis, the magnitude is 4 rad/s, and the direction of gravity is set to -Y. The geometric and material properties of the cable are as follows: The length of the cable is 1 m, the cross-sectional area is $10^{-6} f^2$ m^2, the modulus of elasticity is $10^9 / f^4$ Pa and the density is $8000 / f^2$ kg/m^3, where $f = 5$. The calculated results are compared with (Gerstmayr 2006) as shown in Figure 6. It can be found that the cable component built on MWORKS agrees very well with the results of reference (Gerstmayr 2006).

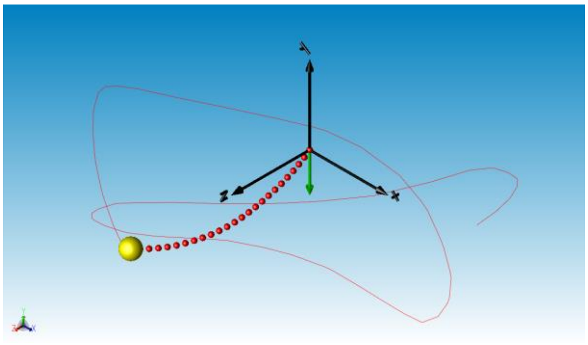


Figure 5. Example 2: 3D flexible cable pendulum.

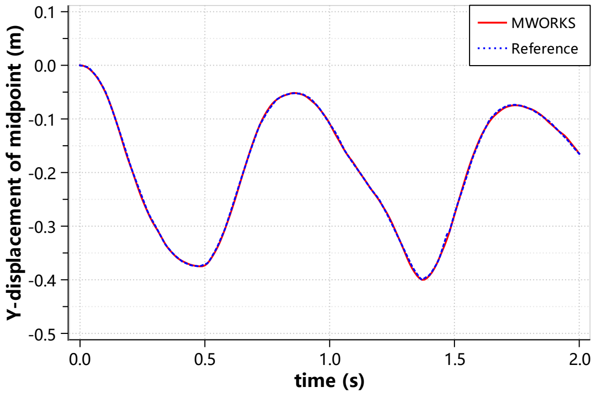


Figure 6. Example 2: Y-displacement of the mid-point of the three-dimensional pendulum as function of time.

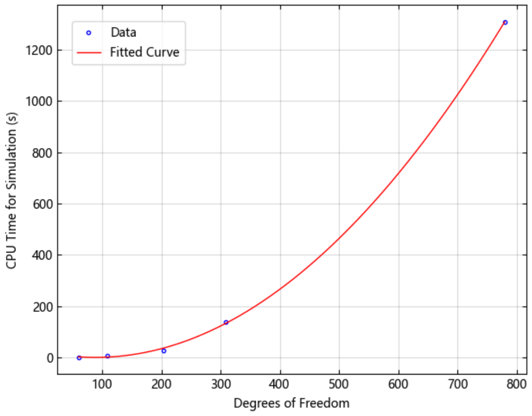


Figure 7. CPU time for simulation as function of degrees of freedom.

To study the simulation performance for the presented model, we recorded the CPU time for simulation for Example 2 with increasing degrees of freedom. The time integration algorithm set to Dassl and the simulation stop time set to 2 seconds. As it can be found in Figure 7 that the computational cost shows a quadratic polynomial relationship with the number of degrees of freedom. For this case four cable elements are enough to get a converged result.

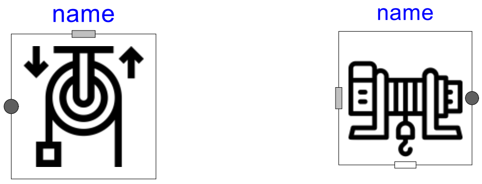


Figure 8. Pulley and winch component icons.

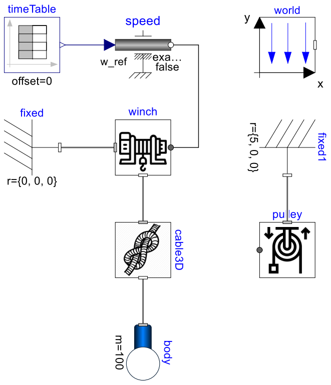


Figure 9. Diagram view of a simple lifting device.

The icons of pulley and winch component are shown in Figure 8, where the frame interface of pulley component and the left frame of winch component is used to connect the mounting position. The lower frame of winch component is used to connect the cable, as shown in Figure 9, where a simple lifting device is constructed. The initial configuration of cable is horizontal, as shown in Figure 10. Figure 12 gives the ball's position and forces exerted on it. After a few swings from side to side, the ball ends up in a straight up and down position.

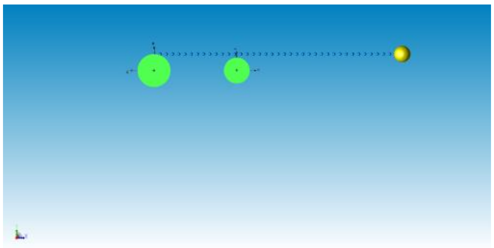


Figure 10. Initial configuration.

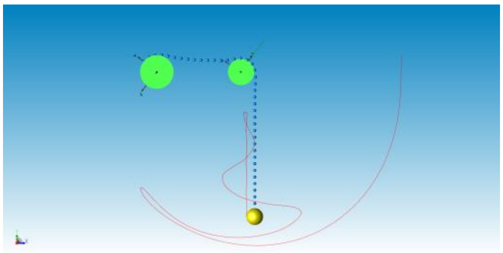


Figure 11. Final configuration at 30 second.

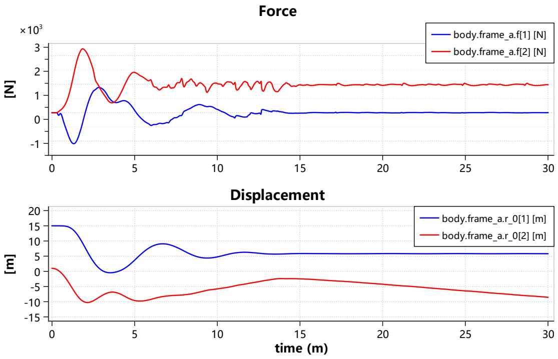


Figure 12. Force and displacement of body as function of time.

The icon of the sliding joint is shown in Figure 13, where the left and right frame represent two ends of the cable, and the lower frame is used to connect the object suspended. An example of zip line is constructed as shown in Figure 14, simulating a rigid body with a horizontal initial attitude sliding on a cable under the effect of gravity. The initial and final configuration of objects are given in Figure 15 and Figure 16 respectively. It can be found in Figure 17 that the suspension is finally stabilized near the midpoint of the cable after several swings.

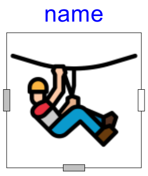


Figure 13. Sliding joint component icon.

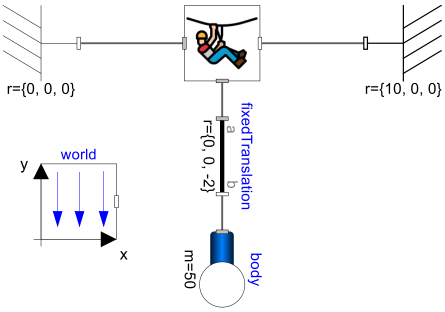


Figure 14. Diagram view of a zip line model.

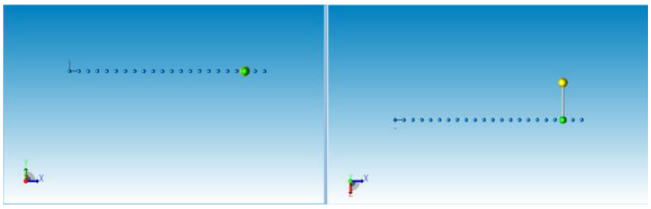


Figure 15. Initial configuration (left: front view, right: top view).

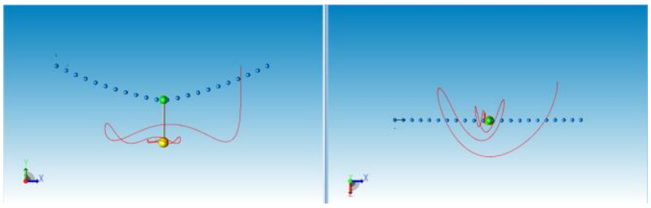


Figure 16. Final configuration at 20 second (left: front view, right: top view).

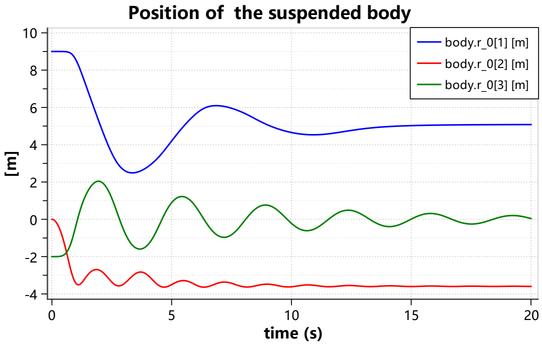


Figure 17. Body's position as function of time.

4 Examples for Application

In this section, several engineering applications have been built in the MWORKS.Sysplorer simulation environment. The three most representative cases have been selected for presentation.

4.1 Case 1: Belt Drives

The belt drive model, as shown in Figure 18, consists of an active pulley, a passive pulley, and a conveyor belt.

The active pulley is driven by a rotational speed signal and the passive pulley is connected to a prismatic joint, which is driven by a position signal, simulating the tensioning process of the conveyor belt. The material and geometrical properties of the belt are as follows: modulus of elasticity 5.8e6 Pa, density 3500 kg/m^3, belt thickness 0.006 m, width 0.007 m, initial length 0.724 m. In this case the belt is discretized into 15 cable elements and the contact detection points are selected at 90 points at equal intervals. Figure 19 shows the axial stresses in the conveyor belt during tensioning and operation. Figure 20 gives the reaction forces applied to the active and passive pulleys, which are symmetrical.

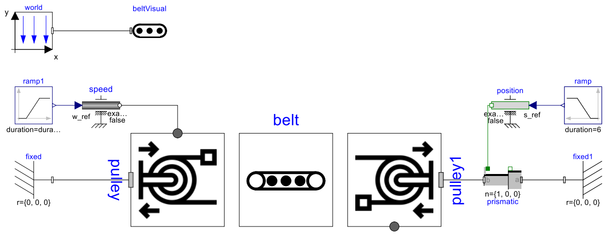


Figure 18. Diagram view of belt drive model.

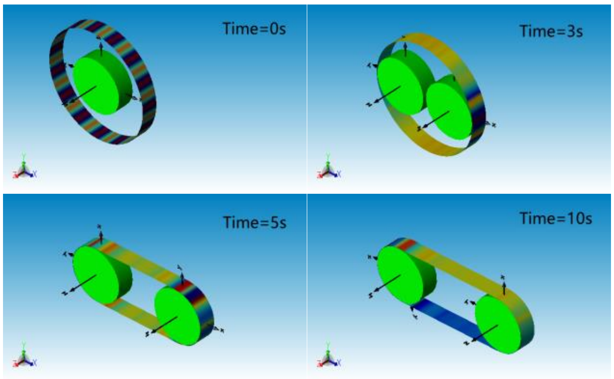


Figure 19. Stress clouds at different seconds.

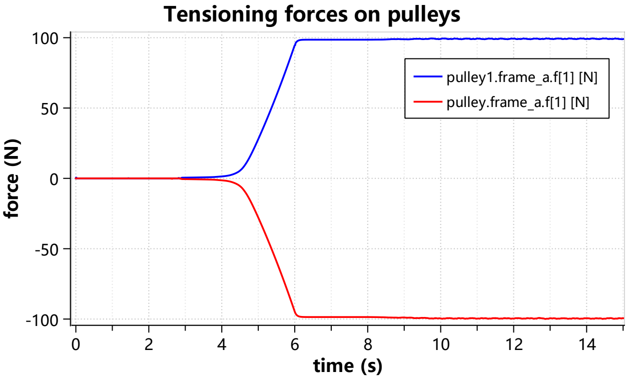
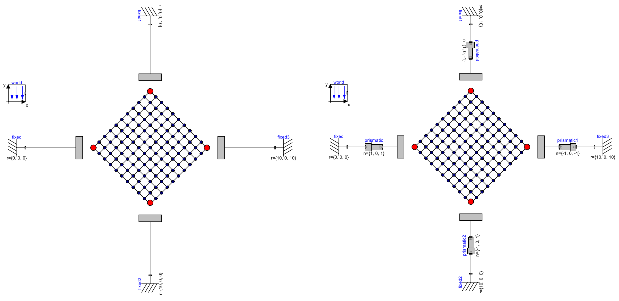


Figure 20. Reaction forces on pulleys.

4.2 Case 2: Cable Nets

The cable net model is assembled from several single cables, with position constraints imposed by Lagrange multipliers at the intersection of the cables. Four corner points of the net are exposed through the frame interface, which can be connected to parts, mechanical joints, and force elements. The dynamic behavior of nets in different scenarios can be easily simulated by changing the boundary conditions at the four corner points, as shown in Figure 21. The net configurations at different times are shown in Figure 22 and Figure 23.



(a) (b)

Figure 21. Diagram view of net models in different scenarios: (a) four corners fixed, (b) four corners moving along the diagonal.

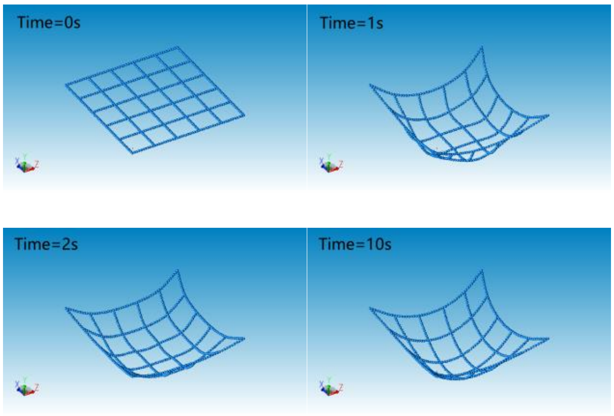


Figure 22. Net with four corners fixed.

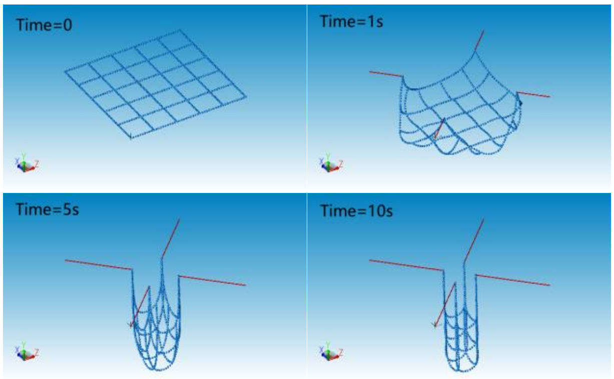


Figure 23. Net with four corners moving along the diagonal.

4.3 Case 3: Gantry Crane

Gantry crane is a kind of port lifting equipment, mainly used for outdoor loading and unloading operations of cargo yards. The simple gantry crane model built in this paper, as shown in Figure 24, mainly consists of two cables, two winches, a container, and a gantry crane rack. It simulates the movement of a container under the combined action of the winches and the rack. Figure 25 illustrates the operating process of the gantry crane. Figure 26 gives the position of the container during loading and unloading, from the initial position {7.5, 12, 0} to the final position {-6.5, 12, 10}.

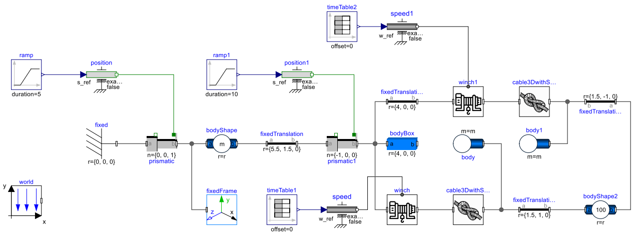


Figure 24. Diagram view of gantry crane model.

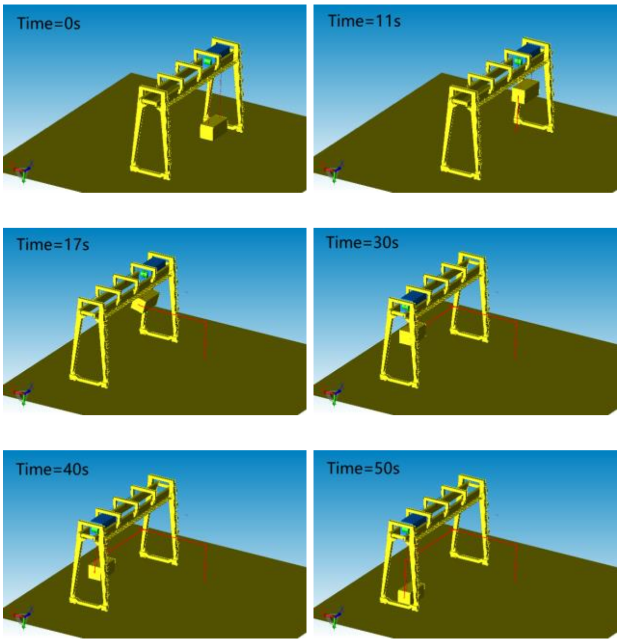


Figure 25. Gantry crane operating processes at different times.

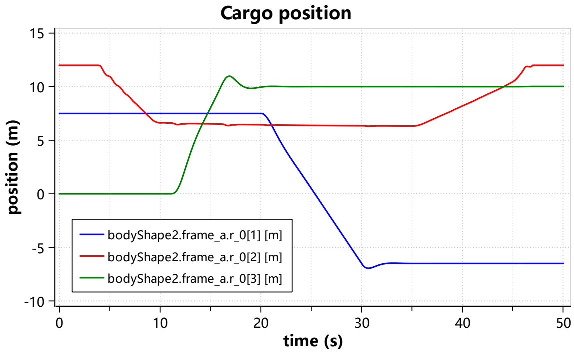


Figure 26. Cargo position as function of time.

5 Conclusion

The simulation of cable drive system covers the area of rigid body dynamics, flexible body dynamics, contact mechanics, etc. In this paper, the three-dimensional flexible cable model is established based on the absolute nodal coordinate formulation. Using the Modelica Multibody Library, the rigid-flexible coupling analysis is accomplished. Utilizing the contact force models, the contact behavior between cables and pulleys are realized. What have not yet been covered in this paper are the self-contact of cables, efficient contact detection algorithms, advanced cable elements (e.g., ALE-ANCF elements), etc. It is noted that, MWORKS.Sysplorer provides an extensive and in-depth platform for modelling cable drive systems.

The content covered in this paper is a prototype of a commercial cable pulley library. There are still some areas of improvement. For example, we did not establish the contact relationship between cables and pulleys at the graphical level. It concerns the ease of use of the library. Another point is that the initial configuration of the cable can only be specified as a straight segment at this moment. In addition, when the winch is involved in too many turns of the cable, the calculation cost brought about by contact forces should not be underestimated. More advanced technologies such as ALE (Arbitrary-Lagrange–Euler) elements will be considered in the future.

References

Aufaure, M. (1993). A finite element of cable passing through a pulley. *Computers & Structures*, 46(5), 807-812. DOI: 10.1016/0045-7949(93)90143-2.

Berzeri, M., & Shabana, A. A. (2000). Development of simple models for the elastic forces in the absolute nodal co-ordinate formulation. *Journal of sound and vibration*, 235(4), 539-565. DOI: 10.1006/jsvi.1999.2935.

Botta, E. M., Sharf, I., & Misra, A. K. (2017). Contact dynamics modeling and simulation of tether nets for space-debris capture. *Journal of Guidance, Control, and Dynamics*, 40(1), 110-123. DOI: 10.2514/1.G000677.

Ferretti, G., Schiavo, F., & Vigano, L. (2005, March). Object-oriented modelling and simulation of flexible multibody thin beams in modelica with the finite element method. In *4th Modelica Conference, Hamburg-Harburg, Germany*.

Gerstmayr, J., & Shabana, A. A. (2006). Analysis of thin beams and cables using the absolute nodal co-ordinate formulation. *Nonlinear Dynamics*, 45, 109-130. DOI: 10.1007/s11071-006-1856-1.

Gerstmayr, J., & Irschik, H. (2008). On the correct representation of bending and axial deformation in the absolute nodal coordinate formulation with an elastic line approach. *Journal of Sound and Vibration*, 318(3), 461-487. DOI: 10.1016/j.jsv.2008.04.019.

Heckmann, A., Otter, M., Dietz, S., & López, J. D. (2006). The DLR FlexibleBody library to model large motions of beams and of flexible bodies exported from finite element programs. In Proceedings (pp. 85-95).

Likins, P. W. (1967). Modal method for analysis of free rotations of spacecraft. *Aiaa Journal*, 5(7), 1304-1308. DOI: 10.2514/3.4188.

Liu, C., Tian, Q., Yan, D., & Hu, H. (2013). Dynamic analysis of membrane systems undergoing overall motions, large deformations and wrinkles via thin shell elements of ANCF. *Computer Methods in Applied Mechanics and Engineering*, 258, 81-95. DOI: 10.1016/j.cma.2013.02.006.

Lugrís, U., Escalona, J. L., Dopico, D., & Cuadrado, J. (2011). Efficient and accurate simulation of the rope–sheave interaction in weight-lifting machines. Proceedings of the Institution of Mechanical Engineers, Part K: *Journal of Multi-body Dynamics*, 225(4), 331-343. DOI: 10.1177/146441931140322.

Pfeiffer, F., & Glocker, C. (Eds.). (2000). *Multibody dynamics with unilateral contacts* (Vol. 421). Springer Science & Business Media.

Rouvinen, A., Lehtinen, T., & Korkealaakso, P. (2005). Container gantry crane simulator for operator training. Proceedings of the Institution of Mechanical Engineers, Part K: *Journal of Multi-body Dynamics*, 219(4), 325-336. DOI: 10.1243/146441905X63322.

Shabana, A. A. (1996). Finite element incremental approach and exact rigid body inertia.

Shabana, A. A. (1996). An absolute nodal coordinate formulation for the large rotation and deformation analysis of flexible bodies. *Technical Report# MBS96-1-UIC, Department of Mechanical Engineering, University of Illinois at Chicago*.

Shabana, A. A. (1997). Flexible multibody dynamics: review of past and recent developments. *Multibody system dynamics*, 1, 189-222.

Shan, M., Guo, J., & Gill, E. (2020). An analysis of the flexibility modeling of a net for space debris removal. *Advances in Space Research*, 65(3), DOI: 1083-1094. 10.1016/j.asr.2019.10.041.

Sugiyama, H., Escalona, J. L., & Shabana, A. A. (2003). Formulation of three-dimensional joint constraints using the absolute nodal coordinates. *Nonlinear Dynamics*, 31, 167-195.

Development of a novel quasi-2D PEM Electrolyzer Model in Modelica

Ansgar Reimann[1] Paul Kohlenbach[2] Lars Röntzsch[3]

[1]Thermodynamic Converters, Fraunhofer IEG, Germany, ansgar.reimann@ieg.fraunhofer.de
[2]Mechanical Engineering / Renewable Energy, BHT Berlin, Germany, kohlenbach@bht-berlin.de
[3]Thermal Energy Technology, BTU Cottbus, Germany, lars.roentzsch@b-tu.de

Abstract

To increase the efficiency of PEM electrolysis, simulation models are required that accurately describe the system's electrochemical and thermal behavior in a computationally efficient manner and are thus suitable for developing control strategies. Therefore, a quasi-2D PEM electrolyzer model is presented in this paper, which is a compromise between the previously developed models regarding their model complexity. The electrochemical behavior is described with equations commonly used in the literature and the thermal behavior with correlations for gas-liquid heat transfer. Preliminary validation indicates that the model can describe the electrochemical behavior and thermal dynamics of a PEM electrolysis stack with good accuracy.

Keywords: PEM electrolysis, dynamic modeling, quasi-2D, gas-liquid heat transfer

1 Introduction

Hydrogen will play a decisive role in the decarbonization of future energy systems. Consequently, the number of electrolyzers worldwide will have to increase significantly in the future to be able to produce the required quantities with low emissions. According to the National Hydrogen Strategy of the German Government, a hydrogen demand of approx. 90 to 110 TWh a^{-1} is expected by the year 2030. Generation plants with a total installed capacity of 5 GW are planned to meet this demand (BMWK 2020). The European Commission (2020) even expects a total installed capacity of 40 GW in the EU.

Proton exchange membrane (PEM) electrolysis is of particular importance here because of its suitability for coupling with volatile sources of electrical energy due to its rapid start-up and shutdown behavior and its partial and overload capability. But according to the current state of technology, only stack efficiencies between 56 and 74 % based on the lower heating value of hydrogen are achieved (Tjarks 2017). Approximately one-third of the electrical energy supplied is thus converted into heat. This heat is currently mostly dissipated directly to the environment via heat exchangers. The overall efficiency and cost-effectiveness of PEM electrolysis could be significantly increased by using this waste heat. However, to exploit this unused potential, models are needed that accurately describe the heat transfer processes in the electrolyzer. In particular, for the development of control strategies, models are needed that can realistically represent the thermal dynamics of electrolysis stacks.

While their electrochemical behavior has been extensively studied, their dynamic thermal behavior has been mostly simplified. A large number of models use the so-called lumped parameter approach, where incoming and outgoing heat fluxes are calculated based on the assumption that the electrolysis stack has a uniform temperature at each time step (Crespi et al. 2023; Espinosa-López et al. 2018; García-Valverde, Espinosa and Urbina 2012; Sood et al. 2020). This type of model has already been implemented in Modelica by Webster and Bode (2019). They are computationally efficient, but cannot represent the heat transfer within the stack and require intensive experimental studies for parameterization. On the other hand, there are multiple models describing and investigating the heat transfer in the electrolysis cell using complex 3D finite volume approaches (Ma et al. 2021; Toghyani, Afshari and Baniasadi 2019; Zhang and Xing 2020). Although these models represent heat transfer very accurately, they are not suitable for dynamic simulation over longer periods due to their complexity.

In the field of fuel cell modeling, some authors discretize the cell components in only one dimension and then couple the discretized cell components with each other. These approaches are called 1D+1D or quasi-2D models (Gong et al. 2022; Tang et al. 2017). Some similar approaches have also been developed in the field of PEM electrolysis but without heat transfer description in the flow channels (Kim, Park and Lee 2013; Lin and Zausch 2022). Therefore, a quasi-2D model of a PEM electrolyzer is presented in this work, which can describe the heat transfer processes in the individual cells and thus the thermal dynamics of the entire stack.

DOI
10.3384/ecp20463

2 Model Description

2.1 General Structure

Figure 1 shows the basic structure of a PEM electrolysis cell. In an electrolysis stack, several cells are connected in series, with the bipolar plate of the cathode serving as the bipolar plate of the next cell's anode.

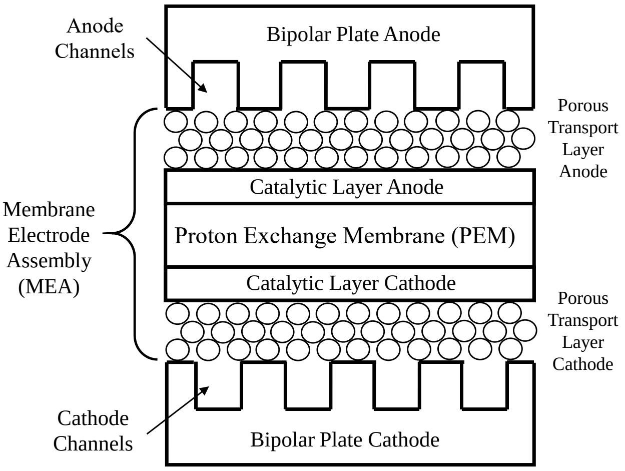

Anode Channels
Bipolar Plate Anode
Porous Transport Layer Anode
Membrane Electrode Assembly (MEA)
Catalytic Layer Anode
Proton Exchange Membrane (PEM)
Catalytic Layer Cathode
Porous Transport Layer Cathode
Cathode Channels
Bipolar Plate Cathode

Figure 1. Basic structure of a PEM electrolysis cell.

In the present model, a parallel flow field design is assumed and the cells are discretized in flow direction only. Temperatures, flow velocities, etc. perpendicular to the channel orientation are thus assumed to be uniform in the respective volumes. Furthermore, the porous transport and catalytic layers as well as the PEM are combined to a uniform thermal mass and together represent the Membrane Electrode Assembly (MEA), where the conversion of water to hydrogen and oxygen takes place.

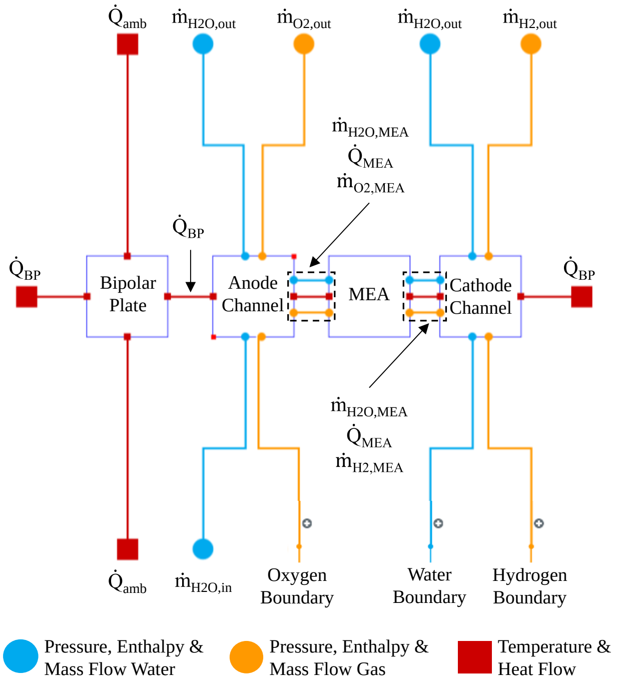


Figure 2. Structure of a single cell.

Figure 2 shows the discretized electrolysis cell's structure in the Modelica development environment. The *TILSuite* package by TLK Thermo GmbH serves as the model basis for the fluid data calculation and the creation of boundary conditions. To be precise, the *Connector*, *Boundary*, *Splitter*, *Joiner*, and *TILMedia* substance data models were used.

The anode and cathode channel volumes each have a gas and a water inlet and outlet at the top and bottom, allowing the individual volumes to be interconnected. Except for the anode's water inlet, all inlets of the first cell volumes are provided with boundaries whose mass flow is equal to zero, since process water is usually the only incoming mass flow. The MEA is connected to the flow channel volumes via three connectors to describe the gas, water, and heat exchange between them. The heat ports on the left and right connect the bipolar plate and cathode flow channel volumes to the neighboring cells. The heat ports at the top and bottom connect the bipolar plate to the ambient.

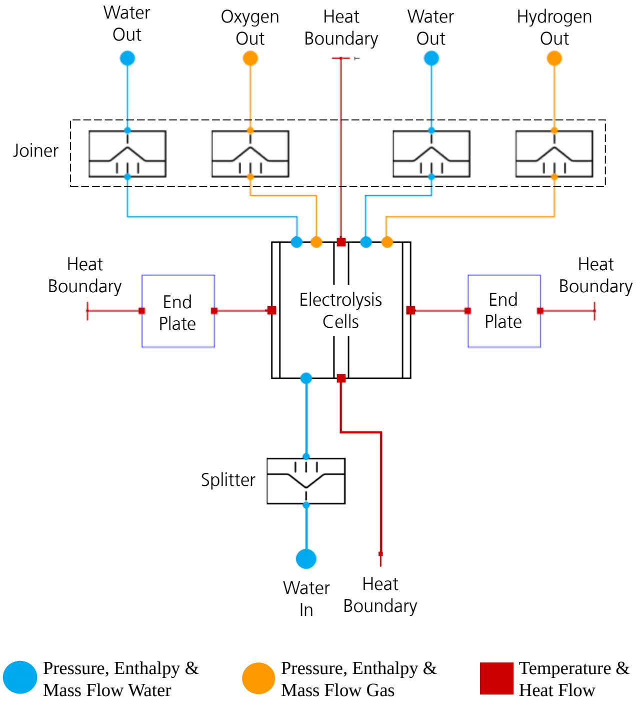


Figure 3. Structure of the stack.

Based on the single-cell model, the stack is now discretized in cell direction. For this purpose, the left and right heat ports of the individual cells are connected. The flow inputs and outputs are connected using so-called joiners and splitters (Figure 3). Joiners add the individual cells' mass flows and form the arithmetic average of their temperatures. Splitters distribute the incoming mass flow evenly over the cells. The heat ports of the first and last cells are connected to the end plates. These have a significantly larger volume than the bipolar plates and are therefore considered separately. All remaining heat ports are connected to heat boundaries representing the ambient temperature. Due to the asymmetric structure of the single cell, a discretization according to Figure 3 would model a redundant bipolar plate. This was solved in the original model by adding a single cell without a bipolar plate but is not shown in Figure 3 for reasons of clarity.

The following sections explain how the individual components can be described mathematically.

2.2 Bipolar / End Plate

The bipolar and end plates of the electrolyzer are treated as lumped capacitance masses C_{th} with a uniform temperature T at every timestep. It is assumed that they are made of titanium. The temperature can be calculated through the following ordinary differential equation:

$$C_{th}\frac{dT}{dt} = \sum_j \dot{Q}_{BP,j} + \sum_i \dot{Q}_{amb,i} \tag{1}$$

The heat flows to the flow channels $\dot{Q}_{BP,j}$ are calculated in the respective cells. $\dot{Q}_{amb,i}$ describes the heat flows to the surrounding ambient. They can be defined as:

$$\dot{Q}_{amb} = A\alpha(T_{amb} - T) \tag{2}$$

For the heat transfer coefficients, values are taken from the work of Tjarks (2017). There, α_{bp} = 2.5 W m^{-2} K^{-1} for the edges of the bipolar plates and α_{ep} = 3.6 W m^{-2} K^{-1} for the end plates were calculated, which is in good accordance with the values determined for alkaline electrolyzers by Diéguez et al. (2008).

2.3 Anode / Cathode Flow Channel

Because of the production of hydrogen and oxygen in the MEA, gas-liquid flow occurs in the flow channels. The following section describes the mass balance, energy balance, and the calculation of heat transfer in them. Pressure loss, on the other hand, is ignored and all fluids are assumed to be incompressible. Therefore, the momentum balance is not presented.

In the anode flow channel, incoming process water gets mixed with oxygen from the MEA. In addition, water flows into the MEA due to its electrochemical conversion and electro-osmosis. In the cathode flow channel, water and hydrogen enter from the MEA. Therefore, the mass balances of the anode and the cathode flow channel volumes can be described with equations 3 and 4.

In reality, hydrogen and oxygen cross the MEA as well due to diffusion and pressure differences. In this work, however, these mass flows are neglected in the calculation of flow conditions and heat transfer in the channels. They will be considered later when describing the gas flows into the flow channels, to be able to represent the effective hydrogen and oxygen production correctly.

$$\begin{aligned}\dot{m}_{H2O,in} + \dot{m}_{O2,in} + \dot{m}_{H2O,out} + \dot{m}_{O2,out} \\ + \dot{m}_{H2O,MEA} + \dot{m}_{O2,MEA} = 0\end{aligned} \tag{3}$$

$$\begin{aligned}\dot{m}_{H2O,in} + \dot{m}_{H2,in} + \dot{m}_{H2O,out} + \dot{m}_{H2,out} \\ + \dot{m}_{H2O,MEA} + \dot{m}_{H2,MEA} = 0\end{aligned} \tag{4}$$

The steady-state energy balance can be formulated via equation 5 with the sum of enthalpy flows into and out of the flow channel and the heat flows to the MEA and bipolar plate. A dynamic energy balance was not introduced since the storage capacity of the flow channel volumes is assumed to be negligible. Furthermore, it is assumed that both fluids in the respective flow channels have the same temperature when leaving the channel volume ($T_{gas,out} = T_{H2O,out}$).

$$\sum_i \dot{m}_{in,i}h_i + \sum_j \dot{m}_{out,j}h_j + \dot{Q}_{BP} + \dot{Q}_{MEA} = 0 \tag{5}$$

The heat flows $\dot{Q}_{MEA}$ and $\dot{Q}_{BP}$ can be described in analogy to equation 2. The fluid temperatures in the volumes T_{fluid} are defined as the arithmetic average between the inlet and outlet of the flow channel volume ($T_{fluid} = 0.5T_{fluid,in} + 0.5T_{fluid,out}$). The overall flow channel volume temperature T_{fc} again is defined as the arithmetic average between the gas and water temperature ($T_{fc} = 0.5T_{H2O} + 0.5T_{gas}$).

Because the flow channels are not in contact with the MEA and bipolar plate over the complete cell area A_{cell}, the correction factor n_A is introduced, which is assumed to be $n_A = 0.6$ (Figure 4). Consequently, the contact area with the MEA becomes $A_{MEA} = A_{cell} \cdot n_A$. Since the flow channels are in contact with the bipolar plate on three sides, the contact area with the bipolar plate becomes $A_{bp} = 3A_{cell} \cdot n_A$ if the flow channels are assumed to be quadratic. To reduce the model complexity, the heat transfer between the MEA and bipolar plate is neglected. The flow channels are assumed to have a thickness of w_{fc} = 1 mm.

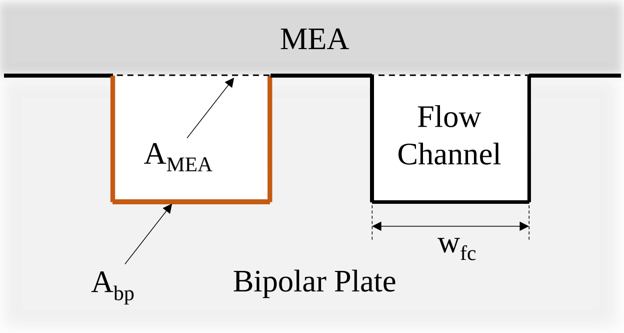


Figure 4. Basic flow channel geometry.

For the heat transfer coefficient calculation in the channel volumes, the general correlation for heat transfer in vertical channels with gas-liquid flow derived by Shah (2018) is used. There, α_{LS} is first calculated as if the gas phase were not present. Then, the heat transfer coefficient of the gas-liquid mixture α_{TP} is calculated using the velocity ratio between the pure gas and pure liquid phase u_r. There are three different formulations for α_{TP} depending on the pure liquid's Reynolds number R_{LS}:

For $15 < Re_{LS} < 175$:

$$\alpha_{TP} = \alpha_{LS}(1 + u_r)^{0.25} \tag{6}$$

For $Re_{LS} \leq 15$:

$$\alpha_{TP} = 0.75\alpha_{LS}(1 + u_r)^{0.25} \tag{7}$$

For $Re_{LS} > 175$:

$$\alpha_{TP} = \alpha_{LS}\frac{E(414 + 89.4u_r^{0.49})}{(365 + u_r^{0.49})} \tag{8}$$

Factor E is calculated using the Froude number of the pure liquid phase Fr_{LS} (equation 9). For $Fr_{LS} > 10$, it becomes $E = 1$.

$$E = \max(0.7Fr_{LS}^{-0.36}, 1.41Fr_{LS}^{-0.15}, 1) \tag{9}$$

The velocities in the flow channels are determined by their average single-phase volume flows and the total flow channel cross-sectional area. However, this velocity calculation is only valid if there is a parallel flow field design. Since the flow channels are assumed to be quadratic, the hydraulic diameter is $d_h = w_{fc}$.

The heat transfer coefficient for the liquid phase is calculated by the correlation of Sieder and Tate (1936) for the laminar and by the correlation of Dittus and Boelter (1985) for the turbulent regime, where L is the total channel length, λ_{LS} is the thermal conductivity and Pr_{LS} is the Prandtl number of the pure liquid (equations 10 and 11). Because the transition from laminar to turbulent occurs at significantly lower Reynolds numbers in gas-liquid flow than in single-phase flow, the correlation for laminar flow applies only up to a Reynolds number of $Re_{LS} < 170$.

$$\alpha_{LS,lam} = 1.86\left(Re_{LS}Pr_{\mathrm{LS}}\left(\frac{d_h}{L}\right)\right)^{\frac{1}{3}}\frac{\lambda_{LS}}{d_h} \tag{10}$$

$$\alpha_{LS,turb} = 0.023Re_{LS}^{0.8}Pr_{LS}^{0.4}\frac{\lambda_{LS}}{d_h} \tag{11}$$

2.4 Membrane Electrode Assembly

In the MEA, the conversion of water to hydrogen and oxygen takes place. The efficiency of this process depends on the cell voltage V_{cell}, which can be described as the sum of the open-circuit voltage V_{ocv}, the activation overvoltage V_{act}, and the ohmic overvoltage V_{ohm} (equation 12). The concentration overvoltage is neglected in this work because of its minimal effects at typical operating densities (Espinosa-López et al. 2018).

$$V_{cell} = V_{ocv} + V_{act} + V_{ohm} \tag{12}$$

The open-circuit voltage describes the electromotive force that is required to start gas production (equation 13). It depends on the reversible cell voltage V_{rev}, the partial pressures of hydrogen, oxygen, and water vapor in the MEA ($p_{H2,MEA}$, $p_{O2,MEA}$, and p_{H2O}), and its temperature T_{MEA}. R = 8.3145 J mol^{-1} K^{-1} and F = 96 485 C mol^{-1} represent the universal gas constant and Faraday's constant. The reversible cell voltage can be described as a function of MEA temperature using the standard temperature T_{std} = 298.15 K and the reversible cell voltage at standard conditions V_{std} = 1.23 V (equation 14).

$$V_{ocv} = V_{rev} + \frac{RT_{MEA}}{2F}\left(\ln\left(\frac{p_{H2,MEA}p_{O2,MEA}^{0.5}}{p_{H2O}}\right)\right) \tag{13}$$

$$V_{rev} = V_{std} - 0.0009(T_{MEA} - T_{std}) \tag{14}$$

To determine the partial pressures of hydrogen and oxygen in the MEA, their partial pressures in the flow channels $p_{H2,cat}$ and $p_{O2,an}$ must be quantified first. They can be calculated via Dalton's Law using the absolute pressures in the flow channels p_{an} and p_{cat} and the water vapor partial pressure (equations 15-17). The formulation for the water vapor partial pressure is taken from the work of Biaku et al. (2008) and calculated in standard atmospheres (atm).

$$p_{H2,cat} = p_{cat} - p_{H2O} \tag{15}$$

$$p_{O2,an} = p_{an} - p_{H2O} \tag{16}$$

$$p_{H2O} = 6.11 \cdot 10^{-3}\exp\left(17.27\frac{T_{MEA} - 273.15}{T_{MEA} - 34.85}\right) \tag{17}$$

When the electrolyzer is in operation, a pressure difference is established, so the partial pressures of hydrogen and oxygen in the MEA are higher than in the flow channels. The factors A_{cat} and A_{an} describe their linear dependency on the current density i (equations 18 and 19). In the work of Schalenbach et al. (2013), they are specified as A_{cat} = 2.4 bar cm^2 A^{-1} and A_{an}= 2.8 bar cm^2 A^{-1}.

$$p_{H2,MEA} = p_{H2,cat} + A_{cat}i \tag{18}$$

$$p_{O2,MEA} = p_{O2,an} + A_{an}i \tag{19}$$

The activation overvoltage describes the energy that is required to start the electrochemical reaction at the anode and cathode. According to Espinosa-López et al. (2018), it can be calculated only considering the activation overvoltage at the anode since it is significantly larger than at the cathode (equation 20). It depends on the charge transfer coefficient α_{an}, and the exchange current density $i_{0,an}$. The latter is temperature-dependent and can be defined using an Arrhenius expression, with $i_{0,an,std}$ being the exchange current density at standard conditions and

E_{exc} the activation energy required for the electron transport in the anode electrode (equation 21).

$$V_{act} = \frac{RT_{MEA}}{2\alpha_{an}F} \operatorname{asinh}\left(\frac{i}{2i_{0,an}}\right) \tag{20}$$

$$i_{0,an} = i_{0,an,std} \exp\left(-\frac{E_{exc}}{R}\left(\frac{1}{T_{MEA}} - \frac{1}{T_{std}}\right)\right) \tag{21}$$

The ohmic overvoltage describes the voltage loss due to the electrolyzer components' resistance to electric flow. Following Ohm's law, the overvoltage is defined as the product of current density and the sum of electrical resistances. According to Olivier, Bourasseau and Bouamama (2017), it is valid to consider the membrane's electrical resistance R_{mem} as the only resistance since it is the dominant factor (equation 22). The membrane's electrical resistance can be expressed in terms of the membrane thickness δ_{mem} and its protonic conductivity σ_{mem}. The membrane thickness is assumed to be δ_{mem} = 183 µm, which is equivalent to the thickness of a Nafion™ 117 membrane (Chemours 2023). The protonic conductivity can be described with an Arrhenius expression as a function of membrane temperature, with $\sigma_{mem,std}$ being the protonic conductivity at standard conditions and E_{pro} the activation energy required for the electron transport in the membrane (equation 23).

$$V_{ohm} = R_{mem} i = \frac{\delta_{mem}}{\sigma_{mem}} i \tag{22}$$

$$\sigma_{mem} = \sigma_{mem,std} \cdot \exp\left(-\frac{E_{pro}}{R}\left(\frac{1}{T_{MEA}} - \frac{1}{T_{std}}\right)\right) \tag{23}$$

To define the MEA's mass balance, the produced and permeated fluid flows have to be determined. The produced oxygen and hydrogen flow $\dot{m}_{H2,prod}$ and $\dot{m}_{O2,prod}$ and the consumed water flow $\dot{m}_{H2O,cons}$ can be calculated through the electrical current density and the respective molar masses M (equations 24-26). The Faraday efficiency is not introduced, since it depends mainly on the permeated mass flows, which are calculated separately.

$$\dot{m}_{H2,prod} = \frac{iA_{cell}}{2F} M_{H2} = \dot{n}_{H2,prod} M_{H2} \tag{24}$$

$$\dot{m}_{O2,prod} = \frac{iA_{cell}}{4F} M_{O2} = \dot{n}_{O2,prod} M_{O2} \tag{25}$$

$$\dot{m}_{H2O,cons} = \frac{iA_{cell}}{2F} M_{H2O} = \dot{n}_{H2O,cons} M_{H2O} \tag{26}$$

According to Fick's law, the permeated oxygen and hydrogen flows can be described using the membrane's permeability to hydrogen and oxygen ε_{H2} and ε_{O2}, its thickness, and the oxygen and hydrogen partial pressures in the MEA, assuming that the partial pressures of the permeated gases are small in comparison to the product gas partial pressures (equations 27 and 28). Schalenbach et al. (2013) determined $\varepsilon_{H2} = 4.65 \cdot 10^{-11}$ mol cm^{-1} s^{-1} bar^{-1} and $\varepsilon_{O2} = 2 \cdot 10^{-11}$ mol cm^{-1} s^{-1} bar^{-1} for the permeability of a Nafion™ 117 membrane at T_{MEA} = 80 °C. In reality, these values are temperature-dependent, however, they are assumed to be constant in this work since electrolyzers are operated to a large extent at membrane temperatures close to 80 °C.

In addition, the water mass flow $\dot{m}_{H2O,ed}$ is transported from the anode to the cathode through the MEA due to electro-osmosis (equation 29). The factor n_{ed} describes the percentage of proton transport through the membrane that involves water molecules. In the work of Santarelli, Torchio and Cochis (2006) it was given as n_{ed} = 0.27.

$$\dot{m}_{H2,per} = \varepsilon_{H2} \frac{p_{H2,MEA}}{\delta_{mem}} A_{cell} M_{H2} \tag{27}$$

$$\dot{m}_{O2,per} = \varepsilon_{O2} \frac{p_{O2,MEA}}{\delta_{mem}} A_{cell} M_{O2} \tag{28}$$

$$\dot{m}_{H2O,ed} = n_{ed} \frac{iA_{cell}}{F} M_{H2O} \tag{29}$$

It is assumed that the product gases exit the MEA completely saturated with water. According to Dalton's law, the water vapor mass flows $\dot{m}_{vap,an}$ and $\dot{m}_{vap,cat}$ can be calculated via the mass flows of the product gases and the pressure ratio between the water vapor partial pressure and the pressure in the anode and cathode, leading to the following equations for the water vapor mass flows:

$$x_{H2O,an} = \frac{p_{H2O}}{p_{an}} \tag{30}$$

$$x_{H2O,cat} = \frac{p_{H2O}}{p_{cat}} \tag{31}$$

$$\dot{m}_{vap,an} = x_{H2O,an} \frac{\dot{n}_{O2,prod}}{1 - x_{H2O,an}} M_{H2O} \tag{32}$$

$$\dot{m}_{vap,cat} = x_{H2O,cat} \frac{\dot{n}_{H2,prod}}{1 - x_{H2O,cat}} M_{H2O} \tag{33}$$

Consequently, the mass flows into and out of the MEA can be described by the following balance equations:

$$\dot{m}_{H2O,an} = \dot{m}_{H2O,cons} + \dot{m}_{H2O,ed} + \dot{m}_{vap,tot} \tag{34}$$

$$\dot{m}_{O2,an} = \dot{m}_{O2,prod} - \dot{m}_{O2,per} \tag{35}$$

$$\dot{m}_{H2O,cat} = \dot{m}_{H2O,ed} \tag{36}$$

$$\dot{m}_{H2,cat} = \dot{m}_{H2,prod} - \dot{m}_{H2,per} \tag{37}$$

To determine the MEA's temperature and thus the product gas temperatures and the heat flows into the flow channels, the heat flow rate generated by the electrolysis reaction due to overvoltages $\dot{Q}_{ely}$ must be known. It depends on the current density and the difference between cell and thermoneutral voltage V_{tn} = 1.48 V.

$$\dot{Q}_{ely} = (V_{cell} - V_{tn}) \cdot i \cdot A_{cell} \tag{38}$$

In addition, a latent heat flow $\dot{Q}_{out,lat}$ is removed from the MEA due to water saturation of the product gases. It can be determined by multiplying the vapor mass flows and the water enthalpy of vaporization $\Delta H_{vap} = 40.65$ kJ mol^{-1}.

The MEA itself is described as a lumped capacitance mass analogous to the bipolar / end plates. It is assumed that the porous transport layers represent its only relevant thermal mass. The material is assumed to be titanium. The porous transport layers are filled with a certain percentage of water. The porosity value of $\Phi = 0.37$ is taken from the work of Grigoriev et al. (2009) and the total thickness is assumed to be $w_{MEA} = 1$ mm.

The energy balance can be calculated from the sum of the incoming and outgoing enthalpy flows, the heat flow generated by the electrolysis reaction, the total heat flow into the flow channels $\dot{Q}_{MEA,tot}$, and the latent heat flow removed by the vapor mass flows (equation 39). It is assumed that the temperature of the outgoing mass flows is equal to the MEA's operating temperature ($T_{fluid,out} = T_{MEA}$).

$$C_{th,MEA}\frac{dT_{MEA}}{dt} = \sum_i \dot{m}_{in,i}h_i - \sum_j \dot{m}_{out,j}h_j + \dot{Q}_{ely} - \dot{Q}_{MEA,tot} - \dot{Q}_{out,lat} \tag{39}$$

3 Model Validation

To validate the simulation model, experimental data from a 1 kW PEM electrolysis test stand at the City University of Applied Sciences Bremen was used. Although the measured data are not ideal for validating the present model due to a lack of large load steps, they can be used to demonstrate the general functionality of the model. The electrolyzer's technical specifications and simulation parameters required for the validation are listed in Table 1.

Table 1. Technical specifications & simulation parameters.

Parameter	*Value*	*Unit*
Ambient Temp.	22	°C
Max. Power	1.88	kW
Max. Current	75	A
Max. Voltage	25	V
Number Cells	10	-
Discretization Cell	5	-
Cell Area	30	cm^2
Pressure Anode	1	bar
Pressure Cathode	5	bar
Water Mass Flow	105	l h^{-1}
L x W x H	174 x 107 x 110	mm

Based on the stack dimensions it is assumed that the bipolar plates have a thickness of $w_{bp} = 3$ mm and the end plates of $w_{ep} = 60$ mm. The test series used for the validation primarily served to determine the current-voltage characteristic of the electrolyzer. For this purpose, current densities from i = 0,1 A cm^{-2} to i = 2.5 A cm^{-2} were set and the cell voltages and water temperatures were measured.

Figure 5 shows the stack's current-voltage characteristic at $T_{MEA} \approx 65$ °C and the pressures listed in Table 1. To derive the missing parameters α_{an}, $i_{0,an,std}$, E_{exc}, E_{pro}, and $\sigma_{mem,std}$ from this curve for the simulative mapping of the current-voltage relationship as described in section 2.4, the *SciPy* Python package was used. Its *curve_fit* function performs a non-linear least squares analysis to fit a set of m observations with a model that is non-linear in n unknown parameters. The fitted curve and the missing parameters are presented in Figure 5. They show a high agreement with the corresponding values from the literature review conducted by Espinosa-López et al. (2018).

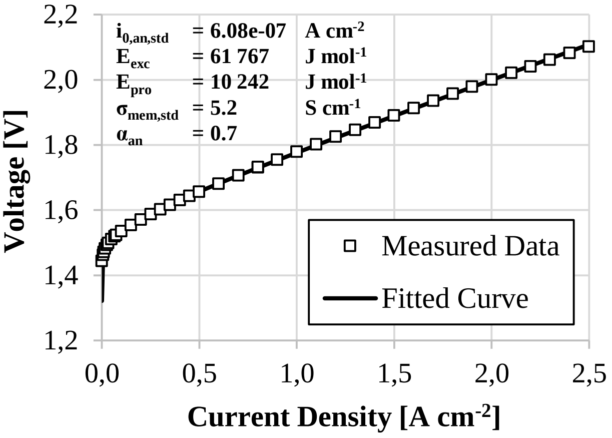


Figure 5. Current-voltage characteristic and fitted parameters.

After finalizing the model parameterization with the calculated parameters, a simulation was performed using the input current profile with which the current-voltage characteristic was determined. The current profile and the resulting measured and simulated stack voltages are shown in Figure 6. It can be seen that in both simulation and measured data, the stack voltage drops and rises with decreasing and increasing current following the current-voltage characteristic. The simulation result shows high accuracy with a mean absolute error of $\Delta V_{MAE} = 0.088$ V and a maximum deviation of $\Delta V_{max} = 0.40$ V.

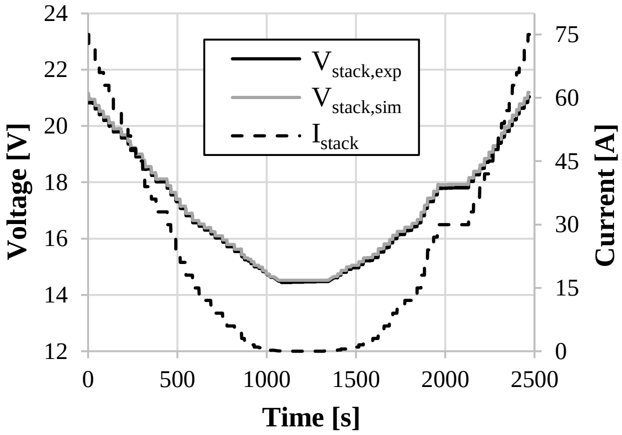


Figure 6. Input current and measured vs. simulated stack voltage.

To verify the thermal modeling, the measured anode water outlet temperature was compared with the simulated one. For the simulation, the measured anode inlet temperature was used. The result is shown in Figure 7. It can be seen that the temperature difference between the inlet and outlet in both simulation and measured data drops and rises as the current decreases and increases due to varying stack heat production. The simulated and measured temperature curves agree to a large extent, the mean absolute error is $\Delta T_{MAE} = 0.239$ K, and the maximum deviation $\Delta T_{max} = 0.630$ K.

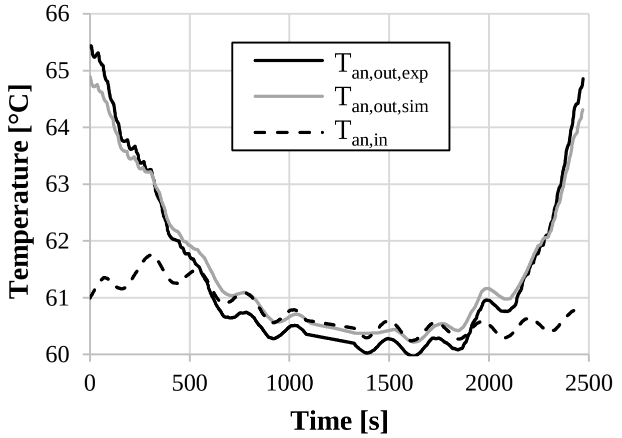


Figure 7. Anode inlet temperature and measured vs. simulated anode outlet temperature.

4 Discussion

Using the obtained experimental data, the general model functionality was successfully demonstrated. However, minor differences between simulation results and experimental data were observed during the simulation of the anode water mass flow's outlet temperature. As the specifications of the employed temperature sensors were unknown, the deviations could be within their measurement inaccuracies. Furthermore, the water mass flow was not measured during the experiment but determined afterward by metering. Therefore, an incorrect mass flow rate may have been used for the simulation. This is also indicated by the fact that the stack's thermal energy balances calculated from the experimental data, once using mass flow and temperature difference and once using equation 38, show significant differences, which cannot be justified by additional heat losses or sources.

Having said this, it is important to note that the presented model is a work in progress. During the development and parameterization, a multitude of assumptions and simplifications were made, e.g. the cell element dimensions, the parallel flow field design, and the steady-state energy balance in the flow channels. Also, as mentioned before, it should be highlighted that the experimental data utilized for validation was not ideal for assessing the dynamic thermal behavior, as it lacked significant load variations.

Furthermore, the model incorporates calculations for substance transport through the membrane to realistically capture the quantities of produced oxygen and hydrogen. However, product mass flows, impurity gas fractions and cathode water mass flow were not measured during the experimental investigation. Therefore, it is necessary to conduct more comprehensive validation studies in the future, aiming to verify the accurate representation of all physical phenomena within the model.

5 Conclusion

In this paper, a quasi-2D model of a PEM electrolyzer was presented. The individual cells were discretized in flow direction of the flow channels, and the electrochemical and thermal behavior was described using analytical equations. Furthermore, the cells in the stack were thermally coupled to each other.

To validate the developed model, experimental data from a 1 kW PEM electrolyzer stack at the City University of Applied Sciences Bremen were utilized. Comparison of the measured data with simulation results demonstrated high accuracy in capturing the electrochemical behavior of the stack. Smaller deviations between thermal simulation results and measurement data can most likely be justified by imprecise data acquisition.

However, not all modeled physical phenomena could be validated using experimental data. In addition, a large number of assumptions were made regarding the design of the stack, which could not be substantiated. Further experimental investigation will be necessary in the future to comprehensively validate the model.

Acknowledgements

This work has been supported by the Federal Ministry of Education and Research as part of the IntegrH2ate Project (grant number 03HY127B). The authors would like to thank Prof. Dr. Lars Jürgensen and Nicol Daniela Jaramillo Rodriguez from the City University of Applied Sciences Bremen for the recording and provision of the experimental data and Aline Luxa from the Fraunhofer Institute for Wind Energy Systems for her support in obtaining experimental data.

SPONSORED BY THE

Federal Ministry
of Education
and Research

References

Biaku, C., N. Dale, M. Mann, H. Salehfar, A. Peters and T. Han. “A Semiempirical Study of the Temperature Dependence of the Anode Charge Transfer Coefficient of a 6kW PEM Electrolyzer.” In: *International Journal of Hydrogen Energy* 33, no. 16 (2008): pp. 4247–54. DOI: 10.1016/j.ijhydene.2008.06.006.

BMWK. “Die Nationale Wasserstoffstrategie.” Berlin, June 2020. Accessed July 26, 2022. https://www.bmwk.de/Redaktion/DE/Publikationen/Energie/die-nationale-wasserstoffstrategie.html.

Chemours. “Nafion™ Sulfonic Membranes.” Accessed June 2, 2023. https://www.nafion.de/products/sulfonic-membranes.

Crespi, E., G. Guandalini, L. Mastropasqua, S. Campanari and J. Brouwer. “Experimental and Theoretical Evaluation of a 60 KW PEM Electrolysis System for Flexible Dynamic Operation.” In: *Energy Conversion and Management* 277 (2023): p. 116622. DOI: 10.1016/j.enconman.2022.116622.

Diéguez, P. M., A. Ursúa, P. Sanchis, C. Sopena, E. Guelbenzu and L. M. Gandía. “Thermal Performance of a Commercial Alkaline Water Electrolyzer: Experimental Study and Mathematical Modeling.” In: *International Journal of Hydrogen Energy* 33, no. 24 (2008): pp. 7338–54. DOI: 10.1016/j.ijhydene.2008.09.051.

Dittus, F. W. and L.M.K. Boelter. “Heat Transfer in Automobile Radiators of the Tubular Type.” In: *International Communications in Heat and Mass Transfer* 12, no. 1 (1985): pp. 3–22. DOI: 10.1016/0735-1933(85)90003-X.

Espinosa-López, M., C. Darras, P. Poggi, R. Glises, P. Baucour, A. Rakotondrainibe, S. Besse and P. Serre-Combe. “Modelling and Experimental Validation of a 46 KW PEM High Pressure Water Electrolyzer.” In: *Renewable Energy* 119 (2018): pp. 160–73. DOI: 10.1016/j.renene.2017.11.081.

European Commission. A hydrogen strategy for a climate-neutral Europe. 2020. Accessed November 2, 2022. https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:52020DC0301.

García-Valverde, R., N. Espinosa and A. Urbina. “Simple PEM Water Electrolyser Model and Experimental Validation.” In: *International Journal of Hydrogen Energy* 37, no. 2 (2012): pp. 1927–38. DOI: 10.1016/j.ijhydene.2011.09.027.

Gong, Z., B. Wang, K. Wu, T. Miao, K. Yang, S. Zhai, R. Ma, F. Gao and K. Jiao. “A 1 + 1-D Multiphase Proton Exchange Membrane Fuel Cell Model for Real-Time Simulation.” In: *IEEE Transactions on Transportation Electrification* 8, no. 2 (2022): pp. 2928–44. DOI: 10.1109/TTE.2021.3115794.

Kim, H., M. Park and K. S. Lee. “One-Dimensional Dynamic Modeling of a High-Pressure Water Electrolysis System for Hydrogen Production.” In: *International Journal of Hydrogen Energy* 38, no. 6 (2013): pp. 2596–2609. DOI: 10.1016/j.ijhydene.2012.12.006.

Lin, N. and J. Zausch. “1D Multiphysics Modelling of PEM Water Electrolysis Anodes with Porous Transport Layers and the Membrane.” In: *Chemical Engineering Science* 253 (2022): p. 117600. DOI: 10.1016/j.ces.2022.117600.

Ma, Z., L. Witteman, J. A. Wrubel and G. Bender. “A Comprehensive Modeling Method for Proton Exchange Membrane Electrolyzer Development.” In: *International Journal of Hydrogen Energy* 46, no. 34 (2021): pp. 17627–43. DOI: 10.1016/j.ijhydene.2021.02.170.

Olivier, P., C. Bourasseau and P. B. Bouamama. “Low-Temperature Electrolysis System Modelling: A Review.” In: *Renewable and Sustainable Energy Reviews* 78 (2017): pp. 280–300. DOI: 10.1016/j.rser.2017.03.099.

Santarelli, M. G., M. F. Torchio and P. Cochis. “Parameters Estimation of a PEM Fuel Cell Polarization Curve and Analysis of Their Behavior with Temperature.” In: *Journal of Power Sources* 159, no. 2 (2006): pp. 824–35. DOI: 10.1016/j.jpowsour.2005.11.099.

Schalenbach, M., M. Carmo, D. L. Fritz, J. Mergel and D. Stolten. “Pressurized PEM Water Electrolysis: Efficiency and Gas Crossover.” In: *International Journal of Hydrogen Energy* 38, no. 35 (2013): pp. 14921–33. DOI: 10.1016/j.ijhydene.2013.09.013.

Shah, Mirza M. “General Correlation for Heat Transfer to Gas–Liquid Flow in Vertical Channels.” In: *Journal of Thermal Science and Engineering Applications* 10, no. 6 (2018). DOI: 10.1115/1.4040652.

Sieder, E. N. and G. E. Tate. “Heat Transfer and Pressure Drop of Liquids in Tubes.” In: *Industrial & Engineering Chemistry* 28, no. 12 (1936): pp. 1429–35. DOI: 10.1021/ie50324a027.

Sood, S., O. Prakash, M. Boukerdja, J.-Y. Dieulot, B. Ould-Bouamama, M. Bressel and A.-L. Gehin. “Generic Dynamical Model of PEM Electrolyser Under Intermittent Sources.” In: *Energies* 13, no. 24 (2020): p. 6556. DOI: 10.3390/en13246556.

Tang, T., S. Heinke, A. Thüring, W. Tegethoff and J. Köhler. “A Spatially Resolved Fuel Cell Stack Model with Gas–liquid Slip Phenomena for Cold Start Simulations.” In: *International Journal of Hydrogen Energy* 42, no. 22 (2017): pp. 15328–46. DOI: 10.1016/j.ijhydene.2017.03.236.

Tjarks, G. H. *PEM-Elektrolyse-Systeme zur Anwendung in Power-to-Gas Anlagen.* Schriften des Forschungszentrums Jülich. Reihe Energie & Umwelt Band 366. Jülich: Forschungszentrum Jülich GmbH, Zentralbibliothek, 2017.

Toghyani, S., E. Afshari and E. Baniasadi. “Three-Dimensional Computational Fluid Dynamics Modeling of Proton Exchange Membrane Electrolyzer with New Flow Field Pattern.” In: *Journal of Thermal Analysis and Calorimetry* 135, no. 3 (2019): pp. 1911–19. DOI: 10.1007/s10973-018-7236-5.

Webster, J. and C. Bode. “Implementation of a Non-Discretized Multiphysics PEM Electrolyzer Model in Modelica”. In: *Proceedings of the 13th International Modelica Conference*, Regensburg, Germany, March 4–6 (2019): pp. 833–840. DOI: 10.3384/ecp19157833.

Zhang, Z. and X. Xing. “Simulation and Experiment of Heat and Mass Transfer in a Proton Exchange Membrane Electrolysis Cell.” In: *International Journal of Hydrogen Energy* 45, no. 39 (2020): pp. 20184–93. DOI: 10.1016/j.ijhydene.2020.02.102.

Modelica Association Standards and Surrogate Modeling to Enable Multi-Fidelity Simulations

Olle Lindqvist[1] Robert Hällqvist[2,4] Raghu Chaitanya Munjulury[3,4]

[1]FCS Verification & Validation, Saab Aeronautics, Sweden
[2]System Simulation and Concept Development, Saab Aeronautics, Sweden, robert.hallqvist@saabgroup.com
[3]Technical Management & Maintenance, Saab Aeronautics, Sweden
[4]Division of Fluid and Mechatronic Systems (FLUMES), Linköping University, Sweden

Abstract

System simulations are particularly useful when analyzing complex systems. Simulations are often cheaper and safer than physical tests of the actual system(s) of interest. Models can additionally be created for systems that do not exist to find solutions that are impossible to analyze experimentally in, for example, early life-cycle stages. Models used in system simulations require appropriate input data to give results with the required fidelity and, in the end, credibility. Integration is often challenging as each system commonly constitutes contributions from several engineering domains. Relying on relevant open standards for information exchange is seen as a means of mitigation. The results of the presented work encompass a developed methodology that allows Computational Fluid Dynamics (CFD) results to be integrated into a simulator using system identification and open standards. Reduced Order Models (ROMs) are generated based on results from a CFD analysis. These ROMs are coupled to lumped parameter system simulation models through the mechanisms of the System Structure and Parameterization (SSP) and Functional Mock-up Interface (FMI) standards. In addition, several important factors to consider before using the proposed methodology are presented. These include the intended use of the ROMs, knowing the flow inside the system, what resources are available, and any potential licensing issues

Keywords: FMI, SSP, CATIA, CFD, System Identification, Neural Networks, Co-simulation

1 Introduction

Complex systems often need to be analyzed using models in order to exploit, understand, and manage emergent behavior. The use of models and simulations instead of physical experiments to analyze systems introduces a number of different benefits in relation to these needs. For instance, models can be created as cheaper and safer alternatives to testing the corresponding physical systems. It is also possible to create models, and multiple models coupled in simulators, of systems that do not exist yet, for which experimentation would be impossible (Ljung and Glad 2003). Dynamic simulation models are models which show how modeled system properties change over time (Ellner and Guckenheimer 2011). Such models are typically described mathematically by differential and algebraic equations. Models in general are typically developed to fulfill some specified purpose. This purpose often requires information gathered from different engineering disciplines. To achieve this exchange efficiently, standardized means to communicate and share digital artifacts across modeling domain boundaries are necessary (Hällqvist, Munjulury, et al. 2021).

At Saab Aeronautics, dynamic models are used during the development of many aircraft systems and sub-systems, both on their own in local desktop environments but also in various co-simulation constellations at different levels of abstraction (from hereon referred to as simulators) (Hällqvist 2023; Steinkellner 2011; H. Andersson 2012). The sharing of digital artifacts between different engineering domains (be it models, simulators, or simulation results) is often challenging partly as a consequence of a need to use the best-suited software, deployed on the most suitable target, for each engineering domain to address different simulation purposes. The export and integration of models have typically, for example, the *Gripen E program* (Saab Group 2023), been performed through in-house developed standards. While these standards are used successfully, maintaining them is difficult and a time-consuming process. Native tool support for in-house standards, not adopted by the community as a whole, is challenging to motivate tool vendors as the tool vendors are bound by their overall customer needs. This challenge only becomes bigger if considering the long life cycles of aircraft as life-cycle information needs to be aggregated, with traceability links, and made available to the end user and decision-makers. Failure to address any of these highlighted aspects may, in the end, lead to sub-optimal designs founded on simulation results with in-accurate credibility (Hällqvist, Munjulury, et al. 2021).

DOI
10.3384/ecp20473

1.1 Contributions

The presented work provides a summary of a master thesis project conducted at SAAB Aeronautics & Linköping University (Lindqvist 2022). A main result of the work is a proposed methodology for creating portable ROMs based on CFD results. In addition, several important factors to consider before using the proposed methodology are presented. These include the intended use of the ROM, a need to know the flow to be analyzed, what resources are available both during model development and end-use, licensing set-ups of tools to be used, etc. The results of the work demonstrate that steady-state data, of internal flow systems, can be used as intended for transient system simulations. The ROMs created using the method gave results that were generally close to the corresponding handbook equations available in the literature, see for example Miller (D. Miller 1990) for a comprehensive theoretical background on internal flow systems.

2 Theoretical Background

This paper incorporates techniques from several fields of research and industry: data-driven methods (system identification) for surrogate modeling purposes, physics-based lumped parameter modeling, CFD, and standardized co-simulation. The work strives to employ a set of standards, exemplified by a set of relevant tools, to address a need identified in the industry. This need is summarized through a development methodology where the best-suited domain-specific tool is used for each engineering task, where each contributing digital artifact and the model-based decision is fully traceable to the end result.

2.1 Utilized Standards

Three different standards jointly enable the interoperability presented herein: the FMI standard (FMI Development Group 2020), the SSP standard (Modelica Association 2019), and the ISO 10303-21 standard also known as Standard for the Exchange of Product Data (STEP) (International Standards Organization (ISO) 2016). All these three specifications strive to provide formats for a neutral, vendor-independent, and platform-independent information exchange and model-based decision-making.

The primary objective of the SSP standard is to establish standardized means for linking simulation models; in the end resulting in portable and executable set of coupled models. This standard offers a nested hierarchical definition of systems and subsystems included within the set. In contrast, the FMI standard focuses on facilitating the exchange of the constituent models, and their interfaces, specific to different engineering domains. The FMI standard additionally provides mechanisms for flexible management of Intellectual Property (IP) and simulation execution. The SSP standard provides a framework for standardized connection, configuration, and exchange of a connected group of models, which collectively form, what here is referred to as, a simulator application. To convey this information, the SSP standard outlines various Extensible Markup Language (XML) schema. The composition of the simulator is stored in the System Structure Description (SSD) format, the parameter values in the System Structure Parameter Values (SSV) format, and the associations of these values with the individual executable models in the System Structure Parameter Mapping (SSM) format. All these different artifacts are exploited in the presented research. The ISO 10303-21 standard specifies a standardized file format for exchanging and representing product data across different Computer Aided Design (CAD) and Computer-Aided Manufacturing (CAM) systems, particularly focusing on data capturing three-dimensional geometric representations. STEP is, unlike the FMI and SSP standards, not an open standard available to all engineers. It is however widely adopted by tool vendors in the geometry and CFD domains. In this work, the STEP format is exploited to exchange geometry models between these two engineering domains. There are several additional features presented in the STEP specification that could contribute to achieving traceable and credible simulations, for example, the *Application protocol 209* for specification and exchange of solver related information (Lanza et al. 2018). These features are however not considered herein, and investigations of their usefulness within the presented context is left for future research.

2.2 Modeling and Simulation Tools

The work expands on a simulator that has been successively developed by Saab Aeronautics in order to capture and communicate industrial requirements on Modelica Association Standards, and the corresponding tool support, that jointly provide technology deduced as essential when developing complex systems (Hällqvist, Naeser, et al. 2022; Lind and H. Andersson 2011). The simulator in focus, see Hällqvist et al. for a detailed description (Hällqvist, Munjulury, et al. 2022), incorporates digital artifacts developed in Dymola (*Dymola User Manual* 2016), OpenModelica (Fritzson et al. 2005), Matlab/Simulink, and CATIA. This work adds high-fidelity information, obtained through CFD analysis, conducted with the Altair suite of tools. The CFD simulations were performed using the Finite Element Method (FEM)-based Altair AcuSolve CFD-solver (Altair 2023). Furthermore, meshing and post-processing were performed using the built-in tools of AcuSolve.

2.3 Systems simulation

In order to simulate complex systems using mathematical models, it is necessary to use some form of computer software. Modelica is one of the, at Saab Aeronautics, commonly used software languages for physics-based modeling and simulation. Physics-based system simulation models are constructed by connecting components, or blocks, in order to transfer information between them

DOI
10.3384/ecp20473

(Modelica Association 2023). Components for use in creating the systems are contained in various modeling libraries.

The simulation of a system is performed by first converting the code of the different components into a system of differential equations. First, the equations are sorted based on the flow of information between them. Second, the system of equations are simplified in order to reduce the computational time. Finally, the equations are solved numerically (Fritzson 2003). Special care needs to be taken if the system contains both fast and slow dynamics. This can cause the solution of the differential equations to become inaccurate or unstable (Ljung and Glad 2003).

2.4 System Identification

System Identification is the process of taking some higher-order data and creating a ROM based on it. The data used could be experimental, from in-situ measurements, or virtual from some higher fidelity simulation. ROMs can be created through various different methods, many of them described in detail by Ljung in (Ljung 1999b). The system identification procedure has three main components (Ljung 1999a): a data set, one or more candidate models to describe the relationship between input and output, and some selected technique for evaluating which model best fits the data. As an example, consider some time-series data with recorded inputs $u(t)$ and outputs $y(t)$. One way of modeling the relationship between them is through a simple difference equation

$$\begin{aligned} y(t)+a_1y(t-1)+\cdots+a_ny(t-n) &= \\ b_1u(t-1)+\cdots+b_nu(t-n) \end{aligned} \tag{1}$$

where a and b are some unknown parameters. In order to calculate $y(t)$, it can simply be isolated on the left-hand side resulting in

$$y(t) = \phi^T(t)\Theta \tag{2}$$

where $\Theta = [a_1 \dots a_n \quad b_1 \dots b_n]^T$ and $\phi(t) = [y(t-1) \dots y(t-n) \quad u(t-1) \dots u(t-n)]^T$. The goal of the system identification procedure is to find the values of the unknown parameters in Θ so that $y(t)$ fits with the recorded data. This can be done through, e.g., statistical methods or machine learning algorithms (Sjöberg et al. 1995). Thus $u(t)$ and $y(t)$ are the data set, Equation 1 is the candidate model, and the method used to calculate Θ is the final step in the above list.

2.5 Neural Networks to realize ROMs

One of the methods for identifying data-driven mathematical models is to train a neural network (Chen, Billings, and Grant 1990) on available data. Neural networks are machines or computer software that solve tasks by imitating the way a brain works. The neural network is built up of interconnected neurons, or nodes. A neural network needs to be trained in order to gain the necessary knowledge to solve a problem. This is done by modifying the strength, or *weight*, of the connections between the neurons. Each neuron has an activation function that determines the output of the neuron based on the strength of the input signals entering it. Some common activation functions are the Rectified Linear Unit (ReLU), logistic, and hyperbolic tangent functions (Haykin 1999). Neural networks can be useful for training on non-linear problems and for mapping the input and output signals of unknown systems. This makes them well suited for system identification tasks (Haykin 1999). There exist many examples in the literature of system identification performed using, for example, Multi-Layer Preceptrons (MLPs) (A. Parlos et al. 1991) (Fernandez, A. G. Parlos, and Tsai 1990).

2.6 Computational Fluid Dynamics

The Navier-Stokes equations are the partial differential equations that describe the motion of viscous fluid substances. Since there exist no known analytical solutions to the Navier-Stokes equations, they have to be solved numerically. This is known as CFD. In CFD, the continuity and Navier-Stokes equations are spatially, and sometimes temporally, discretized to allow for iterative, numerical solutions to be computed (Anderson, John D. 1995). There are several CFD methods available, with two of the most commonly used being the Finite Volume Method (FVM) and the FEM. The difference between these two different methods lies in how the governing equations are discretized.

For the FEM, the weak forms of the governing equations are discretized over the entire fluid domain through, for example, the Galerkin method (Donea, Jean and Huerta, Antonio 2003) (Fontes 2018). However, the convective term $(\nabla \cdot u)u$ is non-linear and gives non-symmetrical coefficient matrices, a problem that only gets bigger with increasing Reynolds numbers and turbulent flows. Thus, special techniques need to be used in order to stabilize the solution (Bathe, Klaus J. 2014). In contrast, the finite volume method discretizes the equations for each control volume (mesh element). This means that the solution will be stable because the flow will naturally be conserved for each element. From a practical point of view, FEM can achieve a higher order of accuracy for the discretization, however, this will also lead to a higher computational cost (Fontes 2018).

In order to spatially discretize the fluid domain, there are several mesh element types to choose from. 2D elements can be, for example, triangular or quadrilateral, while their 3D counterparts can be tetrahedral or hexahedral, etc. The choice of element type depends on the geometry of the domain. Triangular and tetrahedral elements are better at capturing curved and complex geometries, while quadrilaterals and hexahedrals can cover the same domain with fewer elements (Versteeg and Malalasekera 2007).

3 Method

This section describes how the work was carried out and how the previously described theory was tailored and applied to the use-case of this paper. The section covers the three main areas of the presented research: CFD, system identification, and integration of information from contributing disciplines. A description of the resulting application example simulator is also included.

3.1 Application example

The implementation of the targeted aircraft cooling system Modelica model is described in detail by Hällqvist et al. in (Hällqvist, Munjulury, et al. 2021). This particular model represents one essential part of a broader simulator that incorporates models and information from the engineering domains of hardware and physics-based modeling, control development and software modeling, architecture and requirements modeling, and geometry modeling using 3D CAD. The aircraft cooling system constituent piping and the corresponding internal flow is in focus here. The pipe component pressure drop is modeled as a function of the mass flow,

$$\Delta p = \frac{(z + c \cdot l/D)}{A^2 \cdot 2\rho} \dot{m}^2, \tag{3}$$

as described by, e.g., Miller in (D. S. Miller 1990). Here z is a parameter used to account for the pressure loss of pipe features such as bends and changes in the area, c is the friction coefficient, A is the pipe's cross-sectional area, and l is the pipe length. The pipe is connected to a heat exchanger at its outlet and a consumer of cooling power at the inlet. The inputs to the pipe inlet are pressure, mass flow, and enthalpy. Among the specified component outputs are pressure loss, fluid density, viscosity, etc. The parameters for the different system simulation components are automatically imported from the CAD geometry of the system. The Modelica model has been implemented in Dymola, using the in-house developed component library Modelica Fluid Lite (Eek, Gavel, and Ölvander 2017). The boundary conditions for the simulation consist of flight data (altitude and Mach number), and the heat load from the consumer. The atmosphere is modeled using the International Standard Atmosphere (ISA). The atmospheric conditions impact the friction heating and heat transfer from the aircraft to the surroundings. The equations describing this are available in (Hällqvist, Munjulury, et al. 2021). The cooling system incorporated software then regulates the flow to keep the fluid temperature in the feed line at 20 °C.

3.2 Proposed Methodology

A methodology was developed based on the CFD and system identification work done during the thesis (Lindqvist 2022). CFD, System Identification, and ROM implementation make up its three primary stages. The methodology is depicted in more detail in Figure 1, along with the steps that are part of each phase. There are, in addition, several crucial considerations that should be made both before and throughout the work, including the intended purpose of the finished ROM, how the system will function, and the resources that will be accessible.

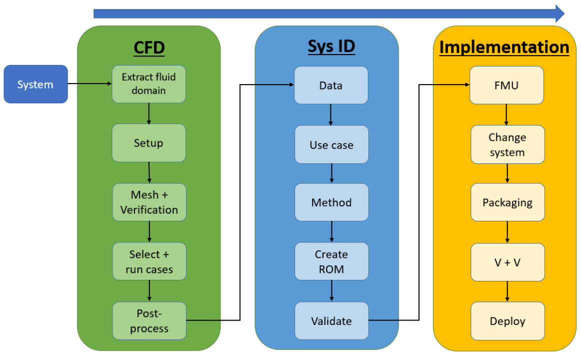


Figure 1. Detailed view of the methodology and its included steps.

The entire procedure is affected by the model's intended use. The appropriate CFD and system identification techniques, for instance, depend on whether the system's transient behavior needs to be accounted for or if steady-state characteristics are sufficient. For the CFD task, understanding the flow within the system that is being modeled is essential in order to, for example, choose the best turbulence model, meshing approach, and to determine whether the findings are plausible or not. In this context, the terms available resources and available software are used interchangeably. This will affect the number and sophistication of CFD simulations that can be performed, the system identification techniques that are accessible, etc. The following are the steps for each phase in Figure 1,

Stage 1: CFD

1. **Extract fluid domain**: Extraction of the fluid domain from the geometry of the system to be simulated is the first step in any CFD analysis. The domain could be divided into sections in order to focus on any interesting flow features while reducing the computational cost. Knowing the flow is crucial in this situation.

2. **Setup**: The setup of the CFD solver must be chosen in the next stage. This includes choosing a turbulence model, whether to execute steady-state or transient simulations, etc. Here, understanding the flow is equally crucial for choosing the modeling strategy that best depicts the anticipated flow characteristics.

3. **Mesh and Verification**: The mesh will need to meet different requirements depending on the turbulence model that is used, such as near-wall resolution. To make sure that it can handle any extreme scenarios, mesh verification should be done in the flow where the maximum turbulence is anticipated.

4. **Select + run cases**: If the system's operational domain is known, an optimization method can be used to determine the ideal number of cases to cover the domain with the fewest number of simulations necessary. The operating domain can be expanded to accommodate system modifications, as was done for this study, to make the final ROM more adaptable.

5. **Post-process**: Depending on how much information the solution uses provides, post-processing the findings may be more or less challenging. When certain variables must be calculated using user-defined expressions, both the workload and the chance of error rise.

CFD can involve a lot of iterations. If, e.g., the findings of the post-processing indicate that a different turbulence model, is required, some processes might need to be repeated.

Stage 2: System Identification
The system identification steps largely follow the process outlined in Section 2.4.

1. **Data**: The data obtained from the CFD simulations need to be imported to the system identification tool used. For steady-state data, it may be necessary to add an "artificial" time-vector for each case, as many system identification techniques assume that such a vector is available.

2. **Method**: Here *method* denotes the decision on what candidate models and evaluation methods to use. The choice is likely dependent on what software that is chosen and available for the task at hand. In this scenario, knowledge about the system and flow can help with deciding, e.g., whether to use linear or nonlinear models.

3. **Create ROM**: It is probable that the creation of the ROM will be highly iterative. It may be necessary to change the candidate models or evaluation in order to get a model that fits the CFD data.

4. **Validate**: The ROM should be validated against higher-order data that was not used to create it. In this study, the validation data consisted of CFD results for the same system at different flow cases.

Stage 3: Implementation:

1. **Functional Mock-up Unit (FMU)**: In order to enable co-simulation, the ROM is exported as a FMU. This allows the ROM to be integrated into any FMI supporting system simulation environment.

2. **Change system**: In this study, the goal of the new ROM was to replace a component in a system model. Thus, the system model had to be modified to accept the new ROM. This step is not applicable when creating ROM for a completely new system model.

3. **Packaging**: The necessary model parameters, specifying the configuration or variant, for the FMUs are specified in an SSV file. If a system of several FMUs is to be simulated, it needs to be packaged as an SSP.

4. **Verification & Validation (V&V)**: The new model or system needs to be verified, to make sure that it reflects its specification, and validated to make sure that it fulfills its intended use.

5. **Deploy**: The model can now be deployed and used for the intended system simulation.

4 Application example

The geometry modeling was performed in CATIA and the resulting geometry model serves as the foundation for the CFD analysis. The CFD simulations were performed using the FEM-based Altair AcuSolve CFD-solver. Meshing and post-processing were also performed using the built-in tools in AcuSolve. These tools were used for convenience, as the Altair tool suite also includes system identification and modeling tools.

4.1 Geometric Modeling

Figure 2 illustrates the feed and return lines of the aircraft cooling system. The starting point of the feed line is connected to a heat exchanger, while the endpoint is attached to a consumer of cooling power, such as a radar. To replicate the flow of the fluid within the pipeline, the CAD model was used to extract the fluid domain using CATIA. The CAD geometry of the fluid domain was then imported into AcuSolve to initiate the flow scenario configurations.

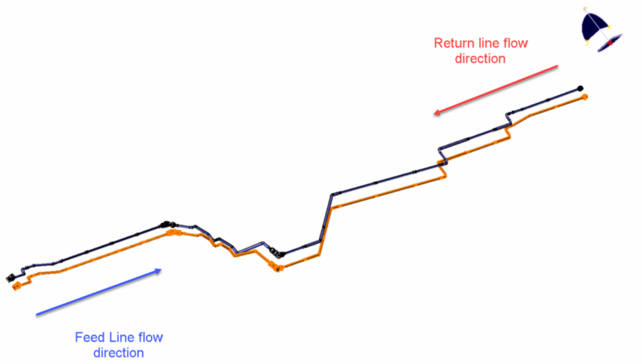


Figure 2. Pipes in the cooling system. The blue pipe is the feed line, while the return line is orange.

To decrease the computational cost of the simulations, the fluid domain of the return line was split into five distinct parts. These pipe sections were specifically selected due to their characteristics that could result in excessive pressure loss and turbulence, such as abrupt expansions and contractions, and acute bends. To ensure fully developed flow at the inlet and avoid reverse flow at the outlet, extensions were added to the inlets and outlets of all pipe sections.

The initial section (heron referred to as *Section 1*) focused on the inlet of the return line. It consisted of a 0.02m diameter pipe connected to a 0.008m diameter pipe through a fitting. The sudden contraction of the fitting was anticipated to cause separation and intensify the turbulence and pressure loss in this section. Since the section is symmetric along its axis, only a quarter of the fluid domain was simulated. The entire length of this section amounted to approximately 0.26m.

Section 2 and Section 3 (Section 2 is shown in Figures 3) exhibited resemblance. Each part comprised a pipe with a diameter of 0.008m and several closely located bends. Consequently, it was probable that turbulence would not disperse between the bends, which would result in an inaccurate estimation of pressure drop. Symmetry could be employed to simplify Section 3.

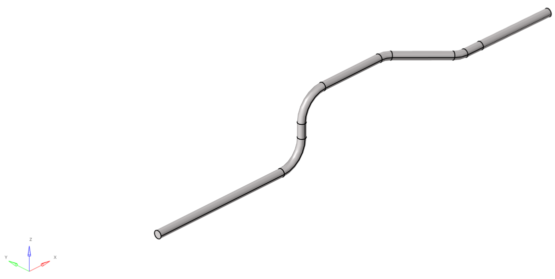


Figure 3. Section 2 of the return line, the flow direction is from right to left.

Section 4, depicted in Figure 4, constituted a complicated portion of the routing located approximately at the midpoint of the return line. A short pipe with numerous bends was linked to two additional linear sections using two fittings. Due to the intricate shape of this section, simplification through symmetry was infeasible. Furthermore, this was the most extensive section that was extracted and required the most computational resources to simulate. The complete length of this segment of the conduit was approximately 1.24m.

The last section dealt with the exit point of the return line (Figure 5). It showcased abrupt enlargements, reductions, and a 90-degree elbow. The inlet was elongated by 0.1m, whereas the outlet was lengthened by 0.2m in order to ensure fully developed flow at the inlet and outlet. This part could also be simplified using symmetry. The overall distance of the pipe in this segment was roughly 0.44m.

A test segment was additionally established to investigate the turbulence and pressure drop caused by the pipe fittings. This segment was comparable to Section 5, except for the appended outlet extension with a length of 50 pipe diameters, and the pipe diameter decreased back to 0.008m. The test segment is illustrated in Figure 6.

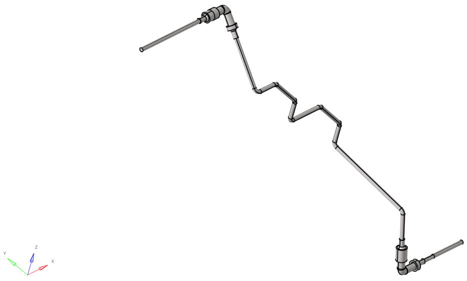


Figure 4. Section 4 of the return line, the flow direction is from right to left.

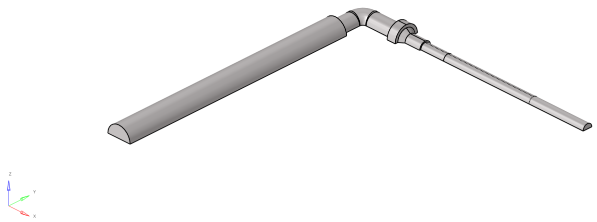


Figure 5. Section 5 (outlet) of the return line, the flow direction is from right to left.

4.2 Meshing

Tetrahedral elements were primarily used to mesh the sections, ensuring the capture of their intricate geometries. However, on the inlet extensions where the flow was expected to be fully developed, hexcore elements were employed, effectively reducing the total element count. To obtain acceptable y^+-values and capture the boundary layer, inflation layers were utilized. Furthermore, body of influence sizing was applied to refine the mesh in areas where significant gradients were anticipated, like sudden expansions. Figure 7 illustrates the general features common to all the meshes generated, including quad- and tri-surface meshes, the body of influence sizing around vital flow features, and inflation layers.

To verify the meshes, three different mesh sizes were evaluated for each section, and the solution results were compared. All mesh verification simulations were conducted using the fluid Dowcal 10 (Dow 2023) at 20°C with an inlet mass flow of 0.4kg/s. This resulted in a density of 1082 [kg/m^3] and a dynamic viscosity of 0.005kg/ms, as shown in Figure 8. Table 1 displays the element numbers employed for each section. Additionally, to ensure a smooth Eddy Viscosity Ratio (EVR) gradient from the wall to the bulk flow, the near-wall mesh's resolution was also evaluated.

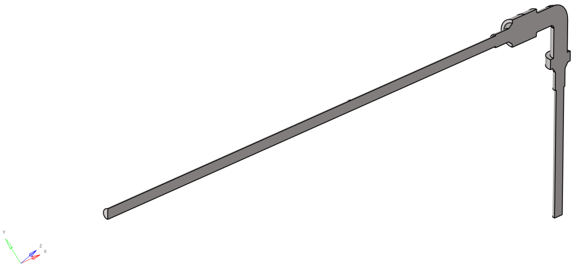
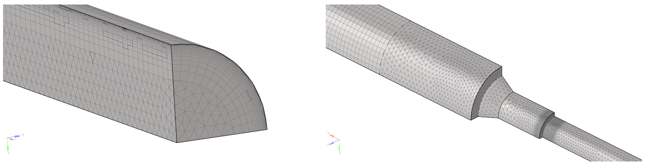


Figure 6. Test section with outlet extension with length of 50 pipe diameters, to investigate flow downstream of pipe fitting.

Section	No. of elements
1	204000
2	773000
3	489000
4	4180000
5	862000

Table 1. Number of elements in the selected mesh for each section.



(a) Mesh on section 1 showing inflation layers and bulk mesh.

(b) Surface mesh on section 1 showing a transition from quad to tri elements, and body of influence sizing.

Figure 7. Examples of the generated meshes.

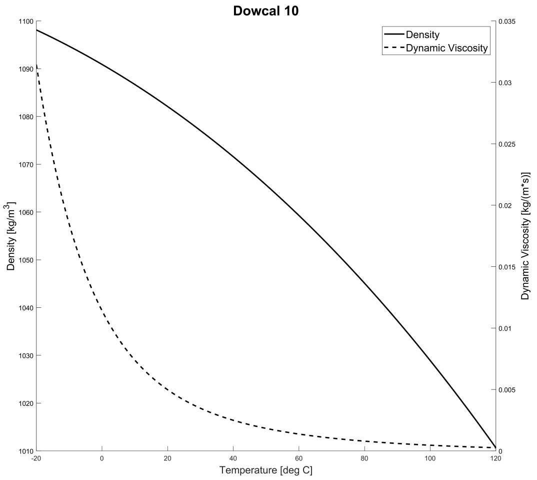


Figure 8. Density and dynamic viscosity as functions of temperature for Dowcal 10.

4.2.1 Solver Setup

Altair AcuSolve utilizes an implicit FEM-based solver with steady-state time-stepping. The turbulence model $k-\omega$ SST was chosen for the simulations, because of its good performance near walls, in adverse pressure gradients, and in separated flow. The boundary conditions for all sections were similar. At the inlet, the mass flow was specified, while the outlet was set to zero gauge pressure. The turbulence parameters at the inlet were automatically calculated. The pipe walls were given a wall roughness height of $2e-5$m, the same as in the Dymola model. Symmetry was applied to the symmetry planes if available. Heat transfer through the walls of the pipe was neglected. The material was set to the water-glycol mixture Dowcal 10, the same fluid used in the Modelica model. The density and dynamic viscosity variations for Dowcal 10 as functions of temperature are shown in Figure 8. In order to ensure convergence of the solutions, residuals, and monitor points throughout the domains were checked. AcuSolve also uses another convergence metric called the *Solution Ratio*, which measures the difference in results between iterations (*AcuSolve Residual Computation* 2013).

4.2.2 Post-processing

The pressure drop Δp for the different sections was investigated by taking the difference in total pressure between the inlet and outlet. For routing Section 5, the pressure drop was defined as the difference between the point of lowest total pressure and the outlet. This is because the sudden expansion in the fitting causes the pressure to decrease rapidly to a minimum in this routing location. Thus, the pressure drop is the pressure required to get it back to zero at the outlet. Another example of calculating the pressure drop for a sudden expansion can be found in (Roul and Dash 2009).

4.3 System Simulation models - romAI & Lookup Table (LUT)

In order to integrate the romAI models in a system simulation context, a new pipe model had to be created. The purpose of this pipe model would be to combine the romAI models for the pipe sections of the pipe where CFD simulations had been run, with handbook equations for the rest of the pipe. This pipe model would then be exported as a FMU for integration using the FMI standard. The requirements for the FMUs were to use the FMI 2.0 standard for co-simulation and to be license free, enabling wide-spread use throughout the organization. The new model was created in Altair Activate. Figure 9 shows an overview of the romAI-based model.

The inputs provided by the surrounding modeled system are mass flow, enthalpy, and pressure entering the pipe model. The mass flow and enthalpy outputs from the pipe were also set to equal the input values, implying continuity and ideal thermal insulation. In order to calculate the pressure drop in the pipe, the input enthalpy first need to be converted into fluid density and viscosity, which are the inputs required by both the romAI model and the

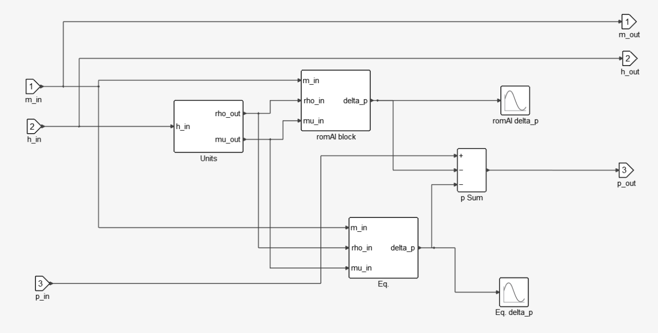


Figure 9. Overview of the romAI-based pipe model.

handbook equations. This conversion is conducted in block labelled `Units` in Figure 9. The equation block `Eq` receives the mass flow, density, and viscosity to calculate the pressure drop via Equation 3. The pressure drops calculated are then subtracted from the input pressure, to give the output pressure.

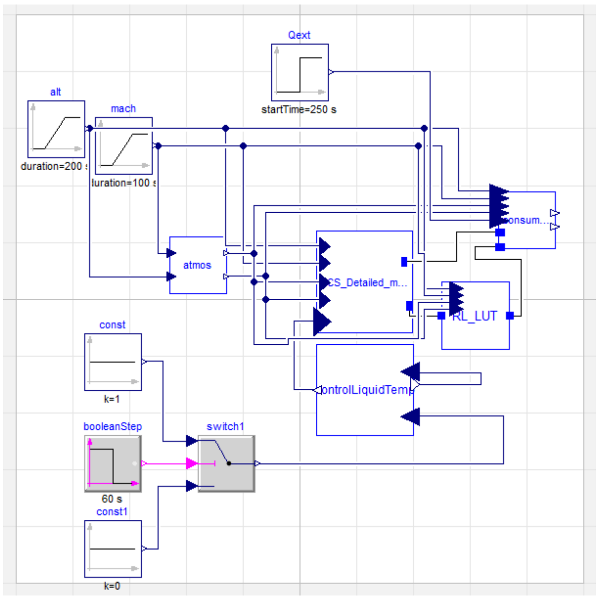


Figure 10. Overview of the new cooling system. RL_LUT is the new LUT-based pipe model (the romAI model was integrated in the same way).

The new FMU is integrated into the simulator by removing the `pipeB` component from the legacy model, and then inserting the new FMU as shown in Figure 10. The complete system is then re-packaged as a SSP file together with the necessary SSV data etc.

The simulation of the updated simulator was performed using the OMSimulator tool (Ochel et al. 2019), executed through dedicated Python scripts. The altitude, Mach number, and consumer heat load boundary conditions were varied over time to simulate hypothetical flight missions. The results from the updated simulator were then compared to the legacy version.

5 Results

This section presents the outcomes of applying the technique outlined in the preceding section. The produced ROMs and the output data from the CFD simulations are both included. The method for CFD-based system identification is offered as a last step.

5.1 Design of experiments

The flow cases were selected in order to give the maximum coverage of the pipe systems Operational Domain (OD). The coverage was defined using the modified nearest-neighbor coverage metric described in (Atamturktur et al. 2015). A reduced metric value implies an increase in OD coverage. Figure 11 shows the operating points of the pipe system with regard to mass flow and fluid temperature. These operating points were calculated while using a specific pump connected to the cooling system. The outlier at $\dot{m} = 0.36$kg/s and $T = 40$°C is a result of the simulation not reaching steady-state. In order to make the model more generalized (e.g., be applicable for evaluating different pump alternatives), a rectangular operational domain was created by connecting the maximum and minimum values in Figure 11. This gave the OD the limits of $0.25 < \dot{m} < 0.38$kg/s, and $-20 < T < 120$°C. Between one and 15 points (flow cases) were placed in the OD using Complex-RF optimization (Krus and J. Andersson 2003), striving to minimize the modified nearest-neighbor coverage metric. Figure 12 shows the change in coverage for an increasing number of points. A lower nearest neighbor coverage metric η_c indicates better coverage of the domain. Based on the result, ten cases were selected, see Figure 13. An additional ten randomly selected cases were run to use as validation for the created FMU. All 20 cases are shown in Table 2.

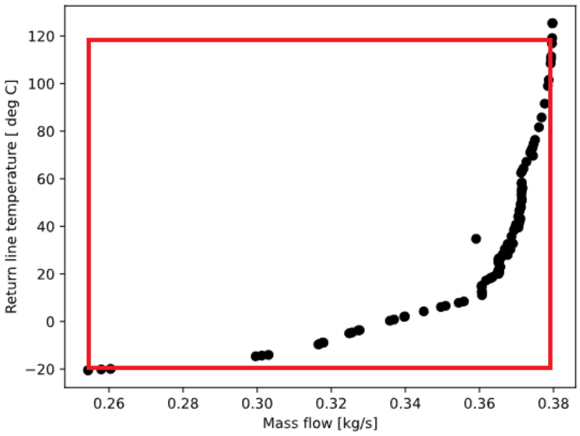


Figure 11. Operational domain of the pipe system. Black points indicate the operating points for a specific pump. The red rectangle is the created operational domain.

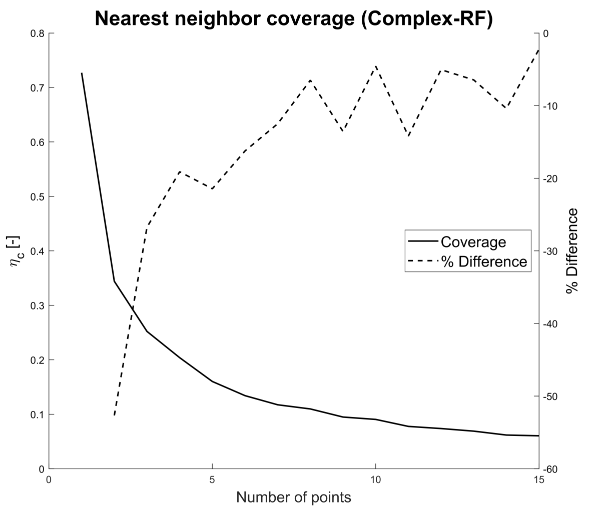


Figure 12. Nearest neighbor coverage for the operational domain. A lower η_c indicates better coverage.

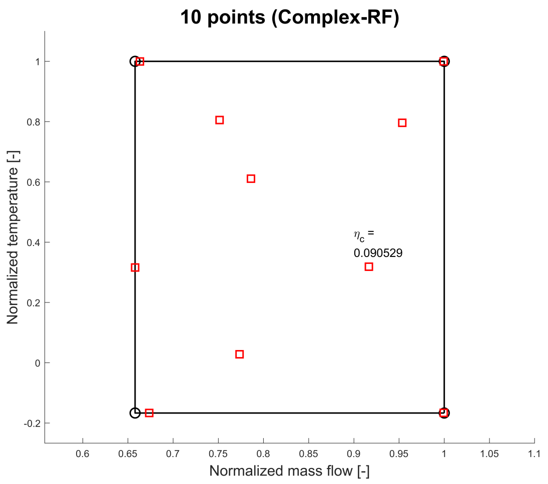


Figure 13. The 10 cases selected for the steady state model creation. The mass flow and temperature have been normalized with their maximum values (0.38 [kg/s] and 120 [°C] respectively).

	Case	$\dot{m}_{norm}$ [-]	T_{norm} [-]	$\dot{m}$ $[kg/s]$	T $[degC]$	ρ $[kg/m^3]$	μ $[kg/ms]$
Input	1	0.999	1.000	0.380	119.99	1010.66	0.000257
	2	0.663	1.000	0.252	119.94	1010.70	0.000257
	3	0.954	0.796	0.362	95.53	1032.69	0.000519
	4	0.658	0.316	0.250	37.93	1072.74	0.00266
	5	0.774	0.028	0.294	3.38	1089.52	0.00980
	6	0.786	0.611	0.299	73.30	1050.05	0.000948
	7	0.754	0.805	0.287	96.63	1031.76	0.000504
	8	0.674	-0.167	0.256	-19.98	1098.11	0.03141
	9	0.999	-0.166	0.380	-19.93	1098.09	0.03133
	10	0.917	0.319	0.348	38.26	1072.56	0.002633
Validation	11	-	-	0.309	116	1014.47	0.00029
	12	-	-	0.334	96	1032.30	0.000513
	13	-	-	0.268	80	1045.08	0.000791
	14	-	-	0.274	28	1078.09	0.003719
	15	-	-	0.295	42	1070.42	0.002337
	16	-	-	0.356	1	1090.50	0.010907
	17	-	-	0.378	-13	1095.77	0.021543
	18	-	-	0.256	31	1076.52	0.003351
	19	-	-	0.328	115	1015.41	0.000299
	20	-	-	0.340	56	1061.88	0.001534

Table 2. Selected flow cases. Cases 1-10 were selected to optimize coverage to be used as input data for the ROM creation. Cases 11-20 were selected randomly for validation purposes.

	Pressure drop [Pa]				
Case	Section 1	Section 2	Section 3	Section 4	Section 5
1	13575	42584	32194	130447	15789
2	6589	19670	15101	59767	6970
3	14008	43059	32784	127623	14006
4	8615	28097	21591	78081	6441
5	15240	53233	40537	142794	8808
6	10555	32886	25349	94778	9353
7	9067	28032	21388	82144	8771
8	20130	73221	55077	181600	6854
9	34975	129844	98110	330255	15023
10	15748	50652	38909	143317	15023
11	9703	29390	22914	89460	10400
12	12025	37156	28283	110235	11907
13	8450	26382	20227	76801	7598
14	10816	35672	27361	98710	7709
15	11413	36773	28254	103944	8935
16	21884	75769	57738	204107	12985
17	29551	108363	82113	281642	14814
18	9362	30791	23622	85180	6683
19	10880	33029	25044	100190	11755
20	14126	44520	34192	128187	12058

Table 3. Steady-state CFD results for the pressure drop through each section.

5.2 CFD Outcome

Table 3 shows the outcomes for all the steady-state CFD cases run for each pipe segment. Cases 1-10 are the cases that were utilized to make the ROMs, while cases 11-20 were utilized for validation. From the outcomes, it may be presumed that the cases with the lowest temperatures and highest mass flows give the highest pressure drop. This demonstrates that it is the friction along the line walls that contributes most to the losses. For more detailed of CFD results, see (Lindqvist 2022).

5.3 Created ROMs

Figure 14 shows the calculated pressure drops for the model of routing Section 1 created using romAI, compared to the validation CFD data. The blue line shows the results for the ROM, while the green line is the results from the validation data. Figure 15 shows the Δp output from the romAI and 8x8 lookup table run for a test case with the mass flow and enthalpy inputs as sine curves. The lookup table result differs at most between $2-3$kPa from the romAI model. Additional system identification results can be found in (Lindqvist 2022).

5.4 System Simulations

The pressure loss for the different return line pipe models over a range of mass flow and temperature values is shown in Figure 16. The models all have a similar pressure drop when the temperature is around 20 °C. As the temperature decreases, the pressure drops for the romAI and LUT models increase. The Modelica pressure drop decreases until Re $<$ 2000, when it too starts to increase. For the low-temperature flows, the romAI and LUT models consistently give a higher pressure drop than the cur-

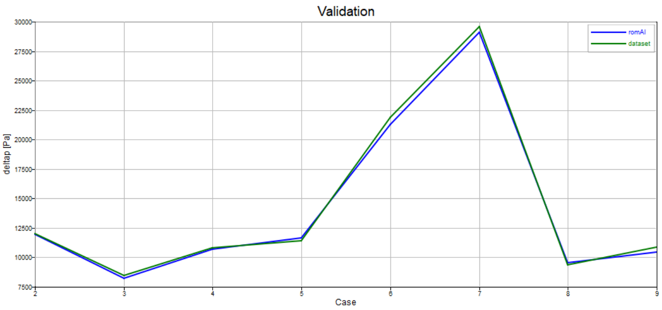


Figure 14. Comparison against validation data for Section 1.

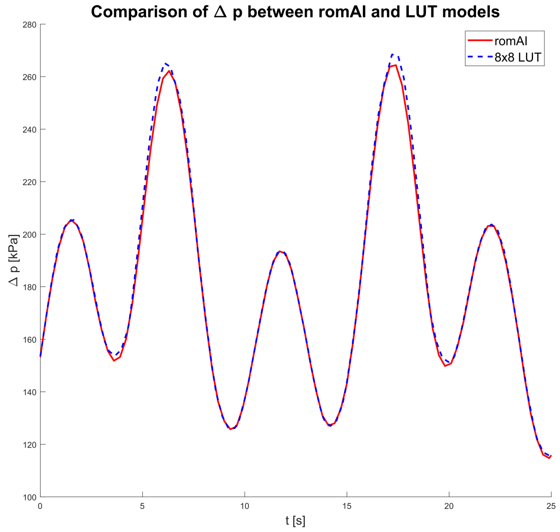


Figure 15. Comparison between 8x8 LUT and romAI model.

rent model. The difference in the romAI and LUT responses to the changing flow conditions is because the LUT model has been implemented using dynamical model components, while the romAI model contains no dynamics. Thus, the romAI model's response is instant. It should be noted that the temperature here goes lower than the lower limit of the operational domain of the created ROMs. Figure 16 mainly shows the similarity between the romAI and LUT results and the behavior of the Modelica pipe model at low Re.

6 Discussion and Conclusions

The work presented herein aims to fill a gap in the industry, enabling the adaption of model detail to the required level in model and simulator development. The presented use-case entails the incorporation of information of high-fidelity analysis using CFD in the systems' simulation domain efficiently. However, a number of different application areas exist. The work demonstrates an efficient means of producing Reduced Order Models from high-fidelity data, and such surrogates have a wide range of applications. One concrete example, to be exploited in the European Defense Fund Project-NEUMANN (European Defence Fund 2022), relates to adapting models for applicability in the model-based design of energy management control strategies in the aerospace domain.

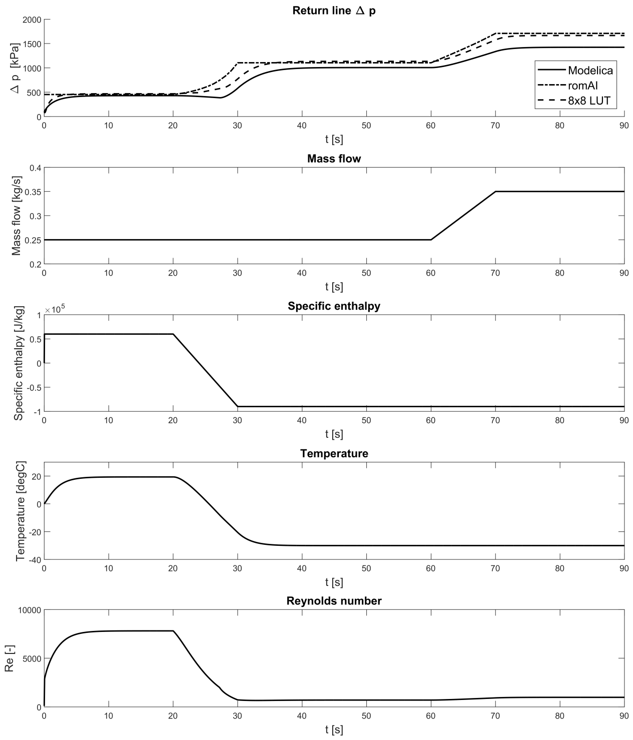


Figure 16. Comparison between current Modelica, romAI, and LUT models.

References

AcuSolve Residual Computation (2013). Altair University.

Altair (2023-06-13). *Altair AcuSolve: General Purpose Fluids and Thermal Simulation*. URL: https://www.altair.de/Product/AcuSolve/ (visited on 2023-06-13).

Anderson, John D. (1995). *Computational Fluid Dynamics: The Basics with Applications*. 1st ed. McGraw-Hill.

Andersson, Henric (2012). "Variability and Customization of Simulator Products : A Product Line Approach in Model Based Systems Engineering". PhD thesis. Linköping University, Machine Design, p. 83. ISBN: 978-91-7519-963-4.

Atamturktur, Sez et al. (2015). "Defining coverage of an operational domain using a modified nearest-neighbor metric". In: *Mechanical Systems and Signal Processing* 50, pp. 349–361.

Bathe, Klaus J. (2014). *Finite Element Procedures*. 2nd ed. K.J. Bathe, Watertown, MA.

Chen, Sheng, Stephen A Billings, and PM Grant (1990). "Nonlinear system identification using neural networks". In: *International journal of control* 51.6, pp. 1191–1214.

Donea, Jean and Huerta, Antonio (2003). *Finite Element Methods for Flow Problems*. 1st ed. Wiley.

Dow (2023-08-14). *Dowcal 100 Heat Transfer Fluids*. URL: https://www.dow.com/en-us/pdp.dowcal-100-heat-transfer-fluid.244104z.html#overview (visited on 2023-06-13).

Dymola User Manual (2016-09). Dassault Systèmes AB.

Eek, Magnus, Hampus Gavel, and Johan Ölvander (2017). "Definition and implementation of a method for uncertainty aggregation in component-based system simulation models". In:

Journal of Verification, Validation and Uncertainty Quantification 2.1.

Ellner, Stephen P and John Guckenheimer (2011). *Dynamic models in biology*. Princeton University Press.

European Defence Fund (2022). *NEUMANN-Novel Energy and propUlsion systeMs for Air dominance, SELECTED PROJECTS EUROPEAN DEFENCE FUND (EDF) 2021*. URL: https://defence-industry-space.ec.europa.eu/system/files/2023-01/Factsheet_EDF21_NEUMANN_0.pdf (visited on 2023-08-03).

Fernandez, Benito, Alexander G Parlos, and Wei Kang Tsai (1990). "Nonlinear dynamic system identification using artificial neural networks (ANNs)". In: *1990 IJCNN international joint conference on neural networks*. IEEE, pp. 133–141.

FMI Development Group (2020-12-15). *Functional Mock-up Interface for Model Exchange and Co-Simulation*. Report 2.0.2.

Fontes, Ed (2018-11). *FEM vs. FVM*.

Fritzson, Peter (2003). *Principles of Object Oriented Modeling and Simulation with Modelica*. 1st ed. Wiley.

Fritzson, Peter et al. (2005-12). "The OpenModelica Modeling, Simulation, and Software Development Environment". In: *Simulation News Europe* 44, pp. 8–16. ISSN: 0929-2268.

Hällqvist, Robert (2023). *On the Realization of Credible Simulations in Aircraft Development : Efficient and Independent Validation Enabled by Automation*. Linköping Studies in Science and Technology. Dissertations: 2279. Department of Management and Engineering, Linköping University Electronic Press. ISBN: 9789179295981.

Hällqvist, Robert, Raghu Chaitanya Munjulury, et al. (2021). "Engineering Domain Interoperability Using the System Structure and Parameterization (SSP) Standard". In: *Proceeding of the 14th International Modelica Conference*.

Hällqvist, Robert, Raghu Chaitanya Munjulury, et al. (2022). "Realizing Interoperability between MBSE Domains in Aircraft System Development". In: *Electronics* 11.18. ISSN: 2079-9292. URL: https://www.mdpi.com/2079-9292/11/18/2901.

Hällqvist, Robert, Johan Naeser, et al. (2022-09-04). "Heterogenous System Modelling in Support of Incremental Development". In: *Proceedings of the 33rd Congress of the International Council of the Aeronautical Sciences*. Stockholm, Sweden.

Haykin, Simon (1999). *Neural Networks: A Comprehensive Foundation*. 2nd ed. Prentice Hall.

International Standards Organization (ISO) (2016-03). *Industrial automation systems and integration, Product data representation and exchange*. Part 21: Implementation methods: Clear text encoding of the exchange structure.

Krus, Petter and Johan Andersson (2003). "Optimizing optimization for design optimization". In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 37009, pp. 951–960.

Lanza, R. et al. (2018). "Relating structureal test and FEA data with STEP AP209". In: *Advances in Engineering Software* 127.2019, pp. 96–105. DOI: 10.1016/j.advengsoft.2018.08.005.

Lind, Ingela and Henric Andersson (2011). "Model Based Systems Engineering for Aircraft Systems – How does Modelica Based Tools Fit?" In: *Proceedings of the 8th Modelica Conference*. Dresden, Germany.

Lindqvist, Olle (2022). "A Method of CFD-based System Identification for Standardized Co-Simulation." In.

Ljung, Lennart (1999a). *System Identification: Theory for the User*. 2nd ed. Prentice Hall. ISBN: 0-13-656695-2.

Ljung, Lennart (1999b). *System identification: theory for the user*. 2nd ed. Prentice Hall. ISBN: 0136566952.

Ljung, Lennart and Torkel Glad (2003). *Modellbygge och Simulering*. 2nd ed. Studentlitteratur. ISBN: 9789144024431.

Miller, Donald (1990). *Internal Flow Systems*. Cranfield, Bedford: BHRA (Information Services. ISBN: 978-0956200204.

Miller, Donald S. (1990). *Internal Flow Systems*. 2nd ed. Miller Innovations. ISBN: 978-0-9562002-0-4.

Modelica Association (2019-03-05). *System Structure and Parameterization*. Report 1.0.

Modelica Association (2023-03-09). *Modelica-A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.6*. URL: https://specification.modelica.org/maint/3.6/MLS.pdf.

Ochel, Lennart et al. (2019-03-04). "OMSimulator – Integrated FMI and TLM-based Co-simulation with Composite Model Editing and SSP". In: *Proceeding of the 13th International Modelica Conference*. DOI: 10.3384/ecp1915769.

Parlos, A et al. (1991). "Recurrent multilayer perceptron for nonlinear system identification". In: *IJCNN-91-Seattle International Joint Conference on Neural Networks*. Vol. 2. IEEE, pp. 537–540.

Roul, Manmatha K and Sukanta K Dash (2009). "Pressure drop caused by two-phase flow of oil/water emulsions through sudden expansions and contractions: a computational approach". In: *International Journal of Numerical Methods for Heat & Fluid Flow*.

Saab Group (2023). *Gripen E program Update 2021*. URL: https://www.saab.com/newsroom/stories/2021/june/gripen-e-program-update-2021 (visited on 2023-08-14).

Sjöberg, J. et al. (1995). "Nonlinear Black-Box Modeling in System Identification: a Unified Overview". In: *Automatica* 31, pp. 1691–1724.

Steinkellner, Sören (2011). "Aircraft Vehicle Systems Modeling and Simulation under Uncertainty". Licentiate thesis. Linköping University, Division of Machine Design. ISBN: 9789173931366.

Versteeg, H K and W Malalasekera (2007). *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. 2nd ed. Pearson. ISBN: 9780131274983.

Distributed Parameter Pneumatics

Felix Fischer[1] Katharina Schmitz[1]

[1]RWTH Aachen University – Institute for
Fluid Power Drives and Systems (ifas), Germany, {f.fischer,katharina.schmitz}@ifas.rwth-aachen.de

Abstract

Pneumatics is a branch of engineering that deals with the use of pressurized air or gases to create mechanical motion. It involves the study and application of systems and components such as air compressors, valves, cylinders, and actuators to control and transmit power through the use of compressed air. For highly dynamic events in pneumatic systems, such as fast switching processes in automation technology, lumped-parameter simulation is not sufficient to correctly calculate the pressure build-up in pipes. The propagation and reflections of different pressure waves and refraction waves cannot be accounted for by the zero-dimensional models provided by the `Modelica.Fluid` library. Therefore, this paper presents a method for calculating such events using the finite volume method. The library presented in this paper uses Godunov's scheme and an arbitrary Riemann solver and gas model to calculate the time evolution inside 1D or 2D discretized pneumatic components, as well as systems composed of these components.

Keywords: pneumatics, simulation, partial differential equation, distributed parameters, library

1 Introduction

The *Distributed Parameters Pneumatics* library allows for the calculating of transient events in pneumatic systems composed of pipes, valves, open and closed endings as well as connecting components such as T-connectors. These components are described in partial differential equations. These kinds of models consider the spatial distribution of parameters, such as temperature, density, and pressure; they are called *distributed parameter* models. This paper describes the theoretical foundation for the library as well as the core details of its implementation.

The theoretic background of the library discussed in this paper is introduced in section 2. The details of the implementation using Modelica is presented in section 3. The components of the library are validated using the analytical results of Sod-like tests, see section 4, as well as using experiments in section 5.

1.1 Motivation

While the *Modelica fluid* library is a suitable tool for calculating slow pressure changes in fluid systems used in hydraulics, pneumatics, and process engineering, it cannot be used to describe the fast fluctuations in pneumatic systems in the build-up phase of increasing or decreasing pressure. This is important, for example, when analyzing the interaction of different fast-moving actuators in an automation system. Another example of an application outside of pneumatics where the description of highly dynamic movements of gases is essential is gas transport in process engineering.

`Modelica.Fluid` uses zero-dimensional components, described by a single set of ordinary differential equations. These equations calculate changes in the quantities pressure p, temperature T, and density ρ only in terms of time and not of space. Therefore, these components can only describe the overall change of the averaged quantities inside of them over time. These types of models are called *lumped parameter* systems.

This is especially relevant for directional components like T- or X-pieces. The ideal T-connectors implemented in *Fluid* do not distinguish between the different directions. Therefore, a pressure wave entering an X-piece would be transmitted immediately to all three other sides without any difference and without any time delay between them. In a real X-piece, most of the pressure wave propagates to the opposite side.

A possible solution for this problem is the discretization of a single pipe component into a set of smaller pipes, which are described by the same underlying ordinary differential equations as the primitive component. This approach is called the finite volume method for solving partial differential equations. *Modelica Fluid* provides a discretized model with `Modelica.Fluid.Pipes.DynamicPipe`.

1.2 Sod Test

The standard method for the evaluation and test of finite volume method for gas dynamics is the Sod Test. This test is a fictional shock tube experiment. At time $t = 0$, the left half of the tube contains an ideal gas at high pressure, whereas the right half contains gas at a lower pressure. Both halves have an equal diameter and are directly connected. Initially, the gas in the whole tube is at the same temperature.

While the original paper by Sod uses a certain set of fixed and dimensionless initial pressures p, densities ρ and velocities $v = 0$, the analytical solution is known and well studied for every chosen set of start values (Sod 1978). Thus, can be used for the verification of gas dynamics simulations. The set-up of a Sod-like test and the pressure and rarefaction waves occurring in this test can be seen in Figure 1.

The standard method of initialization *Modelica Fluid* uses the *system* model. It includes the initial pressures as well as the initial temperature and the initial velocities. The

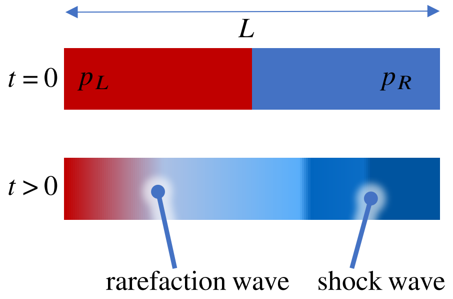


Figure 1. Set up and time evolution of the Sod test

density ρ can be calculated from the pressure and the temperature using the chosen gas law (e.g., the ideal gas law). In this work, the same method of initialization is used and therefore the systems are initialized using realistic values rather than abstract standard values. The start values are listed in Table 1. The Medium is Air. In the one-dimensional case, the diameter of the pipe has no influence on the results.

Table 1. Parameters of Sod-like test used in this work

Description	*Symbol*	*Start value*
Initial temperature	T	20 °C
Pressure in the left half	p_L	6 bar
Pressure in the right half	p_R	1 bar
Length of the pipe	L	2 m
Discretization	N	100
Modelica solver	DASSL	
Tolerance		1×10^{-6}

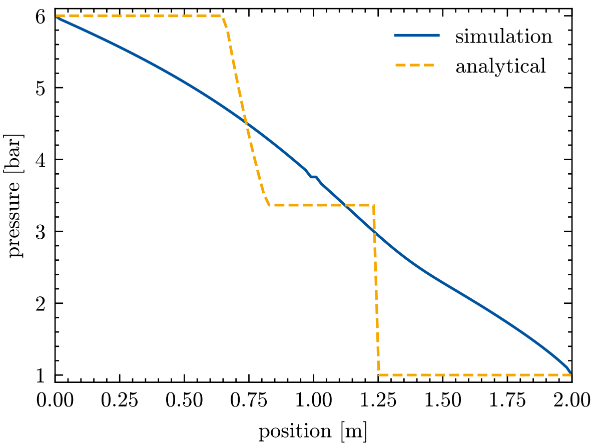


Figure 2. Sod test of staggered `DynamicPipe` at 1 ms

A sod test of the staggered model `DynamicPipe` from *Modelica Fluid* can be seen in Figure 2. There is a large discrepancy between the analytical result and the simulation (Isaac Backus 2017). Thus, this library is not suitable for highly dynamic gas simulations.

2 Theoretic Background

This section contains a brief introduction into the theoretic background of the simulation library presented in this work.

2.1 Euler Equations

This library is based on the conservative formulation of the Euler equations. The Euler equations are a simplified version of the Navier-Stokes equations. They describe the flow of a fluid without considering the thermal conductivity and viscosity.

Conservative formulations of partial differential equations guarantee that the amount of conserved quantities entering a finite volume is equal to the amount leaving it. Even if the discretization is very broad, the total and the local balances of these quantities are conserved. In non-conservative schemes, the quantities are not conserved and therefore there can be erroneous sources and sinks in the calculated solution. Therefore, the error in the conserved quantities is only acceptable, if the grid is fine enough (Ferziger, Perić, and Street 2020).

The conservative formulation of the one-dimensional Euler equations takes the following form (Toro 2009, p. 30):

$$0=\frac{\partial}{\partial t}\mathbf{U}+\nabla\mathbf{F}(\mathbf{U}) \tag{1}$$

$$\mathbf{U}:=(\rho,\rho u,E)^{\intercal} \tag{2}$$

$$\mathbf{F}:=(\rho u,\rho u^2+p,u(E+p))^{\intercal} \tag{3}$$

$$E:=\rho\left(\frac{1}{2}u^2+e\right) \tag{4}$$

In these equations, ρ is the density of the gas, u is the velocity and p is the absolute pressure. The total energy E (see Equation 4) can be separated into a kinetic and an internal component. The specific internal energy $e(p,\rho)$ is a function of the density and the absolute pressure. The expression for e is dependent on the gas model used in the simulation (Toro 2009). The entries of $\mathbf{U}$ shown in Equation 2 are called conserved variables, whereas p, u, and ρ are called primitive variables. The vector $\mathbf{F}$ shown in Equation 3 is called the flux vector and is a function of $\mathbf{U}$.

2.2 Riemann Problem

In finite volume methods, each discrete volume has only one value for $\mathbf{U}$ (see Equation 2) at each time step. Therefore, if two neighboring cells differ in any primitive variable, there will be a discontinuity in the initial condition of the partial differential equation. Such a partial differential equation with an initial condition that is constant everywhere except for a single discontinuous jump is called a *Riemann problem*.

The Riemann problem can be imagined as a miniature version of the Sod test at every cell boundary, see Figure 1. Analogous to the shock tube, the analytical solution is known and can be used to calculate the time evolution of every single finite volume.

The finite volume method used in this work is *Godunov's scheme*. In Godunov's scheme, the spatial derivative in Equation 1 is calculated numerically by evaluating the flux vector **F** at the intercell boundary. The one-dimensional Godunov's method is given by (Toro 2009, p. 177):

$$\frac{\partial}{\partial t}\mathbf{U}(t,x) = -\frac{1}{\Delta x}\left(\mathbf{F}_{i+\frac{1}{2}} - \mathbf{F}_{i-\frac{1}{2}}\right) \tag{5}$$

$$\mathbf{F}_{i\pm\frac{1}{2}} := \mathbf{F}\left(\mathbf{U}_{i\pm\frac{1}{2}}\right) \tag{6}$$

$$\mathbf{U}_{i\pm\frac{1}{2}} := \mathbf{U}\left(t, x_i \pm \frac{\Delta x}{2}\right) \tag{7}$$

$$\Delta x := x_{i+1} - x_i \tag{8}$$

Here, t is the continuous time and x_i the discrete position of a specific finite volume with index i. $x \pm \frac{\Delta x}{2}$ represents the position of the boundary between the volume at x_i and the neighboring volume at $x_i \pm \Delta x$, see Figure 3.

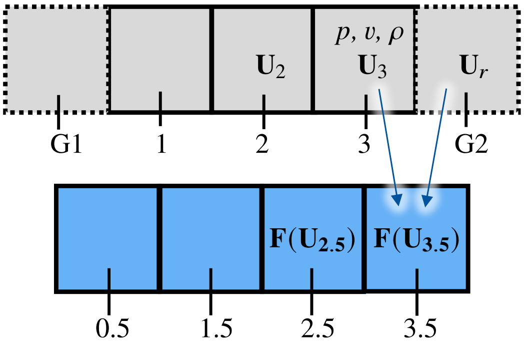


Figure 3. Discretizations used in Godunov's scheme, including ghost cells

The numerical method used to calculate $\mathbf{F}(\mathbf{U}(t,x \pm \Delta x))$ is called a *Riemann solver*. If the value **U** at the intercell boundary is calculated using the analytical solution, the solver is called an exact solver, otherwise, it is called an approximate solver. Compared to approximate Riemann solvers, exact solvers are more complex to construct and implement, and they are more computationally expensive. This work uses the approximate Harten-Lax-van-Leer-Contact (HLLC) and local Lax-Friedrichs solver (Toro 2016). These solvers take only the two neighboring volumes into account and are therefore called first-order methods, as shown in Equation 9.

$$\mathbf{F}_{i+\frac{1}{2}} = f(\mathbf{U}_i, \mathbf{U}_{i+1}) \tag{9}$$

The function f represents the first-order approximate Riemann solver. The time derivative in Equation 5 is solved by the Modelica solver.

2.3 Two Dimensional Formulation

To create a system that is more complicated than a single straight pipe with linear components connected to it, two-dimensional components are needed. These 2D components must be discretized in two dimensions, and a two-dimensional formulation of the finite volume method is needed. The main difference between the one- and two-dimensional algorithms is the additional component of the velocity in the y-direction which must be considered. Therefore, the state vector **U** needs an additional fourth component.

The conservative formulation of the two-dimensional Euler equations can be described as the following (Toro 2009, p. 104):

$$0 = \frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_x}{\partial x} + \frac{\partial \mathbf{F}_y}{\partial y} \tag{10}$$

$$\mathbf{U} := (\rho, \rho u_x, \rho u_y, E)^\intercal \tag{11}$$

$$\mathbf{F}_x := \left(\rho u_x, \rho u_x^2 + p, \rho u_x u_y, u_x(E+p)\right)^\intercal \tag{12}$$

$$\mathbf{F}_y := \left(\rho u_y, \rho u_x u_y, \rho u_y^2 + p, u_y(E+p)\right)^\intercal \tag{13}$$

Here, u_y is the gas velocity in y-direction (Schulz-Rinne, Collins, and Glaz 1993).

This differential equation is solved numerically using a two-dimensional Godunov's scheme, as described in Equation 5. In this case, the flux vectors at all four intercell boundaries have to be calculated using a 2D version of a Riemann solver. In this work, an adapted version of the local Lax-Friedrichs solver has been used for two 2D discretized meshes.

3 Implementation in Modelica

This section describes how this library is structured and how the finite volume method described in section 2 is implemented in Modelica.

3.1 General Structure

The general structure closely follows the structure of the `Modelica.Fluid` library, to keep the library as compatible with it as possible. The models make use of replaceable packages and inheritance to keep the code reusable. Due to this structure and the use of a sub-package, it is possible to quickly exchange the selected gas model or the Riemann solver. The internal structure of the models is based on the finite volume library presented by Sielemann (Sielemann 2012b).

The package `TransientPneumatics` is separated into three sub-packages:

1. `Parts`: This package contains the useable components in this library, such as pipes and valves. Furthermore, it contains the sub-package `Base` in which the one- or two-dimensional discretized pipe sections are located. These discretized sections are used by the regular parts as attributes.
2. `Solvers`: This package contains the different selectable Riemann-Solvers (`OneD`, `TwoD`) and gas models (`Media`).
3. `Systems`: This package contains the base template for new systems as well as simulation models for testing this library.

3.2 Components

The different components contain the initialization, the border conditions, and the connectors. The discretization and the solutions of the partial differential equations are delegated to the reusable models in the sub-package `TransientPneumatics.Parts.Base`. The general structure of a component is shown in Listing 1.

Listing 1. Basic structure of a component in this library

```
within TransientPneumatics.Parts;
model Part "Example of a part"
  import Modelica.Units.SI;
  replaceable package Medium
    = TransientPneumatics
        .Solvers.Media.Ideal.Example
    constrainedby TransientPneumatics
        .Solvers.Media.Base
    "Medium";
  replaceable package Solver
    = TransientPneumatics.Solvers.HLLC(
    redeclare package Medium = Medium)
    constrainedby
         TransientPneumatics.Solvers.OneD
    "Solver";
  Modelica
      .Fluid.Interfaces.FluidPort_a left(
    redeclare package Medium=Medium)
    "Left connector";
  Modelica
      .Fluid.Interfaces.FluidPort_b right(
    redeclare package Medium=Medium)
    "Right connector";
  parameter Integer N = 20
  "Number of finite volumes";
  public
    SI.AbsolutePressure pressure[N]
    "Absolute pressure in component";
  protected
    Base.PipeBase discretization(
      redeclare package Medium=Medium,
      redeclare package Solver=Solver,
      N=N)
      "Reusable base model";
end Part;
```

The base models in `Parts.Base` have the following structure:

Listing 2. Basic structure of the base model containing the logic of every pipe section

```
within TransientPneumatics.Parts.Base;
model PipeBase
  "Base class for pipe sections"
  import Modelica.Units.SI;
  replaceable package Medium
    = TransientPneumatics
        .Solvers.Media.Ideal.Example
    constrainedby TransientPneumatics
        .Solver.Media.Base "Medium";
  replaceable package Solver
    = TransientPneumatics.Solvers.HLLC(
    redeclare package Medium = Medium)
    constrainedby TransientPneumatics
        .Solvers.OneD "Solver";
  parameter Integer N = 20
    "Number of finite volumes";
  parameter SI.Length L = 1
    "Length of the pipe";
  parameter SI.Diameter diameter = 0.001
    "Diameter of pipe";
  protected
    final SI.Area cross_section
      = Modelica
          .Constants.pi * diameter^2 / 4
      "Cross section of pipe";
    final SI.Length delta_x = L/N;
    Real U[
        N, 3] "Conserved variable vectors";
  public
    Medium.ThermodynamicState
      volume_left, volume_right;
    SI.MassFlowRate
      flow_rate_left, flow_rate_right;
    // finite volumes at the border
    Medium.ThermodynamicState volume[N]
        "Records containing
             the primitive variables";
    SI.MassFlowRate flow_rate[N]
        "Mass flow rate in each volume";
    // kept public for initialization
  equation
    // the
        differential equations are put here
end PipeBase;
```

The primitive variables pressure and density are contained in the record `ThermodynamicState`. Additionally, the gas velocity u can be calculated using the mass flow rate $\dot{m}$ and the cross-section a:

$$u = \frac{\dot{m}}{\rho a} \tag{14}$$

Based on the primitive variables, the conserved variables $\mathbf{U}$ can be calculated according to Equation 2 in the function `Solver.primitiveToConserved`. The set of equations in Listing 2 is chosen to avoid the implementation of a function for the conversion from the conserved variables to the primitive variables, because there is no function to set a `thermodynamicState` record from the pressure and the density in the Modelica standard library (Sielemann 2012a). The function `Solver.monotoneFlux` calculates the flux vector as a function of the neighboring cells according to the solver, as seen in Equation 9.

The implementation of the sets of differential equations into Modelica is shown in Listing 3.

Listing 3. Set of differential equation to be solved in each pipe section

```
equation
for i in 1:N loop
  U[i] = Solver.primitiveToConserved(
    volume[i], flow_rate[i],
    cross_section);
end for;
// Godunov's method
// left border
der(U[1]) = 1 / delta_x * (
  Solver.monotoneFlux(
    volume_left, flow_rate_left,
    volume[1], flow_rate[1],
    cross_section)
  - Solver.monotoneFlux(
    volume[1], flow_rate[1],
    volume[2], flow_rate[2],
    cross_section));
// center section
for i in 2:N-1 loop
  der(U[i]) = 1 / delta_x * (
    Solver.monotoneFlux(
      volume[i-1], flow_rate[i-1],
      volume[i], flow_rate[i],
      cross_section)
    - Solver.monotoneFlux(
      volume[i], flow_rate[i],
      volume[i+1], flow_rate[i],
      cross_section));
end for;
// right border
// analogous to left border
```

3.2.1 2D Components

The two-dimensional base models are implemented analog to Listing 2; in this case, the conserved variables are stored in an $N \times M$ array: `U[N, M, 3]`. For 2D components, it needs to be differentiated between pipe section with round or rectangular cross-sections. When implementing rectangular cross-sections, it must be considered, that depending on the discretization the cross-section for flow in the x-direction can differ. In the case of circular cross-sections, the cross-section $a(y)$ for flow in the y-direction (perpendicular to the orientation of the pipe) depends on the y-position, as indicated in Figure 4.

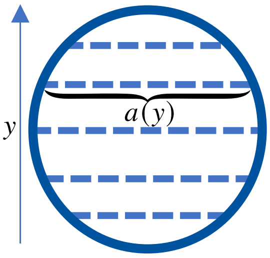


Figure 4. Cross-section of circular 2D pipes

3.3 Connectors

Each component is connected to the neighboring components using connectors. The connectors are the same connectors used in `Modelica.Fluid`. In this work, the ghost cell method is used to connect the connectors to the finite volumes. This approach is based on the work presented by López (López 2006). Due to the connectors being identical to the default ports, it is possible to connect the distributed parameter components developed in this work to the concentrated parameter components contained in `Modelica.Fluid`. This is demonstrated in Figure 5, where a graphical representation of a system in OpenModelica's OMEdit can be seen.

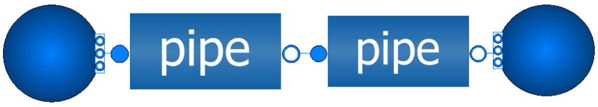


Figure 5. Two distributed parameter pipes connected to two concentrated parameter endings from `Modelica.Fluid` in OMEdit using connectors

The ghost cells on the left side of the base pipe are defined by the additional values `volume_left` and `flow_rate_left`. The variables for the right ghost cells are accordingly named `volume_right` and `flow_rate_right`. These values are related to the connectors by the following equations in the model of the parts shown in Listing 1:

Listing 4. Relation between the ghost cells and the connectors

```
discretization.volume_left
  = Medium.setState_ph(
    left.p, actualStream(left.h_outflow));
discretization.volume_right
  = Medium.setState_ph(
    right.p, actualStream(
      right.h_outflow));
discretization.flow_rate_left
  = left.m_flow;
discretization.flow_rate_right
  = - right.m_flow;
left.m_flow = discretizaion.flow_rate[1];
right.m_flow = -discretiation.flow_rate[N];
left.h_outflow = Medium.specificEnthalpy(
  discretization.volume[1]);
right.h_outflow = Medium.specificEnthalpy(
  discretization.volume[N])
```

In this work, the direction of flow is defined as left to right. A negative mass flow rate would therefore indicate a flow from right to left. Therefore, the sign for flow entering from the right connector must be inverted.

3.4 Pipe Endings

After enough time, every wave propagating in a finite pipe will hit either a closed or an open pipe ending. In both cases, the wave will be reflected on the ending and another wave will travel in the opposite direction. These open and closed boundary conditions are implemented into this library as separate models using the ghost cell approach shown in (Kratschun 2020).

3.4.1 Closed Ending

In the case of the reflection at a closed ending, the reflective boundary condition states that there is no velocity component in the x-direction at the boundary. This can be implemented using with the ghost cell approach by adding another volume with equal, but opposite velocity to the neighboring volume (LeVeque 2012, Chapter 7). Analogous to the connectors, a reflection on the left border of a part (Listing 1) can be implemented by adding the following lines:

Listing 5. Reflection on a closed ending at the left border of a part

```
discretization.volume_left
```

```
  = discretization.volume[1];
discretization.flow_rate_left
  = -discretization.flow_rate[1];
```

3.4.2 Open Ending

In the case of reflection at an open end, the reflective boundary condition states that there can be any velocity at the boundary, but there are conditions on the intensive variables on the volume next to the open end. The ghost cell at this end has the same velocity as the connected end, but depending on the direction of the flow, the temperature, and the pressure are fixed. Due to conservation of energy, the gas velocity at the end will overshoot compared to the wave packets inside the tube. Thus, the reflection at an open tube end can be intuitively understood to be caused by the discontinuity in acoustic impedance. The open end can be implemented by using the following additional lines:

Listing 6. Reflection on an open ending at the left border of a part

```
parameter SI.AbsolutePressure p_start
  = system.p_ambient
  "Environmental pressure";
parameter SI.Temperature T_start
  = system.T_ambient
  "Environmental temperature";
equation
  if discretization.flow_rate[1] > 0
  then
    discretization.volume_left
      = Medium.setState_pT(
        p_start, T_start);
  else
    discretization.volume_left
      = Medium.setState_pT(
        p_start, discretization.T[1]);
  end if;
  discretization.flow_rate_left
    = discretization.flow_rate[1];
```

3.5 Valve

Sod-like tests can be used for the analytical validation of a model. When trying to replicate a shock tube experiment on a test rig, a rapid 2/2-valve is needed, preferable switch times below 1 ms. But even in that case, the valve will have an influence on the propagation of the pressure waves, which cannot be neglected. Therefore, a model for valves is needed for the experimental validation of this library.

In this work, the valve is modeled as a plate orifice with changing diameter (Kratschun 2020). An orifice can be implemented into a finite volume method by calculating the flux vectors F based on the orifice equation (Schmitz 2022). The orifice equation is only valid for ideal gases, and it depends on the isentropic exponent κ.

$$\kappa = \frac{C_p}{C_v} \tag{15}$$

C_p is the heat capacity at constant pressure and C_v is the heat capacity at constant volume. For ideal gases, κ is constant. If the pressures are not very high, the ideal gas law is usually a good approximation. Therefore, κ is calculated as the fraction of heat capacities for all gas laws and is potentially not constant.

The orifice equation is only valid for stationary gases. To include dynamic effects, this work uses the total pressure instead of the static absolute pressure if the velocity is directed towards the orifice:

$$p_{\text{tot.}} = p + \rho u^2/2 \tag{16}$$

According to the orifice equation, gas will only ever flow from the side with higher pressure to the side with lower pressure. In the following, it is assumed that the pressure to the left of the orifice p_l is higher than the pressure to the right p_r. These equations can be implemented bidirectionally with Modelica by using conditional statements.

The velocity of the gas flowing through the orifice u_v increases when the pressure ratio Π decreases up to a critical ratio $\Pi_{\text{crit.}}$. For lower ratios, the velocity remains constant (Schmitz 2022, p. 45):

$$\Pi := \frac{p_r}{p_l} \tag{17}$$

$$\Pi_{\text{crit.}} = \left(\frac{2}{\kappa+1}\right)^{\frac{\kappa}{\kappa-1}} \tag{18}$$

$$u_v = c_d a_o \sqrt{p_l \rho_l} \cdot \begin{cases} \psi(\Pi_{\text{crit.}}) & \Pi \leq \Pi_{\text{crit.}} \\ \psi(\Pi) & \Pi > \Pi_{\text{crit.}} \end{cases} \tag{19}$$

$$\psi(\Pi) := \sqrt{\frac{2\kappa}{\kappa-1}\left(\Pi^{\frac{2}{\kappa}} - \Pi^{\frac{\kappa+1}{\kappa}}\right)} \tag{20}$$

Here, ρ_l is the pressure to the left of the orifice and a_o is the area of the opening of the orifice. c_d denotes the discharge coefficient and is dependent on the geometry of the orifice. The components of the new flux vector can thus be calculated depending on the condition of the flow:

$$F_1 = u_v \rho_l \tag{21}$$

$$F_2 = \begin{cases} p_l \Pi_{\text{crit.}} & \Pi \leq \Pi_{\text{crit.}} \\ p_r & \Pi > \Pi_{\text{crit.}} \end{cases} \tag{22}$$

$$F_3 = u_v \left(\rho_l \left(\frac{1}{2} u_v^2 + e(p_l, \rho_l)\right) + F_2\right) \tag{23}$$

In Equation 23, $e(p,\rho)$ denotes the specific internal energy as a function of the absolute pressure and the density of the gas.

When a pressure wave reaches a partially closed valve or an orifice from either direction, some fraction of the wave will be reflected at the orifice while another fraction will be transmitted according to the orifice equation. To include this effect into the model of the orifice, this work uses the combined flux approach presented in (Kratschun 2020):

$$\mathbf{F} = \frac{a_o}{a} \mathbf{F}_c + \left(1 - \frac{a_o}{a}\right) \mathbf{F}_o \tag{24}$$

Where $\mathbf{F}_c$ is the flux created by the reflection on a closed ending, (see subsection 3.4) and $\mathbf{F}_o$ orifice flux presented in this section.

3.6 Stability of the Simulation

The simulations using the staggered components in this library are stable. Usually, when using explicit schemes for hyperbolic differential equations, the Courant-Friedrichs-Lewy (CFL) condition must be obeyed as a necessary condition for stability (Courant, Friedrichs, and Lewy 1928, Page 61):

$$\frac{|u|\Delta t}{\Delta x} \leq 1 \tag{25}$$

To enforce or check this condition, it would be necessary to check the current simulation time step Δt in run time, which is not possible in Modelica. Therefore, the finite volume method should not be used with explicit Modelica solvers like the Euler method. In this work, the implicit DASSL solver has been used (Petzold 1982), which is unconditionally stable, although slower than typical explicit solvers. In case of an event in Modelica, DASSL will reduce to the Euler method. Therefore, the stability of the simulation can only be guaranteed by avoiding any events, or by using an a-stable solver. An example for an a-stable solver are implicit Runge-Kutta methods.

4 Analytical Validation

The different components, solvers, and gas laws are validated using the analytical results of the sod test described in subsection 1.2. A graphical representation of the system can be seen in Figure 5. In this system, two pipes with equal lengths are initialized with different pressures and connected using a connector in the center. The simulation parameters are listed in Table 1.

4.1 Solvers

A comparison of the different solvers can be seen in Figure 6. The results with both solvers follow the analytical solution. The simulated solutions are approaching the analytical result with increasing degree of discretization. Thus, the solvers have been implemented successfully.

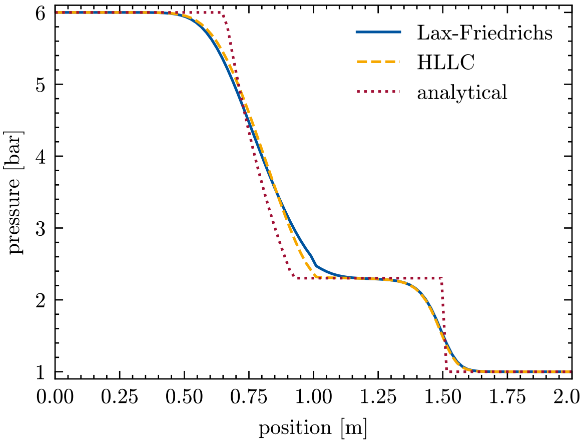


Figure 6. Comparison of Sod tests with the local Lax-Friedrichs Solver and the HLLC solver at 1 ms

The results by the HLLC solver are closer to the analytical solution at every point. The difference in calculation time between both solvers is negligible. For this reason, the HLLC solver is preferred for all simulations in this library.

4.2 Gas Laws

In this work, multiple gas laws have been implemented. Besides the ideal gas law, the Van der Waals gas law has been implemented as an example of a real gas law. The main difference between the different gas laws is the relation between the primitive variables and the specific internal energy $e(p,\rho)$. A comparison of the gas laws at high pressure can be seen in Figure 7. The starting pressure in the left half is 700 bar and on the right half 200 bar. At the initial conditions shown in Table 1, there is no visual difference between both solvers. For the high pressures shown in Figure 7, the results differ significantly. The analytical solution is only based on the ideal gas law and agrees therefore with the calculation based on the same law.

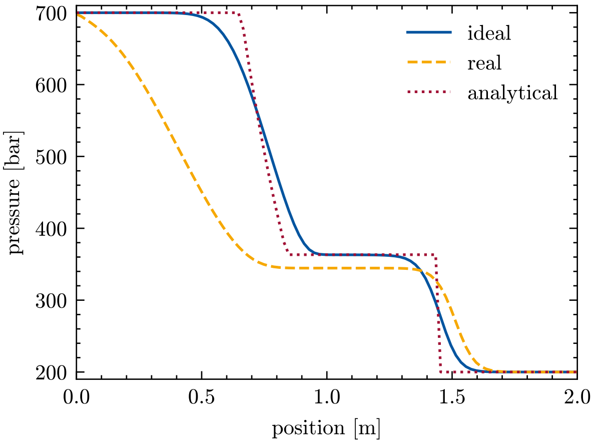


Figure 7. Comparison of Sod tests with real and ideal gas law at a very high-pressure drop at 1 ms

The default record `ThermodynamicState` contained in `Modelica.Fluid` contains the pressure and the temperature as basic variables. This leads to instabilities when solving a system using the DASSL solver of OpenModelica. The *Distributed Parameter Pneumatics* simulations library thus contains an alternative implementation of both `Media` and `ThermodynamicState` with the pressure and the density as internal variables, as seen in Listing 7. With this alternative record, the simulation runs flawlessly.

Listing 7. "Replacement for thermodynamic state record"

```
record ThermodynamicState
    "Custom thermodynamic state model"
  public
    SI.AbsolutePressure p(start = 1e5);
    SI.Density rho(start = 1.2);
end ThermodynamicState;
```

The two needed functions `Media.setState_ph` and `Media.setState_pT` return this record as a function of the pressure and the specific entropy or the pressure and the temperature respectively. In the case of the Van-der-Waals gas equation, there is no closed-form expression for these functions, and they must be computed iteratively. This increases the computation time, which is about a factor of 3 larger compared to the ideal gas law. Thus, for low pressures, which are the most relevant for pneumatic applications, it is preferred to use the ideal gas law with this library.

4.3 2D Components

A sod test for a two-dimensional component with a circular cross-section can be seen in Figure 8. The simulated pipe is circular, so the pressure curve is calculated in the x-direction, parallel to the pipe.

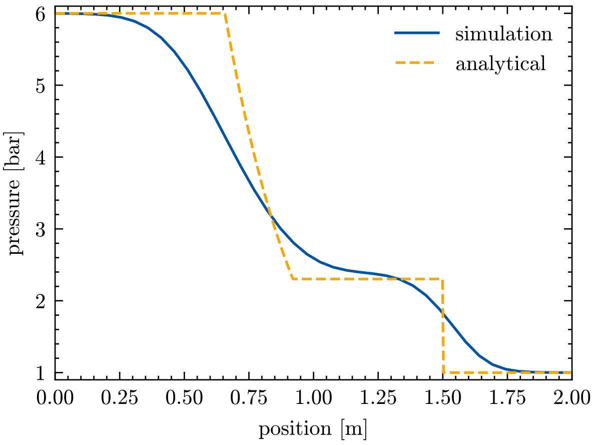


Figure 8. Sod test of a 2D pipe section with circular cross-section at 1 ms

The pipe is discretized into 20 segments in the x-direction and 7 segments in the y-direction. Due to the smaller discretization compared to the simulation shown in subsection 4.1, there is a greater difference between the simulation and the analytical results. In the two-dimensional case, the calculation time is multiple times larger compared to the one-dimensional case. Therefore, simulations with a very high two-dimensional discretization are not feasible on regular hardware. Another reason for the decreased accuracy in this simulation is the use of a 2D-Lax-Friedrichs solver compared to the HLLC solver used in the one-dimensional case. Nevertheless, the simulated result still approaches the analytical result with increasing refinement of the grid. Due to the changing cross-section, it is not possible to perform a useful Sod test in the y-direction.

In the case of the rectangular cross-section, the same validation has been performed in both the x-direction and y-direction and in both cases, the simulation reaches a similar agreement with the analytical results as for the circular cross-section.

The implementation of branched connectors like T-connectors is the principal use of two-dimensional components. These components can be constructed by the combination of several two-dimensional sections with either circular or rectangular cross-sections. The rectangular cross-sections are needed to attach a pipe section to another pipe with an angle of 90° because the pipes with round cross-section taper towards the edge in the y-direction.

5 Experimental Validation

The simulation library presented in this work has been experimentally validated using a test rig for shock test experiments. The experiment allows the validation of components in this library, even if there is no known analytical solution. This is especially relevant for the open and closed pipe endings, as well as for the valve.

A pneumatic circuit diagram of the test rig can be seen in Figure 9. The test rig uses two absolute pressure sensors, one in the high-pressure section and one in the low-pressure section, to measure the environmental pressure and the exact pressure of supplied air. Additionally, there are five highly sensitive piezoelectric relative pressure sensors along the pipe. These sensors measure the exact curve of the pressure at their position.

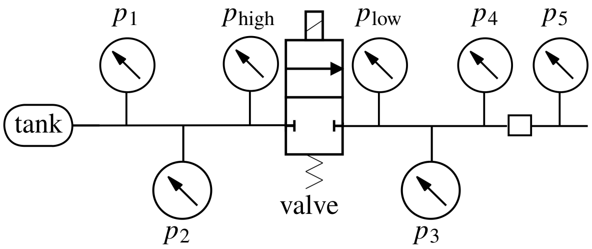


Figure 9. Pneumatic scheme of the test rig used in this work

The low-pressure half is connected to the high-pressure half by a quick-acting pneumatic valve. According to the manufacturer, the valve has a cycle time from closed to open to closed of about 1 ms. In the experiment discussed in this work, the valve starts in its closed state and is then opened and closed as fast as possible. No analytical solution exists for this system. The experimental conditions and the geometry of the pipes are listed in Table 2.

Table 2. Experimental Parameters

Variable	*Value*
Environmental temperature	21 °C
Environmental pressure	1.01 bar
Pressure in the tank	3.15 bar
Length of the left pipe	0.39 m
Length of the right pipe	1.985 m
Diameter of the pipe	0.7 cm
Distance between the valve and p_4	0.59 m
Fluid used in experiment	air

A model of the test rig shown in Figure 9 has been set up using `TransientPneumatics`. The ending of the tank to the left, as well as the connection to the environment to the right, have been modeled using open endings, which have been presented in subsection 3.4.

When implementing the valve, the following parameters can be set in the simulation, see subsection 3.5:

1. The maximal inner diameter of the orifice a_o
2. The discharge coefficient c_d

There is no obvious way to map these parameters to properties of the valve which can be found in its datasheet. The same is true for the signal chosen in Modelica, which controls the degree of opening of the valve. The simulation parameters selected for this model can be found in Table 3.

Table 3. Simulation Parameters

Variable	*Value*
Discharge coefficient c_d	0.68
Inner diameter a_o	5.7 mm
Valve signal	trapezoidal
Opening time	0.4 ms
Open time	0.2 ms
Closing time	0.4 ms
Discretization of the left pipe	40
Discretization of the right pipe	110
Solver	DASSL
Tolerance	1×10^{-6}

A comparison between the measured and the simulated pressure curve at the fourth pressure sensor p_4 can be seen in Figure 10.

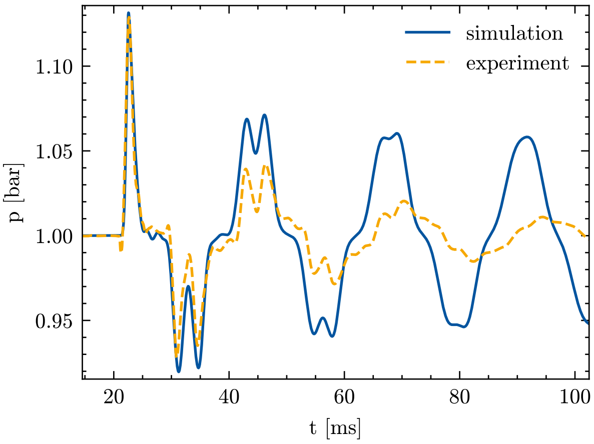


Figure 10. Comparison between the experiment and the simulation shown in Figure 9 measured at sensor p_4

The general shape of the first five double peaks match each other, and therefore it can be concluded, that the reflection at the open ending has been successfully implemented. In the last two peaks, the experiment shows a large dilation of the reflected pressure waves, which is not represented by the simulation. In the experiment, the amplitude of the oscillation decreases at a larger rate compared to the simulation. This is probably due to wall friction and, to a lesser degree, thermal transport through the walls of the pipe. Both effects cannot be yet simulated using this library.

A frequency analysis of the experiment and the simulation is depicted in Figure 11. The first peak corresponds to the principal eigenfrequency of the air column in the right pipe of the test rig. The fundamental tone of a tube of air, which is open at one and closed at the other, can be found approximately by the following equation (Rienstra and Hirschberg 2004):

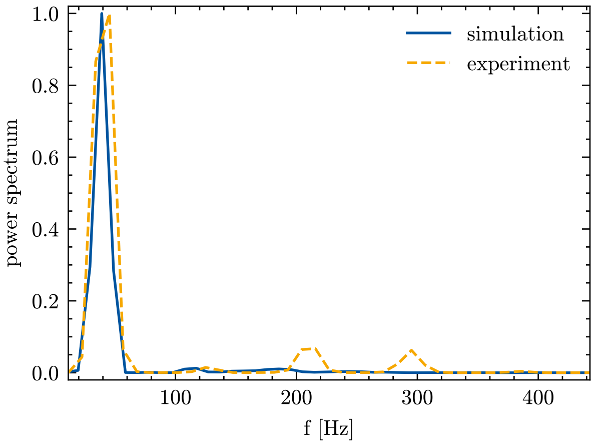


Figure 11. Comparison between the experiment and the simulation of the spectral power spectrum of the signal shown in Figure 10

$$\nu = \frac{V}{4L} \tag{26}$$

Where ν is the frequency of the fundamental tone, $V \approx 343\,\mathrm{m\,s^{-1}}$ is the speed of sound and L is the length of the pipe. This equation yields about 43.2 Hz for this experiment, which is in good agreement with the first peaks seen in Figure 11.

There are secondary peaks at higher frequencies seen in the experimental data, which are not present in the simulation. These peaks are probably due to partial reflection at the drilling holes of the sensors or at short cross-section jumps at the screw fittings. The frequency for two reflections at a closed ending is:

$$\nu = \frac{V}{2L} \tag{27}$$

According to this formula, the frequency for a reflection between the valve and p_4 is approximately 291 Hz; the distance between the valve and the connector in between $p4$ and $p5$ (see Figure 9) is 1.82 m which corresponds to 209 Hz.

Considering the assumed simplifications, this library has been successfully validated experimentally.

6 Summary, Evaluation, and Outlook

In this section, this paper is concluded by a summary of the results, as well as a critical evaluation of the presented method.

The compilation and simulation time for systems with two-dimensional components is considerable. Therefore, this library cannot be used for complex systems with many two-dimensional connections. The presented method allows calculating events in systems with arbitrary gas laws. Therefore, this library can be helpful for calculating gas transport in process engineering. Such systems are smaller and have a simpler topology compared to pneumatic systems, which allows performing the simulation in a short time.

Realistic systems will contain cross-section jumps and elastic tubes, which still need to be implemented. The presented method can further be improved by including wall friction and thermal conduction. It is possible to include wall friction, thermal conduction, and elastic tubing by adding a source term to the differential equations shown in Listing 2. A cross-section jump can be included by using the analytical solution of the Riemann problem at a cross-section jump, similar to the orifice presented in subsection 3.5 (Han, Hantke, and Warnecke 2012).

The library presented in this work allows for the calculation of highly dynamic transient events in pneumatic systems consisting of pipes, valves, connections, and open or closed endings. The parts have been validated using the analytic solution of the Sod tests, as well as experimentally. In general, there is a good agreement between the simulation and the analytical and experimental results.

Acknowledgements

The authors thank the Research Association for Fluid Power of the German Engineering Federation VDMA for its financial support. Special gratitude is expressed to the participating companies and their representatives in the accompanying industrial committee for their advisory and technical support.

Nomenclature

Sym.	Meaning	Sym.	Meaning
a	cross-section	a_o	opening of orifice
C_p, C_v	heat capacity	c_d	disch. coefficient
E	total energy	e	sp. internal energy
$\mathbf{F}$	flux vector	f	general function
L	length of pipe	$\dot{m}$	mass flow rate
p	abs. pressure	t	time
$\mathbf{U}$	conserved var.	u	gas velocity
V	speed of sound	x, y	position
κ	heat cap. ratio	ν	frequency
Π	pressure ratio	ρ	density of gas
ψ	orifice function		

References

Courant, R., K. Friedrichs, and H. Lewy (1928). “Über die partiellen Differenzengleichungen der mathematischen Physik”. In: *Mathematische Annalen* 100.1, pp. 32–74. ISSN: 0025-5831. DOI: 10.1007/BF01448839.

Ferziger, Joel H., Milovan Perić, and Robert L. Street (2020). *Computational methods for fluid dynamics*. Fourth edition. Cham: Springer. ISBN: 978-3-319-99693-6.

Han, Ee, Maren Hantke, and Gerald Warnecke (2012). “Exact Riemann Solutions to Compressible Euler Equations in Ducts with discontinuous Cross-Section”. In: *Journal of Hyperbolic Differential Equations* 09.03, pp. 403–449. ISSN: 0219-8916. DOI: 10.1142/S0219891612500130.

Isaac Backus (2017). *Sod shock tube calculator*. URL: https://github.com/ibackus/sod-shocktube (visited on 2023-05-16).

Kratschun, Filipp (2020). “Transient Pneumatic System Simulation: Transiente Simulation pneumatischer Systeme”. Dissertation. Aachen: RWTH. DOI: 10.2370/9783844073997.

LeVeque, Randall J. (2012). *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press. ISBN: 9780521810876. DOI: 10.1017/CBO9780511791253.

López, José Díaz (2006). “Shock Wave Modeling for Modelica.Fluid Library using Oscillation-free Logarithmic Reconstruction”. In: *Proceedings of the 5th International MODELICA Conference*. Ed. by Martin Otter and Dirk Zimmer. Vienna: The Modelica Association, pp. 641–649. URL: https://modelica.org/events/modelica2006/Proceedings/sessions/Session6b2.pdf.

Petzold, Linda R (1982). *Description of DASSL: a differential/algebraic system solver*. Tech. rep. Sandia National Labs., Livermore, CA (USA).

Rienstra, Sjoerd W and Avraham Hirschberg (2004). “An introduction to acoustics”. In: *Eindhoven University of Technology* 18, p. 19. URL: https://ayeghsoti.com/wp-content/uploads/2019/09/boek.pdf (visited on 2023-06-20).

Schmitz, Katharina (2022). *Fluidtechnik – Systeme und Komponenten*. 1. Auflage. Düren: Shaker Verlag. ISBN: 978-3-8440-8801-4.

Schulz-Rinne, Carsten W., James P. Collins, and Harland M. Glaz (1993). “Numerical Solution of the Riemann Problem for Two-Dimensional Gas Dynamics”. In: *SIAM Journal on Scientific Computing* 14.6, pp. 1394–1414. ISSN: 1064-8275. DOI: 10.1137/0914082.

Sielemann, Michael (2012a). “Device-Oriented Modeling and Simulation in Aircraft Energy Systems Design”. Dissertation. Hamburg: TU Hamburg-Harburg. DOI: 10.15480/882.1111.

Sielemann, Michael (2012b). “High-Speed Compressible Flow and Gas Dynamics”. In: *Proceedings of the 9th International MODELICA Conference*. Ed. by Martin Otter and Dirk Zimmer. Linköping Electronic Conference Proceedings. Linköping: Linköping University Electronic Press, pp. 81–100. ISBN: 978-91-7519-826-2. DOI: 10.3384/ecp1207681.

Sod, Gary A. (1978). “A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws”. In: *Journal of Computational Physics* 27.1, pp. 1–31. ISSN: 00219991. DOI: 10.1016/0021-9991(78)90023-2.

Toro, Elewterio F. (2009). *Riemann solvers and numerical methods for fluid dynamics: A practical introduction*. 3. ed. Berlin and Heidelberg: Springer. ISBN: 978-3-540-49834-6. URL: http://www.loc.gov/catdir/enhancements/fy1109/2009921818-d.html.

Toro, Elewterio F. (2016). “The Riemann Problem”. In: *Handbook of Numerical Methods for Hyperbolic Problems - Basic and Fundamental Issues*. Vol. 17. Handbook of Numerical Analysis. Elsevier, pp. 19–54. ISBN: 9780444637895. DOI: 10.1016/bs.hna.2016.09.015.

A renewable heat plant Modelica library for dynamic optimization with Optimica

Thomas Colin de Verdiere[1,2] Sylvain Serra[2] Sabine Sochard[2] Pierre Garcia[1]
Pierre Delmas[1] Jean-Michel Reneaume[2]

[1]Newheat, France, {thomas.colindeverdiere,pierre.garcia,pierre.delmas}@newheat.fr

[2]Universite de Pau et des Pays de l'Adour, E2S UPPA, LaTEP, Pau, France,
{sylvain.serra,sabine.sochard,jean-michel.reneaume}@univ-pau.fr

Abstract

Almost half of the energy consumed globally is under the form of heat, produced mainly through fossil fuels. Switching to using renewable energy instead is a real challenge. Combining renewable thermal energy with thermal storage is a complex system to operate. To harness the full potential of thermal plants, advanced control strategies need to be implemented. Dynamic real-time optimization (DRTO) seems promising to fine tune controller setpoints of plants. The goal of our study is to ultimately enable DRTO by using Optimica because of its ease of use and Modelica's modularity. This paper presents a Modelica library developed to first perform offline dynamic optimization with Optimica, and would ultimately be used in a DRTO strategy. The library enables to model a renewable thermal plant composed of solar thermals, heat pumps and thermal storages. The model of each subcomponent has been validated. Initial dynamic optimizations of plant operation give promising results.

Keywords: Dynamic optimization, Thermal plant, Optimica, Solar thermal, Heat pump

1 Introduction

Completely replacing fossil fuels for heat production by renewable energy requires combining several heat sources. The integration of thermal renewable energy in conversion/storage/distribution systems, in particular, solar thermal energy, faces several obstacles. Renewable thermal systems have varying thermal inertias and are prone to environmental disturbances. These mixed sources have to be well controlled to maximize performance and competitiveness of the global system.

Combining both a mix of renewable thermal energy sources and large thermal energy storage is quite innovative and has never been implemented in France in large scale heat plants. Such innovative systems have successfully been built and operated in a few Northern European countries. The main learning highlighted by those first plants is the need to fine tune system setpoints in real time to both maximize energy output and minimize operational cost. In China, a study demonstrated that in specific cases of large-scale solar heating systems integrated with water-to-water heat pumps and pit storage, a 16% decrease of the leverage cost of heat (LCOH) can be achieved by operation optimization (Zhang et al. 2023).

Since renewable heat plants are sensitive to external variation such as changing weather, electricity cost, and heat demand, real-time optimization (RTO) is a method that seems perfectly adapted to them. Although it is widely used in the field of chemical engineering, using RTO in the field of energy is quite recent. Adding dynamic optimization would allow to handle the thermal inertias of the system. Then, Dynamic Real-Time Optimization (DRTO) seems particularly adapted to the management of thermal installations combining several renewable production sources.

In our global study, DRTO combined with Non-Linear Programming (NLP) will be used to optimize the operation of a multi-energy heat plant. The optimization will be performed using Modelica language and the Optimica Compiler Toolkit (OCT) from Modelon.

The objective of this optimization is to maximize the heat produced by the plant, while minimizing the operational cost (mainly electric consumption). Weather, electricity, and heat demand forecasts are included in the model. The optimization variables are temperature and power setpoints for each heat source.

The modelling work and initial dynamic optimizations are presented in this paper. DRTO theory and results won´t be discussed in detail in this paper.

1.1 Literature review

JModelica is an open-source platform (Åkesson et al. 2009) for numerically solving large-scale dynamic optimization problems of Modelica models. This tool evolved during the years, in particular by including CasADi. The actual framework JModelica.org (Magnusson and Åkesson 2015) became Optimica Compiler Toolkit (OCT) under a Modelon license in 2020.

JModelica.org is used since its development in many research works, with some in heat production optimization. Runvik et al. (2015) developed a short-term planning optimization for a district heating system solved in two steps, one MILP (Mixed Integer Linear Programming) and one NLP using JModelica. More recently, Rohde et al. (2020) used JModelica to dynamically optimize control setpoints for an integrated heating and cooling system, including heat pump and solar thermal, with thermal energy storages.

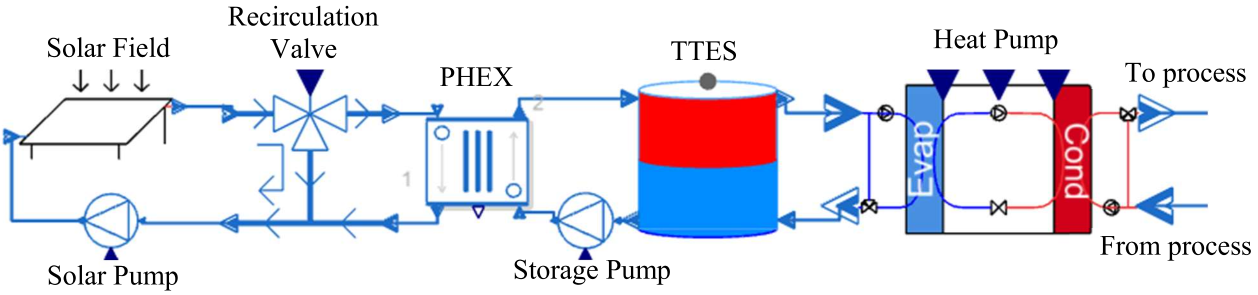


Figure 1. Typical heat plant assembly

Other works used Modelica language to model the heat system but chose other ways to optimize the heat production. Liu et al. (2018) used Dymola to model the system composed of CO_2 heat pump coupled with hot and cold thermal storages, and genetic algorithm to maximize the efficiency of the system.

Dynamic optimization of heat plants including solar thermal and thermal storage using NLP has been also performed by Scolan et al. (2020). They optimized the design and the control of the plant using the optimization software GAMS for modeling and optimization. More recently, Untrau et al. (2023) proposed a DRTO of solar thermal plant including thermal storage also using GAMS, combined with a detailed simulation model of the real plant in Matlab.

1.2 Purpose of this work

Dynamic optimization with Optimica requires dynamic models of the thermal plant formulated with continuous equations; discontinuities would be hardly interpreted by the time discretization method (collocation method) used by OCT. The objective of this research work is to develop a library in Modelica compatible with Optimica to model a thermal plant composed of a solar thermal field, a heat pump, and a thermal energy storage as shown in Figure 1. Modelica libraries already exist and model some of the components of thermal plants, but either are not compatible with Optimica or have too complex fluid modeling, which hinders convergence of the optimization.

This library allows to model and dynamically optimize several plant layouts using the modularity of Modelica.

First, we will present the library and the models used for each component of the thermal plant, then we will discuss the simulation results, and finally present the optimization methodology and the first optimization results.

2 Library

The library is developed in Modelica language using Modelon Impact software.

Each main component of the library and associated controller will be detailed in the following sections. The controllers will be used to initialize the optimization problem as detailed in section 4.

2.1 Hydraulic representation and interfaces

Two different fluids will be used in this library, water on storage and process side, and a glycol-based water solution on solar side. Glycol-based water solution is used to avoid frosting inside the solar field during winter (in European countries).

To simplify modeling, the only hydraulic parameters considered in this library are temperature and mass flow with constant fluid properties. Density ρ and specific heat capacity Cp only vary of few percents over the operating temperature range (40 to 90°C). The main impact of those hypotheses is on the viscosity of the glycol-based water solution, which is varying a lot over the operating temperature range. To further simplify and ensure optimization convergence, pressure loss is not modeled here. This hypothesis leads to inexact electricity consumption of the pump on solar side. However, as the main electricity consumption is from the heat pump, around 15 times more than the solar pump in the plant studied (based on Newheat's proprietary plant design tool), it will only lead to a small error in the overall electricity consumption.

The Modelica connectors used in this library are propagating only temperature and mass flow of the fluid. The mass flow is generated by the pump and is then propagated through each component of the hydraulic loop.

2.2 Solar Field (SF)

The solar thermal collector is the equipment used to transform solar radiation into heat. The collectors modeled here are Flat Plate Collectors (FPC). They provide heat at low temperature (below 100°C). Figure 2 shows the main components of the FPC and how it works.

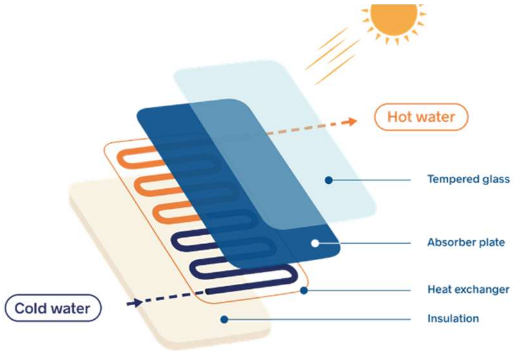


Figure 2. Flat plate collector

The solar field is usually composed by several parallel loops of collectors, and each loop is composed by several collectors in series. This layout is described in Figure 3.

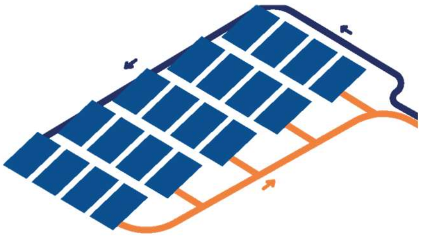


Figure 3. Solar field layout

2.2.1 Model

The model follows the standard ISO/FDIS 9806:2017 (International Standard 2017), using the quasi-dynamic equation of solar thermal collectors. This standard determines characteristic parameters for each collector (η_0, c_1, c_2 and c_5 in our model).

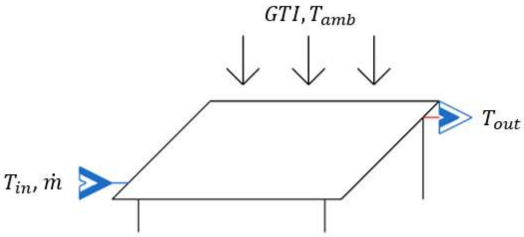


Figure 4. Solar field model

The whole solar field is modeled as an equivalent panel with $Area$ equal to the whole solar field area, assuming a uniform distribution between each panel loop of the solar field, whose power is $\dot{Q_{SF}}$.

The model in Figure 4 is described by equations (1)(2)(3), with GTI the global irradiance received by the collectors, $\dot{m}$ the mass flow inside the solar field, T_{amb} the ambient temperature, and T_{in} , T_{out} , T_m the inlet, outlet and mean temperatures of the solar field.

$$\dot{Q_{SF}} = Area\ (\eta_0\ GTI - c_1\ (T_m - T_{amb}) - c_2\ (T_m - T_{amb})^2 - c_5 \frac{dT_m}{dt}) \tag{1}$$

$$\dot{Q_{SF}} = \dot{m}\ Cp\ (T_{out} - T_{in}) \tag{2}$$

$$T_m = \frac{T_{in} + T_{out}}{2} \tag{3}$$

2.2.2 Control

In most solar thermal plants, the outlet temperature of the solar field is controlled to follow a temperature setpoint which could vary within the year. This temperature is controlled thanks to the mass flow inside the panels. Thus, the mass flow to be provided by the solar pump is calculated in the simulation to get the temperature setpoint at the outlet of the solar field by solving equations (1)(2)(3) without the differential term, as standardly done in solar thermal plants.

2.3 Plate Heat Exchanger (PHEX)

A heat exchanger is needed to transfer heat from the solar field loop (filled with glycol-based water) to the storage and supply loop (filled with water). PHEX are used in most solar thermal plants operating at low temperature (below 100°C).

2.3.1 Model

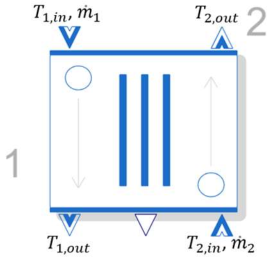


Figure 5. Plate Heat Exchanger model (icon from DLR ThermoFluid Stream Library)

PHEX in Figure 5 is modelled with constant efficiency (ε_{PHEX} between 0 and 1, a value of 0.9 is used to match Newheat's plants data) in equation (4) and is used to have the heat capacity flow ratio equals to 1 in equation(5).

$$\dot{Q} = \varepsilon_{PHEX}\ \dot{m}_1\ Cp_1\ (T_{1,in} - T_{2,in}) \tag{4}$$

$$R = \frac{\dot{m}_1\ Cp_1}{\dot{m}_2\ Cp_2} = 1 \tag{5}$$

The energy conservation gives those latest equations to get the outlet temperatures:

$$\dot{Q} = \dot{m}_1\ Cp_1\ (T_{1,in} - T_{1,out}) \tag{6}$$

$$\dot{Q} = \dot{m}_2\ Cp_2\ (T_{2,out} - T_{2,in}) \tag{7}$$

2.3.2 Control

The mass flow on side 2 will be computed to always respect the equality of the two heat capacity flows as in equation (5).

2.4 Tank thermal energy storage (TTES)

TTES is an essential element of solar thermal plants, it allows desynchronizing solar heat production and process heat demand, as the heat production depends on a fluctuating solar irradiance. During the charge phase the solar field provides heat to the tank (cold water from the bottom is warmed up by the solar field before coming back to the top of the tank). Conversely, during the discharge phase the tank provides heat to the process (hot water from the top of the tank transfers heat to cold water coming from the process).

The tank used in this work is an insulated water tank.

2.4.1 Model

Different tank models are available in the literature (Dumont et al. 2016). The model used in this work is a one-dimensional model called Multi-Node. Tank is vertically discretized in n layers (or nodes) with uniform temperature, considering a constant and incompressible volume of water in the tank (Figure 6).

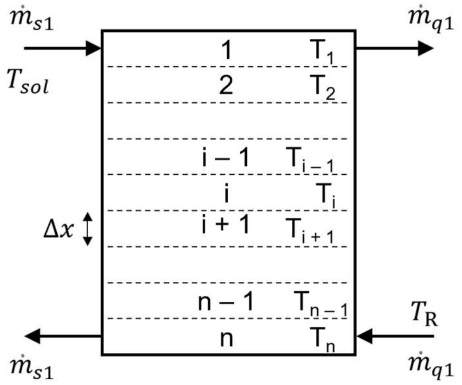


Figure 6. Spatial discretization of TTES (Scolan et al. 2020)

The equation governing the stratified thermal model for conduction and convection is the energy equation (8) (Hawlader et al. 1988).

$$\frac{\partial T(x,t)}{\partial t} + v\,\frac{\partial T(x,t)}{\partial x} = (\alpha + \epsilon_t)\,\frac{\partial^2 T(x,t)}{\partial x^2} + \frac{U\,P}{\rho\,C_p\,S}\,(T_{amb}(t) - T(x,t)) \tag{8}$$

Where $T(x,t)$ is the storage fluid temperature dependent on height x and time t, v the flow velocity, α the thermal diffusivity, ϵ_t the diffusion coefficient due to turbulent mixing caused by buoyancy effect, U the overall heat transfer coefficient, S the tank cross-sectional area, and associated P perimeter.

The tank is discretized into n nodes from equation (8) using a finite difference method (Scolan et al., 2020). Each layer is exchanging heat with its neighboring, spatial derivatives being then expressed by second order finite differences.

In the studied heat plant, solar heat and heat pump could lead to a temperature inversion in the tank (an upper layer which is colder than a lower one). It generates a mixing flows called buoyancy effect or plume entrainment (Hawlader et al. 1988). A model of buoyancy was implemented in Modelica IBPSA library (IBPSA 2013). Based on IBPSA model we implemented an equivalent model computing $\dot{m}_{inv,i}$ as a mixing mass flow between layer i and layer i-1 following equations (9)(10)(11).

$$T_{diff,i} = T_i - T_{i-1} \tag{9}$$

$$\dot{m}_{inv,i} = \begin{cases} z\,T_{diff,i}, & T_{diff(i)} \geq 0 \\ 0, & T_{diff(i)} < 0 \end{cases} \tag{10}$$

$$z = \frac{m_{layer}}{\tau.1K} \tag{11}$$

where τ is a time constant for mixing.

2.5 Water-to-water Heat Pump

The first heat pump used in this work is a water-to-water compression heat pump.

Heat pump could be used in several ways in a thermal plant. In this work, as shown Figure 1, the solar heat stored in the tank is used as cold source (evaporator side); the condenser side is warming up the cold water coming from the industrial process.

2.5.1 Model

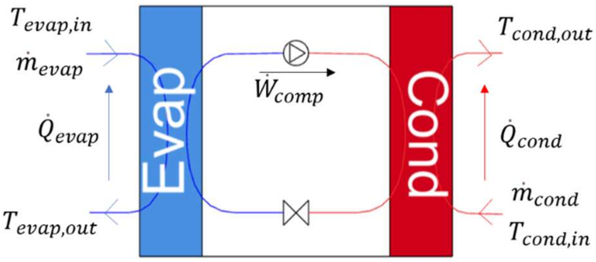


Figure 7. Heat pump model

To model the heat pump, we need a simplified model because detailed one would be too complex for optimization and prevent convergence. We decided to neither model the refrigerant fluid inside the heat pump nor its dynamics.

The heat pump is modeled using Coefficient Of Performance (COP) modeling with evaporator and condenser temperatures mapped on a datasheet. The heat pump can be used at partial load. We first assume that COP does not vary with the load of the heat pump.

- Energy conservation:

$$\dot{Q}_{cond} = \dot{Q}_{evap} + \dot{W}_{comp} \tag{12}$$

$$\dot{Q}_{cond} = \dot{m}_{cond}\,Cp_{cond}\,(T_{cond,out} - T_{cond,in}) \tag{13}$$

$$\dot{Q}_{evap} = \dot{m}_{evap}\,Cp_{evap}\,(T_{evap,in} - T_{evap,out}) \tag{14}$$

- COP modeling:

The COP is defined as:

$$COP = \frac{\dot{Q}_{cond}}{\dot{W}_{comp}} \tag{15}$$

We model the COP as a polynomial expression of temperature lift between evaporator and condenser. This method is used by Fischer et al. (2017) and Ruhnau et al. (2019).

$$COP = a_2\,\Delta T_{lift}^2 + a_1\,\Delta T_{lift} + a_0 \tag{16}$$

$$\Delta T_{lift} = T_{cond,out} - T_{evap,in} \tag{17}$$

Coefficients a_0, a_1 and a_2 are determined using the datasheet of the heat pump at full load.

- Compressor modeling:

The compressor is modeled with a constant efficiency η_{comp} and its available power $\dot{W}_{comp,max}$ depends on the temperature of the fluid inside him. Fischer et al. (2017)

propose to consider only evaporator temperature to determine the available power.

$$\eta_{comp} = \frac{\dot{W}_{comp}}{P_{elec,comp}} \quad (18)$$

$$\dot{W}_{comp,max} = b_1\, T_{evap,in} + b_0 \quad (19)$$

Coefficients b_0 and b_1 are determined using the datasheet of the heat pump at full load.

Partial load y_{pl} is defined as below:

$$y_{pl} = \frac{\dot{W}_{comp}}{\dot{W}_{comp,max}} \quad (20)$$

The electric power of the compressor is considered as an expenditure in the heat plant operation.

2.5.2 Control

To model the COP, we used specific working conditions. All operating points are usually given with a constant temperature difference between the inlet and outlet of evaporator and condenser. We also want to be able to control the evaporator and condenser temperatures.

To control the temperature difference, we need to add a pump, and to control the temperature, we need to add an ideal three-way valve, on each side as shown Figure 8. The valve position and the mass flow are ideally controlled to get the temperature setpoints (this means specification equations are added to the system to be solved).

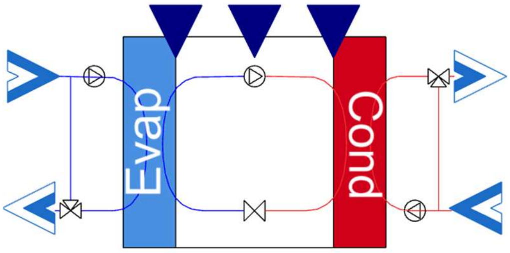


Figure 8. Heat pump with recirculation loop model

The heat pump is turned on when the cold source is warm enough (above the minimal temperature accepted at evaporator) and the warm source is cold enough (below the maximal temperature accepted at condenser). $T_{cond,out}$ and y_{pl} will be computed to follow heat demand temperature and mass flow.

2.6 Pump

The electricity consumption of the pumps will be considered as an expenditure in the heat plant operation. Pumps and heat pump are the main operating cost of the plant.

2.6.1 Model

The electricity consumption of a pump is determined by equation (21) where $P_{elec,max}$ is the electric power measured on the real pump at its maximal mass flow $\dot{m}_{max}$.

$$P_{elec} = P_{elec,max} \left(\frac{\dot{m}}{\dot{m}_{max}}\right)^3 \quad (21)$$

2.6.2 Control

The pump works as a mass flow generator, it directly provides the mass flow desired. It does not need specific control.

3 Simulation results

In previous section, all the models have been detailed. They need now to be validated with experimental results or datasheet, before simulating a full plant.

3.1 Validation

All models (except heat pump) are compared to measurement from solar thermal plants operated by Newheat. For solar field and heat exchangers we used data from Solthermalt, a plant owned by Kyotherm providing heat to a malthouse (Newheat 2020) in Issoudun, France. Tank model has been compared to the one build in Lactosol project, a solar plant providing heat to a whey powder production site (Newheat 2023) in Verdun, France.

Following sections show the result of comparison between the model and the measurements.

In figures below, time of 0.0 day corresponds to 12AM, and 0.5 day to 12PM.

3.1.1 Solar Field

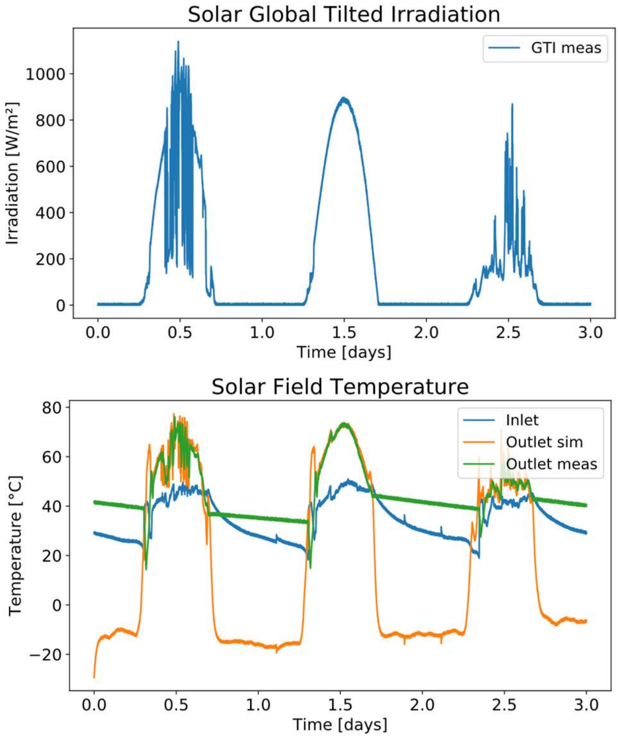


Figure 9. Solar field validation

Figure 9 shows the solar global tilted irradiation and the outlet temperature of a 5000 m² solar field during three days. Inlet temperature and solar irradiation measurements are coming from Solthermalt plant.

The model fits quite well to the measurements except during night periods. The inlet and outlet temperature sensors are in an insulated pipe inside a building; therefore, they do not decrease to the ambient temperature. In the model, the solar field mean temperature decreases to ambient temperature when there is no irradiation. As ambient temperature is around 10°C and measured inlet around 30°C (instead of 10°C if pipes were not insulated), the outlet is logically computed to -10°C to keep the mean of the inlet and outlet temperatures at 10°C. Since there is no mass flow during this period, heat produced by the solar field is not impacted.

3.1.2 Plate Heat Exchanger

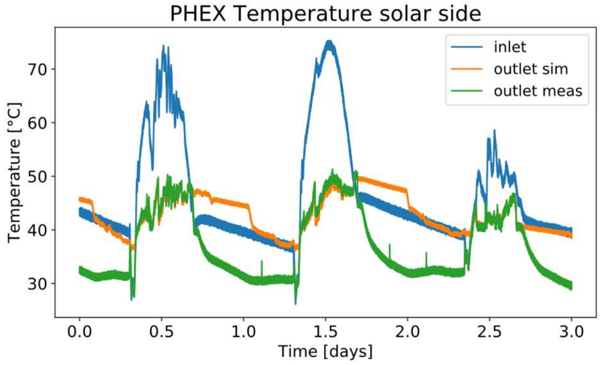


Figure 10. PHEX validation – solar side temperature

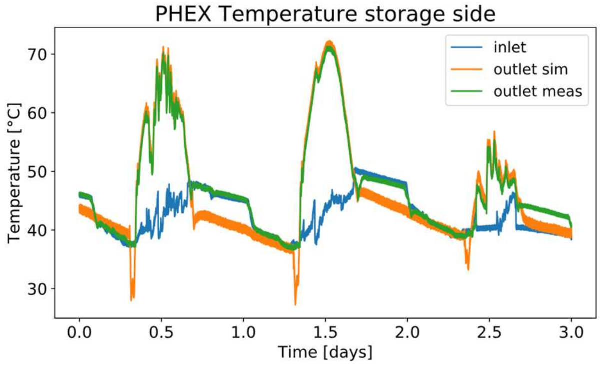


Figure 11. PHEX validation – storage side temperature

Figure 10 and Figure 11 show the inlet and outlet temperature of the plate heat exchanger between the solar loop and the storage. Measurements are coming from the same plant and the same days as for the validation of the solar field.

Same as previous section, the model fits well to the measurements except when the solar and storage pumps are off.

3.1.3 Tank thermal energy storage

Measurements are coming from Lactosol plant. A 3000 m^3 insulated water tank is installed in this solar plant to buffer the heat provided by the solar field and the consumption of the whey powder production site. This 12-meter-high tank is instrumented with 12 temperature sensors (about one every meter).

Two models are simulated and presented, one with 12 layers in Figure 12 and one with 60 layers in Figure 13. Temperature sensors are compared to the temperature at equivalent height in the simulation.

The measurement period is composed of a 9-hour charge phase and an 8-hour discharge phase separated by a 7-hour standby phase.

On the one hand, simulation fits better to measurement with 60 layers than with 12. A higher number of layers in the tank allows indeed higher temperature gradients which are needed to represent the thermocline zone inside the tank. But on the other hand, each layer is adding a state variable to the model and then increasing the size of the optimization problem.

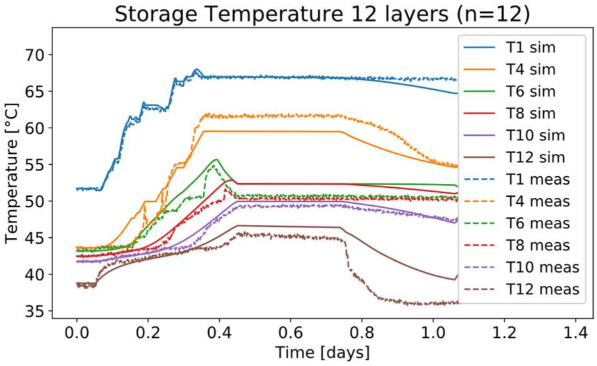


Figure 12. TTES validation – 12 layers

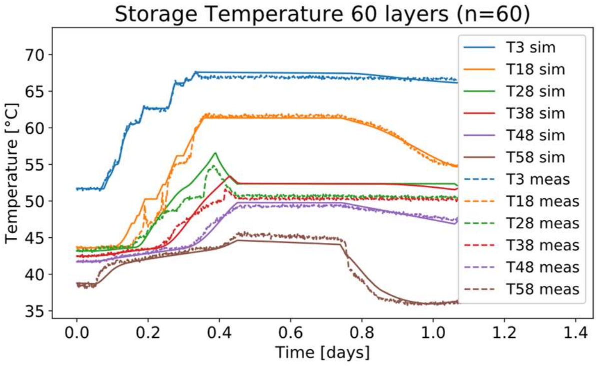


Figure 13. TTES validation – 60 layers

The number of layers will have to be selected carefully to model with a satisfactory accuracy without adding too much complexity to the optimization problem. For optimization we decided to set the number of layers at 10 to reduce the size of the optimization problem.

3.1.4 Heat pump

Figure 14 below compares the COP at full load between the model and the datasheet of an industrial heat pump (WWHS ER3b) made by Ochsner Energie Technik. Temperatures are given for the secondary medium (water) and not the primary medium (refrigerant).

The model fits well the datasheet for mean temperatures but leads to a 6.6% error for cold evaporator temperature.

The model overestimates a bit the COP, which results in an underestimation of the electricity consumption.

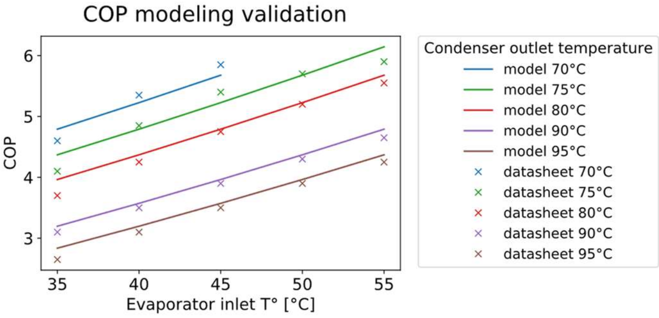


Figure 14. Heat pump COP validation

All component models are now validated separately. The next step is to simulate a full plant with controllers.

3.2 Full plant simulation

The simulated plant is a virtual plant composed of a solar field of 14248 m², a plate heat exchanger with a constant efficiency of 0.9, a 3000 m^3 tank, and a 3.1 MW heat pump. The layout of the plant is described in Figure 1. The heat pump is connected to the tank on the evaporator side while the condenser side is connected to the process.

The heat consumer is represented with a constant heat demand (constant return temperature, and constant mass flow and temperature setpoints). The plant is controlled by the expert rules defined in section 2.

The simulation runs over two spring days, the first one is cloudy and the second one sunny; Figure 15 represents the global irradiation of those two days. An optimization of the control of this plant will be performed on the same days in section 4.3.

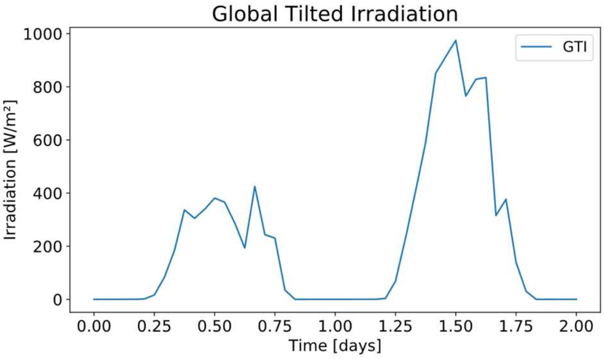


Figure 15. Full plant simulation - Global Tilted Irradiation

Inlet and outlet temperatures of the solar field are presented in Figure 16. The setpoint given to the solar field is 53°C. The outlet of the solar field is following well the setpoint except for the second day where the irradiation is too strong to limit the outlet temperature (because the solar pump is at its maximum speed as it can be seen in Figure 17).

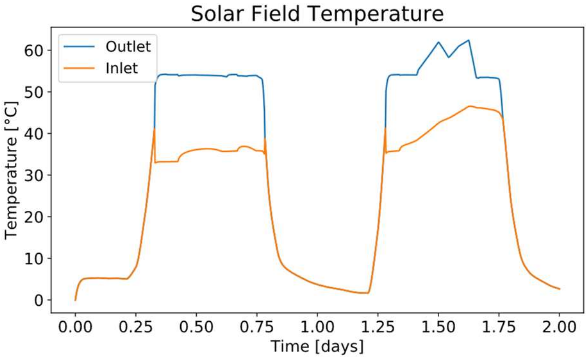


Figure 16. Full plant simulation - Solar field temperature

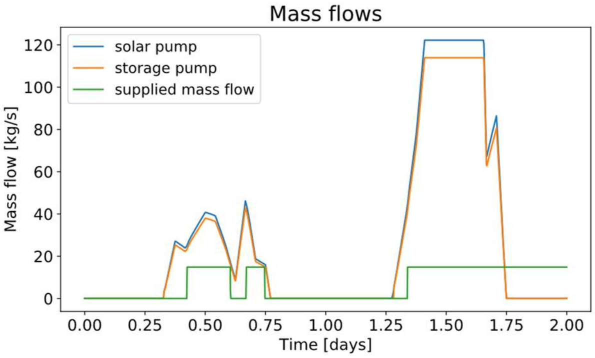


Figure 17. Full plant simulation - Mass flows

The tank is discretized in 10 layers (layer 1 is the top layer and layer 10 is the bottom layer). The first day is too cloudy to fill the tank completely, while the second day allows filling the tank at higher temperature (Figure 18). Evaporator outlet temperature is warmer than tank bottom temperature during the second day, which homogenizes the temperatures of layers from 3 to 10.

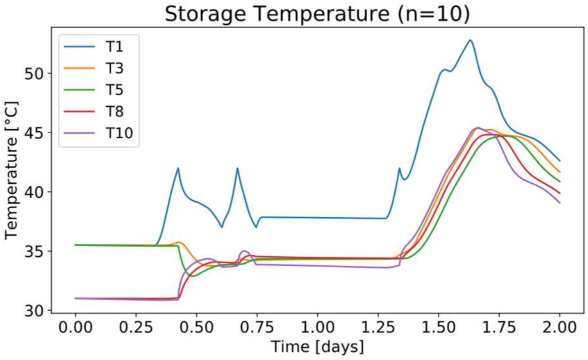


Figure 18. Full plant simulation - Storage temperatures

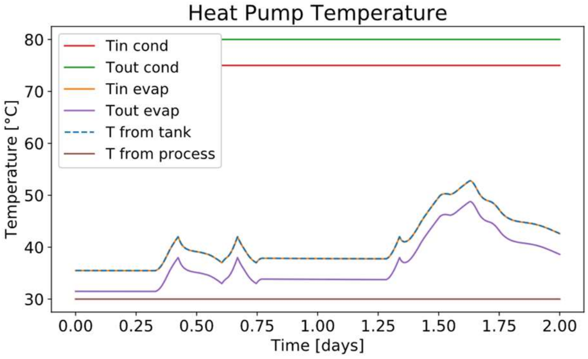


Figure 19. Full plant simulation - Heat pump temperatures

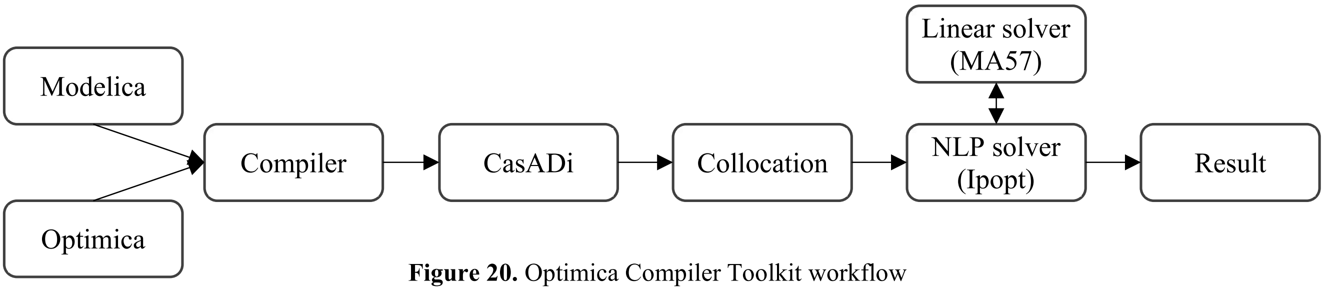


Figure 20. Optimica Compiler Toolkit workflow

Heat pump temperatures are shown in Figure 19. Heat pump is turned on when the top tank layer temperature exceeds the minimal temperature accepted at evaporator inlet. We can see in Figure 17 when the heat pump is on (supplied mass flow at 15kg/s). Finally, the evaporator temperature is controlled to be as high as possible while staying below its maximum (55°C).

The plant behaves as expected with validated models. We developed and validated a library to simulate renewable thermal plants. The next objective is now to optimize control variables and see how behave the models with optimization solver.

4 Optimization

Initial optimization presented in this paper is an offline optimization. It means that the optimizer is not yet connected to the real plant (or a highly detailed model). The goal is to try to optimize the plant in typical days and to define the ideal optimization sequences. Once the offline optimization is working well enough, it will be plugged to the real plant to try real-time optimization (optimization launched every hour considering the changes in the system states and the updated forecasts).

4.1 Tools

The tool used for optimization in this research work is Optimica Compiler Toolkit (OCT) under Modelon license for academic and commercial use. This tool is coming from JModelica.org which became OCT since 2020. Magnusson and Åkesson (2015) presented how JModelica.org is working. OCT workflow is described in Figure 20.

The first step is to describe the continuous models of the system to optimize in Modelica language. Optimica language (which is an extension of Modelica language) will be then used to describe the constraints and objectives of the optimization problem.

Modelica and Optimica models are transformed into optimization problem in the form of DAEs (Differential-Algebraic system of Equations), which is sent to CasADi, before being discretized via orthogonal collocation. Finally, the discretized optimization problem is sent to the solver (IPOPT) which will optimize the control variables of the system.

4.2 Methodology

The objective of optimization is to maximize heat supplied by the thermal plant, while minimizing operational costs. In this first optimization, this equates to maximizing global profit of the plant, electricity price assumed to be fixed. The control variables to be optimized are the outlet temperature of the solar field, evaporator outlet temperature, and the activation of the heat pump.

The optimization is composed of several stages detailed in Figure 21:

- An initial simulation of the plant is done with a control strategy, which would be implemented in a real plant (expert rules detailed in section 2). This simulation provides the optimizer an acceptable solution, which is the starting point of the optimizer.
- Several optimizations are done successively releasing degrees of freedom. Each result is initializing the next optimization. Last optimization result (with all degrees of freedom together) is then used to get optimal trajectories for several setpoints of the plant. This iterative approach was chosen to improve convergence of the optimization, otherwise too complex to be solved directly.

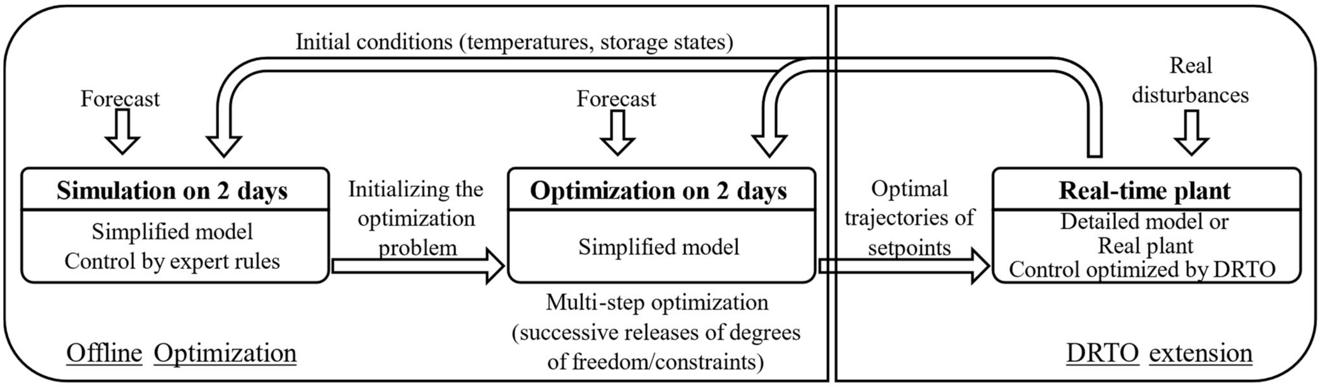

Initial conditions (temperatures, storage states)
Forecast
Forecast
Real disturbances
Simulation on 2 days
Simplified model
Control by expert rules
Initializing the optimization problem
Optimization on 2 days
Simplified model
Optimal trajectories of setpoints
Real-time plant
Detailed model or
Real plant
Control optimized by DRTO
Multi-step optimization (successive releases of degrees of freedom/constraints)
Offline Optimization
DRTO extension

Figure 21. Optimization methodology

- Finally, to perform DRTO, the controller of the real plant (or a highly detailed model) will use the optimal setpoints to operate the plant exposed to real disturbances. This part is not described in this paper.

4.3 First optimization results

In this section is presented the first optimization results using the developed Modelica library. The plant optimized is the one presented in section 3.2. Each figure below compares the standard control strategy used in section 3.2 (named sim) and optimized control (named optim).

The optimization variables are the outlet temperature of the solar field, evaporator temperature, and the activation of the heat pump.

One important thing to remind is that the efficiency of the heat pump (COP) is increasing with evaporator temperature (Figure 14).

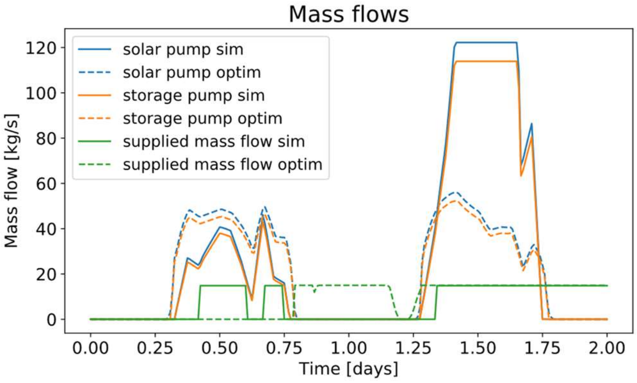


Figure 22. Mass flows of each hydraulic loop. The time scale reads as follow (likewise for all following graphs): 0.5 day = 12PM, 1.0 day = 12AM.

Figure 22 shows the mass flow supplied to the process (coming from the heat pump) shifting from the first day (around 0.5 day) to the first night (around 1.0 day), knowing that heat demand of the process is considered constant. This suggests that the heat pump does not turn on as soon as the tank is filled with enough hot water but rather is led to wait for the night. It allows the tank to rise in temperature during the first day (Figure 23).

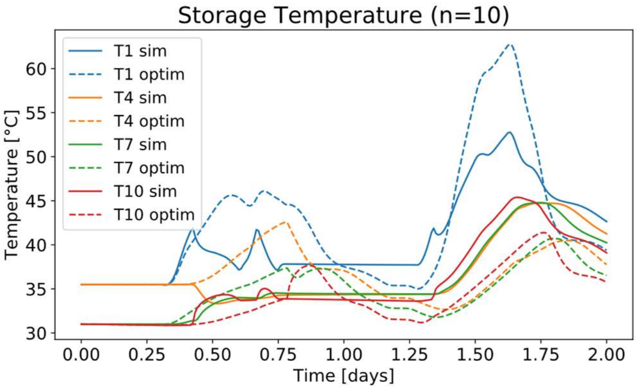


Figure 23. Storage Temperature

Figure 24 shows that the first hours of the night the heat pump is turning on with a higher evaporator temperature (around 42°C) than the rest of the night (around 34°C), which allows to have a better COP the first hours of the night.

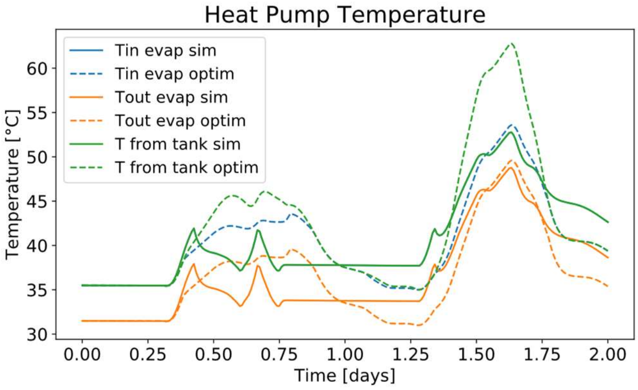


Figure 24. Heat pump evaporator temperature. Tin evap sim and T from tank overlap across the whole time range.

Figure 25 shows that the solar field outlet temperature is different for the two days. The first day, it is lower than the simulation, it leads to less heat losses in the solar field. But the second day this temperature is much higher. This could be explained by the fact that at the end of the second day the tank is full enough to provide heat continuously to the heat pump in both cases: simulation and optimization (Figure 23). Then, the higher is the solar field temperature, the higher the temperature could be at the evaporator, and thus the COP. It also means that, the optimizer does not care about what could happen after the optimization time horizon: we could imagine a third cloudy day which would need to have a tank more filled (with lower temperature) at the end of the second day than the optimization result. This example shows that the result of the optimization may depend on the considered time horizon. Further, to prevent the optimizer to empty the tank at the end of optimization and closer emulate the behavior of a real system, tank state could be constrained to approach the final value obtained through the initial simulation.

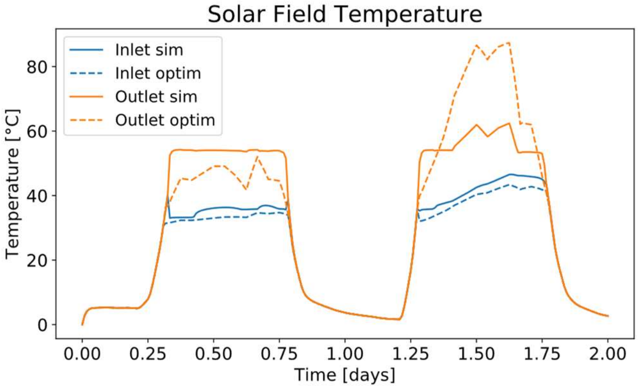


Figure 25. Solar Field Temperature

Finally, Figure 26 shows the instant profit of the plant, depending on the heat supplied and electricity consumption. We can also see the shift of the heat production time from the first day to the first night.

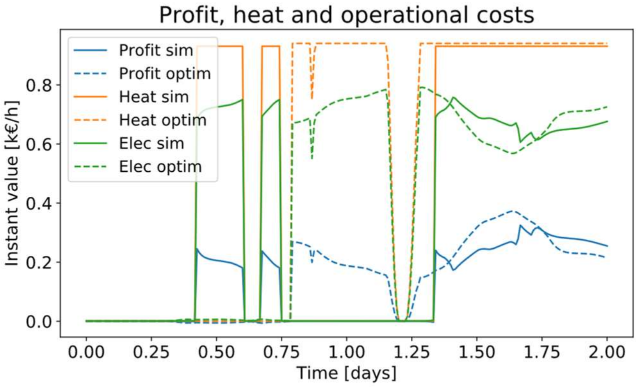


Figure 26. Instant value of the plant

Table 1 gives the improvement provided by the optimizer. The gain seams huge (+21.4% of profit), but it is still not considering what is happening after the two days.

Table 1. Plant economic results for 2 simulated days

	Electric consumption (k€)	Heat supplied (k€)	Profit (k€)
Simulated	15.2	20.6	5.41
Optimized	18.8	25.4	6.56
Gain	+ 23.8%	+23.2%	+21.4%

5 Conclusion and outlook

To conclude, we developed and validated a library which is compatible with Optimica and models a renewable thermal plant. Then we obtained interesting dynamic optimization results of the overall system.

However, the dynamic optimization result is not yet satisfactory since it does not consider the necessity to be able to provide heat after the optimization end time. It could lead to non-optimal results in the real plant, even if only the first hours of the optimization result would be sent to the real plant controller as the optimization will be updated every hour. It also points out the need to perform an offline dynamic optimization based on forecasts on a long enough time horizon before starting DRTO which will correct the control variables trajectories taking into account real disturbances. The final state of the tank could be considered in the optimization objective to get a better result.

We could also consider stratification indicators into the optimization objective, because stratification inside the tank is affecting a lot the efficiency of the storages. Some research works developed indicators to quantify the quality of stratification in thermal storages.

Equipment modeling could also be improved. Efficiency of the PHEX is considered constant. An operating point tabular efficiency could make PHEX model more accurate, with minimal added complexity. A slightly more complex model of COP could decrease error of the COP observed of the heat pump. The current storage model loses accuracy due to low discretization. Other storage models could be considered to either reduce the number of state variables or improve accuracy.

Once the offline optimization will be robust enough, this work will be extended to real-time optimization of the thermal plant. A highly detailed model describing the plant will be needed, so that the optimizer can be run and finetune the plant setpoints in real-time.

Acknowledgements

This work is part of a PhD study. It has been supported by Newheat and the LaTEP.

References

Åkesson, J., et al. (2009). “JModelica---an Open Source Platform for Optimization of Modelica Models”. In: *6th Vienna International Conference on Mathematical Modelling,* Vienna, Austria.

Dumont, Olivier, et al. (2016). “Hot Water Tanks : How to Select the Optimal Modelling Approach?”. In: *Proceedings of CLIMA*.

Fischer, David, et al. (2017). “Model-Based Flexibility Assessment of a Residential Heat Pump Pool.”. In: *Energy*, vol. 118, pp. 853–64. DOI: 10.1016/j.energy.2016.10.111.

Hawlader, M. N. A., et al. (1988). “A Thermally Stratified Solar Water Storage Tank.” In: *International Journal of Solar Energy*, vol. 6, no. 2, pp. 119–38. DOI: 10.1080/01425918808914224.

IBPSA (2013). “IBPSA.Fluid.Storage.BaseClasses.Buoyancy”. URL: https://github.com/ibpsa/modelica-ibpsa/blob/master/IBPSA/Fluid/Storage/BaseClasses/Buoyancy.mo (visited on 2023-06-11).

International Standard (2017). “ISO/FDIS 9806, 2017. Énergie solaire - Capteurs thermiques solaires - Méthodes d’essai”.

Liu, Fang, et al. (2018). “Model-Based Dynamic Optimal Control of a CO2 Heat Pump Coupled with Hot and Cold Thermal Storages”. In: *Applied Thermal Engineering*, vol. 128, pp. 1116–25. DOI: 10.1016/j.applthermaleng.2017.09.098.

Magnusson, Fredrik, and Johan Åkesson (2015). “Dynamic Optimization in JModelica.Org”. In: *Processes*, vol. 3, no. 2, 2, pp. 471–96. DOI: 10.3390/pr3020471.

Newheat (2020). “Malteries Franco-Suisses (Boortmalt Group)”. URL: https://newheat.com/en/projects/french-swiss-malhouses-boortmalt-group/ (visited on 2023-06-11).

Newheat (2023). “Whey Powder Production Site (Lactalis Group)”. URL: https://newheat.com/en/projects/dairy-industry/ (visited on 2023-06-11).

Rohde, Daniel, et al. (2020). “Dynamic Optimization of Control Setpoints for an Integrated Heating and Cooling System with Thermal Energy Storages”. In: *Energy*, vol. 193, p. 116771. DOI: 10.1016/j.energy.2019.116771.

Ruhnau, Oliver, et al. (2019). “Time Series of Heat Demand and Heat Pump Efficiency for Energy System Modeling”. In: *Scientific Data*, vol. 6, no. 1, 1, p. 189. DOI: 10.1038/s41597-019-0199-y.

Runvik, Håkan, et al. (2015). “Production Planning for Distributed District Heating Networks with JModelica.Org”. In: *Proceedings of the 11th International Modelica Conference,* pp. 217–23. DOI: 10.3384/ecp15118217.

Scolan, Simon, et al. (2020). “Dynamic Optimization of the Operation of a Solar Thermal Plant”. In: *Solar Energy*, vol. 198, pp. 643–57. DOI: 10.1016/j.solener.2020.01.076.

Untrau, Alix, et al. (2023). “Dynamic Real-Time Optimization of a Solar Thermal Plant during Daytime”. In: *Computers & Chemical Engineering*, vol. 172, p. 108184. DOI: 10.1016/j.compchemeng.2023.108184.

Zhang, Ruichao, et al. (2023). “Dual-Objective Optimization of Large-Scale Solar Heating Systems Integrated with Water-to-Water Heat Pumps for Improved Techno-Economic Performance”. In: *Energy and Buildings*, vol. 296, p. 113281. DOI: 10.1016/j.enbuild.2023.113281.

Efficient Global Multi Parameter Calibration for Complex System Models Using Machine-Learning Surrogates

Julius Aka[1,2] Johannes Brunnemann[1] Svenne Freund[2] Arne Speerforck[2]

[1]XRG Simulation GmbH, {aka,brunnemann}@xrg-simulation.de
[2]Hamburg University of Technology, {julius.aka,svenne.freund,arne.speerforck}@tuhh.de

Abstract

In this work, we address challenges associated with multi parameter calibration of complex system models of high computational expense. We propose to replace the Modelica Model for screening of parameter space by a computational effective Machine-Learning Surrogate, followed by polishing with a gradient-based optimizer coupled to the Modelica Model.

Our results show the advantage of this approach compared to common-used optimization strategies. We can resign on determining initial optimization values while using a small number of Modelica model calls, paving the path towards efficient global optimization. The Machine Learning Surrogate, namely a Physics Enhanced Latent Space Variational Autoencoder (PELS-VAE), is able to capture the impact of most influential parameters on small training sets and delivers sufficiently good starting values to the gradient-based optimizer.

In order to make this paper self-contained, we give a sound overview to the necessary theory, namely Variational Autoencoders and Global Sensitivity Analysis with Sobol Indices.

Keywords: Sensitivity Analysis, Sobol-Indices, Variational Autoencoders, VAE, Physics-Enhanced Latent Space Variational Autoencoder, PELS-VAE, Model Calibration, Global Optimization, Machine Learning Surrogate

1 Introduction

To enable model based investigation of "real world" technical systems the underlying Modelica system models can quickly grow in size and computational expense. When they are applied in extensive parameter studies, in particular for model calibration or model based optimization, computation becomes a resource intensive task: if the objective function cannot be decomposed into submodel dependencies but depends on the model as a 'whole', then also the whole model needs to be simulated.

In practice optimization based on such models is limited to a few varied parameters and to local, gradient based optimization algorithms. If the modeller has sufficient knowledge on the model, reasonable choices of relevant parameters as well as starting points for the local optimization algorithm can be made from experience. But for complex models this empirical approach may suffer from overlooking parameters and the optimization algorithm running into local minima of the objective function due to the chosen starting points in parameter space.

In this paper we address these issues with a combined approach: A Machine Learning Model, namely a Physics Enhanced Latent Space Variational Autoencoder (PELS-VAE) (Martínez-Palomera, Bloom, and Abrahams 2020; Zhang and Mikelsons 2022) is trained on data generated by the Modelica model. It captures the dependences of model output to the most influential parameters, determined by a preceding sensitivity analysis (Sobol 1993), while requiring a limited set of training data. This surrogate is computationally cheap, and can be used to apply a global optimization algorithm that relies on a large number of model runs. After this global screening, a subsequent local optimization based on the original physical model (polishing) is performed.

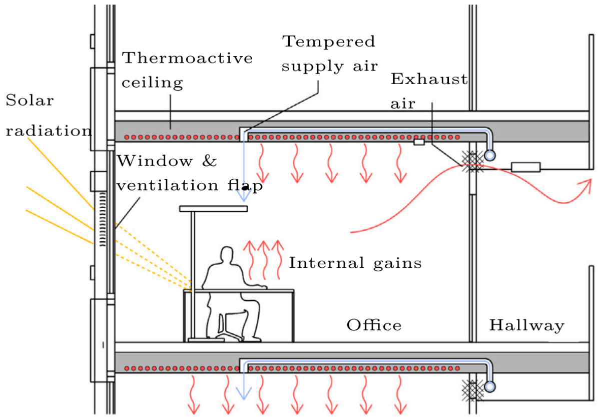


Figure 1. Schematic of standard single office, taken from (Freund and Schmitz 2021)

We choose to test our approach on a computational inexpensive, thermal Modelica model of a single office (Figure 1) with measurement data available for calibration (Freund and Schmitz 2021). Like this, data generation for the machine learning models is fast and we are able to focus on the application of the PELS-

VAE for parameter calibration, while being able to cross check all obtained results against a brute force global optimization based on the original model.

Various Optimization Tools suitable for Modelica models already exist, like the *Dymola Optimization Library*, *GenOpt* (University of California 2023), *ModestPy* for Parameter Estimation with FMUs (Arendt et al. 2018), *AixCaliBuHA* (Wüllhorst et al. 2022) or *ModelOpt* (XRG Simulation GmbH 2023). All of these tools vary in detail, but build on common-known global and local Optimization Algorithms like Particle Swarm Optimization, Genetic Algorithms, Sequential Least Squares or Nelder-Mead Algorithm and do not generate surrogate models.

In contrast to this, surrogate based optimization aims to represent computationally expensive models by the use of a simpler surrogate to significantly save computational resources. Different kinds of surrogate models like linear regression, support vector regression, radial basis functions or kriging (Gaussian process regression) are commonly used (Bhosekar and Ierapetritou 2018). Artificial Neural Network as a generalization of regression models are also a possible surrogate choice. A promising subclass is Bayesian Optimization, which consists of a probabilistic surrogate model and a sequential called loss function that enables optimal, active sampling of the objective function that should be replaced (Shahriari et al. 2016). Bayesian Optimization proved efficient in parameter calibration of a Modelica-modeld HVAC-system (Martinez-Viol et al. 2022). In comparison to these techniques, our approach replaces the actual physical model for a fixed scenario, not the cost function of an optimization objective.

This paper is organised as follows: section 2 introduces the used Modelica model of an office room, the PELS-VAE architecture and training, as well as the applied optimization techniques. In section 3 we present the results of applying our approach for calibration of the Modelica model. Finally we summarize our findings and give an outlook to present and future work in section 4. In Appendix B, we sketch the applied global sensitivity analysis.

2 Methods

2.1 Calibration Problem

The modelled thermal zone is a room of a large-scale office-building ($46\,500\,\mathrm{m}^2$) and high energy efficency (primary energy demand $< 70\,\mathrm{kW\,h\,m^{-2}}$) (Freund and Schmitz 2021). The buildings operation has been explored in previous research projects ((Niemann and Schmitz 2020), (Duus and Schmitz 2021), (Freund and Schmitz 2021)). For example, Model-Predictive-Control (MPC) was used to enhance thermal user-comfort and decrease energy demand (Freund 2023). MPC requires accurate models which can be obtained by calibrating Modelica-Models with measurement data.

A scheme of an office is shown in Figure 1. Heat is supplied by thermal activated ceilings (TAC), i.e. by circulating warm water through pipes in the concrete core of the slabs, and mechanical ventilation with preheated supply air. The large area of the ceilings allows the usage of heat-pumps for low temperature heating, while the high thermal capacity of the concrete slabs enables considerable time delay between heat supply to the ceiling and heat supply to the room. For this building, measurement data is recorded since 2014 at more than 1100 sensors every minute. 32 office spaces are equipped as reference zones with various sensors. (Freund and Schmitz 2021)

For this project, we use the same data than in prior studies (Freund 2023). The calibration target is to fit the model output T_{Air} to the recorded measurement $T_{\mathrm{Air,meas}}$ by adjusting the model parameters $\boldsymbol{\theta}$ within their bounds$[\boldsymbol{\theta}_-, \boldsymbol{\theta}_+]$, employing an error metric such as the Mean Squared Error (MSE):

$$\min_{\boldsymbol{\theta}} \quad \frac{1}{T}\sum (T_{\mathrm{Air}}(t_i) - T_{\mathrm{Air,meas}}(t_i))^2 \qquad (1)$$
$$\text{subject to } \boldsymbol{\theta}_- \leq \boldsymbol{\theta} \leq \boldsymbol{\theta}_+$$

which is in general a constrained, nonlinear optimization problem.

The recorded data consists of several timeseries that serve as an input to the physical model of the thermal zone. The model inputs are outside air temperature T_{A}, supply temperature of the corresponding TAC heating circuit $T_{\mathrm{Sup,TAC}}$, boolean signal of supply $y_{\mathrm{Sup,TAC}}$, supply temperature of mechanical ventilation $T_{\mathrm{Sup,MV}}$, boolean signal of supply $y_{\mathrm{Sup,MV}}$, global solar radiation and occupancy state. For the heat exchange at sun-exposed walls, an equivalent outdoor air temperature $T_{\mathrm{A,Eq}}$ is used. Internal heat gains by persons, lighting or other equipment $\dot{Q}_{\mathrm{Int}}$ are calculated using the by constant heat gain factor multiplied with an heuristic based on measured occupancy state and the buildings electric energy consumption load profile. Internal and external heat gains are split into convective parts acting on the air volume and radiative parts acting on the internal masses. We use data of the identification-timeframe 21.02.2018 - 14.03.2018 (Freund 2023).

2.1.1 Gray-Box Model

In this work, a Gray-Box Model introduced by (Freund and Schmitz 2021) shall be calibrated. Gray-Box Modeling referes to a modeling approach, where a physical model is combined with data-driven approaches. Physical knowledge is used to derive a model structure, while parameters are identified using measurement data (Kathirgamanathan et al. 2021).

The gray-box model (Figure 2) consists of seven resistances and four capacities (R7C4 model). The four

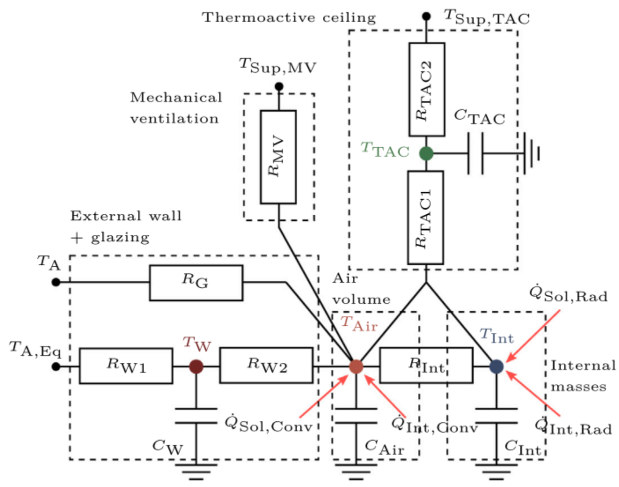


Figure 2. RC network representation of gray-box zone-model with 7 Resistances and 4 Capacitors (R7C4) (Freund and Schmitz 2021).

state variables T_{W} (external wall temperature), T_{Air} (indoor air temperature), T_{Int} (temperature of internal masses) and T_{TAC} (TAC core temperature) correspond to the four thermal capacities $C_{\mathrm{W}}, C_{\mathrm{Air}}, C_{\mathrm{Int}}$ and C_{TAC}.

Based on the EMPA model (Koschenz and Lehman 2000), a simplified model for the TAC is used consisting of two resistances R_{TAC1} and R_{TAC2}. By assuming equal room temperatures below and above the thermoactive ceiling, the two heat flow paths to respectively the room above and below the ceiling can be transformed into a single heat flow path, resulting in a R2C1 TAC model (Sourbron 2012).

The external wall is modeled with two resistances for the envelop (R_{W1} and R_{W2}) and one resistance for the glazing R_{G}. Mechanical ventilation is represented with one resistance R_{MV}. The resistance R_{Int} describes the heat exchange between the air volume and the internal masses. Heat exchange between adjacent zones is neglected, since the heating control is for all zones of a building section the same.

Consequently, the simulation model has 11 parameters (see Table 1). Additionally, we introduce the parameter f_{sol} to tune the fraction of the window projected global radiation flowing to the office and the parameter $\dot{Q}_{\mathrm{Int}}$ as heat gain factor of the heuristic occupancy signal. The initial temperature $T_{\mathrm{TAC}}(t=0)$ of the TAC as the mass with the highest capacity is introduced as a parameter to the optimization problem. Estimated values for these parameters are obtained by using the documentation of constructional elements and values from literature. These estimates are used to generate training data for the autoencoder models, which is for most parameters performed in the range of $\frac{1}{5}$ to 5 times the estimated value. We choose these broad ranges in order to account for situations, where little knowledge on the estimates is available. In practice they should be narrowed as much as possible by available information.

Table 1. Description of the 14 RC-Model Parameters and Corresponding Parameters of the Modelica Model.

RC-Model	Description	Modelica Model
C_{W}	Wall Capacity	`cExt1`
C_{Air}	Air Capacity	`b`
C_{Int}	Internal Masses	`cInt`
C_{TAC}	Thermoactive Ceiling (TAC) Capacity	`cTABS`
R_{TAC1}	TAC Resistance Capacity/Room	`rZone`
R_{TAC2}	TAC Resistance Pipe/-Capacity	`rPipe`
R_{W1}	Wall Resistance Outdoor/Capacity	`rExt1`
R_{W2}	Wall Resistance Capacity/Room	`rExt2`
R_{G}	Window Resistance	`UWin`
R_{MV}	Mechanical Ventilation	`VSup`
R_{Int}	Internal Heat Exchange	`rInt`
f_{sol}	Solar Gain Fraction	`fSol`
$\dot{Q}_{\mathrm{Int}}$	Internal Heat Gains	`qIntOcc`
$T_{\mathrm{TAC}}(t=0)$	Initial Value	`TTABSInit`

The obtained model is exported by using the Functional Mock-up Interface (*FMI*) standard and used in Python-Scripts with *FMPy* (FMPy 2023). We simulate with a time step of 1800 s.

2.2 Physics-Enhanced Latent Space Variational Autoencoder

The general idea of Autoencoders is to encode data of a dataset in a lower-dimensional compression that is sufficient to represent the variation within that dataset. For example, a collection of images of people could be reduced to characteristics like gender, hair color, skin color, pose etc. From this compression, data can be reconstructed with a decoder that learned the influence on the compression of these attribute variations to reconstruct an image from it. In general, the lower-dimensional compression is said to be in a "latent space", i.e. a space whose behavior is hidden and cryptic to us. A Encoder-Decoder Neural Network structure is an unsupervised learning technique. However, the representation of attributes in latent space can be learned, i.e. by a neural network (”Regressor”). By only using the Regressor and the Decoder, new data can be generated, such that an Generative Adversarial Network (GAN) is obtained. A challenge is to chose an adequate dimension for the latent space to prevent the network from just memorizing the data (Jordan 2018a). Various tech-

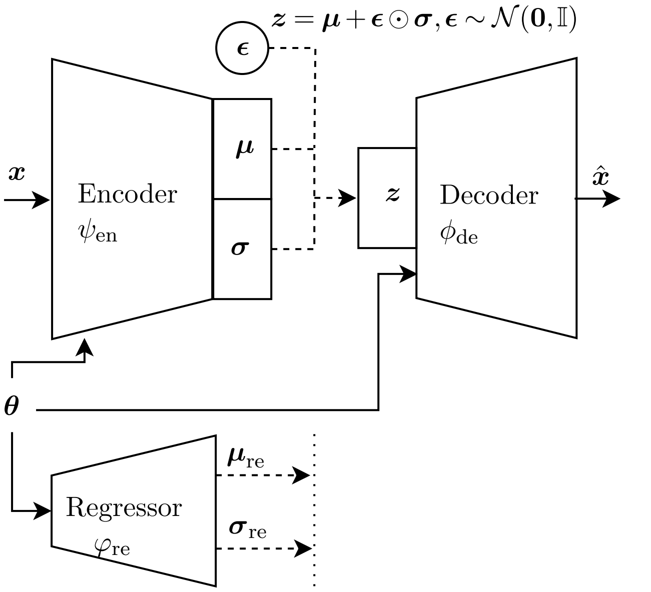


Figure 3. Physics-Enhanced Latent Space Variational Autoencoder (PELS-VAE). The time-series $\boldsymbol{x}$ is introduced to the Encoder ψ_{en}, which transforms it to a latent-space distribution with mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}$, which can be decoded by the Decoder ϕ_{de} by sampling $\boldsymbol{z}$ with the auxiliary Gaussian variable $\boldsymbol{\epsilon}$ to reconstruct the time-series as $\hat{\boldsymbol{x}} = \frac{1}{L}\sum^{L} \phi_{de}(\boldsymbol{z})$. The Regressor φ_{re} is trained simultanously to predict the mean and variance of the latent space distribution. As in (Martínez-Palomera, Bloom, and Abrahams 2020), the physical parameters $\boldsymbol{\theta}$ are introduced to all models. (Zhang and Mikelsons 2022)

niques have been proposed for this regularization, and a widely used approach is to learn probability distributions within the autoencoder structure, making it an Variational Autoencoder (VAE). Hands-on explanation for Autoencoders can be found in (Jordan 2018a), while for VAE in (Jordan 2018b).
Within this paper, we build on the implementation of (Zhang and Mikelsons 2022) to predict time-series $\boldsymbol{x}$ (i.e. our temperature trajectories), with its architecture shown in Figure 3.

For the interested reader, a more detailed explanation of the theory behind the Autoencoder and its training loss function is provided in Appendix A.

2.3 Training Data Generation

To train the PELS-VAE model to mimic the behaviour of the physical model, i.e. learning the behaviour $\boldsymbol{x}(\boldsymbol{\theta})$, the machine learning model needs to be exposed to labeled training data $(\boldsymbol{x}|\boldsymbol{\theta})$. Therefore, we sample n times uniformly in parameter space:

$$\boldsymbol{\theta} \sim \mathcal{U}(\boldsymbol{\theta}_{-}, \boldsymbol{\theta}_{+}) \tag{2}$$

and run the physical Modelica Model to get the posterior $\boldsymbol{x}$ of $\boldsymbol{\theta}$. As the purpose of this paper is to determine possible reduction in required simulations of the physical model, we generate training sets with different sizes in the range (32 to 4096). Validation and test sets have a size of 320 samples. To make results comparable, validation and test sets are the same for all models. The validation set is used to validate the model during the training process to select well-generalizing models and to early stop the training if no further improvement is happening. The test-set is used to determine the final performance of the model, unbiased by the selection through the test set.
The training data should cover well the parameter space as well as the output space, which can be checked by plotting the corresponding confusion plots (combining every θ_i with each other) and plotting all outputs of the physical model. Combining these plots of the model outputs with available measurement data, allows to make a first check if the designed physical model is able to capture the observed behaviour (see Figure 4).

2.4 Optimization-Based Parameter Identification

This paper aims to calibrate a model by minimizing the Mean Squared Error (MSE) between the model output and recorded measurements to determine a globally minimizing parameter combination. An overview of the applied methods is provided in Table 2 and discussed further below.

To demonstrate the superiority of our proposed method over existing optimization techniques, we combine a *FMU* of the Modelica model with selected optimization methods from *SciPy* and compare them with our introduced methods that use the Physics-Enhanced Latent Space Variational Autoencoder (PELS-VAE).

The investigated methods that combine a *SciPy* optimizer with an *FMU* encompass scalar or vector-like objectives, gradient-descent or non-gradient-descent methods, and can be categorized as either local or global optimization techniques. We anticipate gradient-based optimizers to converge quickly and expect further improvements for the LS-TRF approach, which utilizes residuals as the objective, as the optimizer gains more knowledge about the optimization step consequences compared to scalar objectives.
On the other hand, we consider Differential Evolution, a genetic algorithm (GA), a global optimization technique, albeit with the drawback of requiring a higher number of model evaluations.
All local techniques in this study necessitate initial values for the parameters, which may be challenging to derive in practical applications. To address this, we combine each local technique with a multistart approach, where the optimization is initiated n_{start} times using starting values randomly distributed around the given initial parameter values.

Based on these evaluations, we propose to combine a well-trained computationally cheap PELS-VAE

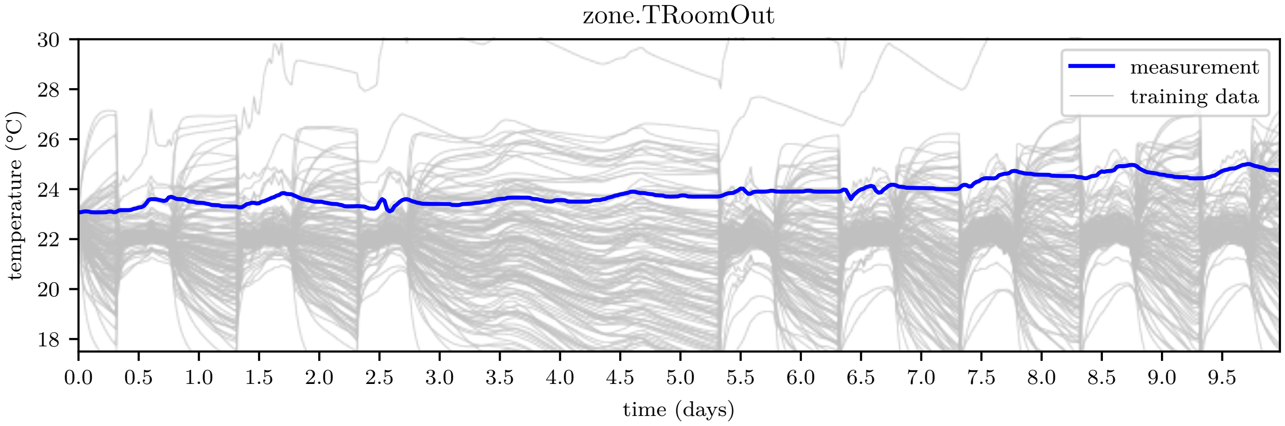


Figure 4. Room temperature trajectories of measurement and training data with sample size $n = 256$, sampled uniformly over parameter space $\boldsymbol{\theta} \sim \mathcal{U}(\boldsymbol{\theta}_-, \boldsymbol{\theta}_+)$

.

Table 2. Optimization methods used in this paper for calibration. If the objective is scalar, a $MSE = \mu(x_{\text{sim}} - x_{\text{meas}})^2$ is used as objective, if the objective is residual, the vector of squared residuals at the simulation time steps $[(x_{\text{sim}}(t_0) - x_{\text{meas}}(t_0))^2, (x_{\text{sim}}(t_0 + \Delta t) - x_{\text{meas}}(t_0 + \Delta t))^2, \ldots]$ is used as objective. For all techniques from *SciPy*, default settings are used. Iterations are limited to reasonable values and tolerances are adapted to the *FMU*-settings.

Method Name	Short Description	Objective	Gradient	Scope
	Methods from SciPy (Virtanen et al. 2020)			
Powell	Conjugate direction method, sequentially performing one-dimensional optimization over an iteratively updated set of direction vectors	Scalar	No	Local
Nelder-Mead	Geometric operations (reflection, expansion, contraction, compression) on a simplex of points (Gao and Han 2012)	Scalar	No	Local
Sequential Least Squares Programming (SLSQP)	Iterative Method for nonlinear constrained optimization that integrates constraints by solving quadratic programming subproblems (Kraft 1988)	Scalar	Yes	Local
Least Squares Trust Region Reflective (LS-TRF)	Gradient-based algorithm, incorporating trust region strategies and reflective boundaries to improve convergence	Residuals	Yes	Local
Differential Evolution (Genetic Algorithm)	Population-based algorithm evolving a population of solution candidates with genetic operations (e.g., mutation)	Scalar	No	Global
	Methods introduced in this paper			
Multistart	first inital value $\boldsymbol{\theta}_{\text{init},0} = \boldsymbol{\theta}_{\text{init}}$, following initial values $j > 0 : \boldsymbol{\theta}_{\text{init,j}} \sim \boldsymbol{\mathcal{N}}(\boldsymbol{\mu} = \boldsymbol{\theta}_{\text{init}}, \boldsymbol{\sigma}^2 = 0.5(\boldsymbol{\theta}_+ - \boldsymbol{\theta}_-))$, repeated until a combination within the bounds $(\boldsymbol{\theta}_-, \boldsymbol{\theta}_+)$ is found.			
GA with PELS-VAE	PELS-VAE coupled with a genetic algorithm	Scalar	No	Global
GA with PELS-VAE + Polish	Phase 1: PELS-VAE coupled with genetic algorithm Phase 2: Polishing Result by LS-TRF with *FMU* of Modelica Model	Scalar	Yes (Phase 2)	Global (Phase 1)

neural-network (i.e. capable of evaluating 10 000 parameter combinations in a few seconds on a GPU) with a genetic algorithm to determine a parameter set that achieves global optimization. Additionally, we propose a 2-Phase approach in which the parameter combination determined by PELS-VAE coupled with a genetic algorithm serves as starting point for a polishing phase. The polishing phase employs the LS-TRF algorithm coupled with the *FMU* of the physical model to be calibrated. This approach is intended to compensate for inaccuracies that may arise when replacing the physical model with a machine learning surrogate model.

3 Results

3.1 Sensitivity Analysis

In order to evaluate the impact of different model parameters to the room temperature, we employ a sensitivity analysis based on Sobol indices as described in Appendix B. The result is shown in Figure 5, where for each time step the Sobol indices are plotted. Obviously the impact of different parameters changes with time: due to heating with TAC and air supply during daytime the ”passive” building properties `UWin` and `rExt1` become less important.

This can be used in order to potentially limit the number of parameters in the overall analysis or in the training of the Autoencoder, as parameter dependencies with large impact are faster learned, that is less training is required (see section 3.2). Often this will be sufficient for the global phase 1 of the optimization approach described in this paper (see section 3.3.3).

3.2 Autoencoder Training

The Autoencoder training was carried out using different numbers of samples n, a varied dimension of the latent space ($\dim(\boldsymbol{z_x})$), and varied dimension of the hidden layers. The analysis, shown in Figure 6, was performed using the same test set ($n = 320$) for all experiments.

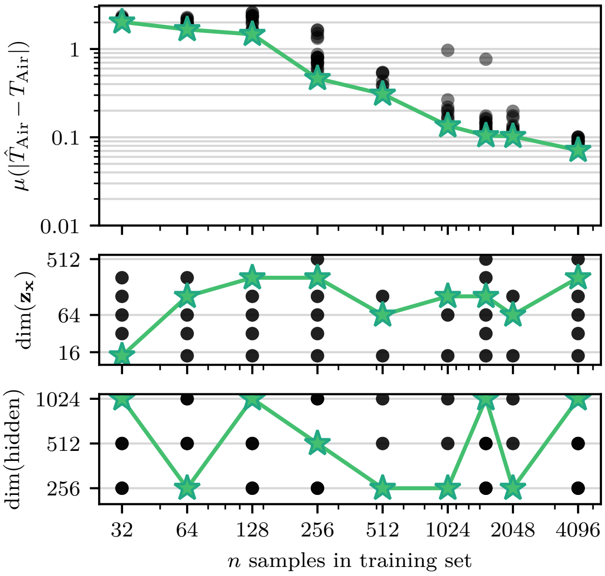


Figure 6. Hyperparameter variation (latent space dimension $\dim \boldsymbol{z_x}$ and dimension of hidden layers) over training sets with different number of samples n, tested with same uniformly sampled test set, all within $[\boldsymbol{\theta}_-, \boldsymbol{\theta}_+]$. The best-performing model for each training dataset size is marked by a star. (training performed for day 8-16 of identification timeframe)

Firstly, the Mean Absolute Error (MAE) was observed to decay with an increasing number of samples. Specifically, for 32 samples, the MAE was approximately 2, which decreased to around 0.07 for 4096 samples. Notably, with 1024 samples, the MAE reached 0.1, and further quadrupling the sample size only resulted in marginal improvements.

Secondly, models trained with different hyperpa-

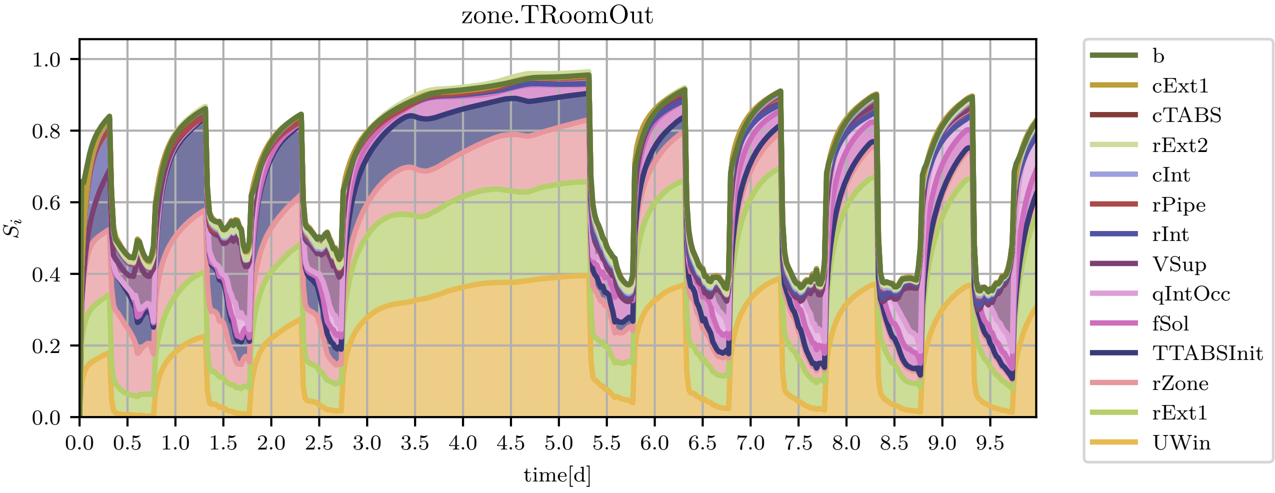


Figure 5. First order Sobol indices for model parameters, plotted ordered by mean value

rameters show variation in MAE. Although at higher numbers of samples the variations may be tolerable, at $n = 256$ the influence of hyperparameter selection lies in the range of 0.4 to 1.1 which might not be appropriate.

Lastly, the dimension of the latent space ($\dim(\boldsymbol{z_x})$) was found to scale with the complexity of the time series. Although the model had 14 parameters, the best latent space dimension are 64, 128, or 256. Reasons for this assumption are that $\boldsymbol{\theta}$ was also directly introduced in the decoder and we found by inspection, the worst performing models had low $\dim(\boldsymbol{z_x})$.

Overall, we find that the influence of chosen hyperparameters changes the training outcome, although its influence is limited, i.e. all hyperparameter sets produced results in comparable ranges with no "failing" hyperparameter sets. We conclude from this, that the Autoencoder training is quiet robust.

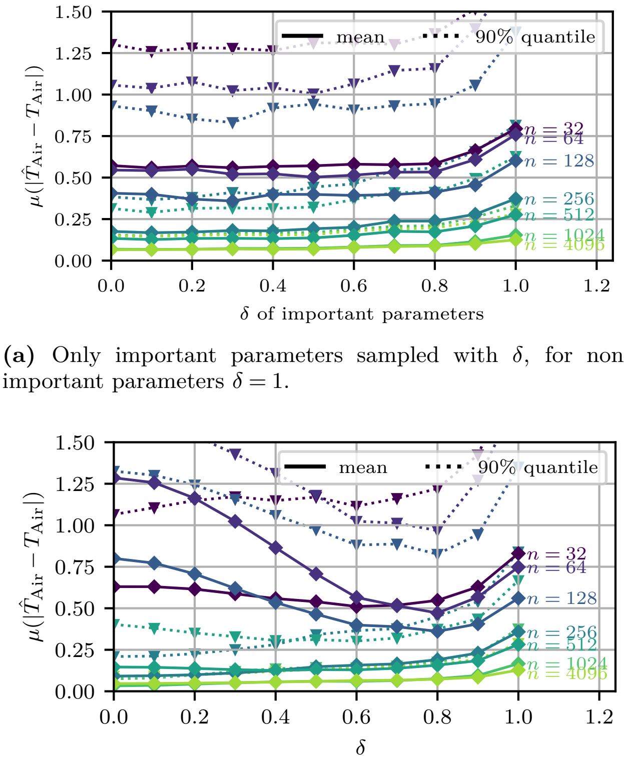
To further assess the performance of the Autoencoder, we have evaluated the prediction error using test sets with varying variability. In the case of large sample sizes n, the median lies at the midpoint of the parameter space of each dimension, approximated as $\tilde{\theta}_i \approx \frac{\theta_{i,+}+\theta_{i,-}}{2}$ with distance of $\Delta\theta_i = \frac{\theta_{i,+}-\theta_{i,-}}{2}$ to each bound. To generate the test sets, we sample with different δ as follows:

$$\boldsymbol{\theta} \sim \mathcal{U}(\tilde{\boldsymbol{\theta}} - \delta\boldsymbol{\Delta\theta}, \tilde{\boldsymbol{\theta}} + \delta\boldsymbol{\Delta\theta}), \quad \delta \in 0,0.1,\ldots,1.0$$

We generate two kinds of test sets: A category in that all parameters are varied and a category in that only the six most important parameters (subsection 3.1, Figure 5) are varied. The Mean Absolute Error (MAE) for these test sets, evaluated on the models trained with the best-performing hyperparameters determined previously, is presented in Figure 7. For the following, we can exclude a discussion about the numerical influence of the parameter value magnitudes as all parameters are normalized by mean and standard deviation before they are fed into the Neural Networks.

First, we take a closer look on the variation of important parameters (7a): We previously observed the MAE in Figure 6 with a variability of $\delta = 100\%$. However, by reducing the variability and excluding the border regions of the parameter space, the prediction error of the Autoencoder decreases. For instance, in the case of a training size of $n = 32$, the error is reduced from 0.8 Kelvin to approximately 0.6 Kelvin at 80% variability.

Figure 7 also includes the 90th percentile of the prediction error. It is evident that certain predictions exhibit considerably higher error than the mean prediction error, which can pose challenges in the optimization process, particularly with Autoencoder models trained on smaller datasets. However, by reducing the variability in the parameter space, the 90th percentile error also decreases.



(a) Only important parameters sampled with δ, for non-important parameters $\delta = 1$.

(b) All parameters randomly sampled.

Figure 7. Mean Absolute Error on randomly sampled test sets with different maximum deviations of parameter combinations from the median value of the training data ($\theta_i \sim \mathcal{U}(\tilde{\theta}_i - \delta\Delta\theta_i, \tilde{\theta}_i + \delta\Delta\theta_i)$). The mean absolute error over all time-series of a test set as well as the 90% quantile is given for the best performing models trained with different numbers of time series. (training performed for day 0-10 of identification timeframe)

Secondly, an analysis is conducted to examine the variation of all parameters, as shown in Figure 7b. Interestingly, it is observed that for small sampling sizes ($n = 32$ to 128), reducing the variability δ leads to an increase in the mean absolute error (MAE), while this trend does not persist for larger sampling sizes. This finding may initially seem counterintuitive, as one might expect that when varying all parameters, the MAE would decrease with overall less variability, compared to varying only the important ones and leaving the others unlimited. However, parameters that are considered less important contribute less to the observed variation in the output of the physical model. Consequently, when training sets are small, the Autoencoder faces challenges in capturing the influence of these less important parameters on the observed trajectories. By limiting the variation of all parameters to, for example, $\delta = 0.2$, a larger proportion of parameters resides in the inner part of parameter space, which can be more difficult for the Autoencoder to learn with small training sets as the majority of variance is produced by the influential parameters.

Consequently, this results in an increase in the mean absolute error.

From this analysis, we conclude that the Autoencoder performs better when predicting parameter combinations $\boldsymbol{\theta}$ that are more centered within the training parameter space. When selecting the bounds $[\boldsymbol{\theta}_-, \boldsymbol{\theta}_+]$, it should be ensured that they are larger than the parameter region where we anticipate the calibrated parameter results to lie. Furthermore, for small number of samples in the training set, the Autoencoder faces difficulties learning properly the influence of less influential parameters on the model output, while learning the impact of the more influential. However, to integrate the influence of the less influential parameters on the model output variation, they should still be sampled during training data generation. This property of the Autoencoder enables to resign on a Sensitivity Analysis before training it.

3.3 Model Calibration

In the following section, we present our findings regarding the curve-fitting methods minimizing MSE between model output and measurement outlined in Table 2. This section is organized as follows: firstly, we present the results obtained from the Optimizers directly coupled to the Modelica Model's FMU, along with the corresponding multistart approach (refer to Table 3), and gain insights to the uniqueness of a solution. Secondly, we showcase the optimization results achieved using the surrogate PELS-VAE Model (see Table 4) and highlight the advantages of our proposed method.

3.3.1 Direct Optimizer Coupling

First of all, one should keep in mind that the measurement signal is prone to error which results from measurement uncertainty of the temperature sensor ($\leq 0.5\,\mathrm{K}$ (Freund 2023)), the data processing and the position of the sensor in the room.
The calibration results obtained from the directly coupled optimizer are presented in Table 3. The majority of methods achieve a final Mean Squared Error (MSE) of approximately 0.01, although they vary significantly in terms of required model calls. Among the local optimizers, LS-TRF achieves the lowest number of iterations, with 261 model calls using the given initial value. SLSQP follows with 3-6 times higher iterations. The none-gradient optimizers Nelder-Mead and Powell perform less efficiently, requiring 5000 model calls (limited by the predefined iteration limit) with the given initial value. The notable difference between SLSQP with a scalar objective and LS-TRF with a vector-like/residual objective can be attributed to the fact that the residual objective allows after calculating the gradient for a more detailed consideration of the consequences of optimizer steps.

Table 3. Calibration Results of Methods which were directly coupled with the Modelica Model's *FMU* with achieved $\mathrm{MSE}_f(\boldsymbol{\theta}_{opt})$.

Method	**model calls**	**MSE**
LS-TRF with initial guess	261	0.0117
SLSQP with initial guess	650	0.0149
LS-TRF with 16 starts	3486	0.0114
Nelder-Mead with initial guess	5000	0.0126
Powell with initial guess	5000	0.0881
LS-TRF with 32 starts	6311	0.0114
LS-TRF with 64 starts	12 612	0.0113
SLSQP with 16 starts	18 245	0.0117
Nelder-Mead with 16 starts	76 483	0.0117
Powell with 16 starts	79 120	0.0134
SLSQP with 64 starts	86 322	0.0114
Nelder-Mead with 64 starts	286 728	0.0110
Powell with 64 starts	31 9322	0.0125
Diff. Evolution with FMU	748 020	0.0104
Nelder-Mead with 256 starts	1 188 496	0.0111

When initial values are poorly known, global optimization strategies help to find the global minimum of a function. The Differential Evolution (Genetic) Algorithm uses unsurprisingly a high number model calls, namely 0.7 million. The introduced multistart-approach also quickly scale the number of required model calls, i.e. for the best performing algorithm LS-TRF 6311 calls with 32 different initial values.

3.3.2 Uniqueness of Solution

To gain more insights into the uniqueness of the solution to our optimization problem, a more detailed analyis of the best-performing algorithm LS-TRF was performed. To perform this a benchmark, a high number of starts (256) was chosen. The identified parameter combinations results were clustered with K-Means Clustering around common centroids (Pedregosa et al. 2011) with the 3 largest groups depicted in 8a, while the 5% best solutions are shown in 8b. From these results, we can infer two insights: First, the optimization problem is ambiguous: one parameter can compensate for the effect for another, e.g. in 8b, a high capacity `cExt1` of the external wall can compensate for low heat resistance `rExt2` and vice-versa. Furthermore, the optimization problem is clearly non-convex, i.e. it hast multiple local minima and the identified parameter combination is depending of the initial value when using local optimizer, which can be seen by the difference between the MSE of the best 5% results with 0.0114 and the average value of 0.0364. If the the problem would be convex, every initial value should lead to the same solution. Therefore, multiple parameter combinations might lead to equally well performing calibrated models.

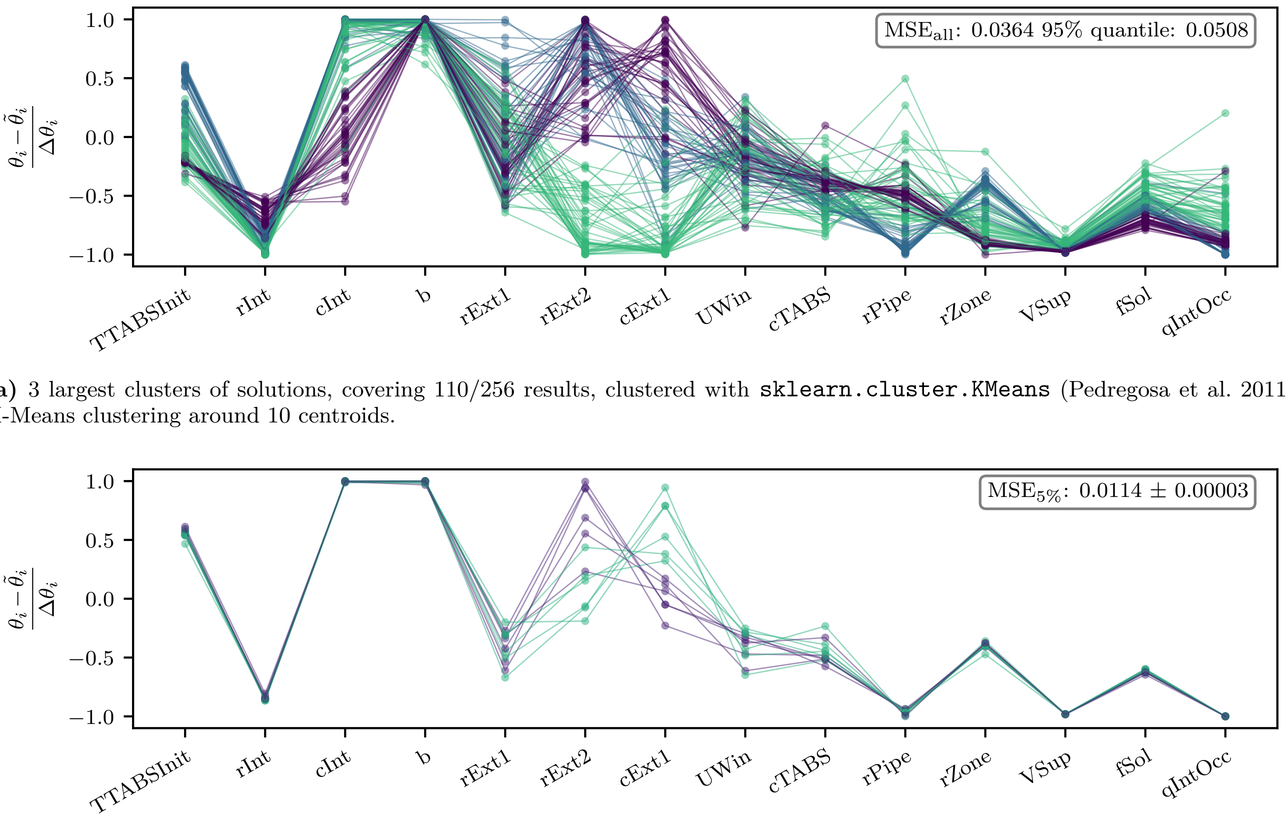
3.3.3 Calibration with Surrogate PELS-VAE Model

The results of the model calibration performed with the surrogate PELS-VAE Model are shown in Table 4 and Figure 9. For the MSE calculated based on the Autoencoder prediction ($\mathrm{MSE}_{\boldsymbol{\phi}_{\mathrm{de}},\boldsymbol{\varphi}_{\mathrm{re}}}(\hat{\boldsymbol{\theta}}_{opt})$), MSE-values comparable to the direct coupling of Optimizer and Modelica-Model (≈ 0.01) are achieved. However, as shown in subsection 3.2, the Autoencoder is prone to prediction errors, i.e. for some parameter combinations, the predicted room temperature trajectories are more faulty than others. Because of this, the MSE of the parameter combination determined by the GA and the Autoencoder, denoted as $\hat{\boldsymbol{\theta}}_{\mathrm{opt}}$, calculated with the Modelica-Model f, $\mathrm{MSE}_f(\hat{\boldsymbol{\theta}}_{opt})$, can be considerably larger than the predicted $\mathrm{MSE}_{\boldsymbol{\phi}_{\mathrm{de}},\boldsymbol{\varphi}_{\mathrm{re}}}(\hat{\boldsymbol{\theta}}_{opt})$. This effect occurs at low numbers of training samples and decreases with higher sampling n_{train}, i.e. a MSE-gap of 1.05 to 3.44 at $n_{\mathrm{train}} = 32$ to 128 is reduced to a gap of 0.02 to 0.22 at $n_{\mathrm{train}} = 512$ to 4096. Although this increase might be negligible at low magnitude, for the results obtained with low number of training samples it might not be appropriate.

Table 4. Calibration Results of PELS-VA coupled with Differential Evolution Genetic Algorithm (GA) for different number of training samples n_{train}, MSE calculated by PELS-VAE ($\mathrm{MSE}_{\boldsymbol{\phi}_{\mathrm{de}},\boldsymbol{\varphi}_{\mathrm{re}}}(\hat{\boldsymbol{\theta}}_{opt})$) and with FMU ($\mathrm{MSE}_f(\hat{\boldsymbol{\theta}}_{opt})$) to determine prediction error introduced by the Autoencoder, number of steps n_{opt} of polishing with LS-TRF and achieved $\mathrm{MSE}_f(\boldsymbol{\theta}_{opt})$.

n_{train}	MSE PELS-VAE + GA	MSE with FMU	n_{opt}	MSE after polish	n_{total}
32	0.0121	1.0623	228	0.0121	260
64	0.0128	1.6390	124	0.0128	188
128	0.0122	3.4509	171	0.0122	299
256	0.0117	0.7498	306	0.0117	562
512	0.0124	0.2486	139	0.0124	651
1024	0.0133	0.0816	65	0.0133	1089
1536	0.0128	0.0298	80	0.0128	1616
2048	0.0135	0.2080	65	0.0135	2113
4096	0.0118	0.0326	201	0.0118	4297

To compensate for that, polishing of the achieved results with the LS-TRF Algorithm, (local, gradient-based, vector-like objective), is performed. This process is illustrated in Figure 9. For all sampling sizes,



(a) 3 largest clusters of solutions, covering 110/256 results, clustered with `sklearn.cluster.KMeans` (Pedregosa et al. 2011) K-Means clustering around 10 centroids.

(b) Best 5% of solutions

Figure 8. Identified normalized parameters for Least-Squares Trust Region Reflective Algorithm with 256 starts.

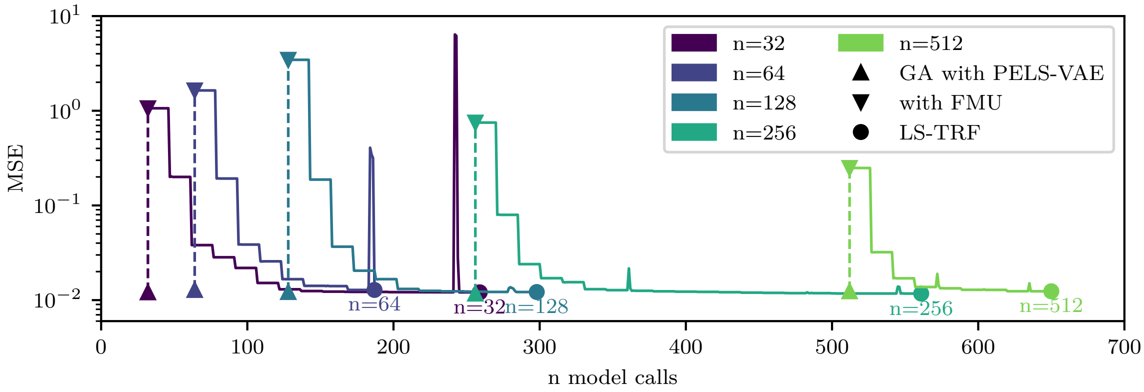


Figure 9. Calibration Results of PELS-VA coupled with Genetic Algorithm (GA), including the prediction gap and the MSE-trajectory during the LS-TRF Optimization.

the MSE is reduced considerable to a magnitude of $\text{MSE}_f(\boldsymbol{\theta}_{opt}) \approx 0.012$. More important, comparing the results for $n_{\text{train}} = 32, 64, 128$, a MSE comparable to that of the LS-TRF directly coupled with the Modelica model with an initial guess is achieved (Table 3), while requiring less or comparable model calls. This effect could be explained as following: As the Autoencoder learns the model reaction on different parameter combinations, especially for the most influential parameters (see subsection 3.2), it allows for a "screening" of parameter space to find a good starting point for the following gradient-based optimization with the exact Modelica-Model. At higher sampling sizes, the prediction gap decreases, which results in the number of model calls reduced as well.

Depending on the number of training samples, one might argue at which point we achieve a screening which is sufficient to call this approach a "global method".

To stress the advantage of this proposed novel method of model calibration: Using the Autoencoder allows a *screening of parameter space*, which *relieves* us of the *burden of finding an initial value* for the optimization, that could potentially even lead us into the "trap" of a local minimum.

4 Conclusion

In this paper, we address the challenges associated with physics based Modelica models increasing in complexity and computational expense regarding optimization-based multi parameter calibration. To overcome these issues, we present a novel approach that enables computationally efficient parameter calibration by using a Machine-Learning Surrogate.

To showcase our developed method, we use a simple thermal zone model implemented in Modelica, which allows to focus on the analysis of the proposes method.

The used Machine-Learning Surrogate is a Physics-Enhanced Latent Space Variational Autoencoder (PELS-VAE). It provides efficient model regularization and robust training. We propose to combine a PELS-VAE trained on a small dataset with a Genetic Algorithm (as PELS-VAE inference is computational cheap) to screen parameter space for well-performing parameter regions. To achieve best-performing results, we furthermore propose to polish the achieved result with a gradient-based residual-objective optimizer (LS-TRF).

To compare our approach to existing alternatives, we have tested a variety of optimizers and found significant variation in number of required model calls and strong dependence on initial values. When moving towards global optimization, the usage of multi-start approaches or global optimizer quickly scales significantly the number of model calls, making this potentially infeasible for computational demanding system models.

We were additionally able to show that the chosen optimization problem is non-convex and has ambiguous solutions.

We also perform a detailed analysis of the PELS-VAE application. By analyzing the training process, we find that hyperparameter variation has limited impact on the training process, i.e. we have a robust training, while predicting time-series that are more centered within the training parameter space exhibit considerably lower prediction error.

Our results provide evidence that even PELS-VAE trained with small datasets (32-128 samples) and resulting high prediction errors proved effective to screen parameter space for initial values which are then used in a gradient based optimizer. We provide indications that the PELS-VAE is able to capture the impact of most-influential parameters on small training sets. Comparing to the best-performing optimizer with the need for an initial value, we were able to show that our initial value free method achieved comparable MSE with comparable number

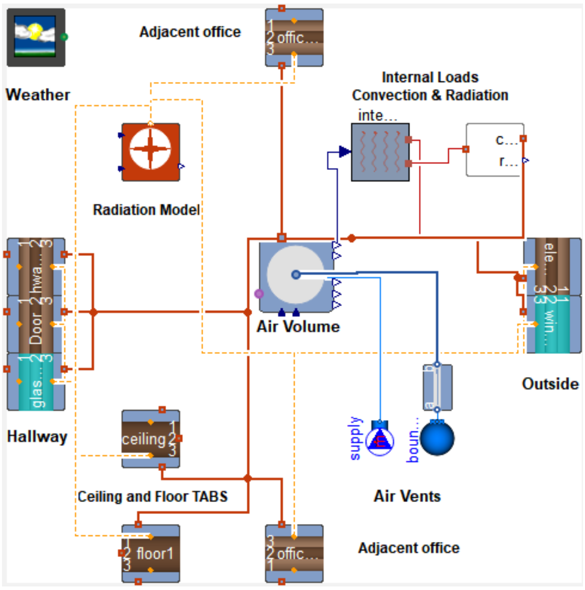


Figure 10. Physics based model of the office with XRG-simulation's HumanComfort Library

of model calls.

In summary, our proposed method offers an effective solution for calibrating complex models. Using the PELS-VAE models allows for a screening of parameter space with a low number of model calls, and relieves us from the burden of fining suitable initial values for local optimizers.

For future work, our method will be applied to other examples like a White-Box Model of the office (see Figure 10) to prove its suitability for various kind of optimization problems. Furthermore, the training process could be improved by adaptive online data generation, narrower parameter ranges, other layers in the network and embedding of multiple Modelica model outputs.

Acknowledgements

Funded by the European Union
NextGenerationEU

Supported by:
Federal Ministry for Economic Affairs and Climate Action
on the basis of a decision by the German Bundestag

A major part of this work has been carried out within the the PHyMoS research project (Proper Hybrid Models for Smarter Vehicles) funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under grant number 19I20022F. J.A. and J.B. thank Yi Zhang and Lars Mikelsons for numerous insightful discussions and support with code examples for implementing PELS-VAE. Moreover J.A. and J.B. thank the XRG team for support. We also thank the anonymous referees for valuable remarks.

References

Arendt, Krzysztof et al. (2018). "ModestPy: an open-source python tool for parameter estimation in functional mock-up units". In: *Proceedings of the 1st American Modelica Conference*. Modelica Association and Linköping University Electronic Press, pp. 121–130.

Bhosekar, Atharv and Marianthi Ierapetritou (2018). "Advances in surrogate based modeling, feasibility analysis, and optimization: A review". In: *Computers & Chemical Engineering* 108, pp. 250–267. DOI: https://doi.org/10.1016/j.compchemeng.2017.09.017.

Burgess, Christopher P. et al. (2018). *Understanding disentangling in β-VAE*. arXiv: 1804.03599 [stat.ML].

Duus, Kristian and Gerhard Schmitz (2021). "Experimental investigation of sustainable and energy efficient management of a geothermal field as a heat source and heat sink for a large office building". In: *Energy and Buildings* 235, p. 110726. DOI: 10.1016/j.enbuild.2021.110726.

FMPy (2023-06-10). *FMPy, A free Python library to simulate Functional Mock-up Units (FMUs)*. URL: https://github.com/CATIA-Systems/FMPy (visited on 2023-06-10).

Freund, Svenne (2023-03). "Modellbasierte Prädiktive Regelung komplexer gebäudetechnischer Anlagen zur Optimierung der Energieeffizienz und des Komforts". PhD thesis. DOI: 10.15480/882.5018.

Freund, Svenne and Gerhard Schmitz (2021-03). "Implementation of model predictive control in a large-sized, low-energy office building". In: *Building and Environment* 197, p. 107830. DOI: 10.1016/j.buildenv.2021.107830.

Gao, Fuchang and Lixing Han (2012-05). "Implementing the Nelder-Mead simplex algorithm with adaptive parameters". In: *Computational Optimization and Applications* 51, pp. 259–277. DOI: 10.1007/s10589-010-9329-3.

Hart, Joey (2018). "Sobol' Indices for Sensitivity Analysis with Dependent Inputs ". In: *SIAM Conference on Uncertainty Quantification, Garden Grove, California, USA*. fetched June, 10th 2023. URL: https://www.pathlms.com/siam/courses/7376#.

Jordan, Jeremy (2018a-03-18). *Introduction to autoencoders*. URL: https://www.jeremyjordan.me/autoencoders/ (visited on 2023-06-10).

Jordan, Jeremy (2018b-03-19). *Variational autoencoders*. URL: https://www.jeremyjordan.me/variational-autoencoders/ (visited on 2023-06-10).

Kathirgamanathan, Anjukan et al. (2021). "Data-driven predictive control for unlocking building energy flexibility: A review". In: *Renewable and Sustainable Energy Reviews* 135, p. 110120. DOI: 10.1016/j.rser.2020.110120.

Kingma, Diederik P and Max Welling (2022). *Auto-Encoding Variational Bayes*. arXiv: 1312.6114 [stat.ML].

Koschenz, M. and B. Lehman (2000). "Thermoaktive Bauteilsysteme tabs". In: *EMPA Energiesysteme/Haustechnik*. 1st ed. Dübendorf.

Kraft, Dieter (1988). "A software package for sequential quadratic programming". In: *Forschungsbericht-Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*.

Martínez-Palomera, Jorge, Joshua S. Bloom, and Ellianna S. Abrahams (2020). *Deep Generative Modeling of Periodic Variable Stars Using Physical Parameters.* arXiv: 2005.07773 [astro-ph.IM].

Martinez-Viol, Victor et al. (2022). “Automatic model calibration for coupled HVAC and building dynamics using Modelica and Bayesian optimization”. In: *Building and Environment* 226, p. 109693. ISSN: 0360-1323. DOI: https://doi.org/10.1016/j.buildenv.2022.109693.

Murphy, Kevin P. (2022). *Probabilistic Machine Learning: An introduction.* MIT Press. URL: probml.ai.

Murphy, Kevin P. (2023). *Probabilistic Machine Learning: Advanced Topics.* MIT Press. URL: http://probml.github.io/book2.

Niemann, Peter and Gerhard Schmitz (2020). “Impacts of occupancy on energy demand and thermal comfort for a large-sized administration building”. In: *Building and environment* 182, pp. 1–15. DOI: 10.15480/882.2857.

Odaibo, Stephen (2019). *Tutorial: Deriving the Standard Variational Autoencoder (VAE) Loss Function.* arXiv: 1907.08956 [cs.LG].

Pedregosa, F. et al. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

SALib (2023-06-10). *SALib, Sensitivity analysis library for systems modeling.* URL: https://salib.readthedocs.io/en/latest/ (visited on 2023-06-10).

Saltelli, A. et al. (2008). *Global Sensitivity Analysis: The Primer.* Chichester (England): Wiley. ISBN: 978-0-470-05997-5.

Shahriari, Bobak et al. (2016). “Taking the Human Out of the Loop: A Review of Bayesian Optimization”. In: *Proceedings of the IEEE* 104.1, pp. 148–175. DOI: 10.1109/JPROC.2015.2494218.

Sobol, I.M. (1993). “Sensitivity Estimates for Nonlinear Mathematical Models”. In: *Mathematical Modelling and Computational Experiments* 4, pp. 407–414.

Sourbron, M (2012). “Dynamic thermal behaviour of buildings with concrete core activation”. Dissertation. Katholieke Universiteit Leuven.

University of California, The Regents of the (2023-08-08). *GenOpt Generic Optimization Program.* URL: https://simulationresearch.lbl.gov/GO/index.html (visited on 2023-08-08).

Virtanen, Pauli et al. (2020). “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Wüllhorst, Fabian et al. (2022). “AixCaliBuHA: Automated calibration of building and HVAC systems”. In: *Journal of Open Source Software* 7.72, p. 3861. DOI: 10.21105/joss.03861.

XRG Simulation GmbH (2023-08-08). *ModelOpt.* URL: http://xrg-simulation.de/de/produkte/applications/modelopt (visited on 2023-08-08).

Zhang, Yi and Lars Mikelsons (2022-07). “Sensitivity-Guided Iterative Parameter Identification and Data Generation with BayesFlow and PELS-VAE for Model Calibration”. In: DOI: 10.21203/rs.3.rs-1898389/v1.

A Physics Enhanced Latent Space Variational Autoencoder (PELS-VAE)

This is a explanation with more detail but simplifications intended for the Modelica Community to gain understanding. For theory without simplifications, please refer to (Kingma and Welling 2022), (Murphy 2022) and (Murphy 2023).

Machine Learning can be done with a probabilistic perspective, such that the quantities of interests are modeled as random variables (Murphy 2022). In stochastic variational inference, it is assumed that the data $\boldsymbol{x}$ to be learned has initially emerged from a latent variable distribution $p(\boldsymbol{z})$ with a conditional probability density function $p(\boldsymbol{x}|\boldsymbol{z})$ (Kingma and Welling 2022).

To model this process within means of unsupervised learning, we have to infer the stochastic latent variable $\boldsymbol{z}$ by a recognition model $p_*(\boldsymbol{z}|\boldsymbol{x})$ from a data sample $\boldsymbol{x}$ and then reconstruct the data with a generation model $p_*(\boldsymbol{x}|\boldsymbol{z})$. As both the true recognition and generation model are inaccessible to us, we model them by using Neural Networks; $q_{\psi_{\text{en}}}(\boldsymbol{z}|\boldsymbol{x})$ for the recognition model and $p_{\phi_{\text{de}}}(\boldsymbol{x}|\boldsymbol{z})$ for the generation model.

For training probabilistic models, one commonly tries to maximize the marginal likelihood of the data, $\sum_{i=0}^{N} \log p(\boldsymbol{x}_i)$. This is the likelihood the network structure assigns to the probability density function at $\boldsymbol{x}_i$, if $\boldsymbol{x}_i$ was inserted. One can think of this as following: If the probability of a data sample $\boldsymbol{x}_i$ is high, the information content it carries is low, i.e. the characteristics are learned by the probability distributions which are modeled by the Neural Networks ψ_{en} and ϕ_{de}. (Odaibo 2019).

One can show (Odaibo 2019) that the right hand side of

$$\begin{aligned}\log p(\boldsymbol{x}_i) \geq -\mathbb{KL}(q_{\psi_{\text{en}}}(\boldsymbol{z}|\boldsymbol{x}_i)\|p(\boldsymbol{z})) \\ +\mathbb{E}_{\boldsymbol{z}\sim q_{\psi_{\text{en}}(\boldsymbol{z}|\boldsymbol{x}_i)}}(\log p_{\phi_{de}}(\boldsymbol{x}_i|\boldsymbol{z})) = \mathcal{G} \quad (3)\end{aligned}$$

is a lower bound (namely the evidence lower bound $\mathcal{G}$, ELBO) to the likelihood of a data sample $\log p(\boldsymbol{x}_i)$, which we seek to maximize. In general, the used network structure is a deterministic one, i.e. $y = f(x)$. To use the network for the approximation of probablistic distributions, two tricks are applied. First, the encoder model ψ_{en} is used to predict the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}$ for a data sample $\boldsymbol{x}_i$ of the distribution $p(\boldsymbol{z})$, which is prescribed to be a multivariate gaussian (i.e. normal) distribution, i.e. $\boldsymbol{z} \sim \boldsymbol{\mathcal{N}}(\boldsymbol{\mu}, \boldsymbol{\sigma})$. Assuming this, an analytical expression for the Kullback-Leibler-Divergence-Term ($\mathbb{KL}$) in Equation 3 can be found (Odaibo 2019). Then, sampling of a random, normal distributed auxiliary variable $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \mathbb{I})$ is required to obtain samples $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$, where $\odot$ represent element-wise multiplication. Using this, an estimate for the second term in Equation 3, the reconstruction likelihood, can be found. The reconstruction term can be determined from the network structure (Kingma and Welling 2022), however, in our model, we follow the approach of (Martínez-Palomera, Bloom, and Abrahams 2020) and use the negative mean squared error between prediction and ground truth, $\mathrm{MSE}(\hat{\boldsymbol{x}}, \boldsymbol{x})$ as representation of the reconstruction likelihood. Taking this together, the evidence lower bound for our network becomes for a training sample $\boldsymbol{x}_i$ with L samples in latent space

$$\begin{aligned}\mathcal{G} \approx \sum_{j=1}^{\dim(\boldsymbol{z})} \frac{1}{2}\Big[1 + \log(\sigma_j^2) - \sigma_j^2 - \mu_j^2\Big] \\ - \beta\ \mathrm{MSE}(\frac{1}{L}\sum_{l=0}^{L} \hat{\boldsymbol{x}}_{i,l}, \boldsymbol{x}_i) \qquad (4)\end{aligned}$$

which should be maximized. When minimizing in a Optimizer, we should do this with $\mathcal{L} = -\mathcal{G}$. Furthermore, the hyperparameter β is added to help disentangling the latent space distribution $\boldsymbol{z}$ (Burgess et al. 2018). Finally (Martínez-Palomera, Bloom, and Abrahams 2020) introduce the physical parameters $\boldsymbol{\theta}$ as inputs for all sub-models.

To use the Variational Autoencoder as a generative model, the representation of the Modelica model parameters $\boldsymbol{\theta}$ within the latent space must be traceable, which is why (Zhang and Mikelsons 2022) added a regression model in a Teacher-Student Architecture. The regression model tracks $\boldsymbol{\theta}$ in latent space, i.e. $\boldsymbol{\mu}_{\mathrm{re}}, \boldsymbol{\sigma}_{\mathrm{re}} = \varphi_{\mathrm{re}}(\boldsymbol{\theta})$. MSE-losses of this model are added to the loss-function to train all models simultaneously. Further details of the implementation can be found in (Zhang and Mikelsons 2022). The overall objective function becomes with this for a batch size N

$$\begin{aligned}(\psi_{\mathrm{en}}, \phi_{\mathrm{de}}, \varphi_{\mathrm{re}}) = \operatorname{argmin} \sum^{N} \quad \mathcal{L} \ + \ \mathrm{MSE}(\boldsymbol{\mu}, \boldsymbol{\mu}_{re}) \\ + \ \mathrm{MSE}(\boldsymbol{\sigma}, \boldsymbol{\sigma}_{re}) \quad (5)\end{aligned}$$

Finally, a well-trained PELS-VAE can replace the physical model by determining the latent space representation of the physical parameters with the regressor model (6): the time-series $\boldsymbol{x}$ is reconstructed after sampling multiple times $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ by the decoder as the mean of the outputs (7).

$$\{\boldsymbol{\mu}_{\mathrm{re}}, \boldsymbol{\sigma}_{\mathrm{re}}\} = \varphi_{\mathrm{re}}(\boldsymbol{\theta}) \qquad (6)$$

$$\hat{\boldsymbol{x}} = \frac{1}{L}\sum^{L} \phi_{de}(\boldsymbol{z}) \qquad (7)$$

B Global Sensitivity Analysis

In the following we sketch the idea of Sobol indices. For a more elaborate and mathematical sound introduction we refer to Hart (2018) or the book by Saltelli et al. (2008). A comprehensive implementation of the required functions is available in SALib (2023).

Consider X to be a real continuos random variable with probability density function $p_X(x)$, such that $\int_{-\infty}^{\infty} p_X(x)dx = 1$. X can be thought of as a specific measurement setup, giving a measured value x each time the experiment is carried out. If a large number of experiments is conducted and the obtained results x are collected into a histogram, then this histogram resembles the probability density distribution $p_X(x)$, that characterizes the experiment X. The probability P of measuring x inside the interval $[T_1, T_2]$ is $P(x \in [T_1, T_2]) = \int_{T_1}^{T_2} p_X(x)dx$. We define the **mean** μ_X of x as $\mu_X = \mathbb{E}_{p_X}[X] = \int_{-\infty}^{\infty} x \cdot p_X(x)dx$. The **variance** of x is defined as $\mathrm{Var}(X) = \mathbb{E}_{\shortmid\mathbb{X}}[(X - \mathbb{E}(X))^2] = \int_{-\infty}^{\infty} (x - \mu_X)^2 \cdot p_X(x)dx$. For a function $f(X)$ of the random variable X one can equivalently define its mean value $\mu_F = \mathbb{E}_{p_X}[f(X)] = \int_{-\infty}^{\infty} f(x) \cdot p_X(x)dx$ and its variance $\mathrm{Var}_{p_X}(F) = \int_{-\infty}^{\infty} (f(x) - \mu_F)^2 \cdot p_X(x)dx$.

Now assume that a system model is evaluated on a fixed scenario (=fixed time series of boundary conditions) for different variations of its model parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_r)$. If we look at a specific output y_t of the model at a specific timestep t then we can interpret our model as a map $f : \boldsymbol{\theta} \to y_t(\boldsymbol{\theta})$. Now assume for a moment that the parameters $\boldsymbol{\theta}$ are an r-dimensional random variable Θ with known probability density function $p_{\boldsymbol{\Theta}}$ and f to be square integrable. Then it can be shown (Sobol 1993) that

$$\begin{aligned} f(\boldsymbol{\theta}) &= f_0 + \sum_{i=1}^{r} f_i(\theta_i) + \sum_{1 \le i < j \le r} f_{i,j}(\theta_i, \theta_j) \\ &\quad + \ldots + f_{1,2,\ldots,r}(\theta_1, \theta_2, \ldots, \theta_r) \\ &= f_0 + \sum_{k=1}^{r} \sum_{|\boldsymbol{u}|=k} f_{\boldsymbol{u}}(\boldsymbol{\theta}_{\boldsymbol{u}}) \qquad (8) \end{aligned}$$

where in the last line we summarize the previous line by the sum over the multi label $\boldsymbol{u}$ representing all possible subsets $\boldsymbol{u} \subseteq \{\theta_1, \ldots, \theta_r\}$ having $|\boldsymbol{u}| = k$ elements. Moreover we have the special expectation value functions

$$\begin{aligned} f_0 &= \mathbb{E}_{p_{\boldsymbol{\Theta}}}[f(\boldsymbol{\theta})] \\ f_i(\theta_i) &= \mathbb{E}_{p_{\boldsymbol{\Theta}}}[f(\boldsymbol{\theta})|\theta_i] - f_0 \\ f_{i,j}(\theta_i, \theta_j) &= \mathbb{E}_{p_{\boldsymbol{\Theta}}}[f(\boldsymbol{\theta})|\theta_i, \theta_j] - f_i(\theta_i) - f_j(\theta_j) - f_0 \\ &\vdots \\ f_{\boldsymbol{u}}(\boldsymbol{\theta}_{\boldsymbol{u}}) &= \mathbb{E}_{p_{\boldsymbol{\Theta}}}[f(\boldsymbol{\theta})|\boldsymbol{u}] - \sum_{k=1}^{|\boldsymbol{u}|-1} \sum_{\substack{\boldsymbol{v} \subset \boldsymbol{u} \\ |\boldsymbol{v}|=k}} f_{\boldsymbol{v}}(\boldsymbol{\theta}_{\boldsymbol{v}}) - f_0 \end{aligned}$$

with the conditional expectation values $\mathbb{E}_{p_{\boldsymbol{\Theta}}}[f(\boldsymbol{\theta})|\boldsymbol{u}] = \int_{-\infty}^{\infty} f(\boldsymbol{\theta}) p_{\boldsymbol{\Theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta}_{\boldsymbol{\theta}\setminus\boldsymbol{u}}$, where $\boldsymbol{\theta} \setminus \boldsymbol{u}$

is the complement of $\boldsymbol{u} \subseteq \{\theta_1, \ldots, \theta_r\}$. Now if all elements $\{\theta_1, \ldots, \theta_r\}$ are statistically independent then it can be shown (Sobol (1993)) that

$$\mathrm{Var}(f(\boldsymbol{\theta})) = \sum_{k=1}^{r} \sum_{|\boldsymbol{u}|=k} \mathrm{Var}\left(f_{\boldsymbol{u}}(\boldsymbol{\theta}_{\boldsymbol{u}})\right) \tag{9}$$

that is the overall variance $\mathrm{Var}(f(\boldsymbol{\theta})) = \mathbb{E}_{p_{\boldsymbol{\Theta}}}\left[(f(\boldsymbol{\theta}) - f_0)^2\right]$ is the sum of all variances of the subset functions $f_{\boldsymbol{u}}$. Then the Sobol index $S_{\boldsymbol{u}}$ of the subset $\boldsymbol{u} \subseteq \{\theta_1, \ldots, \theta_r\}$ measures the relative contribution of $\boldsymbol{\theta}_{\boldsymbol{u}}$ to the total variance of $f(\boldsymbol{\theta})$:

$$S_{\boldsymbol{u}} = \frac{\mathrm{Var}\left(f_{\boldsymbol{u}}(\boldsymbol{\theta}_{\boldsymbol{u}})\right)}{\mathrm{Var}(f(\boldsymbol{\theta}))} \tag{10}$$

Moreover the total Sobol index $T_{\boldsymbol{u}}$ measures the relative contribution of all members of $\boldsymbol{u}$ to the total variance of $f(\boldsymbol{\theta})$:

$$T_{\boldsymbol{u}} = \sum_{\boldsymbol{v} \cap \boldsymbol{u} \neq \emptyset} S_{\boldsymbol{v}} \tag{11}$$

Finally the first order Sobol indices are those $S_{\boldsymbol{u}}, T_{\boldsymbol{u}}$ which are defined on single element subsets $|\boldsymbol{u}| = 1$, that is $\boldsymbol{u} = \{\theta_1, \{\theta_2\}, \ldots, \{\theta_r\}\} = \{\theta_k\}$. Then the Sobol indices S_k, T_k measure the importance or sensitivity of $\mathrm{Var}(f(\boldsymbol{\theta}))$ to $\{\theta_k\}$:

- the first order Sobol index S_k measures the contribution of θ_k
- the total Sobol index T_k measures the contribution of all interactions involving θ_k

The previous discussion is valid for a specific model output y_t at a single time step t so far. The extension to time series $\{t_1, t_2, \ldots, t_n\}$ is straight forward: one simply computes the Sobol indices for all parameters at each time step. In this way one can observe the impact of different model parameters to the chosen output y at several times. This is especially important for transient scenarios, as the sensitivity of the model output to the values of a specific model parameter may vary over time, as can be seen in Figure 5.

DOI
10.3384/ecp204107

Fast Charge Algorithm Development for Battery Packs under Electrochemical and Thermal Constraints with JModelica.org

Alberto Romero[1] Johannes Angerer[1]

[1]Kreisel Electric, Austria, {alberto.romero, johannes.angerer}@kreiselelectric.com

Abstract

Strict operating boundaries on commercial lithium ion cells are defined to mitigate the effect of aging and avoid safety hazards like, the appearance of lithium plating during fast charge, which can lead to internal short circuit and subsequent thermal runaway. Most studies so far have focused on the single cell charging problem because the temperature difference between cells within a battery pack is often considered small, and therefore optimal charging profiles can be extrapolated from single cell investigations. In practice, temperature spread can reach up to 10 K from coldest to warmest points in the pack, and at least 5 K between same position of different cells. With this in mind, a Nonlinear Model Predictive Control (NMPC) scheme is proposed that considers both electrochemical and thermal constraints at pack level, establishing, at least on a theoretical basis, the practical limits of fast charge. An electrochemical cell model and the pack thermohydraulic balance equations were modeled using Modelica. The NMPC implementation is carried out using JModelica.org to find the optimal control actions, and includes the closed loop control problem on a high fidelity plant model. We demonstrate how active thermal management, i.e., controlling the fluid inlet temperature, is critical to reducing charging times below 40 min (from 5% to 80% state of charge), and discuss some challenges when using online optimization-based control techniques.

Keywords: Li-ion battery pack, fast charge, constrained control, temperature spread, FMI

1 Introduction

The performance and lifetime of lithium-ion battery packs strongly depend on the operating conditions, typically determined and/or limited by the user's needs and the auxiliary systems, i.e., the thermal management system (TMS) and the battery management system (BMS). Moreover, operating limits and control strategies may change over time to accommodate for changes in the battery state of health (SOH). Therefore, an operating strategy that meets the required performance and lifetime must be established around the individual cells, the packs built upon them, and the subsystems responsible for adjusting the boundary conditions and applying the constraints under which cells operate.

1.1 Compromises between performance and lifetime

The battery operation strategy modifies or adjusts the performance in the short term, for example, tightening the operating power envelope, thus reducing peak temperatures, in order to achieve a desired lifetime (Barreras, Raj, and Howey 2018). Another example of adjusting thermal and electrochemical limits is the so-called extreme fast charge (XFC) (Yang, T. Liu, et al. 2019), where the cell operating temperature is increased to enhance the electrochemical dynamics to ensure safety requirements. The negative effect in lifetime of higher temperatures is compensated by a significant shorter charging times, which is considered a critical requirement in certain applications, like electric vehicles (EV).

1.2 Temperature limits for commercial Li-ion cells

Modern lithium-ion cells can nevertheless operate over a wide range of temperatures, typically from -30°C to 60°C. A common upper limit of commercial cells can be found around 80°C (Groß and Golubkov 2021), while the Department of Energy of the United States (DOE) established the maximum operating cell temperature at 52°C (Keyser et al. 2017). But already within these limits, and especially beyond them, different degradation mechanisms lead to the progressive deterioration of the performance, reducing the life time of the cells beyond practical or economical criteria. According to information summarized in (Keyser et al. 2017), cell lifetime doubles approximately for each 13K temperature reduction: if 10 years of lifetime is achieved operating at 20°C, the same cell under the same current load would last less than 5 years at 35°C.

Early studies on lithium-ion batteries established the ideal operating temperature range between 25°C and 40°C for a *"good balance between performance and life"*, as well as a module to module temperature spread below 5K (A. A. Pesaran 2002). More recently, temperature limitations have been established using different guidelines to improve safety and performance: maximum temperature 40°C, minimum temperature -30°C, maximum (internal cell) temperature difference 10K, and mean temperature between 25°C and 30°C (M. Sievers, U. Sievers, and Mao 2010).

Recent efforts in quantifying the actual thermal performance of battery packs have been done. Wassiliadis et al.

DOI
10.3384/ecp204121

(2022) determined that sensors located at different cells within a battery module of an electric vehicle (with bottom plate liquid cooling) measured a temperature spread below 2 K during a DC fast charging (maximum C-rate below 1C). Given the large cell format of their test, the maximum difference between the coldest and warmest points of the module (i.e., the absolute difference) could indeed be closer to the difference between cell sensors and fluid inlet temperature, which for their fast charge test is a difference of up to 20 K. On a similar but only simulated case, J. Wang et al. (2020) report between 3.5 and 5 K absolute difference during a 2C discharge depending on the design of the cooling channels and the mass flow rate of the fluid. In practice, absolute temperature spread in real packs are likely to reach 10 K, although efforts to keep it below 5 K is the general consensus, whether absolute or cell to cell spread.

1.3 Solving the fast charge problem

Temperature limits during charge may differ significantly from those while discharging due to the possibility of lithium plating. This negative side-reaction usually takes place at low temperatures, but it can also appear at room temperature for moderate to high charging C-rates (Yang and C.-Y. Wang 2018). As indicated by Yang, T. Liu, et al. (2019), it is desirable to relax the upper temperature limits while charging in order to improve performance at the expense of a marginally higher aging to ensure safety.

The problem fast charge (i.e., how to tackle its complexity and produce safe and fast charge profiles) has been addressed in the literature with different methods, and today vehicle manufacturers have developed practical approaches that consider not only the battery limits, but also the TMS, BMS, on-board converters, power grid (and charger), the local environmental conditions and the user driving needs. The scientific literature has mainly addressed the fast charge problem at cell level (for an application in Modelica, see Romero, Goldar, and Garone (2019)), but studies at pack level that address the effect and limits of TMS are less abundant.

With exclusive focus on cell level fast charge, recent efforts on cell modelling in various spatial and physic domains have led to the conclusion that Li-ion cells can be safely charged below 20 min (0-80% SOC). Frank et al. (2022) established (simulation results only) a theoretical minimum of 18 min for 18650 and 21700 cylindrical cells with conventional tab design, and 13 min for the larger 4680 format with tabless technology; according to the authors, cooling limitations bring the values closer to 20 min and 16 min, respectively. However, the anode potential constraints are set to 0 mV, which leaves no safety margin for the possibility of lithium plating. With the purpose of avoiding lithium plating through a safety buffer (e.g., 20 mV) Yin and Choe (2020) optimized a combined fast charge profile with periodical discharge pulses that favour lithium stripping, i.e., the recovery of already plated lithium. Together with offline and online optimization methods, which include the selection of the optimal temperature boundary, the authors prove experimentally that 18 min is possible (0-80%) with lifetime degradation similar to 1C CCCV (1C constant current, followed by constant voltage) protocol (47 min, 0-80%). It is in general acknowledged, nevertheless, that these fast charge speeds are hardly attainable at pack level, where cell heterogeneities and challenges cooling technologies play a critical role (Tomaszewska et al. 2019).

Modelica has seen a growing number of libraries and studies dedicated to battery systems. The reader is referred to validated libraries reported in Dao and Schmitke (2015), Uddin and Picarelli (2014), Gerl et al. (2014), Bouvy et al. (2012), Brembeck and Wielgos (2011), Einhorn et al. (2011), and Janczyk et al. (2016), as well particular applications on fuel economy (Batteh and Tiller 2009; Spike et al. 2015), thermal management (Bouvy et al. 2012), cell modelling and coolant analysis (Krüger, M. Sievers, and Schmitz 2009),and battery aging (Gerl et al. 2014; Stüber 2017). More recently, (Groß and Golubkov 2021) developed a comprehensive Li-ion library that includes not only electrical cell models, but also thermal runaway (TR) and propagation dynamics, i.e., equations that capture the chemical reactions once an onset temperature is reached.

Completing the single cell level optimal charging analysis presented in (Romero, Goldar, and Garone 2019), this work addresses the limits of fast charge at pack level on an immersion cooled battery with dielectric fluid under electrochemical and thermal constraints. Such cooling approach puts the fluid in direct contact with the cells, which results in higher heat transfer compared to indirect cooling. We make use of Model Predictive Control (MPC) (Camacho and Alba 2013), implemented using the tool JModelica.org (Andersson et al. 2011; Magnusson and Åkesson 2015). A validated functional mock-up unit (FMU) (Blochwitz et al. 2011) is used as a plant model, while a simplified, yet nonlinear model of the pack written in the Modelica language with the same inputs (current, fluid flowrate and inlet temperature) is considered.

The reminder of the paper is organized as follows. Section 2 describes the electrochemical cell model used to model the internal states associated to lithium plating. Section 3 introduces the proposed MPC scheme, describing the cost function, the prediction model, and the plant model. The first part of section 4 explores the optimal profiles under a different set of constraints solved as an offline optimization problem, and then presents the NMPC results accompanied with a discussion related to constraints saturation. This paper is closed with the conclusion section completed with future paths to be investigated.

2 Electrochemical cell model

To support its development activities around battery pack design and simulation, Kreisel Electric (2023) has been working with different Li-ion cell model paradigms, including Equivalent Circuit Models (ECM), Equivalent Hydraulic Model (EHM), Single Particle Model (SPM), and higher level detail models like the P2D Neuman-Fuller-Doyle model. For most of the electrothermal simulations, we rely on ECM-based battery packs, and resort to the light weight EHM when some electrochemical state information is needed, for example in fast charge analyses, the focus of the present paper.

The EHM is based on the original work of (Manwell and McGowan 1993), where the hydraulic analogy is used to describe the dynamics of charge moving between volumes of active material. One recent use of this analogy on Li-ion batteries was proposed by Couto et al. (2016), although derivations of similar models can be found in different sources (Y. Li et al. 2019). The EHM is equivalent to the second order Padé approximation and valid for current pulses with frequencies below 0.5mHz (0.002rad/s) (Forman et al. 2011). Knowing that the 1C/1C cycle results in a frequency of 0.14mHz (charge and discharge included, 1h long each), aging protocols including high charging, steady currents of up to 3.6C fall well under the validity range of the EHM to accurately predict lithium plating.

Consequently with the model choice, the following assumptions must be considered:

1. 0D electrochemical and thermal dynamics
2. Homogeneous behaviour in electrode and separator
3. Fast positive electrode dynamics
4. Constant lithium concentration in electrolyte
5. Temperature dependent exchange current density
6. Heat transfer dominated by side liquid cooling

The EHM considers two electrochemical states, the bulk concentration and the surface concentration, in representative solid particles of the positive and negative electrodes. They are normalized with the maximum concentration ($c_{\mathrm{s,max}}$) and denoted by SOC and CSC respectively. The model considers as input the normalized current (I) using the electrode area (A_{cell}), to provide a form factor independent calculation.

$$\frac{d\,\mathrm{SOC_n}}{dt} = -\gamma I \tag{1}$$

$$\frac{d\,\mathrm{CSC_n}}{dt} = \frac{g}{\beta(1-\beta)}(\mathrm{SOC_n} - \mathrm{CSC_n}) - \frac{\gamma}{1-\beta} I \tag{2}$$

$$\mathrm{SOC_p} = \rho\,\mathrm{SOC_n} + \sigma \tag{3}$$

$$\mathrm{CSC_p} = \mathrm{SOC_p} \tag{4}$$

$$V = U_{\mathrm{p}} - U_{\mathrm{n}} + \eta_{\mathrm{p}} - \eta_{\mathrm{n}} - (R_{\mathrm{f}} + R_{\mathrm{cc}} A_{\mathrm{cell}}) I \tag{5}$$

$$V_{\mathrm{n}} = U_{\mathrm{n}} + \eta_{\mathrm{n}} \tag{6}$$

$$\eta_{\mathrm{p,n}} = \frac{RT}{\alpha F} \sinh^{-1}\left(\frac{\theta_{\mathrm{p,n}} I}{\sqrt{\mathrm{CSC_{p,n}}(1-\mathrm{CSC_{p,n}})}} \right). \tag{7}$$

Table 1 summarizes the most relevant model parameters and exact or reference values for energy cells (C.-H. Chen et al. 2020). The actual values of such parameters used in this work are not disclosed. Moreover, since the diffusion dynamic of the cathode is assumed to be orders of magnitude faster, only the negative electrode is modelled.

Table 1. Cell model parameters

Parameter	*Units*	Value
Particle radius, R_{s}	[μm]	5
Electrode thickness, l	[μm]	80
Diffusion coefficient, D	[m^2/s]	1e-15
Active material vol. fraction, ε	%	75
Specific interfacial area, a	[m^2/m^3]	45e3
Effective reaction rate, r_{eff}	[$\frac{A}{m^2}(\frac{m^3}{mol})^{1.5}$]	7e-6
Maximum concentration $c_{\mathrm{s,max}}$	[mol/m^3]	30e3
Electrolyte concentration, c_{e0}	[mol/m^3]	1200

The relationships of these parameters with the proper parameters of the EHM system are the following

$$\gamma = \frac{3}{R_{\mathrm{s}}\, a\, F\, l\, c_{\mathrm{s,max}}} \quad g = \frac{147}{20}\tau \quad \beta = \frac{7}{10}$$
$$\tau = \frac{R_{\mathrm{s}}^2}{D} \qquad a = 3\frac{\varepsilon}{R_{\mathrm{s}}} \qquad \theta = \frac{1}{2\, a\, l\, r_{\mathrm{eff}}\, c_{\mathrm{e0}}^{1/2}\, c_{\mathrm{s,max}}}.$$

For more information regarding the cell electrochemical models and the equations used in this paper the reader is referred to Romero, Goldar, and Garone (2019), Dao and Schmitke (2015), Chaturvedi et al. (2010), and M. Sievers, U. Sievers, and Mao (2010).

The thermal model assumes lumped properties collapsing on the cell centre, i.e., the warmest area. The thermal resistance consists of a serial sum of convection and conduction terms (external and internal heat flow respectively)

$$m_{\mathrm{cell}}\, C_{\mathrm{p,cell}} \frac{dT}{dt} = i(V - (U_{\mathrm{p}} - U_{\mathrm{n}}) + T(\frac{\partial U_{\mathrm{p}}}{\partial T} - \frac{\partial U_{\mathrm{n}}}{\partial T})) \quad - \frac{1}{R_{\mathrm{th}}}(T - T_{\mathrm{amb}}) \tag{8}$$

$$A = \pi D H_{\mathrm{c}} \quad R_{\mathrm{th}} = (\frac{1}{4\pi H_{\mathrm{c}} k} + \frac{\ln(D_{\mathrm{can}}/D))}{2\pi H_{\mathrm{c}} k_{\mathrm{can}}} + \frac{1}{hA}) \tag{9}$$

where R_{th} is the thermal resistance of the lumped thermal model of the cell, h is the heat transfer coefficient w.r.t. the liquid cooled section, and $\frac{\partial U_{\mathrm{p(n)}}}{\partial T}$ define the so called entropic heat of the positive (negative) electrode (Dao and Schmitke 2015). We note here that this model approximates the behaviour of an infinite cylinder with homogeneous heat generation. In reality, the active cooled length is limited to a portion of the total height of the cell, while the rest is cooled passively by natural convection with the surrounding air. The can conductivity is high enough, and its thickness is so small, as to neglect its contribution in the total resistance. The entropic heat, as well as the heat transfer to the air, are also considered negligible. Thus, the simplified model can be reduced to

$$m_{\text{cell}}\, C_{\text{p,cell}}\, \frac{dT}{dt} = i\,(V - (U_{\text{p}} - U_{\text{n}})) - U\,A\,(T - T_{\text{amb}}), \tag{10}$$

where U is the so called overall heat transfer coefficient.

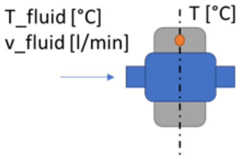


Figure 1. Single Cell thermal model.

3 Model Predictive Control scheme

MPC has been selected as a control paradigm to adjust the inputs, denoted as $u(t)$, which in the general case of a battery pack operation with liquid cooling consists of current, fluid flowrate and fluid inlet temperature. Additional, non-manipulated inputs or disturbances, can be the ambient temperature or the parasitic loads connected to the battery pack. For simplicity, we neglect the effect of the latter, and limit the control inputs to the battery current and the fluid inlet temperature. Moreover, we consider that internal states are observable in practice, but in a real implementation a well tuned estimation method (e.g., Kalman-Filter) must be used.

The nature of the system is non-linear, not only from the coupling between electrochemical and thermal model (the heat source is proportional to i^2, i being the current of the cell or pack), but also because the product of flowrate (considered however constant in the present work) and temperature difference in the heat exchanged. The latter can be simplified for the single cell case, and decoupling and linearisation of the state space system could be solved in a decentralized fashion as reported in Romero, Goldar, Couto, et al. (2019). Therefore, in general, nonlinear solvers are needed in the optimization problem, specially when pack-level control is considered.

3.1 Optimization problem

The on-line nonlinear optimization problem subject to constraints that can be written as

$$\begin{aligned} \min_{u(t)} \quad & \int_{t_0}^{t_\text{f}} [(\text{SOC}(t) - \text{SOC}_{\text{ref}})^2 + k_{\text{T}}(\text{T}(t) - \text{T}_{\text{init}})^2]dt \\ \text{s.t.} \quad & \text{model dynamicsconstraints} \\ & \text{electrochemical constraints} \\ & \text{thermal constraints.} \end{aligned} \tag{11}$$

The first row in Equation 11 is the integral cost over the horizon determined between t_0 and t_f. Its first term penalizes the difference between the SOC at time t and the desired reference SOC$_{\text{ref}}$. An additional cost term is added to bring the cell/pack temperature to a desired value for storage or before discharge begins. In addition to the model dynamics itself, two type of constraints are considered: electrochemical constraints on the anode potential (V_n) to avoid lithium plating, and thermal constraints including maximum and minimum cell temperature ($T_{\max}$, $T_{\min}$), as well as maximum temperature spread within the pack (T_{spread}).

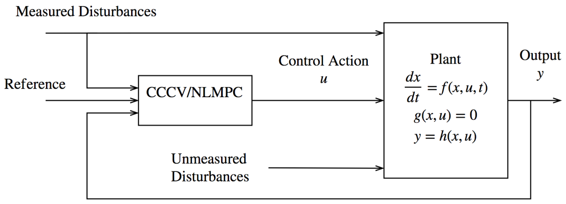


Figure 2. Model Predictive Control Scheme

3.2 Prediction Model

The optimization class extends the pack model `PackEHMT`, i.e., the prediction model, which includes all state and output dependencies with the input variables. The following listing is part of the optimization class `EHMTVpack_OptMPC`, which includes a constraint section that defines the limits of operation for the cell and pack.

Listing 1. Optimization class `EHMTVpack_OptMPC`

```
optimization EHMTVpack_OptMPC (
   objectiveIntegrand =
     (SOC - SOC_ref)^2 + 1e-8*(T-T_init)^2,
   startTime = 0, finalTime = 1000)

  extends PackEHMT(
    CSC(fixed=true), CSCn_0=0.05,
    SOC(fixed=true), SOCn_0=0.05,
    T(fixed=true), Tm(fixed=true),
    Tf(fixed=true), T_init = 298.15);
  // ...

  // Example of limit values:
  parameter Real SOC_ref = 0.80;
  parameter Real SOC_max = 0.65;
  parameter Voltage V_max = 4.2;
  parameter Temperature T_max = 450;
  parameter Temperature Tf_max = 450;
  parameter Temperature Tspread_max = 10;
  parameter Temperature DTf_low = 10;
  parameter Temperature DTf_high = 25;
  parameter Current i_max = 20;
  parameter Voltage Van_min = 0.1;

equation
  U_n = ...;
  U_p = ...;
  Van = R * T / alpha / F * Modelica.Math.
      log(theta_n * (I) / sqrt(CSC * (1 -
      CSC)) + sqrt(1 + (theta_n * I / sqrt(
      CSC * (1 - CSC))) ^ 2)) + U_n;

constraint
  SOC <= SOC_max;
  CSC <= SOC_max;
  SOC >= 0.0001;
  CSC >= 0.0001;
  Van >= Van_min;
  Tfin >= T_amb - DTf_low;
  Tfin <= T_amb + DTf_high;
  (Tf - T) <= Tspread_max;
  -(Tf - T) <= Tspread_max;
  i <= i_max;
  T <= T_max;
  Tf <= Tf_max;
  V <= V_max;

end   EHMTVpack_OptMPC;
```

The core model of the battery pack is the cell model `CellEHMT_base`, where the main electrical and electrochemical parameters and equations are defined. The only exception is lack of a cell temperature model. This and the temperature balances of the full pack form the model class `PackEHMT` as shown in Listing 2. Figure 3 shows the approach to simplify the pack equations, where the cells between the first and last are lumped into a single thermal node. Despite its simplicity, this approximation allows us to obtain an inlet fluid temperature for the last cell, and yields a level of fidelity for the pack model sufficiently accurate for an MPC scheme.

Listing 2. Pack model class `PackEHMT`

```
model PackEHMT

  extends CellEHMT_base; // includes state
      variable T
  //...
  parameter Integer nmid = 400 "Cells in
      the middle";
  Power Q(start = 0);
  Power Qm(start = 0);
  Power Qf(start = 0);
  Temperature Tf2(start = T_init);
  Temperature Tf3(start = T_init);
  Temperature Tf4(start = T_init);
  Temperature Tm(start = Tm0) "Temperature
      cells in the middle";
  Temperature Tf(start = Tf0) "Temperature
      last module cell";
  parameter MassFlowRate mfr = 0.001 "Mass
      flow rate";
  inputTemperature Tfin "Inlet fluid
      temperature";

equation
  //...
  V = U_p - U_n + ...;
  Q = 1/(1/(h * A) + 1/G_rad)*(T-(Tfin+Tf2)
      /2);
  Q = mfr*Cpf*(Tf2-Tfin);
  Qm = nmid*1/(1/(h * A) + 1/G_rad)*(Tm-(
      Tf3+Tf2)/2);
  Qm = mfr*Cpf*(Tf3-Tf2);
  Qf = 1/(1/(h * A) + 1/G_rad)*(Tf-(Tf4+Tf3
      )/2);
  Qf = mfr*Cpf*(Tf4-Tf3);
  P_loss =  i * (V - U_p + U_n);
  der(T) = (P_loss - Q)/(M*Cp);
  der(Tm) = (nmid*P_loss - Qm)/(M*Cp*nmid);
  der(Tf) = (P_loss - Qf)/(M*Cp);

end PackEHMT;
```

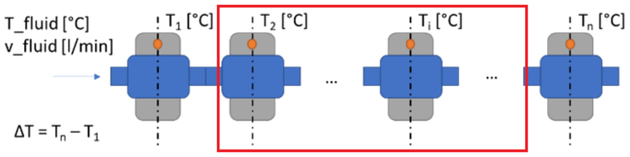


Figure 3. Simplified pack model with lumped dynamics within the rectangle.

3.3 Plant Model

For the plant model, a high definition, 1D battery pack nonlinear model is used. The pack consists of several stacks connected hydraulically in parallel and electrically in series. Each stack consist of several modules, made of staggered groupings of 36 cells secured within a cooling enclosure, which allows for a dielectric fluid to circulate in contact with the surface of the cells using immersion cooling technology (Kastler and Menzl 2021). More information about a similar stack can be found in the work of Kasper et al. (2023).

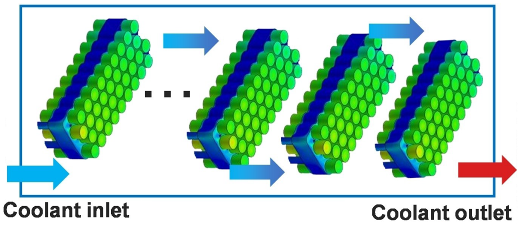


Figure 4. Stack formed by a variable number of modules

The maximum voltage of the pack's energy content is 60 kWh. A detailed view of an arbitrarily long stack is shown in Figure 4. This pack model, of which an FMU was created and integrated in the main simulation loop, uses a validated ECM cell model without aging dynamics, with a discretized model consisting in 9 sub-volumes (3 divisions in radial direction and 3 in axial).

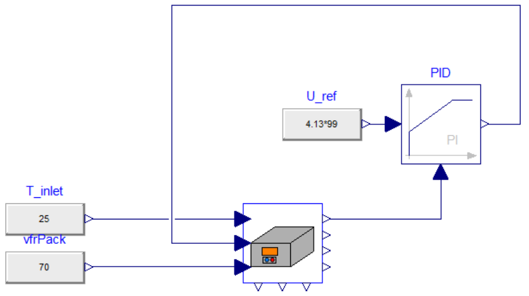


Figure 5. Pack model tested with a CCCV charge using a limited PI

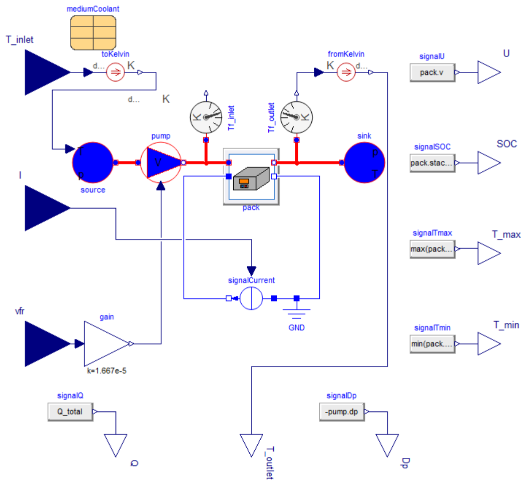


Figure 6. Pack model interface detail

4 Case studies

To illustrate what an optimal operation strategy looks like and how it is calculated, a series of optimization problems are solved, first solving the off-line, fast charge problem, and then a closed-loop NMPC with state feedback on a realistic plant model. We are concerned in this work with the optimal charge, i.e., the overall control strategy including the discharge phase of the cycle is part of ongoing investigations. The main control parameters needed in JModelica.org are shown in Table 2.

First, the optimal constrained fast charge profile of a 5 Ah, 21700 format cylindrical single cell with immersion cooling is computed and compared with standard charging protocols with passively cooled cell. The optimization is carried out under several constraints involving voltage, temperature, and electrochemical limits, that prevent premature aging and lithium plating. Subsequently, the optimal profile for the battery pack, based on the same cell, is calculated without and with additional temperature spread limit. In all cases the same `EHMTVpack_OptMPC` class, where only the constraints (upper and lower values) are adjusted for each of the cases described.

Table 2. NMPC controller setup

Variable	*Value*	*Units*
t_f	1000	[s]
SOC_{ref}	0.665	[-]
n_e	100	[-]
n_{cp}	1	[-]
H	1000	[s]
Δt_{MPC}	10	[s]
Δt_{sim}	1	[s]
solver	IPOPT	

Finally, the NMPC scheme proposed in the previous section is used to determine the impact of imperfect state feedback and control horizon on the constraint satisfaction and the controller performance.

4.1 Single cell optimal charge

We begin by comparing the conventional charge protocol in three basic situations: passive cooling with 1C charge CCCV, and immersion cooling at different C-rates: 1C/2C CCCV. Passive cooling is defined by a boundary condition defined by the overall heat transfer coefficient (U) equal to 1 W/m^{2}K over the whole surface of the cell, which is the case of a slightly insulated cell subject to natural convection heat transfer. For immersion cooling, a value of $U = 200$ W/m^{2}K has been chosen.

4.1.1 Cooling system comparison

Figure 7 illustrates a typical 1C-CCCV charge profile of a commercial Li-ion cell. Starting at 7.5% SOC, it reaches 80% in 45 min. From top to bottom, the subplots contain state of charge and critical surface concentration (expressed as percentage at target SOC stoichiometric), current and voltage, anode potential, and cell temperature. Passively cooled cells experience high peak temperatures during charge. Figure 7 shows that, for slightly insulated cells, temperature reaches 25 K above the ambient temperature (fluid at 20°C) at the end of the CC phase. On the positive side, the anode potential remains above 42 mV thanks to improved dynamics at higher temperature.

Immersion cooling, as shown in Figure 8, improves thermal management, i.e., the ability to bring the temperature of cells to a desired reference. There is no decrease in charging time compared with passive cooling when charging at 1C because temperature or electrochemical limits are not achieved in both passive or immersion. Special care should be taken not to cross the anode potential limits at lower temperatures and higher currents. This is illustrated in Figure 8, showing an anode potential margin of 18 mV. Depending on the expected fidelity of the electrochemical model, this may not be sufficient to ensure total lithium plating avoidance. In this case 4 K of peak cell temperature above ambient is achieved. Although a higher fluid flowrate is possible, the temperature reduction due to improved heat transfer will lead to a further drop in anode potential, and therefore higher risk of plating.

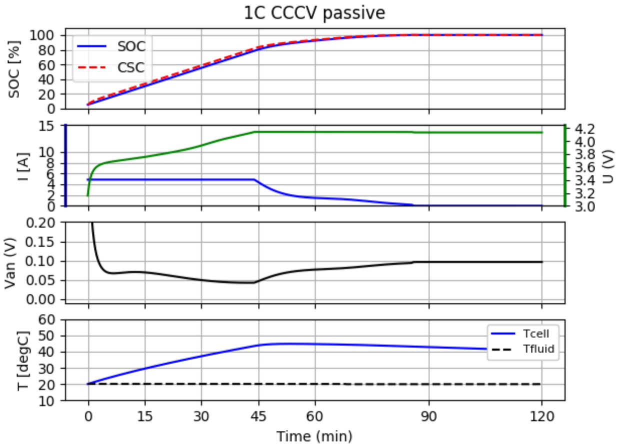


Figure 7. 1C-CCCV, single cell, passive cooling (T_{fluid} refers here to the environmental temperature)

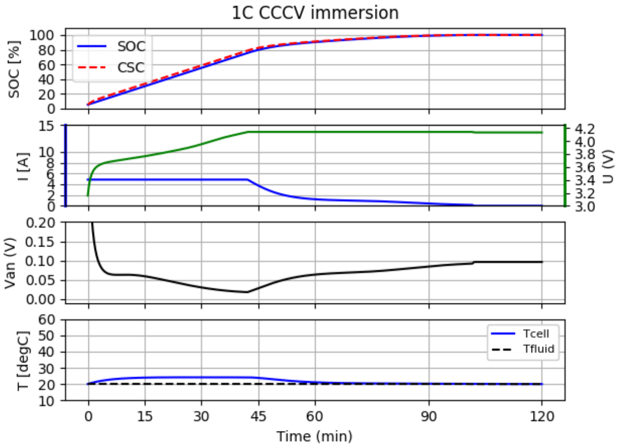


Figure 8. 1C-CCCV, single cell, immersion cooling

4.1.2 C-rate comparison

An increase in C-rate improves charging time significantly, from 45 min at 1C to 26 min at 2C (from 7.5% to 80% SOC). Only 5 mV of margin w.r.t. plating, and 33°C peak temperature set a limit in performance for safety and lifetime, but again model inaccuracies and cell-to-cell variations at BOL would encourage additional improvements to this profile, specially when considering charge at pack level.

4.1.3 Optimal constrained profile, 2C maximum C-rate

When constraints are present (40 mV for the anode potential, 45°C), the only way to reduce the charging time is to increase the fluid temperature so that the cell properties are enhanced. Figure 10 shows the calculated optimal current and temperature profile, which brings the charging time to 28 min, only 8% higher than the 2C-CCCV profile. The fluid temperature can vary +25/-10 K around the nominal value 20°C. It is worth noting the optimal trajectory of the fluid temperature, which brings the temperature of the cell to the maximum level after a series of swings, and finally brings the cell to the nominal value even before the charge is completed. This of course depends on the controller setup, i.e., the weightings of the cost function. It must be noted that the time derivatives of the fluid temperature are limited in practice, for example, by the heating/cooling devices mounted on the vehicle. The consideration of such limits are beyond the scope of this work. If the fluid temperature is kept at 20°C at all times, the charging time increases to 44 min, just above the 1C-CCCV protocol.

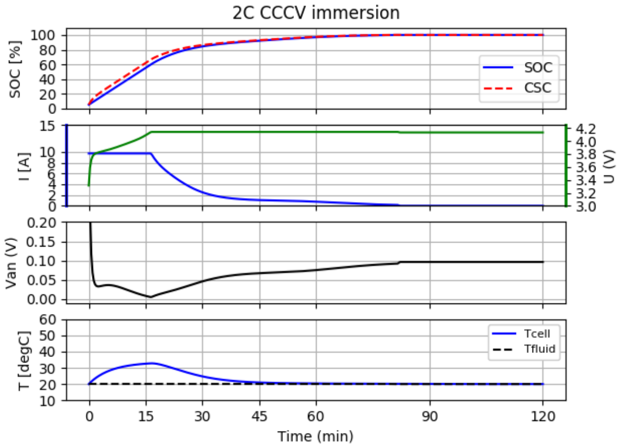


Figure 9. 2C-CCCV, single cell, immersion cooling

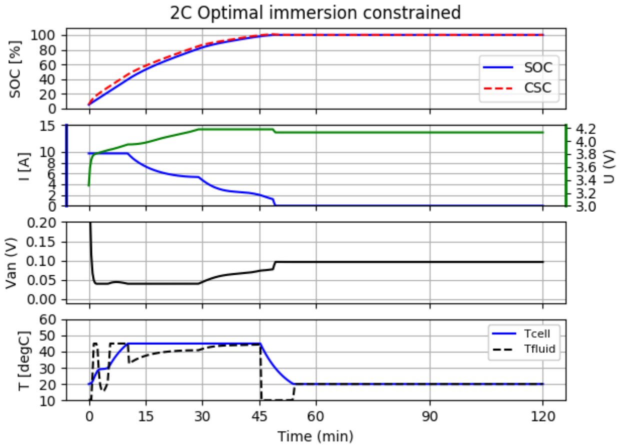


Figure 10. 2C Optimal profile, immersion cooling with temperature control

4.2 Pack level offline charge optimization

The obtained profiles are valid only for a pack where all the cells face the same boundary conditions, which in practice is generally not possible. The optimization algorithm can control the temperatures of all the cells (provided the first and last cell hold the extreme temperature values), as well as the temperature spread in the pack. For simplicity, we control the first and last cell's temperature, as well as the absolute value of their temperature difference (T_{spread}).

4.2.1 No inlet fluid temperature control, no temperature spread control

We present first the case in which a pack is charged and only the current is manipulated. The only constraint that is not considered is the temperature spread. The maximum temperature is nevertheless not active, while the anode potential constraint is active for most of the charge, before the CV phase begins at about 53 min. The charging time (7.5%-80% SOC) is 44 min.

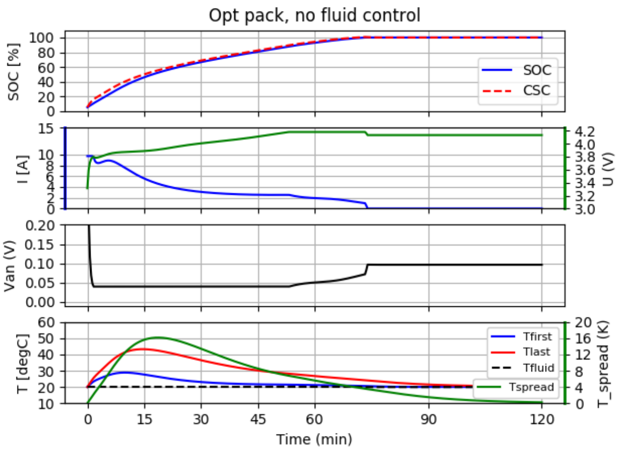


Figure 11. 2C Optimal profile, immersion cooling at pack level without fluid temperature control

4.2.2 Fluid temperature control, temperature spread control option

When the fluid temperature is amenable to manipulation, results become more interesting. Charging time is reduced to 34 min (Figure 12) when temperature spread is not included, and 36 min otherwise (Figure 13), which indicates that controlling temperature spread is marginally difficult if the inlet temperature can be controlled. These values are 21% and 29% higher than the single cell case. Incidentally, the maximum cell temperature constraint becomes active at some point due to increased inlet fluid temperature. Further limitation of the pack temperature spread down to 5 K leads to a charging time of 43 min, a 20% increase.

4.3 Pack level NMPC scheme

The results concerning the online fast charge optimization using NMPC are presented in this section. Some implementation details to be taking into account when utilizing this scheme on a real battery pack are also discussed. Figure 14 represents the same offline problem described in the last example (pack level constrained optimization with manipulated fluid temperature), now from a more realistic perspective. It should be noted, notwithstanding, that further limitations in pressure drop and fluid temperature ramps may slow down the overall charging operation.

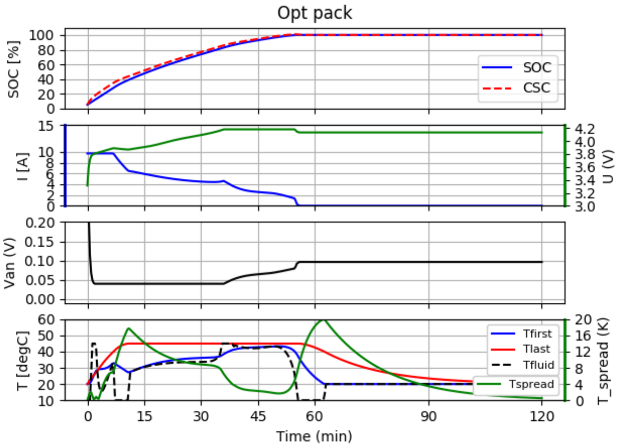


Figure 12. 2C Optimal profile, immersion cooling at pack level with fluid temperature

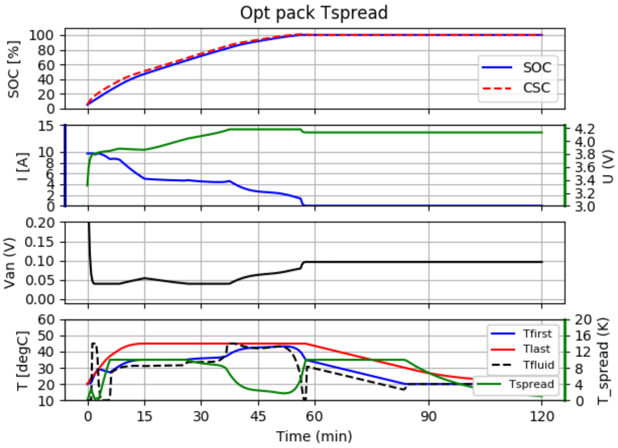


Figure 13. 2C Optimal profile, immersion cooling at pack level with fluid temperature and temperature spread control

The control horizon chosen in this work is 1000s, i.e., 100 steps of 10s each. The total computational time per step remained over the complete integration loop below 2 s for the device used (Windows system, processor Intel i7, 32 GB RAM, overall usage less than 20%). The total charging time (7.5%-80% SOC) is slightly increased up to 37 min. If the control horizon decreases to 100s, the computational time is reduced ten-fold, but the myopia of the controller leads to a charging time of 63 min, not being able to avoid temperature constraint saturation. This highlights the need for sufficient computing power.

Another limitation arises from model inaccuracies in the cool-down part after the charge (beyond 60 min), which leads to an increased temperature spread that would violate the controller's constraints. This helps us introducing how the scheme leads to constraint saturation when feeding back the plant's actual temperatures. Without explicit handling of such saturation, state values of the plant

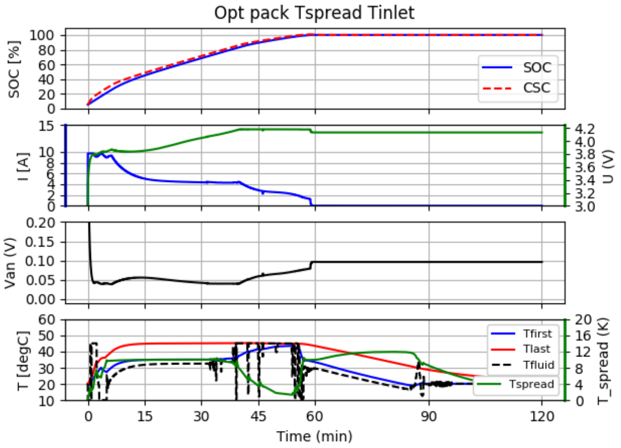


Figure 14. 2C NMPC profile, immersion cooling at pack level with temperature spread and temperature spread control

may initialize the optimization problem from an infeasible point. Correcting the state slightly to always stay within the limits of the saturation is proposed for the maximum temperature, which leads in this problem to satisfactory results, as seen in Figure 14, since plant and prediction models similar dynamics. However, it is clear from this figure that the temperature spread is violated during the cool-down phase. Saturation is resolved by fixing either the minimum or maximum temperature, and afterwards the remaining one considering the limited spread. Not dealing with spread saturation leads to slightly higher charging time (38 min), and future work will be devoted to examine better options to include a robust approach that ensures feasibility.

5 Conclusions

Fast charging of battery packs present a rich set of design and operational challenges. In this paper, it has been shown that active thermal management is critical to achieve competitive charging speeds in combination with optimization-based control algorithms. Unlike previous works tackling only single cell level operation, this work has demonstrated that the objective of less than 20 min pack-level fast charge (0-80%) is not yet attainable. In fact, we proved that even with high-effective immersion cooling and optimization-based algorithms, the charging time from cell to pack is expected to increase by more than 40%. In summary, further improvements from the current state-of-the-art on cell design, cell-to-pack integration, and thermal management are needed. Ongoing extensions for the current formulation include the addition of a flow-pressure model and a more realistic approach of the available heating/cooling power for thermal management, so that constraints in pack pressure drop, volumetric flowrate, and fluid inlet temperature ramps can be applied.

References

Andersson, Joel et al. (2011). “Integration of CasADi and JModelica. org”. In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 063. Linköping University Electronic Press, pp. 218–231.

Barreras, Jorge Varela, Trishna Raj, and David A Howey (2018). “Derating strategies for lithium-ion batteries in electric vehicles”. In: *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, pp. 4956–4961.

Batteh, John and Michael Tiller (2009). “Implementation of an extended vehicle model architecture in modelica for hybrid vehicle modeling: development and applications”. In: *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*. 043. Linköping University Electronic Press, pp. 823–832.

Blochwitz, Torsten et al. (2011). “The functional mockup interface for tool independent exchange of simulation models”. In: *Proceedings of the 8th international Modelica conference*. Linköping University Press, pp. 105–114.

Bouvy, Claude et al. (2012). “Holistic vehicle simulation using Modelica-An application on thermal management and operation strategy for electrified vehicles”. In: *Proceedings of the 9th International Modelica Conference; September 3-5; 2012; Munich; Germany*. 076. Linköping University Electronic Press, pp. 264–270.

Brembeck, Jonathan and Sebastian Wielgos (2011). “A real time capable battery model for electric mobility applications using optimal estimation methods”. In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 063. Linköping University Electronic Press, pp. 398–405.

Camacho, Eduardo F and Carlos Bordons Alba (2013). *Model predictive control*. Springer science & business media.

Chaturvedi, Nalin A et al. (2010). “Algorithms for advanced battery-management systems”. In: *IEEE Control systems magazine* 30.3, pp. 49–68.

Chen, Chang-Hui et al. (2020). “Development of experimental techniques for parameterization of multi-scale lithium-ion battery models”. In: *Journal of The Electrochemical Society* 167.8, p. 080534.

Couto, Luis D et al. (2016). “SOC and SOH estimation for Li-ion batteries based on an equivalent hydraulic model. Part I: SOC and surface concentration estimation”. In: *2016 American control conference (ACC)*. IEEE, pp. 4022–4028.

Dao, Thanh-Son and Chad Schmitke (2015). “Developing Mathematical Models of Batteries in Modelica for Energy Storage Applications”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. 118. Linköping University Electronic Press, pp. 469–477.

Einhorn, M et al. (2011). “A modelica library for simulation of electric energy storages”. In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 63. Linköping University Electronic Press, pp. 436–445.

Forman, Joel C et al. (2011). “Reduction of an electrochemistry-based li-ion battery model via quasi-linearization and pade approximation”. In: *Journal of the Electrochemical Society* 158.2, A93.

DOI
10.3384/ecp204121

Frank, Alexander et al. (2022). “Impact of Current Collector Design and Cooling Topology on Fast Charging of Cylindrical Lithium-Ion Batteries”. In: *ECS Advances* 1.4, p. 040502.

Gerl, Johannes et al. (2014). “A Modelica Based Lithium Ion Battery Model”. In: *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 096. Linköping University Electronic Press, pp. 335–341.

Groß, Christian and Andrey Golubkov (2021). “A Modelica library for Thermal-Runaway Propagation in Lithium-Ion Batteries”. In: *14th Modelica Conference 2021*.

Janczyk, Leonard et al. (2016). “Validation of a Battery Management System based on AUTOSAR via FMI on a HiL platform”. In: *The First Japanese Modelica Conferences, May 23-24, Tokyo, Japan*. 124. Linköping University Electronic Press, pp. 87–94.

Kasper, Manuel et al. (2023). “Calibrated Electrochemical Impedance Spectroscopy and Time-Domain Measurements of a 7 kWh Automotive Lithium-Ion Battery Module with 396 Cylindrical Cells”. In: *Batteries & Supercaps* 6.2, e202200415.

Kastler, Helmut and Kilian Menzl (2021). “Effective Battery Design and Integration of Cylindrical Cells for High Power Applications”. In: *CTI SYMPOSIUM 2019: 18th International Congress and Expo 9-12 December 2019, Berlin, Germany*. Springer, pp. 283–293.

Keyser, Matthew et al. (2017). “Enabling fast charging–Battery thermal considerations”. In: *Journal of Power Sources* 367, pp. 228–236.

Kreisel Electric (2023). *We drive the future*. http://www.kreiselelectric.com. Accessed: 2023-06-14.

Krüger, Imke, Martin Sievers, and Gerhard Schmitz (2009). “Thermal modeling of automotive lithium ion cells using the finite elements method in modelica”. In: *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*. 043. Linköping University Electronic Press, pp. 1–8.

Li, Yang et al. (2019). “Development of a degradation-conscious physics-based lithium-ion battery model for use in power system planning studies”. In: *Applied Energy* 248, pp. 512–525. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2019.04.143. URL: https://www.sciencedirect.com/science/article/pii/S0306261919308049.

Magnusson, Fredrik and Johan Åkesson (2015). “Dynamic optimization in JModelica. org”. In: *Processes* 3.2, pp. 471–496.

Manwell, James F. and Jon G. McGowan (1993). “Lead acid battery storage model for hybrid energy systems”. In: *Solar Energy* 50.5, pp. 399–405. ISSN: 0038-092X. DOI: https://doi.org/10.1016/0038-092X(93)90060-2. URL: https://www.sciencedirect.com/science/article/pii/0038092X93900602.

Pesaran, Ahmad A (2002). “Battery thermal models for hybrid vehicle simulations”. In: *Journal of power sources* 110.2, pp. 377–382.

Romero, Alberto, Alejandro Goldar, Luis D Couto, et al. (2019). “Fast charge of Li-ion batteries using a two-layer distributed MPC with electro-chemical and thermal constraints”. In: *2019 18th European Control Conference (ECC)*. IEEE, pp. 1796–1803.

Romero, Alberto, Alejandro Goldar, and Emanuele Garone (2019). “A model predictive control application for a constrained fast charge of lithium-ion batteries”. In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. 157. Linköping University Electronic Press.

Sievers, Martin, Uwe Sievers, and Samuel S Mao (2010). “Thermal modelling of new Li-ion cell design modifications”. In: *Forschung im Ingenieurwesen* 74.4, pp. 215–231.

Spike, Jonathan et al. (2015). “Holistic Virtual Testing and Analysis of a Concept Hybrid Electric Vehicle Model”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. 118. Linköping University Electronic Press, pp. 537–545.

Stüber, Moritz (2017). “Simulating a variable-structure model of an electric vehicle for battery life estimation using modelica/dymola and python”. In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. 132. Linköping University Electronic Press, pp. 291–298.

Tomaszewska, Anna et al. (2019). “Lithium-ion battery fast charging: a review”. In: *ETransportation* 1, p. 100011.

Uddin, Kotub and Alessandro Picarelli (2014). “Phenomenological Li ion battery modelling in Dymola”. In: *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 96. Linköping University Electronic Press, pp. 327–334.

Wang, Jianguo et al. (2020). “Effect analysis on thermal behavior enhancement of lithium–ion battery pack with different cooling structures”. In: *Journal of energy storage* 32, p. 101800.

Wassiliadis, Nikolaos et al. (2022). “Quantifying the state of the art of electric powertrains in battery electric vehicles: Range, efficiency, and lifetime from component to system level of the Volkswagen ID. 3”. In: *ETransportation* 12, p. 100167.

Yang, Xiao-Guang, Teng Liu, et al. (2019). “Asymmetric temperature modulation for extreme fast charging of lithium-ion batteries”. In: *Joule* 3.12, pp. 3002–3019.

Yang, Xiao-Guang and Chao-Yang Wang (2018). “Understanding the trilemma of fast charging, energy density and cycle life of lithium-ion batteries”. In: *Journal of Power Sources* 402, pp. 489–498.

Yin, Yilin and Song-Yul Choe (2020). “Actively temperature controlled health-aware fast charging method for lithium-ion battery using nonlinear model predictive control”. In: *Applied energy* 271, p. 115232.

Comparative Study and Validation of Photovoltaic Model Formulations for the IBPSA Modelica Library based on Rooftop Measurement Data

Laura Maier[1] Christoph Nytsch-Geusen[2] Lucas Westermann[2] Kushagra Mathur[2]
Michael Wetter[3] Dirk Müller[1]

[1]Institute for Energy Efficient Buildings and Indoor Climate, E.ON Energy Research Center, RWTH Aachen University, Germany, {laura.maier,dmueller}@eonerc.rwth-aachen.de
[2]Institute for Architecture and Urban Planning, Berlin University of the Arts, Germany, {nytsch,k.mathur}@udk-berlin.de
[3]Lawrence Berkeley National Laboratory, Berkeley CA, USA, mwetter@lbl.gov

Abstract

Domain-overarching system models are crucial to investigate sector coupling concepts. Specifically, the coupling of building and electrical energy systems becomes crucial to integrate renewable energy sources such as photovoltaic power systems (PV). For such interdisciplinary models, Modelica is a suitable language. However, most open-source Modelica libraries are either domain-specific or lack simple-to-parameterize PV models. We close this gap by developing a PV model for the IBPSA Modelica Library. The model comprises two I-V-characteristic models and three mounting-dependent cell temperature models. The I-V-characteristic models follow a single- and two-diodes approach. This study uses measurement data from a rooftop PV system in Berlin, Germany, for validation. The focus lies on comparing the implemented single- and two-diodes approach. Results prove that both models accurately calculate the modules' DC power output and cell temperature.

Keywords: PV, Modelica, Validation, Open-Source

1 Introduction

Interconnected systems facilitate the integration of renewable energy sources resulting in a decrease in CO_2 emissions. One important sector with a high emission reduction potential is the building sector (*2022 Global status report for buildings and construction: Towards a zero-emissions, efficient and resilient buildings and construction sector* 2022). The electrification of buildings' energy systems has proven to be a valuable instrument to integrate renewable energy sources from the grid and, hence, interconnect sectors. Consequently, synergy effects arise from the connection of the two sectors while the system complexity increases. Buildings cannot only exploit renewable energy sources from the distribution grid level but also make use of self-generated electricity by, e.g., photovoltaic power plants (PV). PV is already a common way in practice to integrate renewable energy sources on both a small and large scale. Exemplary PV application fields in the building sector are rooftop PV, facade-integrated PV, and stand-alone PV on district level. While designing building energy systems is already a challenge, it becomes even more complex when also considering potential PV integrations. Here, simulation tools support the design and operation of such interconnected systems.

For building energy systems, Modelica has proven to be a suitable modeling language. In this regard, open-source libraries facilitate the knowledge transfer from research to practice and harmonize the modeling process. As an outcome of the Annex 60 project and the subsequent Project 1 (Wetter, Treeck, et al. 2019), five modeling libraries focusing on building performance simulations started their collaboration. Among the five modeling libraries are one core library and four derivative libraries. The four derivative libraries, namely Buildings (Wetter, Zuo, et al. 2014), BuildingSystems (Nytsch-Geusen, Huber, Ljubijankic, et al. 2013), IDEAS (Jorissen et al. 2018), and AixLib (Maier, Jansen, et al. 2023) share a common core library, called Modelica IBPSA Library. These libraries aim to provide models for all relevant domains in building energy systems, namely, HVAC components and subsystems, building envelope models as well as internal and external boundary conditions. In addition, the derivative libraries partially include relevant electrical components, such as battery energy storage and PV models. However, prior to this development, the core library IBPSA neither includes an electrical package nor a PV model. Consequently, the derivative libraries have developed their own modeling approaches. This work aims at harmonizing the existing work by implementing a PV model in the core library.

The development is motivated by the already existing infrastructure between the core and its derivative libraries as well as the wide selection of building energy systems-related models and unified interfaces. We implement and compare two popular modeling approaches for PV systems, namely, the single- and two-diodes model approach.

DOI
10.3384/ecp204131

While the former is an easy-to-parameterize and more simplified model, the latter is a more accurate representation of the physical behavior of the PV cell. Both models are compared and validated based on real measurement data from a rooftop PV system based on thin-film CIGS modules in Berlin, Germany.

2 Related work

This section gives an overview of existing modeling approaches for PV modules. At first, we define relevant criteria for the developed model. Subsequently, popular modeling approaches are discussed including their advantages and disadvantages. Following, existing Modelica models and libraries are presented.

2.1 Requirements

PV systems interact with building energy systems in various ways. While rooftop PV systems are the most common installation type in building energy systems, standalone PV systems play an important role in the context of district energy systems. Consequently, the focus lies on both PV system types. The present study does not focus on PV systems interacting with the thermal building mass, e.g., facade-integrated PV, despite their potential. In addition, this study aims at developing PV system models for the building design and operating phase focusing on energy performance. Consequently, the overall PV system's output power, rather than a detailed cell analysis, is the main performance metric . Since the models aim at supporting both practitioners and researchers, the parameterization effort should account for limited available information. In addition, the model execution time should be short so that different system configurations can quickly be assessed for annual system performance. To summarize, we define the following criteria for the model(s):

1. Representation of a rooftop or stand-alone PV system.
2. Suitability for energy performance evaluation.
3. Modular modeling approach for simple extensions and adaptations.
4. Simple parameterization based on readily accessible data.
5. Fast model execution for annual simulation.

2.2 Photovoltaic modeling approaches

The present study distinguishes the modeling approaches in electrical and thermal model. We first focus on the electrical modeling and discuss thermal models subsequently.

2.2.1 Electrical modeling approaches: I-V characteristic

In the literature, there exist different approaches to model the electrical characteristics of a PV module, the so-called I-V-characteristic. These approaches can be distinguished into empirical and physics-based models. In the context of this study, the latter are further differentiated into single- and two-diodes models, corresponding to the electrical equivalent circuit that they are based on. For the present study, we compare different techniques regarding their accuracy, computational effort, and parameterization effort. Table 1 shows a qualitative comparison based on the following literature review.

Table 1. Comparison of modeling techniques.

Scheme	*Accuracy*	*Comp. effort*	*Parameter.*
Empirical	+	+	-
Single-diode	0	+	+
Two-diodes	+	- to 0	0

Empirical approaches

Two well-known empirical PV modeling approaches are the Sandia PV Array Performance Model (SAPM) (King, Boyson, and Kratochvill 2005) and the Loss Factors Model (LFM) (Sellner et al. 2012; Sutterlueti et al. 2008). To obtain these models, selected points of the electrical characteristic, are determined based on measurement data. Those points cover the maximum power point (MPP), the short circuit current I_{sc} and the open-circuit voltage V_{oc}. In addition, fitting parameters are calculated based on measurement data that consider the change in the I-V characteristic with a change in irradiation and the cell temperature. Stein et al. (2013) and De Soto, Klein, and Beckman (2006) compare empirical models with numerical single-diode models In these experiments, De Soto, Klein, and Beckman (2006) prove that the empirical models outperform the single-diode approach. Their explanation for this phenomena lies in a variety of experimental data that was used to fit the parameters of the empirical models.

Single-diode approach

In contrast to the empirical models, physics-based models are based on the assumption that PV cells can be described as diode circuits. Two approaches are common: The single- and the two-diodes approach (see Figure 1 and Figure 2). The single-diode model assumes that the PV modules' current I_{PV} can be described as the sum of the photo current I_{ph}, the leakage current I_{sh}, and the dark current I_{d} (see Figure 1). I_{d} is opposed to I_{ph}. I_{d} derives from the Shockley equation

$$I_{\text{d}} = I_{\text{s}}(e^{\frac{U+IR_{\text{s}}}{a}} - 1), \tag{1}$$

where a denotes the modified ideality factor

$$a = \frac{N_{\text{s}}\, n_{\text{I}}\, k\, T_{\text{cell}}}{q}, \tag{2}$$

where I_{d} and a are based on the saturation current I_{s}, the ideality factor n_{I}, the elementary charge q, the Boltzmann constant k, and the cell temperature T_{cell}. R_{s} is the serial resistance resulting in a voltage loss, while R_{sh} is the shunt resistance that leads to the leakage current I_{sh}. The result is the I-V characteristic

$$I = I_{\mathrm{ph}} - I_{\mathrm{d}} - I_{\mathrm{sh}}, \tag{3}$$

$$I = I_{\mathrm{ph}} - I_{\mathrm{s}}(e^{\frac{U+IR_{\mathrm{s}}}{a}} - 1) - \frac{U+IR_{\mathrm{s}}}{R_{\mathrm{sh}}}. \tag{4}$$

These equations have five unknown parameters I_{ph}, I_{s}, a, R_{s}, and R_{sh}, which is why they are referred to as 5p modeling approach. This approach is a compromise between accuracy, parameterization effort, and computational effort (see Table 1).

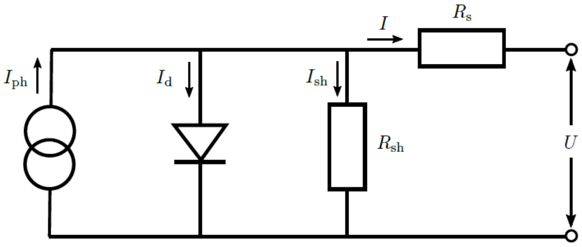


Figure 1. Single-diode equivalent circuit

The five parameters can be computed numerically or analytically. Both approaches are based on the idea that the five unknown parameters are calculated for standard conditions first and their change with changing operating conditions is computed subsequently. For more details, we refer to Duffie and Beckman (2013) for an exemplary numerical solution method and to E. I. Batzelis and Papathanassiou (2015) for an analytical approach. The analytical approach's advantage over the numerical one lies in more robust and quick computation times (E. Batzelis 2019).

Two-diodes approach

The two-diodes model is a refinement of the single-diode approach introducing a second diode into the electrical equivalent representation (see Figure 2).

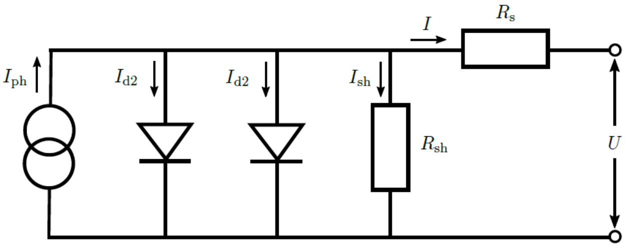


Figure 2. Two-diodes equivalent circuit

According to this model, the I-V characteristic of a PV module can be described as

$$\begin{aligned}0 = I_{\mathrm{ph}} - I_{\mathrm{S1}}(e^{\frac{\frac{U}{n_{\mathrm{ser}}}+\frac{I}{n_{\mathrm{par}}}R_{\mathrm{s}}}{U_{\mathrm{t}}}} - 1) - I_{\mathrm{S2}}(e^{\frac{\frac{U}{n_{\mathrm{ser}}}+\frac{I}{n_{\mathrm{par}}}R_{\mathrm{s}}}{2U_{\mathrm{t}}}} - 1) \\ - \frac{\frac{U}{n_{\mathrm{ser}}}+\frac{I}{n_{\mathrm{par}}}R_{\mathrm{s}}}{R_{\mathrm{sh}}} - \frac{I}{n_{\mathrm{par}}},\end{aligned} \tag{5}$$

$$U_{\mathrm{t}} = k\frac{T_{\mathrm{cell}}}{e}, \tag{6}$$

$$I_{\mathrm{ph}} = (c_1 + c_2\, 0.001 T_{\mathrm{cell}})H, \tag{7}$$

$$I_{\mathrm{S1}} = c_{\mathrm{S1}} T_{\mathrm{cell}}^3\, e^{-\frac{E_{\mathrm{g}}\ e}{k\ T_{\mathrm{cell}}}}, \tag{8}$$

and

$$I_{\mathrm{S2}} = c_{\mathrm{S2}} \sqrt{T_{\mathrm{cell}}^5}\, e^{-\frac{E_{\mathrm{g}}\ e}{2\ k\ T_{\mathrm{cell}}}}. \tag{9}$$

These equations contain six parameters ($R_{\mathrm{s}}, R_{\mathrm{sh}}, c_1, c_2, c_{\mathrm{S1}}, c_{\mathrm{S2}}$), which cannot be taken directly from the module manufacturer's data sheets. Instead, they must be obtained by parameter identification from the module's I-V characteristics for different module temperatures. Consequently, the two-diodes approach results in a high accuracy due to a detailed representation of the I-V characteristic, but it results in a higher computational effort due to an additional parameter fitting process and, hence, a more complicated parameterization.

2.2.2 Cell temperature calculation

The cell temperature affects the I-V characteristic and, hence, the electrical efficiency of the PV module. The absorbed irradiation is partially transformed into electrical and thermal energy within the cell. The module's heat transfer to the ambient is influenced by the wind velocity, the ambient temperature, and the irradiation. The energy balance can be formulated as

$$G_{\mathrm{n}}(\tau\alpha) = \eta_{\mathrm{c}} G_{\mathrm{n}} + U_{\mathrm{T}}(T_{\mathrm{cell}} - T_{\mathrm{ambient}}), \tag{10}$$

where $\tau\alpha$ is the transmission-absorption coefficient of the module that accounts for the transmission of the glazing and the absorption of the anti-reflection layer, η_{c} is the cell efficiency under operating conditions and U_{T} the heat transfer coefficient (Duffie and Beckman 2013; Jakhrani et al. 2011). Duffie and Beckman (2013) and Jakhrani et al. (2011) describe that the energy balance can be solved using the normal operating cell temperature (NOCT) conditions that assume no load conditions, no wind, and a module tilt of 45° as

$$\begin{aligned}T_{\mathrm{cell}} = T_{\mathrm{ambient}} + \\ (T_{\mathrm{NOCT}} - 20\,^{\circ}\mathrm{C})\frac{G_{\mathrm{n}}}{800\,\mathrm{W/m^2}}\frac{U_{\mathrm{T,NOCT}}}{U_{\mathrm{T}}}(1 - \frac{\eta_{\mathrm{c}}}{(\tau\alpha)_{\mathrm{NOCT}}}).\end{aligned} \tag{11}$$

Because some parameters, such as $(\tau\alpha)_{\mathrm{NOCT}}$ and U_{T}, are difficult to assess, some studies neglect their influence by assuming no load conditions and no wind (Romary et al. 2011) or real operating conditions and no wind (Bai et al. 2014). Other studies find empirical relations to consider the wind velocity's influence on the cell temperature (Duffie and Beckman 2013; Romary et al. 2011). Apart from the NOCT-based approaches, which are purely physical or a mix of empirical or physical models, there exist purely empirical approaches, such as described by King, Boyson, and Kratochvill (2005). King, Boyson, and Kratochvill (2005) develop empirical solutions for two different mounting types, i.e., a type with an installation close to ground and the other installed in contact with the ground.

2.3 Existing photovoltaic models in Modelica libraries

The following review investigates already existing open-source PV models in Modelica. We distinguish models that are part of the *Project 1*-related derivative libraries from external libraries. All of the derivative libraries of the *IBPSA* core library contain some type of PV model. While the *AixLib* and *IDEAS* library both contain a model based on the 5p-modeling, i.e., single-diode approach (see subsubsection 3.1.2), the *Buildings* library contains two simplified models assuming constant efficiencies. The model implemented in the *AixLib* library contains different cell temperature models accounting for three mounting types (open rack, close to the ground, and in contact with the ground) following Duffie and Beckman (2013) and King, Boyson, and Kratochvill (2005). In contrast, the models implemented in the *IDEAS* library use a simplified approach for which a heat transfer coefficient needs to be known. Both models assume an internal MPP tracker. In contrast to that, the models implemented in the *BuildingSystems* library cover both single- and two-diodes models with and without internal MPP tracking (Pruthviraj Balekai 2018; Nytsch-Geusen, Huber, and Nie 2013).

In contrast to the IBPSA-related libraries, two specialized open-source libraries for PV system simulation exist: the *PhotoVoltaics* (Brkic et al. 2019) and the *PVSystems* (Villalva, Gazoli, and Ruppert Filho 2009). The former is based on the single-diode model and contains various examples and validation data. It is also based on manufacturer data only but neglects the effect of different mounting types, which affect the cell temperature. In addition, the latter library is also based on the single-diode approach and applies a numerical solution method to obtain the 5 unknown parameters (Villalva, Gazoli, and Ruppert Filho 2009). However, according to Brkic et al. (2019), the library is missing a parameterization support for the parallel and serial resistances. In addition, simulation results for the open-circuit voltage do not necessarily match the points provided in the data sheets (Brkic et al. 2019). As in the case of the *PhotoVoltaics* library, the mounting's effect on the cell temperature and, hence, the module's efficiency is not included. To summarize, most of the implemented models rely on the same physical assumptions, i.e., the single-diode approach. However, literature is currently missing a detailed comparison of the single- and the two-diodes approach. Furthermore, the mounting's effect on the cell temperature has not yet been implemented in any of the analysed models. Finally, an implementation in the IBPSA library facilitates the use in the derivative libraries due to consistent interfaces and parameterization schemes.

3 Methodology

The following section gives an overview of the model implementation. In the last section, we describe the rooftop system in Berlin that is used for the validation.

3.1 Model implementation

3.1.1 Basic model structure

Figure 3 presents the basic model structure. The two model formulation both extend the `PartialPVSystem` model. The weather data, i.e., the dry bulb temperature, the wind velocity, and the global irradiation on the tilted surface are given by existing models of the package `IBPSA.BoundaryConditions`. The `PartialPVSystem` includes three replaceable models, namely the `PartialPVElectrical`, the `PartialPVThermal`, and the `PartialPVOptical`. The `PartialPVElectrical` model calculates the I-V characteristic. The single- and two-diodes modeling approach are implemented. The `PartialPVThermal` model computes the cell temperature, taking into account how the PV mounting type affects the cell temperature due to the wind velocity. The `PartialPVOptical` model calculates the PV-material-specific absorption ratio of the module, which is an input to the thermal and the electrical model. The overall model inputs comprise weather information such as the zenith angle, the incidence angle, the diffuse horizontal irradiation, the irradiation on the tilted surface, the irradiation on the horizontal surface, the dry bulb temperature as well as the wind velocity via real inputs. In addition, the tilt and azimuth angle can be manipulated by real inputs if not used as parameters. The model outputs the DC power using a real output connector. The current and voltage as well as the cell temperature are additional model variables.

3.1.2 Single-diode modeling approach

One of the implemented electrical partial models follows the single-diode approach as described in subsubsection 3.1.2. The model is based on five unknown parameters that need to be estimated. To determine these parameters, two approaches exist: an iterative, numerical solution method and an analytical one. Since the numerical approach is computationally less efficient, and to avoid users having to provide start values for the iteration variables, the implemented model uses the analytical approach. Even though it is less accurate, it does not suffer from initialization problems and is robust (Bai et al. 2014). We implement the approach presented by Bai et al. (2014). It is based on the approximation of the Lambert W func-

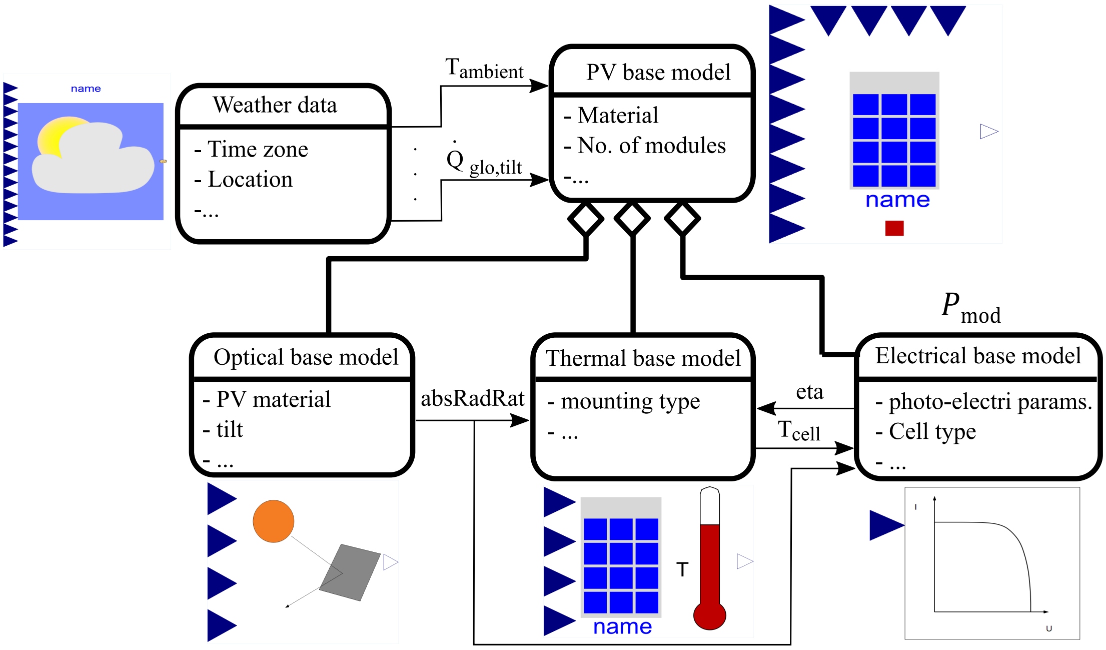


Figure 3. Basic model structure with partial models, main interfaces, and parameters. For simplicity, only the main parameters and variables are displayed.

tion and uses simplifications to derive an explicit formulation of the I-V characteristic based on manufacturer data, only. In addition, it provides explicit formulations for the initialization of the five unknown parameters. For more details on the initialization method, we refer to Bai et al. (2014). The current and voltage at the MPP is deduced as described in Bai et al. (2014),

$$I_{\text{mp}} = I_{\text{ph}}\,(1 - \frac{1}{w}) - a\frac{(w-1)}{R_{\text{sh}}} \tag{12}$$

$$U_{\text{mp}} = a\,(w-1) - R_{\text{s}} I_{\text{mp}}, \tag{13}$$

where w is the wind velocity. Bai et al. (2014) do not consider any temperature- or irradiation-dependence of the parameters. Since the presented model aims to consider these dependencies, we assume for the operating conditions equations based on De Soto, Klein, and Beckman (2006) and Messenger and Abtahi (2017),

$$\frac{a}{a_0} = \frac{T_{\text{cell}}}{T_{\text{cell}}}, \tag{14}$$

$$I_{\text{ph}} = \frac{H_{\text{tilt}}}{H_{\text{tilt,0}}}\,(I_{\text{ph}} + \mu_{\text{I,K}}\,(T_{\text{cell}} - T_{\text{cell,0}})), \tag{15}$$

$$\frac{I_{\text{s}}}{I_{\text{s,0}}} = (\frac{T_{\text{cell}}}{T_{\text{cell,0}}})^3 e^{(\frac{E_{\text{g,0}}}{kT_{\text{cell,0}}} - \frac{E_{\text{g}}}{kT_{\text{cell}}})}, \tag{16}$$

$$\frac{E_{\text{g}}}{E_{\text{g,0}}} = 1 - C\,(T_{\text{cell}} - T_{\text{cell,0}}). \tag{17}$$

We assume that the parallel resistance R_{sh} is temperature invariant, but that it depends on irradiation, and we assume the serial resistance to be constant (De Soto, Klein, and Beckman 2006). Therefore,

$$\frac{R_{\text{sh}}}{R_{\text{sh,0}}} = \frac{H_0}{H}, \tag{18}$$

$$R_{\text{s}} = R_{\text{s,0}}. \tag{19}$$

This approach is purely based on manufacturer data and is, therefore, simple to parameterize. At this point, we highlight that even though the unknown quantities are called parameters in the scientific literature, they are variables within the models. The proposed method includes predefined initialization values for those variables and calculates their values during operation using (14) to (19). Despite its simplicity, the single-diode approach is based on strong assumptions regarding the behavior of the parallel and serial resistance. In this contrast, the two-diodes approach described in the next section provides a more accurate representation.

3.1.3 Two-diodes modeling approach

We implemented the two-diodes model as

$$\begin{aligned} 0 = I_{\text{ph}} - I_{\text{S1}}(e^{\frac{\frac{U_{\text{mp}}}{n_{\text{ser}}} + \frac{I_{\text{mp}}}{n_{\text{par}}} R_{\text{s}}}{U_{\text{t}}}} - 1) - I_{\text{S2}}(e^{\frac{\frac{U_{\text{mp}}}{n_{\text{ser}}} + \frac{I_{\text{mp}}}{n_{\text{par}}} R_{\text{s}}}{2U_{\text{t}}}} - 1) \\ - \frac{U_{\text{mp}}}{n_{\text{ser}}} + \frac{I_{\text{mp}}}{n_{\text{par}}} \frac{R_{\text{s}}}{R_{\text{sh}}} \frac{I_{\text{mp}}}{n_{\text{par}}}, \end{aligned} \tag{20}$$

$$0 = \frac{I_{\text{mp}}}{n_{\text{par}}} - \lambda \frac{I_{\text{Sat1}}}{U_{\text{t}}} e^{\frac{\frac{U_{\text{mp}}}{n_{\text{ser}}} + \frac{I_{\text{mp}}}{n_{\text{par}}} R_{\text{s}}}{U_{\text{t}}}} + \frac{I_{\text{Sat2}}}{2U_{\text{t}}} e^{\frac{\frac{U_{\text{mp}}}{n_{\text{ser}}} + \frac{I_{\text{mp}}}{n_{\text{par}}} R_{\text{s}}}{2U_{\text{t}}} + \frac{1}{R_{\text{sh}}}}, \tag{21}$$

and

$$0 = \frac{U_{\mathrm{mp}}}{n_{\mathrm{ser}}} - \lambda \frac{R_{\mathrm{s}} I_{\mathrm{S1}}}{U_{\mathrm{t}}} e^{\frac{\frac{U_{\mathrm{mp}}}{n_{\mathrm{ser}}} + (\frac{I_{\mathrm{mp}}}{n_{\mathrm{par}}}) R_{\mathrm{s}}}{U_{\mathrm{t}}}} + \frac{R_{\mathrm{s}} I_{\mathrm{S2}}}{2U_{\mathrm{t}}} e^{\frac{\frac{U_{\mathrm{mp}}}{n_{\mathrm{ser}}} + (\frac{I_{\mathrm{mp}}}{n_{\mathrm{par}}}) R_{\mathrm{s}}}{2U_{\mathrm{t}}}} + \frac{R_{\mathrm{s}}}{R_{\mathrm{sh}} + 1}, \tag{22}$$

where the module voltage U_{mp} and module current I_{mp} are unknowns. The calculation of the current I_{mp} and voltage U_{mp} in the MPP uses the method of Lagrange multipliers to find the maximum of power $P_{\mathrm{mp}} = I_{\mathrm{mp}}\, U_{\mathrm{mp}}$. This leads to a system of three equations for the three unknowns $I_{\mathrm{mp}}, U_{\mathrm{mp}}$ and λ.

With the two-diodes model, the electrical behavior of a PV module can be represented more accurately due to the second diode in the equivalent circuit. The drawback is that the six parameters cannot be derived from manufacturer data, as is the case with the single-diode model, but must be determined using optimization. For parameter fitting, we use GenOpt (Wetter 2009).

3.1.4 Thermal modeling approach

To calculate the cell temperature, we implemented three different thermal modeling approaches. These are implemented using replaceable classes of the thermal base model, allowing the user to select the modeling approach that corresponds best to the mounting situation. In addition to the physics-based model for the open rack installation, we also integrated the empirical models of King, Boyson, and Kratochvill (2005). They reflect a mounting close to ground and in contact with ground, respectively. King, Boyson, and Kratochvill (2005) found their cell temperature calculation to be applicable for different cell types.

3.2 Preliminary studies

Both modeling approaches have been partially validated in preliminary studies. The single-diode modeling approach combined with the implemented thermal models has been validated in Maier, Kratz, et al. (2021) using measurement data from the National Institute of Standards and Technology (NIST). The underlying model was integrated in the *AixLib* library (Maier, Jansen, et al. 2023). The measurement data was taken from two different arrays consisting of mono-SI wafer-cell-based modules. We extend the existing work by including an analysis of the thermal model and focusing on thin-film CIGS modules. The two-diodes model has been developed and presented in Pruthviraj Balekai (2018) and implemented in the *BuildingSystems* library (Nytsch-Geusen, Huber, Ljubijankic, et al. 2013). This work enhances the model by implementing it in a modular model structure that forms the basis of the single- and two-diodes modeling approach. In addition, the thermal model has been replaced.

3.3 Validation data

The measured values for the validation of the newly developed single-diode and two-diodes models were obtained using the monitoring system of the rooftop experimental building[1] as shown in Figure 4), which is located on the university campus in Berlin-Charlottenburg. A total of 84 thin-film PV modules (type Solibro SL2 CIGS 110-120[2]) with a total power of 9.24 $\mathrm{kW_{peak}}$ are installed on the roof of the building on two movable single-axis tracking facade elements.

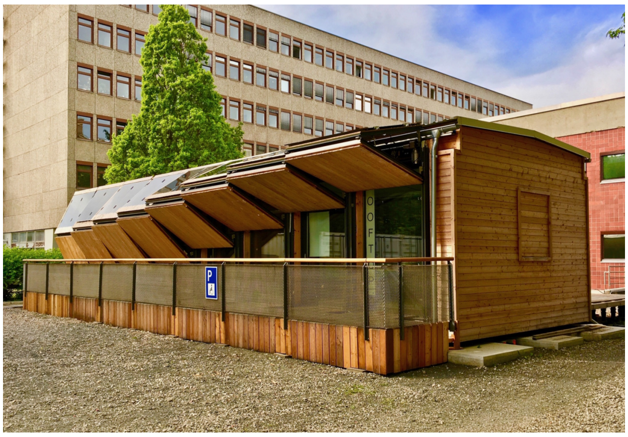


Figure 4. Rooftop building with roof and facade mounted photovoltaic system in Berlin (Germany).

These modules are monitored separately in 42 groups, each with 2 interconnected PV modules using the SolarEdge (SolarEdge 2006) monitoring platform. For the validation, a non-movable, rarely shaded module group on the roof was selected, which has a slight inclination of 2° and is oriented 27.5° to the west. The following variables were recorded in a time interval of 5 min during the monitoring process for this module:

- Module temperature (using a digital sensor on the back of the module),
- total horizontal radiation (using KIPP and Zonen SP Lite Pyranometer),
- wind speed and direction and outside air temperature, using the Netatmo (Netatmo 2011) weather station, and
- electric power per module group (4 x 120 $\mathrm{W_{peak}}$ and 2 x 115 $\mathrm{W_{peak}}$) using the monitoring system as shown in Figure 5.

The data from the Netatmo weather station are extracted using a Python-based data collector which then feeds it to the InfluxDB data base for further processing. The module temperature and total horizontal radiation is collected individually using NodeMCU microcontroller and then fed wireless to InfluxDB via MQTT.

[1] http://www.solar-rooftop.de

[2] https://hanergy.eu/wp-content/uploads/2015/08/Solibro_data-sheet_SL2-F-module_G1-4_100-110-120-125_EN.pdf

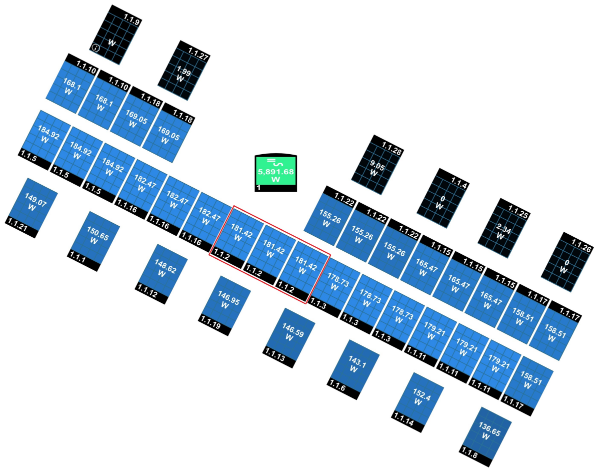


Figure 5. Layout of the PV system with 84 PV modules and generated electricity on April $30^t h$, 2023 (and the selected PV system marked in red rectangle).

4 Results

This section first describes the results from the model implementation and continues by discussing the validation.

4.1 Model application

4.1.1 Single-diode model application

The parameters of the single-diode model is based on the manufacturer data, only. The following five parameters were used to parameterize the 115 Wp (and 120 Wp module, respectively): $I_{\mathrm{sc},0} = 1.69\,\mathrm{A}$ (1.71 A), $V_{\mathrm{oc},0} = 101.2\,\mathrm{V}$ (102.3 V), $eta_0 = 0.122$ (0.128), $\alpha_{\mathrm{Isc},0} = 0.01\,\%/\mathrm{K}$, $\beta_{\mathrm{Voc},0} = -0.27\,\%/\mathrm{K}$, $\gamma_{\mathrm{PMPP},0} = -0.32\,\%/\mathrm{K}$, and $P_{\mathrm{MPP},0} = 115\,\mathrm{W}$.

4.1.2 Two-diodes model application

To determine the parameters of the two-diodes model, a set of U-I characteristics curves for different cell temperatures of the module type Solibro SL2 CIGS 110 were used. We note that we only had access to data for the modules with a peak power of 110 W. However, the target system comprises four modules with a peak power of 115 W and 120 W. Consequently, we cannot directly compare measured with simulated data for the two-diodes model. To still validate the plausibility of the two-diodes model, a second validation set is generated. For the second validation set, the single- and the two-diodes models are both parameterized with 110 Wp modules. This serves as a plausibility check for the two-diodes model. Note, however, that the lack of measurement data for the 110 Wp modules precludes an assessment of the accuracy of the single- vs. two-diodes model.

To obtain the best possible combination of the six model parameters using optimization, a cost function was defined. This function calculates the squares of the difference between the calculated module current with the two-diodes model and the value from the manufacturer curve for 20 different voltage values along the U-I curve. We used the Hooke-Jeeves algorithm from GenOpt, which yields the following parameters values: $R_{\mathrm{s}} = 0.027484527\,\Omega$, $R_{\mathrm{sh}} = 500.0\,\Omega$, $c_1 = 0.0011962052$, $c_2 = 0.001542755$, $c_{\mathrm{S1}} = 9.490919$, and $c_{\mathrm{S2}} = 0.007634368$.

4.2 Model validation and plausibility check

The validation and plausibility check comprises three aspects: (i) the validation of the single-diode model (115 Wp and 120 Wp) using the measurement data (see Section 4.2.1), (ii) the plausibility check of the two-diodes model using the single-diode model (110 Wp) simulation results (see Section 4.2.2), and (iii) the cell temperature validation (see Section 4.2.3). The validation period is from July 27th until August 9th, i.e., it covers 12 d. To evaluate the cell temperature approaches, an exemplary day from the spring period was added.

4.2.1 Validation of the single-diode model

Figure 6 depicts the curves of the simulated and measured DC power output of the six modules for the single-diode model. A comparison of the simulated and the measured DC power for the whole validation period yields an R^2 value of 0.86 and a mean absolute error of 16.6 W. This is rated as a high accuracy. The measurement data contains days with low, medium, and high irradiation, which confirms high model accuracy for different operating conditions. To better understand the data, Figure 7 picks an exemplary day. We select July 30th as an exemplary day because it covers both high and low irradiation periods during one day. The figure shows the simulated and measured DC output power of the modules on the top and the global horizontal irradiation at the bottom. We observe that the simulated and measured DC power generally follow the trend of the global irradiation, as expected. However, at around 3 pm, the measurements show a DC power peak, while the simulation and irradiation data do not. A potential explanation is that the irradiation sensor might detect shading while the observed modules do not. Other possible explanations include imperfect calibration of the irradiation sensor or measurement errors. However, since this behaviors is only observed for a negligible amount of time, the overall measurement data quality is rated high.

4.2.2 Plausibility check for the two-diodes model

Since the measurement data do not cover 110 Wp modules, the two-diodes model is compared to the simulation data of the single-diode model. For this plausibility check, the single-diode models are parameterized based on the manufacturer data of the 110 Wp modules. Figure 8 illustrates the simulated DC powers of both modeling approaches for July 30th. When comparing the simulated outputs, it becomes evident that they are almost identical. The comparison for the whole validation period of 12 d yields an R^2 value of 0.99 and a mean absolute error of 5.5 W. This is a high accuracy and it shows that both models yield similar DC power outputs.

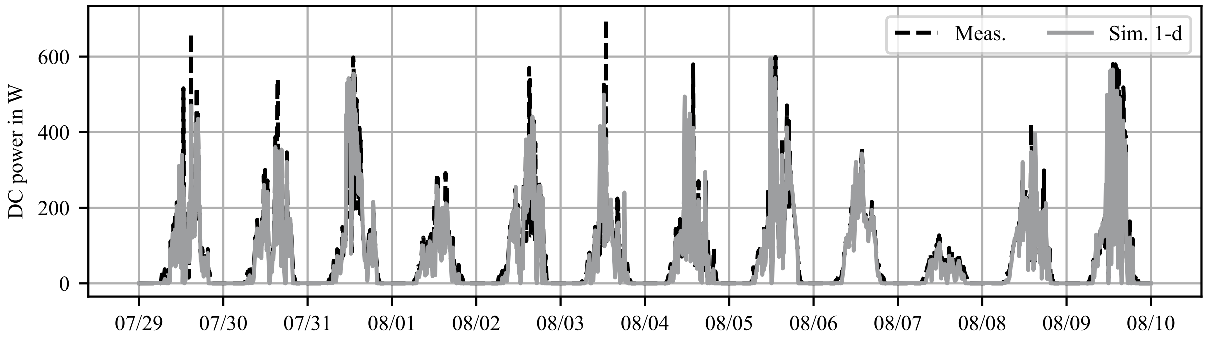


Figure 6. Comparison of measured and simulated DC power output for the validation period for the single-diode approach.

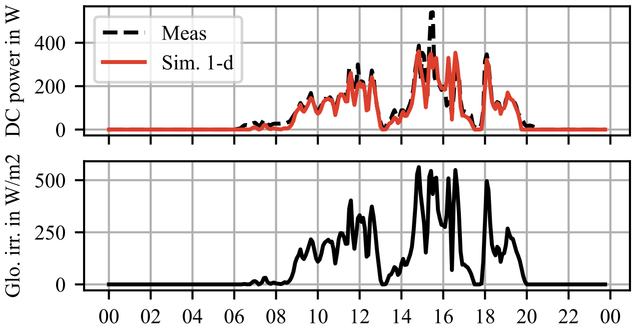


Figure 7. July 30th as an exemplary day to demonstrate the accuracy of the single-diode model.

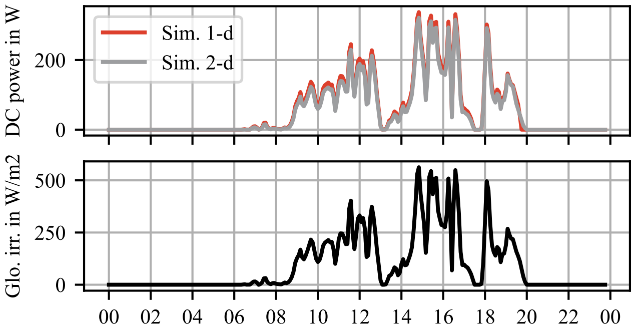


Figure 8. July 30th as an exemplary day to compare the single- and two-diodes model outputs.

4.2.3 Cell temperature validation

In addition to the DC power, we also validated the cell temperature model. Figure 9 shows the curves of the measured cell temperature, the ambient temperature, and the simulated cell temperature for the three modeling approaches (“open rack”, “close to ground”, and “in contact with ground”), as discussed in Section 3.1.4, for a spring (left) and a summer day (right). The thermal model corresponding to the mounting of the use case is “close to ground”, the other modeling approaches are added to the figure for plausibility check. While for the spring day, the thermal model corresponding to the mounting type “close to ground” best predicts the cell temperature, the mounting type “open rack” results in the highest accuracy for the summer day. When taking the R^2 as a KPI, the following values are realized for the spring (and summer) day, respectively: “close to ground”: 0.81 (0.3), “open rack”: 0.7 (0.75), and “in contact with ground”: 0.68 (-0.01). The cell temperature calculations are, among other influences, based on the wind velocity and the ambient temperature. On the summer day, the ambient temperature is higher while the wind velocity is lower. The thermal models seem to overestimate this combined influence on the temperature leading to an overestimation of the temperature. In addition, we observe that the model overestimates the cell temperature during the night time. We find that the assumption of the cell temperature reaching the dry bulb temperature during no-irradiation periods is incorrect. The PV modules experiences radiative heat transfer with the sky leading to lower cell temperatures than the dry bulb temperatures during no-irradiation periods. However, the night time is not relevant for PV system evaluation since no DC power is generated.

Finally, we discuss the CPU time of the model. Both models result in a similar set of equations. While the single-diode model has 544 equations, the two-diodes model results in 534. To compare the CPU times, we simulated both models for the validation period 10 times and took the median of the CPU times. The simulations were done on a Lenovo L480 with an Intel Core i5-8250U CPU and 1.60 GHz. The operating system is Windows 10 and Dymola 2023 was used as simulation environment. Two-diodes model results in average CPU times of 0.19 s and single-diode of 0.2 s Consequently, we rate the models' simulation speed as sufficiently fast for most applications.

5 Limitations

The validations show high accuracy of DC power output for the single-diode model for the thin-film CIGS module. Due to missing measurement data for the 110 Wp modules, we cannot directly validate the two-diodes model making a general comparison between both modeling techniques difficult. The cell temperature validation

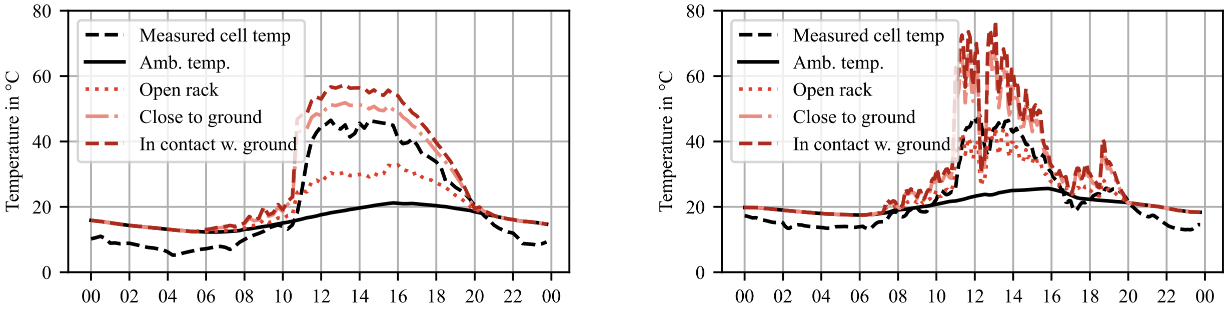


Figure 9. Comparison of measured and simulated cell temperature for a spring day (left) and a summer day (right). Both the two- and single-diode model use the same cell temperature model.

shows that the cell temperature model overestimates the actual temperature, especially during summer days. However, the cell temperature computations only have a small effect on the computed DC power due to small temperature coefficients of the modules ($\alpha_{\mathrm{Isc},0} = 0.01\,\%/\mathrm{K}$ and $\beta_{\mathrm{Voc},0} = -0.27\,\%/\mathrm{K}$). Uncertainty may be introduced by the irradiation sensor not capturing all shading effects of the six modules. This might lead to an overestimation of the power output at certain points as shown in Figure 7.

We focus the validation on the summer period. The validation should be extended to account for colder days and days with almost no direct irradiation. This is explained by the advantage of thin-film CIGS modules that benefit from diffuse radiation. In addition, the cell temperature model validation should be extended to a very cold winter day to capture all relevant effects and better understand how to correctly select an appropriate thermal model. Moreover, the presented models were not yet validated regarding their accuracy in current and voltage estimation. Even though it is possible to model each module separately, we modelled the PV array as one system and neglected the detailed electrical connections between the modules. For such a detailed analysis, we expect the two-diodes model to outperform the simplified single-diode model.

6 Conclusions

We presented a new open-source Modelica model of a PV system that is implemented in the Modelica IBPSA Library. The model includes two typical modeling approaches for the electrical characteristics of PV modules, a single- and a two-diodes approach. To validate the model, real measurement data from a rooftop building in Berlin, Germany, was used. The validated PV modules are thin-film CIGS cells. The comparison is done for a validation period in July and the DC power output and cell temperature are analysed. The investigation reveals that the single-diode model captures the PV DC power output well (see Figure 6) and that the two-diodes model estimates similar DC power outputs (see Figure 8). However, we could not evaluate which modeling technique is favorable since proper measurement data for the two-diodes model were not available, and we therefore only verified its accuracy based on the single-diode model.

The mounting situation of the measured system corresponds to an installation close to ground. The simulation results show that this approach does not necessarily lead to the best cell temperature estimation (see Figure 9). For the regarded spring day, the cell temperature model for the intended mounting type results in the highest accuracy, while for the summer day, the cell temperature model corresponding to an installation with open rack yields the highest accuracy. The thin-film modules are characterized by small temperature coefficients and, hence, their performance decline with increasing cell temperatures is small.

7 Future work

The present paper shows the comparison and validation for a thin-film CIGS PV module. Even though, the single-diode model has been validated before based on mono-Si modules, literature is still lacking a comparison of the two presented modeling approaches for a wider range of typical configurations. These configurations include mono- and poly-Si wafer cells for different setups (i.e., tilts and mountings) as well as orientations and locations. Validation data of different mounting situations could also be used to further validate the cell temperature calculations implemented in the thermal model. Furthermore, our validation does not include a detailed current and voltage analysis due to missing data. Moreover, the validation only focuses on 12 days in July. Even though the validation period was carefully selected to cover different irradiation situations, validation in other seasons would be beneficial.

8 Data availability

The validations were done with the model that is available in the Modelica IBPSA Library and can be found at `https://github.com/ibpsa/modelica-ibpsa/pull/1766` [3]. The measurement data is available in the corresponding IBPSA resources folder and obtained from the UdK Berlin.

[3] The corresponding commit hash is 6e263f8cfa1ea65c0f4f91b610f205ece11e8a51.

Acknowledgements

We gratefully acknowledge the financial support by the Federal Ministry for Economic Affairs and Climate Action (BMWK), promotional reference 03EN3026C. This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231. The described research was also conducted within the project "Eneff: HCBC 1. Umsetzungsphase HochschulCampus Berlin-Charlottenburg" and funded by the Federal Ministry for BMWK, reference 03ET1632B. This work emerged from the IBPSA Project 1, an international project conducted under the umbrella of the IBPSA. Project 1 has developed and demonstrated a BIM/GIS and Modelica Framework for building and community energy system design and operation.

References

2022 Global status report for buildings and construction: Towards a zero-emissions, efficient and resilient buildings and construction sector (2022). Nairobi. ISBN: 978-92-807-3984-8.

Bai, Jianbo et al. (2014). "Development of a new compound method to extract the five parameters of PV modules". In: *Energy Conversion and Management* 79, pp. 294–303.

Batzelis, Efstratios (2019). "Non-iterative methods for the extraction of the single-diode model parameters of photovoltaic modules: A review and comparative assessment". In: *Energies* 12.3, p. 358.

Batzelis, Efstratios I and Stavros A Papathanassiou (2015). "A method for the analytical extraction of the single-diode PV model parameters". In: *IEEE Transactions on Sustainable Energy* 7.2, pp. 504–512.

Brkic, Jovan et al. (2019). "Open Source PhotoVoltaics Library for Systemic Investigations". In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. Linköping Electronic Conference Proceedings. Linköing University Electronic Press, pp. 41–50. DOI: 10.3384/ecp1915741.

De Soto, Widalys, Sanford A Klein, and William A Beckman (2006). "Improvement and validation of a model for photovoltaic array performance". In: *Solar energy* 80.1, pp. 78–88.

Duffie, John A and William A Beckman (2013). *Solar engineering of thermal processes*. John Wiley & Sons.

Jakhrani, Abdul Qayoom et al. (2011). "Determination and comparison of different photovoltaic module temperature models for Kuching, Sarawak". In: *2011 IEEE Conference on Clean Energy and Technology (CET)*. IEEE, pp. 231–236.

Jorissen, Filip et al. (2018). "Implementation and Verification of the IDEAS Building Energy Simulation Library". In: *Journal of Building Performance Simulation* 11 (6), pp. 669–688. DOI: 10.1080/19401493.2018.1428361.

King, David L., William E. Boyson, and J. A. Kratochvill (2005). "SANDIA REPORT SAND 2004-3535 Unlimited Release Printed December 2004 Photovoltaic Array Performance Model". In.

Maier, Laura, David Jansen, et al. (2023). "AixLib: an open-source Modelica library for compound building energy systems from component to district level with automated quality management". In: *Journal of Building Performance Simulation* 0.0, pp. 1–24. DOI: 10.1080/19401493.2023.2250521. eprint: https://doi.org/10.1080/19401493.2023.2250521. URL: https://doi.org/10.1080/19401493.2023.2250521.

Maier, Laura, Michael Kratz, et al. (2021). "Open-source photovoltaic model for early building planning processes: Modeling, application and validation". In: *Building Simulation 2021*. Vol. 17. IBPSA, pp. 2315–2316.

Messenger, Roger A and Amir Abtahi (2017). *Photovoltaic systems engineering*. CRC press.

Netatmo (2011). accessed June 08, 2023. https://www.netatmo.com/.

Nytsch-Geusen, Christoph, Jörg Huber, Manuel Ljubijankic, et al. (2013). "Modelica BuildingSystems – eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme". In: *Bauphysik* 35.1, pp. 21–29. ISSN: 01715445. DOI: 10.1002/bapi.201310045.

Nytsch-Geusen, Christoph, Jörg Huber, and Yue Nie (2013). "Simulation-based design of PV cooling systems for residential buildings in hot and dry climates". In: *13th International Conference of the International Building Performance Simulation Association*.

Pruthviraj Balekai, Pruthviraj (2018). "Energy performance analysis of single axis tracking PV system for the Rooftop Building based on Modelica and openHAB". Master thesis. Berlin, Germany: Technische Universität Berlin.

Romary, Florian et al. (2011). "Thermal modelling to analyze the effect of cell temperature on PV modules energy efficiency". In: *Proceedings of the 2011 14th European Conference on Power Electronics and Applications*. IEEE, pp. 1–9.

Sellner, Stefan et al. (2012). "Understanding PV module performance: Further validation of the novel loss factors model and its extension to AC arrays". In: *27th European Photovoltaic Solar Energy Conference and Exhibition*, pp. 3199–3204.

SolarEdge (2006). Accessed June 08, 2023. https://https://www.solaredge.com/.

Stein, Joshua et al. (2013). *Outdoor PV performance evaluation of three different models: single-diode SAPM and loss factor model*. Tech. rep. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Sutterlueti, J et al. (2008). "Characterising PV modules under outdoor conditions: what's most important for energy yield". In: *Channels* 48.24, p. 24.

Villalva, Marcelo Gradella, Jonas Rafael Gazoli, and Ernesto Ruppert Filho (2009). "Comprehensive approach to modeling and simulation of photovoltaic arrays". In: *IEEE Transactions on power electronics* 24.5, pp. 1198–1208.

Wetter, Michael (2009-05). *GenOpt, Generic Optimization Program, User Manual, Version 3.0.0*. Tech. rep. LBNL-2077E. Berkeley, CA, USA: Lawrence Berkeley National Laboratory.

Wetter, Michael, Christoph van Treeck, et al. (2019-09). *IBPSA Project 1: BIM/GIS and Modelica framework for building and community energy system design and operation – ongoing developments, lessons learned and challenges*. DOI: 10.1088/1755-1315/323/1/012114. URL: https://doi.org/10.1088/1755-1315/323/1/012114.

Wetter, Michael, Wangda Zuo, et al. (2014). "Modelica Buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. ISSN: 1940-1493. DOI: 10.1080/19401493.2013.765506.

Paving the way for Hybrid Twins using Neural Functional Mock-Up Units

Tobias Thummerer[1] Artem Kolesnikov[2] Julia Gundermann[2] Denis Ritz[3] Lars Mikelsons[1]

[1]University of Augsburg, Germany, {tobias.thummerer, lars.mikelsons}@uni-a.de
[2]ESI Germany GmbH, Dresden, Germany, {artem.kolesnikov, julia.gundermann}@esi-group.com
[3]Technische Universität Dresden, Germany, denis.ritz@tu-dresden.de

Abstract

Neural Ordinary Differential Equations (NeuralODEs) open up the possibility to enhance the modeling of dynamical systems, in terms of prediction quality and computation time, as well as shortened development time. Porting NeuralODEs, the combination of an artificial neural network and an ODE solver, to real engineering applications is still a challenging venture. However, we will show that *Neural Functional Mock-up Units* (NeuralFMUs), an evolved subgroup of NeuralODEs that contain *Functional Mock-up Units* (FMUs), are able to cope with these challenges. This paper briefly introduces to the topics NeuralODE and NeuralFMU and describes the procedure and considerations to apply this technique to a real engineering use case. Further, different workflows to apply NeuralFMUs dependent on tool capabilities and use case requirements are discussed. The presented method is illustrated with the creation of a *Hybrid Twin* of an hydraulic excavator arm, which features various challenges such as discontinuity, nonlinearity, oscillations and characteristic maps. Finally, we will show that the *Hybrid Twin* created on basis of measurement data from a real system gives more accurate results compared to a conventional simulation model based on physical equations (*first principle model*).

Keywords: NeuralFMU, NeuralODE, PeNODE, FMI, PhysicsAI, Hybrid Twin, Scientific Machine Learning

1 Introduction

In the following sections, short introductions to the used techniques are given.

1.1 NeuralODE

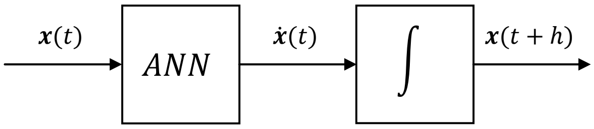


Figure 1. The topology of a NeuralODE. On basis of the system state $\boldsymbol{x}(t)$ the ANN computes the system state derivative $\dot{\boldsymbol{x}}(t)$, which is numerically integrated into the next system state $\boldsymbol{x}(t+h)$ by the ODE solver with step size h.

Since their introduction in 2018, *Neural Ordinary Differential Equations* (NeuralODEs) (Chen et al. 2018) are one of the key techniques for data driven modeling of physical systems. NeuralODEs consist of an *Artificial Neural Network* (ANN) that functions as the right-hand side of an *Ordinary Differential Equation* (ODE), together with an ODE solver to obtain the solution of the ODE, see Figure 1. Training such models on the ODE solution $\boldsymbol{x}$ requires (efficient) differentiation through the ODE solver by *Automatic Differentiation* (AD) or estimating sensitivities with *adjoint sensitivity analysis* (Bittner 1963). As almost any other machine learning model for learning dynamic systems like *recurrent neural networks* or *long short-term memory* networks in (Champaney et al. 2022), plain NeuralODEs need a significant amount of data and are only able to learn physical effects that are represented as part of this data. In real world engineering, there is often far more system knowledge available, which is only partially included or not included at all in the NeuralODE training data set. This knowledge can be used to drastically improve training, by incorporating it into the NeuralODE model itself, for example in the form of differential equations. The resulting structure, consisting of ANNs, differential equations and a numerical ODE solver, is further referred to as *Physics-enhanced Neural Ordinary Differential Equation* (PeNODE) or synonymously hybrid NeuralODE (see Figure 2). In this case, the ANN needs only to learn the missing physics that is not part of the system of differential equations. Compared to plain NeuralODEs, this allows for the use of smaller ANN

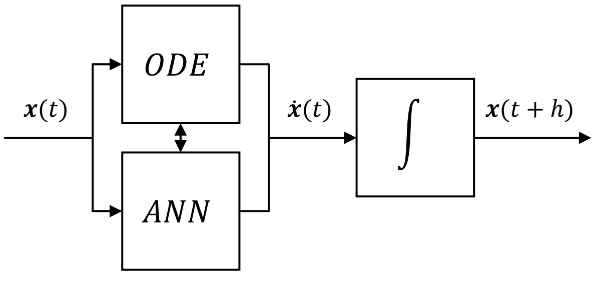


Figure 2. An example topology for a PeNODE. The system dynamics are determined by an ODE and an ANN, for which a variety of different interconnection topologies are possible.

topologies with less trainable parameters, which result in faster training times and often much better convergence. Further, introducing physical equations opens up to much better explainability regarding the learned process by the ANN, because a physical interpretation can be given for signals between ANN and physical equations. In practice, this *incorporated knowledge* might often be the part of the system that is well understood by the engineer and can therefore be modeled easily, for example the kinematics of an industrial robot, whereas the ANN has to learn the remaining physical effects that are challenging to model, like e.g. the friction behavior of the robot joints.

1.2 NeuralFMU

If the concept of a PeNODE shall be applied to real world engineering problems, another issue has to be faced: Physical models, designed in dedicated modeling tools, are not available in a symbolic representation of the equation system that can easily be used as parts of a PeNODE. Even if the symbolic ODE is accessible, large and complex systems often count thousands of equations, which is cumbersome to handle. Therefore, modeling PeNODEs on basis of large systems of equations, which are common in industrial applications, is not practicable out of the box. Fortunately, this can be solved by deploying the model foundation not by a system of equations, but a handy container for such: The *Functional Mock-up Unit* (FMU) (Blochwitz et al. 2011). The FMU type *Model-Exchange* (ME) provides an interface in analogy to an ODE system: On basis of the current time t, the system state $\boldsymbol{x}(t)$ and optional inputs $\boldsymbol{u}(t)$, the system dynamics $\dot{\boldsymbol{x}}(t)$ are calculated. Comparing to the PeNODE model, instead of combining an ODE with an ODE solver and ANN, a ME-FMU is used. This topology (see Figure 3) is referred to as *Neural Functional Mock-up Unit* (NeuralFMU) and was introduced in (Thummerer, Kircher, and Mikelsons 2021).

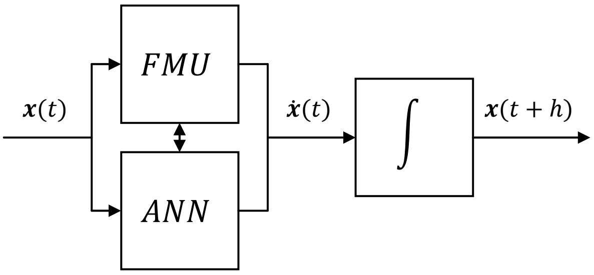


Figure 3. An example topology for a ME NeuralFMU. In this specific case, the system dynamics are determined by an ME-FMU and an ANN in parallel. In general, a wide variety of different interconnection topologies is possible.

Because determination of the loss function gradient for NeuralFMUs (and NeuralODEs in general) is computationally expensive compared to gradient determination in plain ANNs without ODE solvers, efficient differentiation is a necessity to achieve an economic training (and therefore development) time. The *Julia* programming language (Bezanson et al. 2014) offers some of the most powerful frameworks for AD that implement a variety of different approaches, therefore the first library for building and training NeuralFMUs called *FMIFlux.jl*[1] was implemented in this language. As a proof of concept for the applicability of NeuralFMUs in industrial applications, a vehicle longitudinal dynamics model for the prediction of an electric vehicle's energy consumption was enhanced in terms of result accuracy (Thummerer, Stoljar, and Mikelsons 2022). Further, the applicability to medical use cases was shown at the example of a hybrid simulation model of the human cardiovascular system (Thummerer, Tintenherr, and Mikelsons 2021).

2 Method

Deploying a NeuralFMU for a custom use case can be subdivided into three main tasks. First, the transfer of the *First Principle Model* (FPM) from modeling environment to the machine learning environment *Julia* (discussed in subsection 2.1). Second, the actual topology design and training of the NeuralFMU in *Julia* (subsection 2.2) and third, the reimport of the trained *Hybrid Model* (HM) from *Julia* into the original (or another) modeling or simulation environment (subsection 2.3).

2.1 From modeling environment to Julia

Because high performance differentiation is not available in most modeling tools, the FPM needs to be transferred from the original modeling tool into *Julia* (Thummerer, Stoljar, and Mikelsons 2022). This is achieved by using the FMU export functionality of the modeling tool to export an FMU and the *Julia* library *FMI.jl*[2] to import the FMU into *Julia*, see Figure 6 (step 1). After the import of the FPM FMU into *Julia*, the HM can be designed and trained.

2.2 Designing the topology

Basically, a wide range of topologies for NeuralFMUs are thinkable and designing a suitable one might not be intuitive. In the following, different aspects are highlighted and suggestions for decision-making based on requirements are given.

2.2.1 Sequential/Parallel

While the position of the numerical integrator in a NeuralODE is fixed, the positions of the FMU(s) and ANN(s) are not. Here, two main topologies can be distinguished. First, the elements can be connected sequentially, so one element is computing results on basis of intermediate results from another element, see Figure 4. A common use-case is an ANN, that corrects the system dynamics retrieved from an insufficient FMU model. Second, elements can be connected in parallel, so multiple elements

[1] `https://github.com/ThummeTo/FMIFlux.jl` (accessed on May 24, 2023)

[2] `https://github.com/ThummeTo/FMI.jl` (accessed on May 24, 2023)

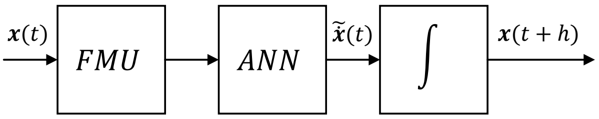


Figure 4. An example for a sequential NeuralFMU topology.

are computing results on basis of the same or different inputs, whereas the results need to be merged, as shown in the introduction in Figure 3. This approach is useful for example if the ANN needs to learn an effect that can be added to the existing dynamics (like many friction effects) or only a subpart of the state derivative vector shall be computed by the ANN, whereas the remaining part is determined by the ODE.

If no decision can be made on how the final effect will influence the system, e.g. because of lack of information regarding the unmodeled system part, both topologies can be used at once with minimal overhead: A topology using gates, as introduced in (Thummerer, Stoljar, and Mikelsons 2022), allows for continuous fading between a sequential and parallel interconnection of the ANN, see Figure 5. Note, that the gate parameters do not need to sum up to 1, but can be treated independently. Dependent on the status of the gates, different edge cases can be parameterized:

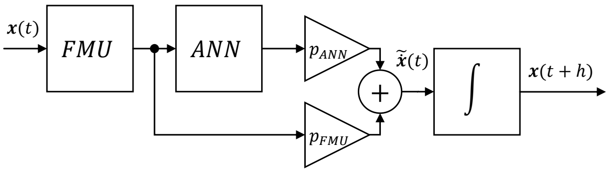


Figure 5. A NeuralFMU using gates. The parameters p_{ANN} and p_{FMU} control how much the ANN dynamics and FMU dynamics contribute to the final system dynamics $\tilde{\boldsymbol{x}}$. For vectors of signals, gate parameters can be used for all signals at once or be extended to a vector parameter, to control each signal's contribution individually.

- For $p_{ANN} = 0$ and $p_{FMU} = 1$, the system behaves like the original FMU without any ANN attached to it.
- For $p_{ANN} \neq 0$ and $p_{FMU} = 0$, the system behaves like a series connection of FMU and ANN.
- For $p_{ANN} \neq 0$ and $p_{FMU} \neq 0$, the system uses the FMU as well as the ANN dynamics. The ANN is able to manipulate the FMU dynamics (series) as well as learning its very own dynamics on given signals (parallel).

Finally, these gate parameters can be trained together with the ANN parameters and so bypass the initialization challenge of (physics-enhanced) NeuralODEs, which was further highlighted in (Thummerer, Stoljar, and Mikelsons 2022).

2.2.2 Data processing between FMUs and ANNs

FMUs and ANNs operate in different numerical ranges. Whereas the signal interface of FMUs is basically only limited by Float64, ANNs suffer if layer inputs become too large or too small dependent on the used activation functions. Small values from the FMU may not sufficiently influence the ANN output, large values lead to saturation if activation functions with limited output are used, like *tanh*, *sigmoid* or *relu*. A straight forward workaround for this is to scale and shift the FMU's output values to fit the activation function of the ANN, which is further described in (Thummerer, Stoljar, and Mikelsons 2022). To restore signal ranges at the ANN outputs, a post-processing transformation can be added here. If the ANN input signals are matching the outputs, this transformation can be initialized as inverse of the pre-processing transformation. The parameters of these pre- and post-processing operations can be initialized to suit known signal ranges and further be optimized together (as forward and reverse transformation) or separately (as individual transformations) with the ANN parameters to adapt well to changing signal ranges that might occur with further training convergence.

2.2.3 Signal selection

By design of the *Functional Mock-Up Interface* (FMI), different signals can be passed between FMU and ANN. Using all available FMU signals (in principle, any system variable can be retrieved by supported FMUs) results in suboptimal training performance, because unnecessary large Jacobian matrices need to be computed during training and the signal information is highly redundant, if besides system states and inputs further dependent variables are connected. Further, connecting signals that have no causal dependency on the physical effects being learned carries the risk of the ANN learning wrong correlations or at least delays training convergence. This motivates to define an efficient and minimal interface. A clever and automatic signal selection during training is an active topic of research. For now, choosing a small set of interface variables is driven by expert knowledge of the system and only some general instructions can be given. One is, that for higher order differential equations (mechanical system are usually modeled as second order ODEs), the order of differentiation of the ODE should be preserved (Thummerer, Stoljar, and Mikelsons 2022). This is achieved by preventing the ANN to modify the derivatives that are also states. For example, consider a translational mechanical system with the states *position* and *velocity* and the state derivatives *velocity* and *acceleration*. If the ANN is allowed to modify the entire derivative vector (here the velocity and the acceleration), the second order ODE disintegrates into a first order ODE, because the state *velocity* does not match the integrated *acceleration* anymore. This can be tackled by allowing the ANN to manipulate only the highest derivatives of the system, in the considered example the derivative *acceleration*, while passing the derivative *velocity* directly to the numerical integrator without modifications.

2.3 From Julia to modeling environment

After training, the HM needs to be reimported from *Julia* back into the original (or another) development tool. For this, three different approaches to integrate the newly learned dynamic effect into the original modeling environment are possible and explained in the following sub sections.

2.3.1 Export the HM as FMU

The entire HM, including the FPM as FMU, can be exported as FMU, see Figure 6 (step 2a). This approach is useful if model development has (almost) finished and no other modifications to the structure needs to be done or if the HM is being used in a different modeling tool that features another modeling language (but support for FMI). From software side, this can be accomplished using the *Julia* library *FMI.jl* or the related sub-library *FMIExport.jl*[3] directly. An example on how to export NeuralFMUs as FMUs is part of the *FMIExport.jl* repository.

From a technical perspective, the FPM FMU (the FMU exported from the original simulation model) is copied to the resources folder of the HM FMU (the FMU being exported from *Julia*) and most of the FMI functions of the HM FMU are directly connected to the embedded FPM FMU. Because these connections are implemented by redirecting function pointers, the execution performance of the hybrid FMU is only influenced by the ANN dynamics without overhead. To induce the modified dynamics to the HM FMU, some functions need to be extended, which is discussed in the following paragraphs at the example of model-exchange FMUs.

Getting system state derivative
The functions `fmi2GetDerivatives` (FMI2) and `fmi3GetContinuousStateDerivatives` (FMI3) need to return the dynamics of the entire HM instead of the FPM FMU. This can be simply achieved by obtaining the FPM FMU state derivative and passing it through the part of the HM topology, that modifies the system state derivative. Finally, the modified state derivatives are returned instead of the derivatives of the FPM FMU.

Setting system state
The functions `fmi2SetContinuousStates` (FMI2) and `fmi3SetContinuousStates` (FMI3) set a new state for the FPM FMU. If the system state is manipulated by the HM topology before being passed to the FPM FMU, the HM FMU state is not matching the FPM FMU state. As a consequence, a new state for the FPM FMU needs to be retrieved by optimization through the topology (Thummerer, Stoljar, and Mikelsons 2022).

Getting/Setting ANN parameters
Before overwriting the corresponding functions, the model description needs to be extended by new parameter identifiers for the ANN parameters. After that, to be able to read and change the ANN parameterization, the functions `fmi2GetReal` and `fmi2SetReal` (FMI2) and `fmi3GetFloat64` and `fmi3SetFloat64` (FMI3) need to be overwritten to get and set corresponding ANN parameters.

Finally, further functions can be overwritten to add additional functionalities but are not highlighted at this point, for example function calls to `fmi2GetDirectionalDerivatives` (FMI2), `fmi3GetDirectionalDerivatives` and `fmi3GetAdjointDerivatives` (FMI3) can easily be chained to the *Julia* AD framework to efficiently retrieve partial derivatives.

2.3.2 Export the ANN only

Export as FMU
By exporting only the trained ANN as FMU, the FPM can further be used in its original format, maintaining the white-box structure that is known by the modeling engineer, see Figure 6 (step 2b). Exporting ANNs as FMUs is supported by *FMIExport.jl*, an example is part of the library repository.

Export in a dedicated format
Further, the ANN can be exported in a dedicated format instead of being compiled into an FMU, see Figure 6 (step 2c). Two different formats are discussed in the following.

From a linguistic point of view, the *Modelica* language is capable of describing any operation that is performed by an ANN layer. Even more uncommon layers, like the gates layer used in the presented use case, can be expressed with simple mathematical operations and therefore with *Modelica*. Currently, the ANN is exported as *Modelica* model by hand, but an automated export is part of an upcoming research project. The *Modelica* ANN can easily be imported into tools that support the *Modelica* language.

Further, also the *Open Neural Network Exchange*[4] (ONNX) can be used to export the ANN separately. ONNX is capable of describing conventional as well as custom layers, as long as they are subdivisible into primitive mathematical operations (like e.g. the gates layer). For *Julia*, an ONNX library is available, called *ONNXNaiveNASflux.jl*[5], that is able to import and export ONNX models including custom layers.

Instead of exporting the ANN structure alongside with the identified parameters, it is also possible to export both elements separately. This can be achieved by importing the ANN topology in the original modeling environment and transferring the parameters separately via an arbitrary file format (like `CSV`, `MAT` or `TXT`).

[3] `https://github.com/ThummeTo/FMIExport.jl` (accessed on May 24, 2023)

[4] `https://github.com/onnx/onnx` (accessed on May 30, 2023)

[5] `https://github.com/DrChainsaw/ONNXNaiveNASflux.jl` (accessed on May 31, 2023)

DOI
10.3384/ecp204141

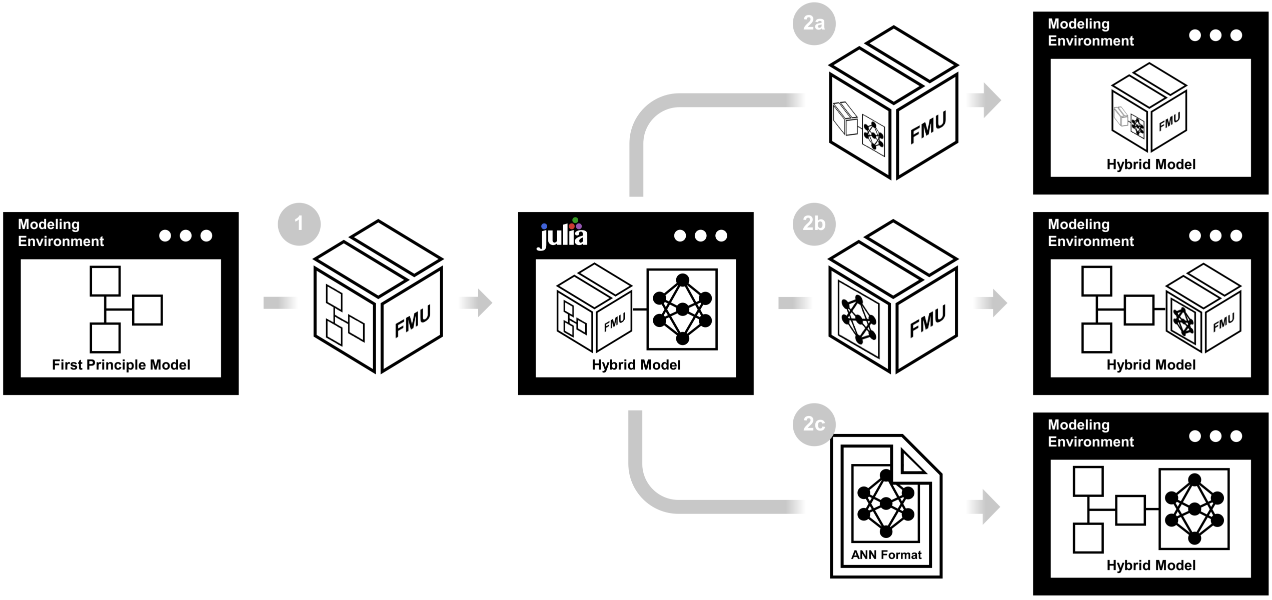


Figure 6. Comparison of different ways on how to deploy NeuralFMUs. Step (1) shows the export of the FPM as FMU from modeling environment and the import of the FMU into *Julia*. After hybrid modeling and training inside *Julia*, the entire HM can be exported as FMU (2a), only the ANN as FMU (2b) or a dedicated ANN format like *Modelica* or *ONNX* can be used for ANN export (2c). Dependent on the chosen exporting step, the further usability of the HM varies.

Reintegration of the exported ANN in the original modeling environment

The exported formats (FMU, ONNX, Modelica) share the input and output structure. When reintegrating them into the original modeling environment they all behave like data driven sub-models such as characteristic maps. The connection to the original FPM depends on the modeling tool's specifics. If the targeted modeling environment is a *Modelica* tool, the following has to be considered: The inclusion of the modified dynamics into the original FPM is not straight forward since the original *Modelica* system is acausal. The definition of variables as states or state derivatives only takes place during compilation. To modify a variable's derivative prior to compilation means to modify/replace equations in the *Modelica* system. This becomes a limitation, since ...

- . . . component models could be protected,
- . . . there are multiple equations to replace or modify, so the solution is not unique,
- . . . sometimes variables/equations do not even exist (but are only created during compilation).

Hence, in general the integration of a causal ANN into the acausal *Modelica* model is not straightforward, and remains a future research topic.

3 Experiments

In the experimental part, the NeuralFMU approach is applied to a subsystem of an excavator model. Figure 7 sketches the excavator arm and the considered subsystem.

3.1 Motivation

The model of the excavator was developed in *SimulationX*[6] during the real excavator's setup period, in which the model was used to test an automatic steering controller. Now, during the operation of the excavator, the model shall be extended and used with measurement data as a *Hybrid Twin* to detect and predict faults (i.e. deviations, leaks in hydraulic components, etc.) (Gundermann et al. 2018). The *Hybrid Twin* concept enhances physics-based models with real data from physical sensors to increase the model's accuracy and prediction capabilities (Chinesta et al. 2019). The advantage of the applied *Hybrid Twin* of the excavator - a condition monitoring system of the latter - is that the real system requires less scheduled maintenance while at the same time can be repaired in time before serious faults occur. One major prerequisite for detecting deviations using *Hybrid Twins* is that the simulated nominal behavior must match the measured data very accurately during operation. This can be acchieved by applying the NeuralFMU method to the system. The results for the bucket cylinder subsystem of the excavator (including reimport to the original model) are outlined below.

3.2 Model selection for NeuralFMU

The complete, comprehensive model of the excavator contains components of various domains of the *SimulationX* libraries, such as *Multi Body Systems* (MBS, i.e. 3D) and 1D mechanics, hydraulics and control signals. The real excavator is equipped with pressure and cylinder position

[6] `www.SimulationX.com` (accessed on May 24, 2023)

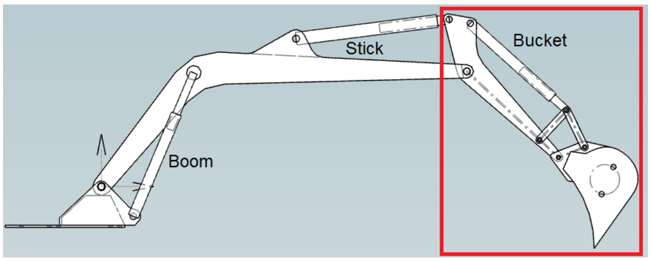


Figure 7. Sketch of the full excavator arm with all three cylinders. For the NeuralFMU approach, the subsystem controlled by the bucket cylinder (red box) is used.

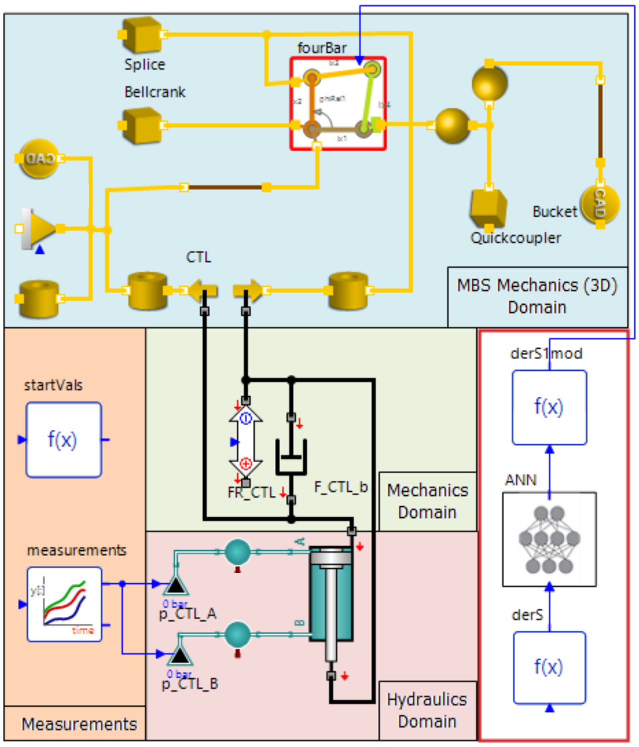


Figure 8. The submodel of the bucket cylinder and mechanics in *SimulationX*. The model shows different domains (color marked): MBS mechanics (blue), 1D mechanics (green), hydraulics (red), signal blocks for measured pressures and initial position values (orange). The original FPM is shaded with the colors of the domains. The extension of the model to a Hybrid Model (cf. subsection 3.4) is highlighted by the red boxes.

sensors, the signals of which can be recorded together with the control signals. This allows to collect data of the motion of a single cylinder as well as of the full motion of the excavator arm. Comparing model with data, one will find that there are deviations, e.g. between simulated and measured cylinder positions.

One source of deviation is the friction in the hydraulic cylinders, which is hard to model. Simple models are not able to describe the process accurately enough, sophisticated models are hard to parameterize, and the modeling task is even more complex since different parts in the cylinder system contribute to the effective friction force. To avoid the influence of interaction with connected components, the model is considered for each cylinder separately. Therefore, the bucket cylinder subsystem and attached mechanics are cut out of the full model and are used with the measured pressure signals in the cylinder chambers as inputs, to focus on the movement of the considered cylinder only. Figure 8 shows the submodel of the bucket cylinder and mechanics, which will be used in experiments, and be referred to as the *First Principle Model* (FPM). The model already includes some generic friction components - a pressure based force `FR_CTL` and a damper `F_CTL_b`, which prevents oscillations. The necessity of including or excluding such components will be examined below.

It shall be mentioned that similar submodels can be created for the other cylinders of the excavator arm, in which case there are more components on the lower end of the kinematic tree which affect the mass and momentum distribution. Note, that the system model is unstable if the center of mass of the rotated components lies above the joint around which the mechanical components rotate when extending or retracting the cylinder. This can lead to difficulties, e.g. when trying to fit the friction force for the boom cylinder. Even with submodels in a stable state, the fitting method is time-consuming and requires knowledge of engineers with system specific experience.

The cylinder chambers in Figure 8 are fed with non-constant pressure signals, taken from real measurements shown in Figure 9. These pressure signals cause the piston in the cylinder to move. As mentioned, the simulated position deviates from the measurement, which can be seen in Figure 14 which spans the same time sequence. The position of the bucket cylinder will be used as training objective in the NeuralFMU optimization. Further, the measurement data is noisy and contains irregular deviations (e.g. position measurement around 470 s), which cannot be removed by standard filters.

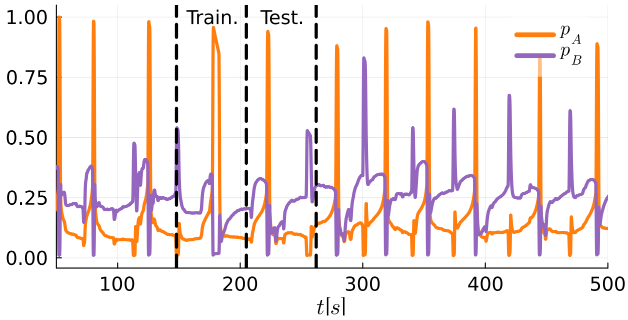


Figure 9. Measured pressure signals p_A and p_B of the two cylinder chambers (normalized, units removed).

3.3 NeuralFMU setup and validation

3.3.1 The FMU of the Bucket model

For applying the NeuralFMU method, an FMU of the FPM is generated, with the top bucket cylinder position `CTL.dx` as the output y. Figure 10 shows two extraction-retraction-cycles of the time segment used to train and test the NeuralFMU. For the NeuralFMU training, a single extraction-retraction-cycle ranging from 148 s to 205 s is selected from the data since it covers most of the possible dynamics of the measured series. A second cycle ranging from 205 s to 262 s is used for testing. Please note, that the amount of data for training is very small in terms of machine learning applications and should be increased to contain every aspect of the effect to be learned. Here however, it is shown that even small and incomplete data sets can be used to significantly improve the simulation model.

Table 1. FMU states $\boldsymbol{x}$ and output y.

Symbol	Name	Description
x_1	`der(fourBar.q[2])`	Angular velocity
x_2	`connection12.x`	Position
x_3	`fourBar.q[2]`	Angle
y	`CTL.dx`	Position

As shown in Table 1, the FMU has three states, which seems not intuitive, since the mechanical system (cf. Figure 8) has only one degree of freedom: the motion around the revolute joint in the four bar. The related states are the angle `fourBar.q[2]` and the corresponding angular velocity `der(fourBar.q[2])`. The third state `connection12.x` is the position of the 1D mechanical connection between force interface (`CTL`) and hydraulic cylinder port B, which is equivalent to the length of the bucket cylinder. This state is introduced since the 3D-1D force interface creates a constraint on the 1D velocity. Hence a differential equation for the 1D position (`connection12.x`) exists. Further, the bucket cylinder position `CTL.dx` is identical to the state `connection12.x` in the considered simulation range. Because of the zero-crossing behavior in `der(fourBar.q[2])`, the FMU has state events that must be handled during simulation to obtain correct simulation results (Blochwitz et al. 2011).

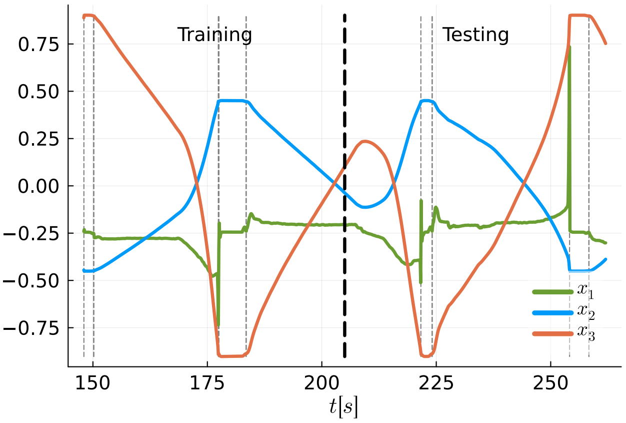


Figure 10. FMU states $\boldsymbol{x}$ (normalized, units removed) during training and testing. State events (discontinuities) are triggered whenever the cylinder reaches or leaves one of the end stops. In the graph, they are shown as gray-dashed horizontal lines.

3.3.2 The definition of the ANN

The definition of the ANN topology bases on past experience and empirical hyper parameter tuning, and is considered as large enough to cover the dynamics, while being trainable in reasonable time. In addition to a core of two dense layers, pre- and post-processing layers are included, as shown in Figure 11. These shift and scale the value of all inputs to a distribution with mean 0 and standard deviation 1 to suit the applied activation function *tanh* of the first dense layer. The figure further shows the additional gates layer introduced in subsubsection 2.2.1 which allows efficient training of the unknown friction effect and a solvable initialization for the system. The final ANN topology with 91 parameters is shown in Table 2.

Table 2. Parameters of the NeuralFMU topology.

Type	Inputs	Outputs	Bias	Num.
Pre-process	3	3	0	6
Dense	3	16	16	64
Dense	16	1	1	17
Post-process	1	1	0	2
Gates	2	1	0	2
			Total:	91

3.3.3 Solver details and loss function

To solve the NeuralFMU in *FMIFlux.jl*, the explicit Runge-Kutta method *Tsit5* (Tsitouras 2011) is used as numerical ODE solver. Using the gradient-based optimization algorithm *Adam* (Kingma and Ba 2017) with step size 10^{-3}, the following loss function of the mean absolute error (*mae*) is minimized during training:

$$mae(\boldsymbol{x}_2, \hat{\boldsymbol{x}}_2) = \frac{1}{n} \sum_{i=1}^{n} \left| x_2^i - \hat{x}_2^i \right|, \tag{1}$$

where x_2^i is the simulated bucket cylinder position `connection12.x` at time instant i, $\hat{x}_2^i$ the corresponding measured position, and n the number of compared measurement points.

3.3.4 Training and testing in Julia

After defining the ANN topology and its training parameters, the NeuralFMU is trained in *Julia* for 2000 epochs on the first extraction-retraction-cycle of the selected time interval ranging from 148 s to 205 s in 0.1 s second increments. Two experiments are considered: The training of a *Hybrid Model* (HM) on basis of the FPM with a simple friction model and on basis of the FPM without friction (undamped system).

FPM with friction model

The results after training of the NeuralFMU are very close to the measurement data, which is confirmed by the small value of the loss function from Equation 1 of 0.0094 m. After training is complete, the NeuralFMU is simulated the second extraction-retraction-cycle in the time interval

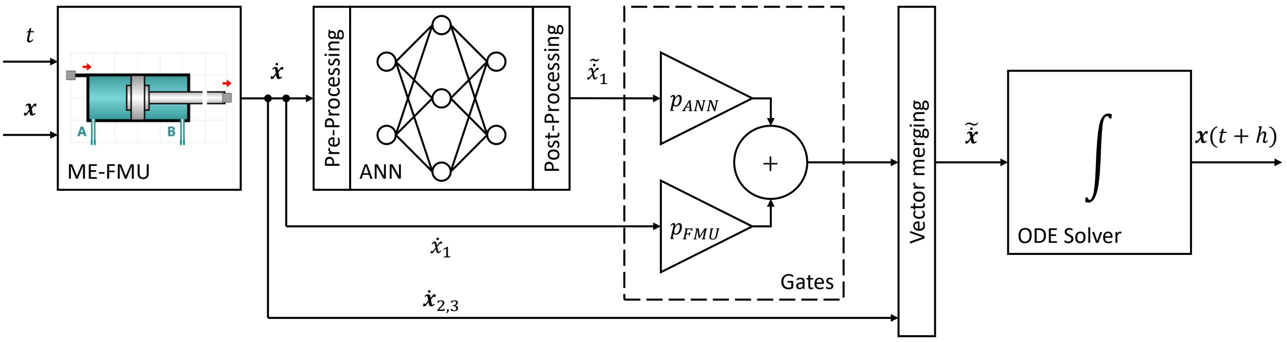


Figure 11. The topology of the used NeuralFMU using gates adapted from (Thummerer, Stoljar, and Mikelsons 2022). The current system state $\boldsymbol{x}$ and time t are passed to the FMU, which computes the system state derivative $\dot{\boldsymbol{x}}$. On basis of the state derivatives from FMU, only the revolute joint acceleration $\tilde{\dot{x}}_1$ is computed by the ANN (featuring pre- and post-processing layers) and linearly combined with the corresponding derivative from the FMU $\dot{x}_1$ in the gates layer. The remaining FMU derivatives $\dot{\boldsymbol{x}}_{2,3}$ only enter the ANN as inputs and are bypassed directly to the final derivative vector $\tilde{\dot{\boldsymbol{x}}}$, to preserve a second order ODE.

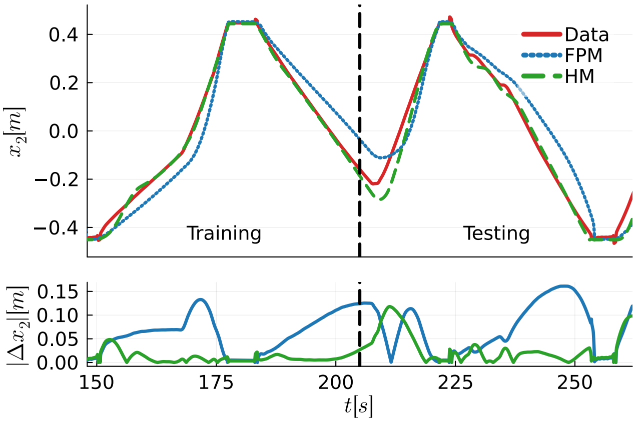


Figure 12. Comparison of bucket cylinder position to data (top) and absolute deviation from data (bottom). The HM is modeled on basis of the FPM with simple friction.

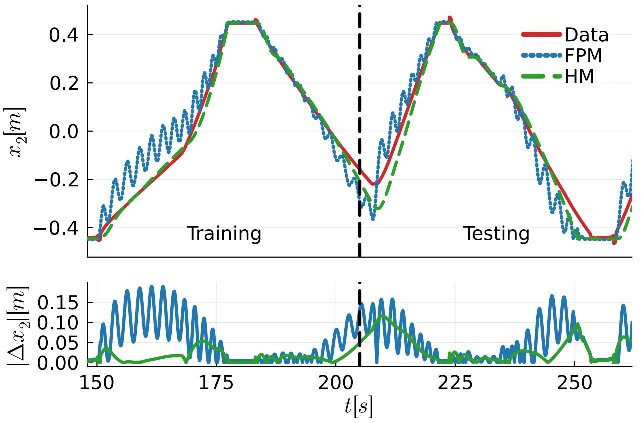


Figure 13. Comparison of bucket cylinder position to data (top) and absolute deviation from data (bottom). The HM is modeled on basis of the FPM without friction.

from 205 s to 262 s, shown in Figure 12. The test result shows that the loss function value of 0.0228 m doubled compared to the training sequence, but it is still three times smaller than the loss function value of 0.0696 m in the original FPM FMU simulation (cf. in Table 3).

FPM without friction model

As introduced, another NeuralFMU is built using an FPM without friction components. Training and testing results for the same sequence are shown in Figure 13. These confirm the fact, that the NeuralFMU can be trained completely without friction components and the ANN can compensate the oscillations of the border-stable system. The loss function value of the NeuralFMU featuring the FPM without friction is 0.0327 m and therefore about 30 % higher than the one trained on basis of the FPM with friction for the same sequence (cf. in Table 3). The accuracy of the case without friction under the same training conditions is lower than the one implementing viscous damping, therefore the model with friction is used for the outlined application.

Table 3. Loss function values.

Model	FPM friction	Training [m]	Testing [m]
FPM	yes	0.0616	0.0696
HM	yes	0.0094	0.0228
FPM	no	0.0543	0.0497
HM	no	0.0157	0.0327

3.4 Hybrid Model in SimulationX

As described in section 2.3.2, one way to reimport the NeuralFMU to the original modeling environment is to export the ANN in a dedicated format and to couple it with the FPM (or an FMU containing the FPM) inside the modeling tool. Thereby after training and testing in *Julia*, the ANN parameters are exported as a `TXT` file and a *Modelica* model with an equivalent network topology (cf. Figure 11) and the parameter values loaded from this text file is created. Listing 1 shows the equation section of the ANN type implemented in *SimulationX*.

Listing 1. Equation section of Modelica type of ANN

```
preProcess  = (dxIn+prePShift).*prePScale;
```

```
dense1      = tanh(preProcess*w1 + b1);
dense2      = dense1*w2 + b2;
postProcess = dense2*postPScale+postPShift;
dxOut={
   gates[1]*dxIn[1]+ gates[2]*postProcess,
   dxIn[2],
   dxIn[3]};
```

Here, `dxIn` and `dxOut` are the state derivatives as calculated by the FPM and modified by the ANN, respectively. `prePShift`, `prePScale`, `w1`, `b1`, `w2`, `b2`, `postPScale`, `postPShift` and `gates` are the optimized parameters of the different layers. To include the ANN in the FPM, equations have to be modified (cf. Sec. 2.3.2). In the bucket system (see Figure 8), this affects only the equation which defines the angular acceleration in the four bar. The variables defining the FMU state derivatives (cf. Table 1) are read from the model components (`derS`), and are passed as input into the `ANN`. The modified `der(fourBar.q[2])` enters the `FourBar` component as an input (named `derS1mod`) and is used instead of the original variable. The equation section in the four bar is modified as written below:

Listing 2. Modification of FourBar equation section

```
der(q[2]) = om[2];     //—>  input 3 to ANN
alp[2] = ...           //—>  input 1 to ANN
//der(om[2]) = alp[2];  replaced by:
der(om[2]) = derS1mod; //<— output 1 of ANN
```

In Figure 8, the extension and modifications of FPM by the ANN are highlighted with red boxes. The bottom right box shows state derivatives (`derS`), `ANN`, and modified state derivative of state x_1 (`derS1mod`). This modified derivative is fed into the modified `fourBar` component (also highlighted with red box).

The output of the HM with a simple friction model simulated in *SimulationX* in the interval from $50s$ to $500s$ is shown in Figure 14. The figure also shows that the bucket cylinder position can be reliably predicted in *SimulationX* for a measurement sequence that was not used to train and test the NeuralFMU. Comparing the results in the time interval for training and testing, the NeuralFMU shows the same cylinder position in *Julia* as well as in *SimulationX* for the original FPM and HM without friction, as can be seen when comparing results in *SimulationX* (Figure 14 and 15) with results in *Julia* (Figure 12 and 13). Figure 15

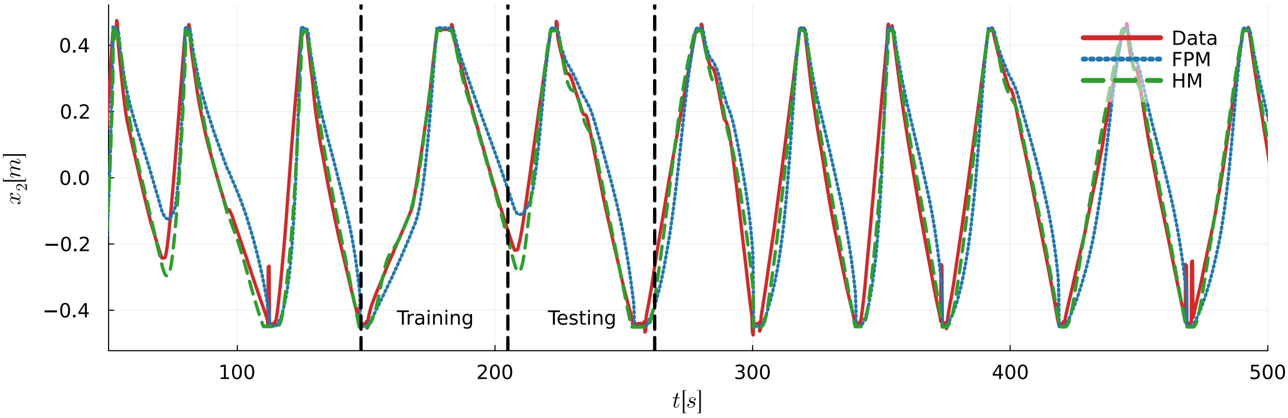


Figure 14. Measured data and simulation results of FPM and HM for the bucket cylinder position for the entire measurement. The HM is modeled on basis of the FPM with simple friction.

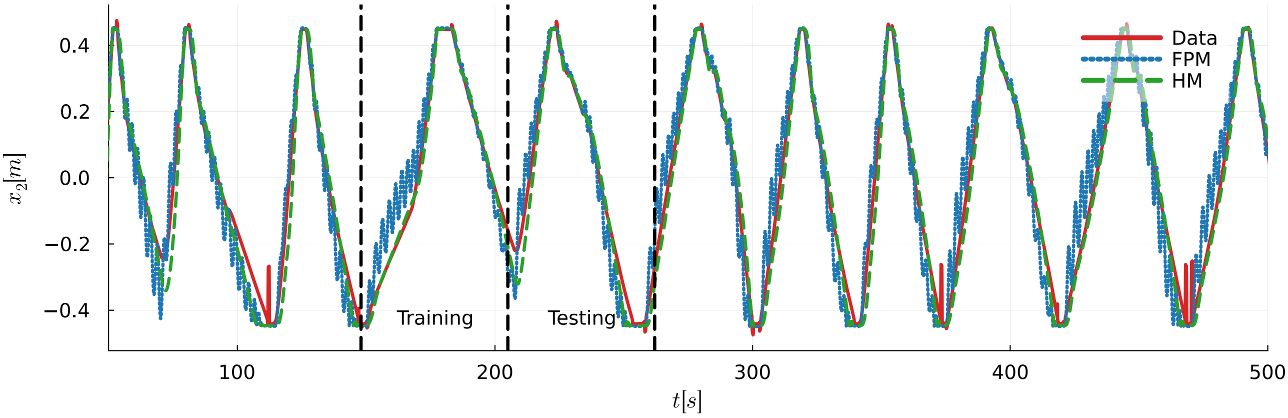


Figure 15. Measured data and simulation results FPM and HM for the bucket cylinder position, for the entire measurement. The HM is modeled on basis of the FPM without friction.

also confirms that the HM without friction is able to damp oscillations for the full measurement sequence.

4 Conclusion

We started by introducing the concept of a NeuralODE and adapted this machine learning model step by step to suit the requirements of industrial engineering applications, resulting in a so called NeuralFMU. We highlighted a generic workflow, that allows for NeuralFMU development in custom applications, dependent on tool capabilities and further model use. Finally, we exemplified the presented theory at a real engineering use-case: The modeling of a *Hybrid Twin* of a hydraulic bucket cylinder to be used for process and failure monitoring.

Next, additional features of the *FMIFlux.jl* package shall be investigated - input values to NeuralFMUs for interaction with other excavator components and the batching method for more effective training on a broader data base. Besides, we plan to apply the workflow to larger submodels and other examples, and to automate parts of the workflow where possible. For the excavator model, the final goal is the creation of the *Hybrid Twin* for the detection of malfuntions in the system.

Besides physical equations, further system knowledge in the form of ODE properties like stability, oscillation capability, stiffness as well as frequency and damping information can be integrated into the PeNODE training process in form of Eigen-informed NeuralODEs (Thummerer and Mikelsons 2023). All listed properties depend on the eigenvalue positions of the system model over time. By computing eigenvalues and rating their positions as part of the (or an additional) loss function, the desired ODE properties can be enforced for the considered *Hybrid Twin* and may further improve training and prediction.

Author Contributions

NeuralFMU method, implementation and support: T.T., L.M.; Experimental results in *Julia* and *SimulationX*: A.K, J.G.; Preparation of excavator model and data: D.R.; Writing: A.K, T.T., J.G., L.M.; All authors have read and agreed to the published version of the manuscript.

Funding

The contributions to this research were partially funded by different projects:

- the ITEA3-Project Unleash Potentials in Simulation (UPSIM) Nr. 19006. `https://www.upsim-project.eu/`
- Bauen 4.0, funded by German Federal Ministry of Education and Research (grant number 02P17D230). `https://www.verbundprojekt-bauen40.de/`
- Proper Hybrid Models for Smarter Vehicles (PHyMoS), funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) (grant number 19I2022A). `https://phymos.de/`

(all accessed on May 23, 2023)

The authors thank the organizations for their funding.

References

Bezanson, Jeff et al. (2014). “Julia: A Fresh Approach to Numerical Computing”. In: *CoRR* abs/1411.1607. arXiv: 1411.1607. URL: http://arxiv.org/abs/1411.1607.

Bittner, L. (1963). “L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, E. F. Mishechenko, The Mathematical Theory of Optimal Processes. VIII + 360 S. New York/London 1962. John Wiley & Sons. Preis 90/–”. In: *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik* 43.10-11, pp. 514–515. DOI: 10.1002/zamm.19630431023.

Blochwitz, T. et al. (2011). *The functional mockup interface for tool independent exchange of simulation models*. URL: https://publica.fraunhofer.de/handle/publica/371243.

Champaney, Victor et al. (2022). “Modeling systems from partial observations”. In: *Frontiers in Materials* 9. ISSN: 2296-8016. DOI: 10.3389/fmats.2022.970970.

Chen, Tian Qi et al. (2018). “Neural Ordinary Differential Equations”. In: *CoRR* abs/1806.07366. arXiv: 1806.07366. URL: http://arxiv.org/abs/1806.07366.

Chinesta, Francisco et al. (2019). “Virtual, Digital and Hybrid Twins: A New Paradigm in Data-Based Engineering and Engineered Data”. In: *Archives of Computational Methods in Engineering*. DOI: 10.1007/s11831-018-9301-4.

Gundermann, Julia et al. (2018). “The Fault library - A new Modelica library allows for the systematic simulation of non-nominal system behavior”. In: *Proceedings of the 2nd Japanese Modelica Conference*. Linköping Electronic Conference Proceedings 148. Linköping, Sweden: Modelica Association and Linköping University Electronic Press, pp. 161–168. DOI: 10.3384/ecp18148161.

Kingma, Diederik P. and Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].

Thummerer, Tobias, Josef Kircher, and Lars Mikelsons (2021-09). “NeuralFMU: Towards Structural Integration of FMUs into Neural Networks”. In: *Proceedings of the 14th International Modelica Conference*. Ed. by Martin Sjölund et al. Linköping Electronic Conference Proceedings 181. Linköping, Sweden: Modelica Association and Linköping University Electronic Press, pp. 297–306. ISBN: 978-91-7929-027-6. DOI: 10.3384/ecp21181297.

Thummerer, Tobias and Lars Mikelsons (2023). *Eigen-informed NeuralODEs: Dealing with stability and convergence issues of NeuralODEs*. arXiv: 2302.10892 [cs.LG].

Thummerer, Tobias, Johannes Stoljar, and Lars Mikelsons (2022). “NeuralFMU: Presenting a Workflow for Integrating Hybrid NeuralODEs into Real-World Applications”. In: *Electronics* 11.19. ISSN: 2079-9292. DOI: 10.3390/electronics11193202.

Thummerer, Tobias, Johannes Tintenherr, and Lars Mikelsons (2021-11). “Hybrid modeling of the human cardiovascular system using NeuralFMUs”. In: *Journal of Physics: Conference Series* 2090.1, p. 012155. DOI: 10.1088/1742-6596/2090/1/012155.

Tsitouras, Ch. (2011). “Runge–Kutta pairs of order 5(4) satisfying only the first column simplifying assumption”. In: *Computers & Mathematics with Applications* 62.2, pp. 770–775. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2011.06.002.

Modeling and simulation of dynamically constrained objects for limited structurally variable systems in Modelica

Robert Reiser[1] Matthias J. Reiner[1]

[1]Institute of System Dynamics and Control, German Aerospace Center (DLR), Germany,
{firstname.lastname}@dlr.de

Abstract

This work introduces a new solution for the modeling and simulation of dynamically constrained objects for limited structurally variable systems purely in Modelica. A combination of a collision detection algorithm, the limitation of collisions, and a method to constrain objects based on forces leads to a constraint network in Modelica. It allows a stable and accurate simulation of applications such as robot tool changers in a flexible way without the need for predefined connections in the model.

Keywords: collision detection, structural variability, constraint force, network, tool change, robotics, Modelica

1 Introduction

Structurally variable systems often occur in different fields of technical problems. One prominent example is robot tool changers. The production industry has an ever-increasing demand for flexibility because manufacturing is shifting from standardized products with high quantities to individual goods. Tool changers are a common method to increase the flexibility of an assembly cell.

Simulation is important for the development and testing of robot cells for example in virtual commissioning (Wünsch 2008). There have been many works about the simulation of robots (Paryanto et al. 2014; Bellmann, Seefried, and Thiele 2020; Reiser et al. 2022) and manipulation (Reiser 2021) in Modelica. Existing tools such as RoboDK, CoppeliaSim, and ANSYS (Li et al. 2016) can be used to simulate tool changers but they do not offer the high degree of flexibility of the Modelica language.

In the Modelica environment (Modelica Association 2017) it is especially challenging to simulate applications such as tool changers since structural variability is not possible, limited to special cases (Stüber 2017) or requires additional effort (Tinnerholm, Pop, and Sjölund 2022). To our knowledge, there has been no work in the area of simulating a tool changer based on Modelica models.

There are also examples in the field of aeronautics, such as the structural changes during runtime in the area of stage separation (Acquatella and Reiner 2014) or for in orbit construction of orbital platforms (Reiner 2022).

This work builds on the previous work of (Acquatella and Reiner 2014) and (Reiner 2022) to make it usable in a wider range of applications, and shows how robot tool changers can be modeled using the proposed technique. We present a new solution for the modeling and simulation of structurally variable systems in Modelica. It combines:

- A collision detection algorithm in Modelica
- A method to limit the number of possible collisions (see Figure 1 and section 3.2 for details)
- The Constraint Force Equation method combined with the Baumgarte stabilization technique

The next section introduces the state of the art. Section 3 presents the new solution for the modeling of structurally variable systems in Modelica. The following section contains the implementation. In Section 5, a tool change process and the transition of a spring-borne object are simulated. In conclusion, the approach is discussed and future developments are considered.

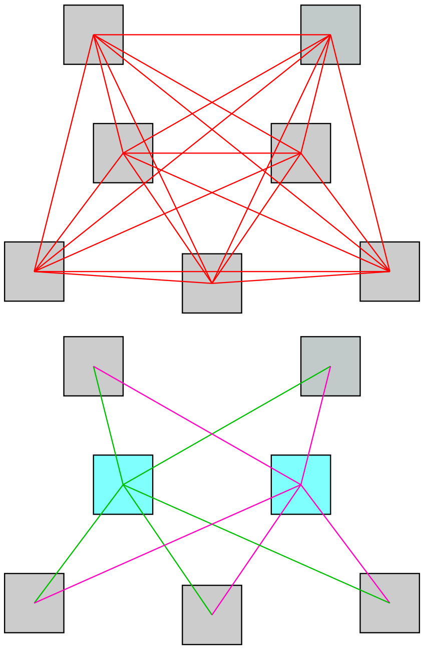


Figure 1. An example showing the limitation of possible collisions. If all collisions are allowed, there are 21 possible collision pairs (top). A restriction divides the objects into two groups and only collisions between blue and grey objects are allowed. Now there are only 10 possible collision pairs (bottom).

2 State of the art

In this section, related existing work is analyzed. This includes an overview of collision detection and the introduction of the Constraint Force Equation (CFE) method and the Baumgarte stabilization technique.

2.1 Collision detection

Collision detection is a complex task. There are special libraries such as the **libccd** (Fiser 2018) to detect collisions between convex shapes. The most common algorithms are the Gilbert-Johnson-Keerthi distance algorithm (GJK) (Gilbert, Johnson, and Keerthi 1988) and the Minkowski Portal Refinement algorithm (MPR) (Snethen 2008).

There are several works about collision detection in Modelica: (Otter, Elmqvist, and López 2005), (Hofmann et al. 2014), and (Elmqvist et al. 2015).

To the knowledge of the authors, the approaches above have in common that they combine Modelica with an external library for the collision detection task. The usage of external function calls can result in delays (i.e. values might be one time step behind in the simulation), additional model complexity, and possible incompatibility when used on different computing platforms.

In the works of (Oestersötebier, Wang, and Trächtler 2014) and (Bortoff 2020), the collision detection is native in Modelica. However, predefined contacts are needed.

2.2 Constraint Force Equation method and Baumgarte stabilization

The Constraint Force Equation (CFE) method was developed at NASA (Toniolo et al. 2008). The aim is to constrain two bodies by applying joint forces to each body. (Acquatella and Reiner 2014) used this method for the modeling and simulation of stage separation dynamics in Modelica and (Reiner 2022) for robot based in orbit construction of orbital platforms. Their solution is however limited because the contact pairs are predefined or use special cases and cannot be changed during runtime. Therefore an application such as a tool changer cannot be modeled easily. A tool is usually connected to more than one robot and a tool holder and the connections change during a process. A more general method is needed.

The Baumgarte stabilization (Baumgarte 1972) is used to stabilize the constraint force equation. The basic formula for the constrained force calculation with Baumgarte damping can be seen in the following equation:

$$\ddot{\xi} + 2\eta\dot{\xi} + \eta^2\xi = 0 \tag{1}$$

ξ represents the (generalized) difference in position and orientation between the two objects and $\eta > 0$ the damping factor, resulting in an asymptotically stable ODE. Note that the constraint is defined as a kinematic condition. Modelica can automatically calculate the resulting forces and torques when the equation is correctly used together with mechanical bodies with mass and/or inertia (see implementation details in later sections).

Since the constraint $\ddot{\xi} = 0$ is defined on the relative acceleration between the to be constrained objects, small numerical errors can lead to drift in the relative generalized velocity $\dot{\xi}$ and position ξ between the objects. Using the additional damping terms in equation 1 for the relative velocities and positions between the objects can reduce this drift substantially. See section 4.6 for more details on the implementation used here.

3 Modeling structurally variable systems in Modelica

The solution for modeling structurally variable systems in Modelica is presented in the following. The section starts with the general idea behind this approach. Furthermore, the method for limiting the number of possible collisions is introduced and a Modelica native collision detection algorithm is presented.

3.1 Idea

The idea is to build a constraint network within Modelica. By forgoing external libraries, the approach is stable and accurate. In general, the CFE method implemented in (Acquatella and Reiner 2014) is combined with a collision detection algorithm. Thus, predefined contact pairs are no longer required. In addition, the number of possible collisions is limited to achieve a higher performance.

3.2 Limitation of possible collisions

Building a general collision detection library in Modelica is challenging. However, it is possible to restrict the scope.

In collision detection, the number of possible collision pairs x depends on the number of objects in the scene n and can be determined by:

$$x = \frac{n!}{k! \cdot (n-k)!} = \frac{n!}{2 \cdot (n-2)!} \tag{2}$$

This is based on the equation to calculate the number of combinations of k from n elements. An example is shown in Figure 1 (top). Seven objects in a scene have 21 possible collision pairs. With an increasing number of objects, the number of pairs increases significantly. To avoid this, the number of possible collisions is limited by dividing the objects in a scene into two groups and allowing only collisions between objects of one group and another.

In the example, one group contains two and the other group five objects. Therefore the number of pairs decreases to ten objects (see Figure 1 (bottom)).

Now the number of possible combinations is:

$$x = n_1 \cdot n_2 \tag{3}$$

where n_1 is the number of objects of the first and n_2 the number of objects of the second group.

The number of collision checks performed during simulation runtime is directly related to the number of possible collision pairs in a scene. In other words, limiting the number of possible collision pairs allows a fast collision detection algorithm in native Modelica code.

3.3 Collision detection in Modelica

The reasons for building a native Modelica collision detection algorithm are a high stability and compatibility of the resulting simulations and a high accuracy of the results.

Implementing a collision detection algorithm (e.g. GJK or MPR) in Modelica in general is not feasible since operations such as the handling of complex 3D models are hardly manageable without external code. Furthermore, such an algorithm would have a weak performance because it is not possible to use all the optimization techniques usually applied in collision detection libraries.

Hence, for this work, the collision detection algorithm is highly restricted. Only the following contact combinations are allowed:

- Sphere and sphere
- Sphere and rectangle surface (with length and width)

This reduces the complexity significantly and leads to a fast calculation of collision checks.

The collision detection between **two spheres** (located at position $p_{Sphere1}$ and $p_{Sphere2}$) is straight forward: Each object has a radius and the collision check between two objects is based on the Euclidean distance. The Euclidean distance $d_{Euclidean}$ is calculated by:

$$d_{Euclidean} = \|p_{Sphere1} - p_{Sphere2}\| \tag{4}$$

Using the sum of the radius $r_{Sphere1}$ and $r_{Sphere2}$ of both objects a collision occurs when the following inequality is fulfilled:

$$d_{Euclidean} < r_{Sphere1} + r_{Sphere2} \tag{5}$$

The collision check between **sphere and rectangle surface** can also easily be calculated. The sphere (located at position p_{Sphere}) is defined by its radius r_{Sphere} and the rectangle (located at position $p_{Rectangle}$) by its length $l_{Rectangle}$ and width $w_{Rectangle}$. The distance vector between the sphere origin and the rectangle origin d_{SP} in the orientation of the rectangle $T_{Rectangle}$ can be calculated by:

$$d_{SP} = \begin{pmatrix} d_{SP1} \\ d_{SP2} \\ d_{SP3} \end{pmatrix} = T_{Rectangle} \cdot (p_{Sphere} - p_{Rectangle}) \tag{6}$$

For the implementation, it is assumed that the rectangle normal is the local z-axis of the object. Now a simple distance inequality can be checked to determine the collision. If all of the following inequalities are fulfilled, a collision occurs between the sphere and the rectangle surface:

$$d_{SP1} < l_{Rectangle} \tag{7}$$

$$d_{SP2} < w_{Rectangle} \tag{8}$$

$$d_{SP3} < r_{Sphere} \tag{9}$$

Since these inequalities are easy to solve in Modelica for a limited number of objects, a native implementation is possible with good computational performance, while still maintaining the flexibility and power of the Modelica language.

4 Implementation

This section shows the implementation of the solution presented in section 3. Figure 2 shows an overview of the implementation in the library browser. The structure consists of three objects (see Figure 3 for details):

- **CollisionCollector** (outer object to store information of the CollisionObjects and ConstrainedObjects)
- **CollisionObject** (lightweight object whose position and orientation are stored in the CollisionCollector)
- **ConstrainedObject** (contains the collision detection algorithm and calculates the constraint forces)

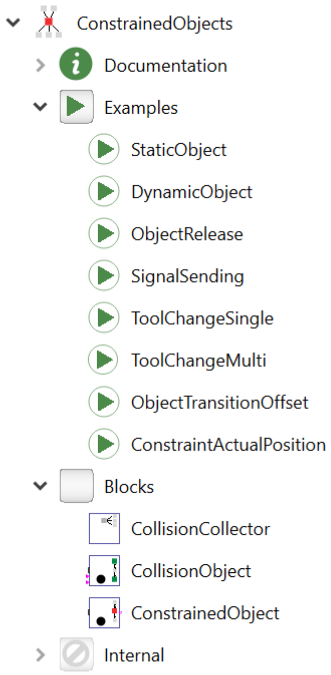
4.1 Objects

This section contains descriptions for the objects of the implemented solution (CollisionCollector, CollisionObject, ConstrainedObject). An overview of all objects and their interaction is illustrated in Figure 3.

4.1.1 CollisionCollector

The **CollisionCollector** is an outer object to store all information of the CollisionObjects. This includes the position, velocity, acceleration, orientation, angular velocity, and angular acceleration of each object. Further information are the collision type (see section 4.5), the shape type (sphere/rectangle) with the related radius, length and width, and the closed indicator.

In addition, the CollisionCollector stores information of the ConstrainedObject. This includes the force and torque calculated in the ConstrainedObject and the ID of the CollisionObject in contact with the ConstrainedObject.



ConstrainedObjects
- Documentation
- Examples
  - StaticObject
  - DynamicObject
  - ObjectRelease
  - SignalSending
  - ToolChangeSingle
  - ToolChangeMulti
  - ObjectTransitionOffset
  - ConstraintActualPosition
- Blocks
  - CollisionCollector
  - CollisionObject
  - ConstrainedObject
- Internal

Figure 2. Overview of the library structure.

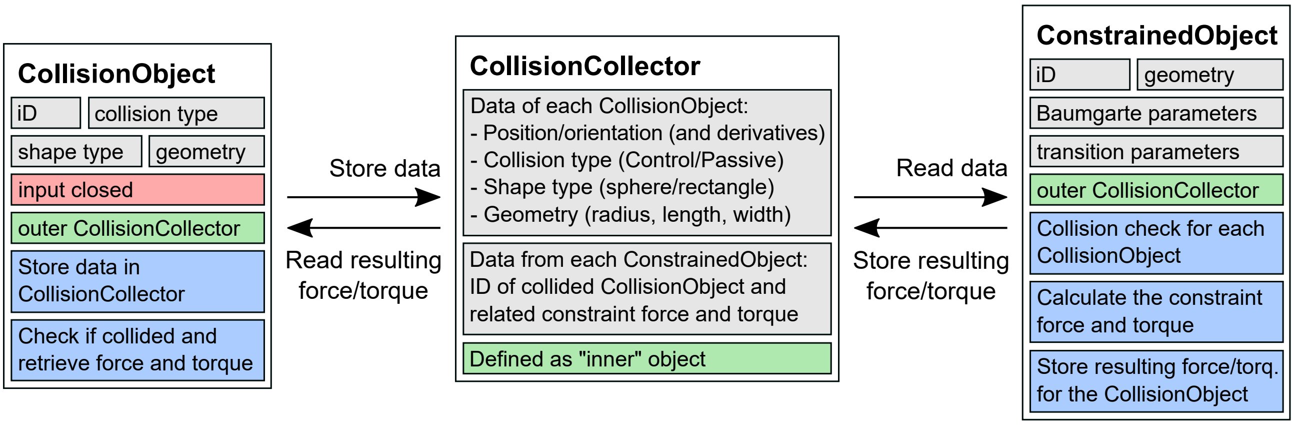

CollisionObject
iD
collision type
shape type
geometry
input closed
outer CollisionCollector
Store data in CollisionCollector
Check if collided and retrieve force and torque
Store data
Read resulting force/torque
CollisionCollector
Data of each CollisionObject:
- Position/orientation (and derivatives)
- Collision type (Control/Passive)
- Shape type (sphere/rectangle)
- Geometry (radius, length, width)
Data from each ConstrainedObject: ID of collided CollisionObject and related constraint force and torque
Defined as "inner" object
Read data
Store resulting force/torque
ConstrainedObject
iD
geometry
Baumgarte parameters
transition parameters
outer CollisionCollector
Collision check for each CollisionObject
Calculate the constraint force and torque
Store resulting force/torq. for the CollisionObject

Figure 3. Overview of the implemented objects with parameters (grey), inputs (red), methods (blue), and the inner/outer dependency (green). The arrows show the interaction based on the data flow. The CollisionObject stores its data in the CollisionCollector. The ConstrainedObject reads this data and performs a collision check for each CollisionObject. If a collision occurs, the ConstrainedObject calculates the constraint force and torque. The resulting force and torque for the related CollisionObject are returned.

4.1.2 CollisionObject

The **CollisionObject** is lightweight and has mainly the aim to store its position and orientation and their derivatives to the CollisionCollector (used as outer object). Parameters are the ID, the collision type (see section 4.5), and the shape type (sphere/rectangle) with the related radius, length, and width. The closed indicator is an input.

4.1.3 ConstrainedObject

The **ConstrainedObject** contains the collision detection algorithm and equations to calculate the constraint forces and torques. It also uses the CollisionCollector as outer object. Parameters are the ID, the radius (the ConstrainedObject is always a sphere), the damping factor for the Baumgarte stabilization (η), the duration for the smooth transition phase, and settings to enable the smooth transition phase and the offset.

4.2 Building the constraint network

The constraint network is built as follows:

- The CollisionObjects store their information in the CollisionCollector. This includes the position and orientation (with velocity and acceleration), radius (or rectangle length and width), collision type, collision shape, and closed indicator.

- Each ConstrainedObject runs a collision check to all CollisionObjects by calculating the corresponding equations from section 3 (it gets the relevant data via the CollisionCollector).

- If a collision occurs, the constraint force and torque are calculated (see section 4.6) to constrain the ConstrainedObject to the related CollisionObject.

- The resulting force and torque for the related CollisionObject is then returned (see section 4.6).

4.3 Boundary conditions

To achieve the procedure in section 4.2, some boundary conditions are necessary:

Only collisions between CollisionObjects and ConstrainedObjects are possible. Two CollisionObjects can't collide. The same applies for two ConstrainedObjects. An example is Figure 1 (bottom), where the blue objects as can be seen as ConstrainedObjects and the grey ones as CollisionObjects.

The ConstrainedObject can only be constrained to one CollisionObject (in other words it can only have a collision with one CollisionObject). This reduces the number of possible combinations significantly and improves the performance, while still allowing to model many relevant scenarios.

4.4 Manual definition of IDs

The *Modelica Language Specification* (Modelica Association 2017) does not provide the capabilities for unique IDs although it has been proposed in the past (Otter, Elmqvist, and López 2005; Hellerer and Buse 2017). To achieve a standard compliant solution, one necessity is the manual definition of unique IDs for each CollisionObject (1 ... n) and ConstrainedObject (1 ... m). In addition, the count must be set for both objects in the CollisionCollector.

4.5 Opening and closing connections

The boundary conditions restrict the ConstrainedObject to only one collision. This leads to the following question: How can tool changers be simulated having contact with both the robot and the holder? A tool changer is only constrained to one other object, the robot or the holder. But there is a transition phase as well.

Therefore additional capabilities are needed. The **CollisionObjects have a mode**, defined by a type. There are two possible types: *Control* and *Passive*. In *Passive* mode, nothing changes for the CollisionObject. In *Control* mode, the CollisionObject only collides if an additional input *closed* is true.

In addition, the **CollisionObjects are prioritized** based on their mode. If a ConstrainedObject collides with two CollisionObjects, the one in *Control* mode is higher prioritized, i.e. the collision occurs with this object.

Now the simulation of tool changers is possible. The tool is attached to a ConstrainedObject held by a CollisionObject in *Passive* mode. A CollisionObject in *Control* mode is attached to the robot flange. When the robot has approached the tool, the *closed* indicator of its CollisionObject switches from false to true. This enforces the ConstrainedObject of the tool to switch its constraint from the holder to the robot. An application of this procedure is demonstrated in section 5.1.

4.6 Calculation and return of the constraint force and torque

Constraining accelerations in a complex inner/outer scenario only leads to forces and torques in the ConstrainedObject. There is a high relevance for the constraint forces and torques in the CollisionObject, e.g. to determine the load on the robot.

Hence the force and torque in the ConstrainedObject must be transferred back to the CollisionObject. This is achieved by using the CollisionCollector. Each ConstrainedObject adds the resulting force and torque for the related CollisionObject and the ID of the related CollisionObject to the CollisionCollector (if a collision occurs). The CollisionObject is then able to check for its own ID in the CollisionCollector and if it occurs it retrieves the stored force and torque and applies it to its frame.

In the case of a tool changer, it makes sense to constrain the tool exactly at the position of the robot tool center point (TCP). In reality, there would be some kind of mechanical flange to fixate the tool exactly there. For other scenarios, it is necessary to constrain one object to another at the contact point. An example is the robot based in orbit construction of orbital platforms (see (Reiner 2022)). For such applications, the position and orientation of the ConstrainedObject should be kept and reaction forces computed accordingly. To achieve this within the same Modelica framework, position and orientation offsets have to be computed at the time t_0 when the collision occurs.

The calculation of the resulting constrained force is described in the following (the calculation of the torque is omitted for brevity).

When a collision is detected as described in section 3.3, the position offset p_{offset} is computed as the difference between the position p_c of the counterpart and the ConstrainedObject itself p_s. The start time t_0 is also saved.

$$p_{offset} = \begin{cases} 0 & \text{if no collision} \\ p_c - p_s & \text{if collsion deteced} \end{cases} \tag{10}$$

Since collisions can occur at high speed between objects this can lead to numerical problems when using fixed-step solvers for quick simulations, especially when elastic systems with weak damping are involved. To elevate this problem a slack or transition function $s_{tr}(t)$ can be enabled (optional) to scale up the constraint forces and torques. If not enabled, $s_{tr}(t)$ is simply set to 1 at all times.

It can be parameterized by its duration t_d (should be chosen as small as possible to achieve a stable simulation). The transition function is a smooth function between zero and one and can be differentiated by Modelica automatically (see equation 11).

$$s_{tr}(t) = \begin{cases} 1 & \text{if } t - t_0 >= t_d \\ \left(\sin\left(\frac{(t-t_0)\cdot\pi}{2\cdot t_d}\right)\right)^2 & \text{if } t - t_0 < t_d \end{cases} \tag{11}$$

Equation 12 shows the constrained equation, which leads to the calculation of the constraint force $f_{con,s}$ acting on the ConstrainedObject itself when connected to a mechanical body.

$$\begin{aligned} s_{tr}(t) \cdot ((\ddot{p}_c - \ddot{p}_s) + 2 \cdot \eta \cdot (\dot{p}_c - \dot{p}_s) \\ + \eta^2 \cdot (p_c - p_s - p_{offset})) = 0 \end{aligned} \tag{12}$$

Equation 13 can then be used to calculate the corresponding reaction force $f_{con,c}$ acting on the CollisionObject using the rotation matrices of both objects (T_c and T_s).

$$f_{con,c} = -(T_c \cdot T_s^T) \cdot f_{con,s} \tag{13}$$

When no constraint is active $f_{con,s}$ is set to zero.

5 Applications

In this section, two examples of dynamically constrained objects are shown, namely the simulation of a tool change process and the simulation of the transition of a spring-borne object with offset. We used Dymola 2023x (64-bit) on Windows 10 with a Rkfix4 solver (0.001 s step size) on an Intel® Core™ i7-11700K workstation.

5.1 Simulation of a tool change process

This example demonstrates the capabilities of the developed solution by simulating a tool change process. It consists of two robots, two tools, and four tool holders. The Modelica model is shown in Figure 5. Both tools are attached to ConstrainedObjects. The robot flanges are connected to CollisionObjects in *Control* mode and the tool holders to CollisionObjects in *Passive* mode.

The flange of Robot1 is moved to the holder of Tool1 (green sphere). Then for the CollisionObject attached to the robot the *closed* input switches from false to true. This causes the ConstrainedObject to switch its constraint from the holder to the robot (see section 4.5 for details). Now equipped with Tool1, Robot1 moves to a different holder (red sphere) and releases Tool1 there. Robot2 does the same simultaneously for Tool2 (equipping at the yellow sphere and releasing at the pink sphere). Subsequently,

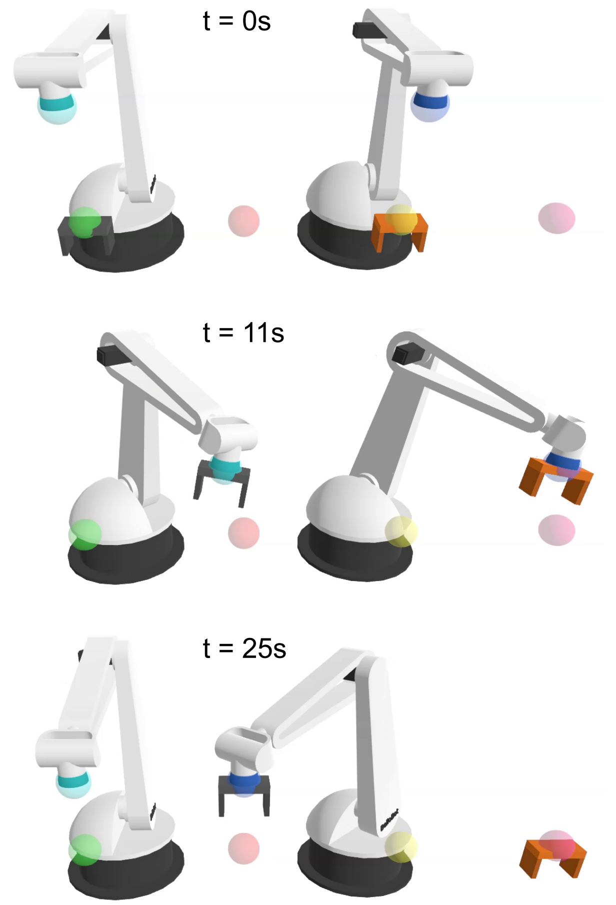


Figure 4. Visualization of the simulation of a tool change process in the DLR Visualization 2 Library. The gray tool is equipped by the left robot and moved to the next holder. The same applies to the right robot and the orange tool ($t = 11s$). In addition, the right robot is equipped with the gray tool ($t = 25s$).

Robot2 is equipped with Tool1 and moved upwards to demonstrate the flexibility of our solution.

The visualization of the final state based on the DLR Visualization 2 Library (Kümper, Hellerer, and Bellmann 2021) is illustrated in Figure 4. Figure 6 shows the results for the constraint forces applied to the ConstrainedObject and to the CollisionObject. The latter represents the resulting forces for the robot when the tool is attached. Simulating the model with 25 s simulation time took 4.5 s.

5.2 Simulation of the transition of a spring-borne object with offset

The second example of dynamically constrained objects is the simulation of a spring-borne object. In the simple scenario, two rigid bodies are connected with a revolute joint. The joint is connected to a spring damper pair. One of the rigid bodies is connected to a ConstrainedObject with enabled transition function (transition duration 0.5s) and offset calculation. The model also contains three different CollisionObjects (all in *Control* mode, i.e. they can be enabled or disabled by the input *closed*).

Figure 7 shows an overview of the scenario. At the

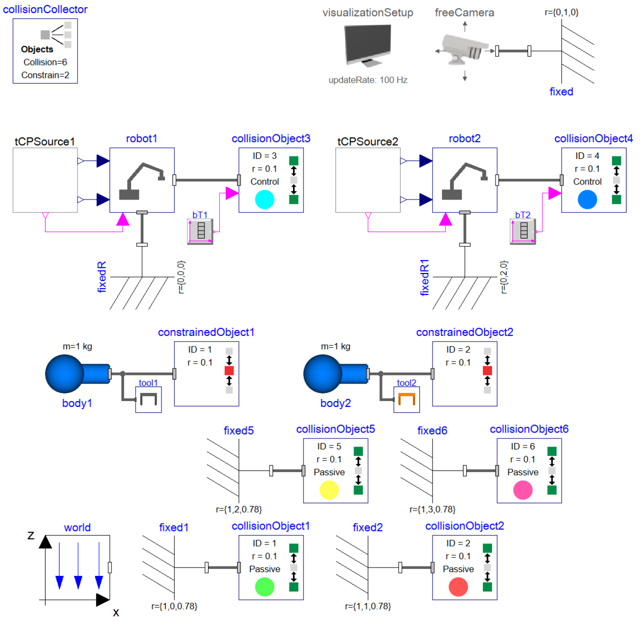


Figure 5. Model for the tool change example. There are two robots equipped with CollisionObjects, two tools attached to ConstrainedObjects, and four tool holders with CollisionObjects. No pre-defined connections are necessary, all components can be added to the model by drag-and-drop.

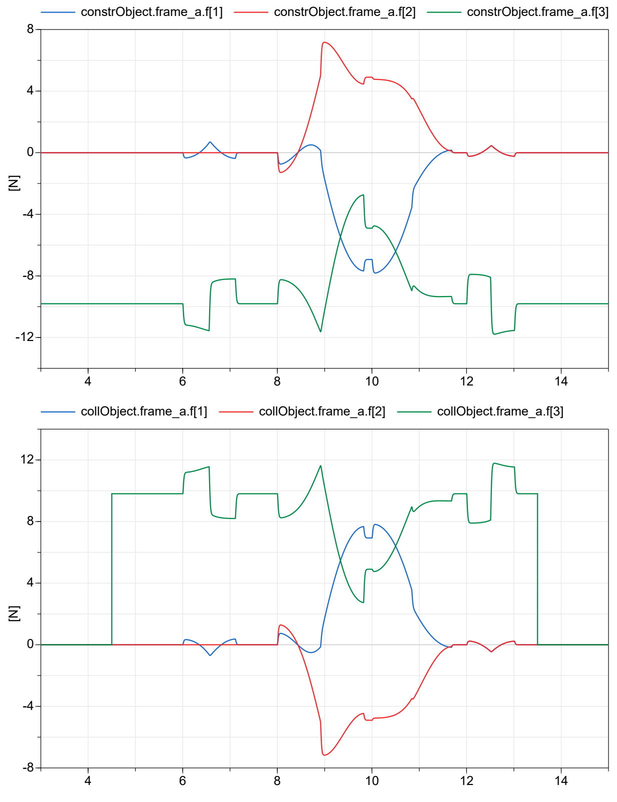


Figure 6. Simulation results for the tool change process. The top shows the forces applied to the ConstrainedObject (connected to the tool) and the bottom shows the forces applied to the CollisionObject (connected to the robot).

DOI
10.3384/ecp204151

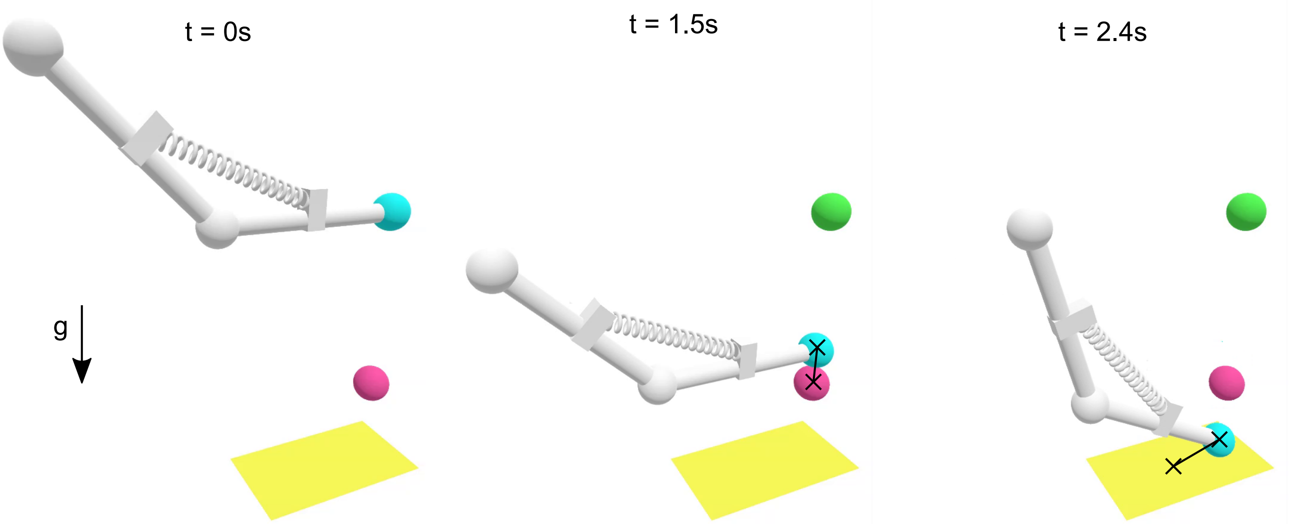


Figure 7. Visualization of the transition of a spring-borne object in the DLR Visualization 2 Library. At first ($t = 0s$), the object is connected to the green sphere. Then it falls and is attached to the pink sphere ($t = 1.5s$). It continues falling and is constrained to the yellow rectangle surface ($t = 2.4s$). The connecting line (black) shows the offset between the origin of both objects.

beginning of the scenario, the ConstrainedObject (blue) is connected to CollisionObject1 (green color). At the time $t = 1s$ the *closed* input of CollisionObject1 is set to false, and the assembly falls down (due to the world gravity in the model) and collides with CollisionObject2 (pink color). At $t = 2.0s$ the input *closed* for this CollisionObject is also set to false, so that the object falls further down until it hits CollisionObject3 (yellow rectangle).

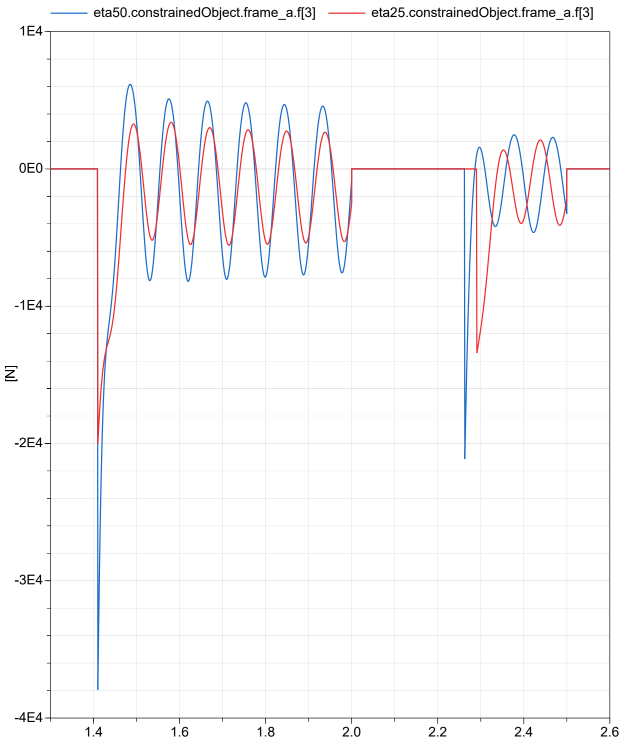


Figure 8. Simulation results for the transition of a spring-borne object with offset. The blue curve shows the constraint force in the z-direction for $\eta = 50$ (damping factor for the Baumgarte stabilization). The red curve shows the result for $\eta = 25$.

The offset from ConstrainedObject (blue sphere) to CollisionObject2 (pink sphere) and to CollisionObject3 (yellow rectangle) is illustrated in Figure 7. Since the use of offsets is enabled in the ConstrainedObject, the sphere is constrained exactly at the contact point. Otherwise, the sphere would be forced to the center of the rectangle (also with the same orientation as the rectangle). At $t = 2.5s$ the input *closed* for CollisionObject3 is also set to false.

The resulting constraint force in the local z-direction can be seen in Figure 8 for two different values of η (see equation 1). The selection of η is unfortunately not straightforward. In principle, it should be set as low as possible and as high as necessary. A high value of η can lead to a numerically stiff system. This can cause problems with numeric integration, especially when fixed-step solvers are used. However, a too-small value for η can result in large deviations between the objects. As shown in Figure 8, the resulting constrained forces can change significantly for different values of η, especially when flexible elements are involved. This can also lead to different behavior in the overall model. The difference for the beginning of the second force spike (for $t > 2.25s$) in the plot results from the higher constrained force (and torque) which affects the flexible element in the system. As such the parameter η is an engineering (control) parameter and has to be chosen problem specific and very carefully. Simulating the model with 3 s simulation time took 0.08 s. It takes 0.44 s to simulate a model with five spring-borne objects and 1.2 s for one with ten objects.

6 Conclusion

In this paper, a new solution for the modeling and simulation of structurally variable systems in Modelica was presented. It combines a collision detection algorithm in native Modelica code, a method to limit the number of possible collisions, the Constraint Force Equation method, and the Baumgarte stabilization. The result is a constraint network within Modelica. It allows the stable and accurate simulation of structurally variable systems in a flexible way (no pre-defined connections are necessary). The ability of the new solution was demonstrated in two examples: the simulation of a tool change process and the simulation of the transition of a spring-borne object with offset. However, the presented approach has some restrictions and limitations: The user has to manually set unique IDs for the objects since it is not (yet) possible within the Modelica language standard and the scalability of the concept is limited. Possible future developments are the support of more geometries for the collision check and external objects to automatically generate unique IDs.

References

Acquatella, Paul and Matthias J. Reiner (2014). “Modelica Stage Separation Dynamics Modeling for End-to-End Launch Vehicle Trajectory Simulations”. In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, pp. 589–598. DOI: 10.3384/ecp14096589.

Baumgarte, J. (1972). “Stabilization of constraints and integrals of motion in dynamical systems”. In: *Computer Methods in Applied Mechanics and Engineering* 1.1, pp. 1–16. ISSN: 0045-7825. DOI: 10.1016/0045-7825(72)90018-7.

Bellmann, Tobias, Andreas Seefried, and Bernhard Thiele (2020). “The DLR Robots library - Using replaceable packages to simulate various serial robots”. In: *Proceedings of Asian Modelica Conference 2020, Tokyo, Japan, October 08-09, 2020*. Ed. by Rui Gao and Yutaka Hirano. Linköping, pp. 153–161. DOI: 10.3384/ecp2020174153.

Bortoff, Scott A. (2020). “Modeling Contact and Collisions for Robotic Assembly Control”. In: *Proceedings of the American Modelica Conference 2020, Boulder, Colorado, USA, March 23-25, 2020*, pp. 54–63. DOI: 10.3384/ecp2016954.

Elmqvist, Hilding et al. (2015). “Generic Modelica Framework for MultiBody Contacts and Discrete Element Method”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*, pp. 427–440. DOI: 10.3384/ecp15118427.

Fiser, Daniel (2018). *libccd: Library for collision detection between two convex shapes*. URL: https://github.com/danfis/libccd (visited on 2023-07-05).

Gilbert, E. G., D. W. Johnson, and S. S. Keerthi (1988). “A fast procedure for computing the distance between complex objects in three-dimensional space”. In: *IEEE Journal on Robotics and Automation* 4.2, pp. 193–203. ISSN: 08824967. DOI: 10.1109/56.2083.

Hellerer, Matthias and Fabian Buse (2017). “Compile-time dynamic and recursive data structures in Modelica”. In: *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools - EOOLT ’17*. Ed. by Dirk Zimmer and Bernhard Bachmann. New York, New York, USA, pp. 81–86. DOI: 10.1145/3158191.3158205.

Hofmann, Andreas et al. (2014). “Simulating Collisions within the Modelica MultiBody library”. In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, pp. 949–957. DOI: 10.3384/ECP14096949.

Kümper, Sebastian, Matthias Hellerer, and Tobias Bellmann (2021). “DLR Visualization 2 Library - Real-Time Graphical Environments for Virtual Commissioning”. In: *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Ed. by Martin Sjölund et al., pp. 197–204. DOI: 10.3384/ecp21181197.

Li, Na et al. (2016). “The Dynamic Simulation of Robotic Tool Changer Based on ADAMS and ANSYS”. In: *2016 International Conference on Cybernetics, Robotics and Control - CRC 2016*. Ed. by Sun Dong, Wei-Hsin Liao, and Sergei Gorlatch, pp. 13–17. DOI: 10.1109/CRC.2016.013.

Modelica Association (2017). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.4: Tech. Rep.* Linköping. URL: https://modelica.org/documents/ModelicaSpec34.pdf.

Oestersötebier, Felix, Peng Wang, and Ansgar Trächtler (2014). “A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces”. In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, pp. 929–937. DOI: 10.3384/ecp14096929.

Otter, Martin, Hilding Elmqvist, and José Díaz López (2005). “Collision handling for the Modelica multibody library”. In: *Proceedings of the 4th International Modelica Conference*.

Paryanto et al. (2014). “Energy Consumption and Dynamic Behavior Analysis of a Six-axis Industrial Robot in an Assembly System”. In: *Procedia CIRP* 23, pp. 131–136. ISSN: 22128271. DOI: 10.1016/j.procir.2014.10.091.

Reiner, Matthias J. (2022). “Simulation of the on-orbit construction of structural variable modular spacecraft by robots”. In: *Proceedings of the American Modelica Conference 2022*, pp. 38–46. DOI: 10.3384/ECP2118638.

Reiser, Robert (2021). “Object Manipulation and Assembly in Modelica”. In: *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Ed. by Martin Sjölund et al., pp. 433–441. DOI: 10.3384/ecp21181433.

Reiser, Robert et al. (2022). “Real-time simulation and virtual commissioning of a modular robot system with OPC UA”. In: *ISR Europe 2022*. Munich: VDE Verlag. ISBN: 978-3-8007-5891-3.

Snethen, Gary (2008). “Xenocollide: Complex collision made simple.” In: *Game programming gems 7*. Ed. by Scott Jacobs. Boston, MA: Charles River Media/Course Technology. ISBN: 9781584505273.

Stüber, Moritz (2017). “Simulating a Variable-structure Model of an Electric Vehicle for Battery Life Estimation Using Modelica/Dymola and Python”. In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, pp. 291–298. DOI: 10.3384/ecp17132291.

Tinnerholm, John, Adrian Pop, and Martin Sjölund (2022). “A Modular, Extensible, and Modelica-Standard-Compliant OpenModelica Compiler Framework in Julia Supporting Structural Variability”. In: *Electronics* 11.11, p. 1772. DOI: 10.3390/electronics11111772.

Toniolo, Matthew et al. (2008). “Constraint Force Equation Methodology for Modeling Multi-Body Stage Separation Dynamics”. In: *Aerospace Sciences Meetings*. DOI: 10.2514/6.2008-219.

Wünsch, Georg (2008). *Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme*. Vol. 215. Forschungsberichte IWB. München: Utz. ISBN: 978-3-8316-0795-2.

A Graph-Based Meta-Data Model for DevOps: Extensions to SSP and SysML2 and a Review on the DCP Standard

Stefan H. Reiterer[1] Clemens Schiffer[1] Mario Schwaiger[1]

[1]Department E, Virtual Vehicle Research,
{stefan.reiterer,clemens.schiffer,mario.schwaiger}@v2c2.at

Abstract

Computer simulation has become a vital tool for modeling complex systems. However, the development and deployment of simulation models often involve multiple stages, tools, and teams, which can lead to significant challenges in maintaining quality, reliability, and efficiency. DevOps, a set of practices that combines software development and IT operations, has emerged as a promising approach to streamline the simulation development. Although most system engineers are not DevOps specialists and there are a lot of manual steps involved when writing build pipelines and configurations of simulations. For this purpose, an abstract graph-based meta-data model was presented in Stefan H. Reiterer, Balci, et al. (2020) to provide an automation framework for DevOps with simulations (see also Stefan H Reiterer, Schiffer, and Benedikt (2022)). In this work we want to continue our investigations by expanding and harmonizing this approach to better work with established standards like SSP, SysML2 and DCP and demonstrating its application on real-life use cases.

Keywords: Continuous Integration, DevOps, MBSE, NoSQL Graph databases, DCP, SysML, UML, SSP

1 Introduction

DevOps is a set of practices, cultural values, and tools that aim to improve collaboration and automation between software development and IT operations teams, with the goal of delivering high-quality software products and services more efficiently and reliably. This approach emphasizes the integration of development, testing, deployment, and monitoring processes to enable faster and more frequent software releases, while maintaining stability and reliability.

Formally Bass, Weber, and Zhu (2015) introduced DevOps as a set of practices intended to reduce the time between committing a change to a system and the change being placed into production while ensuring high quality.

DevOps involves a range of practices, such as continuous integration and continuous delivery (CI/CD), infrastructure as code (IaC), automated testing and monitoring, and often includes agile development methodologies. It also emphasizes the importance of communication, collaboration and shared responsibility between Development and Operations teams and the use of the continuous improvement processes to assess quality and outcomes.

The benefits of DevOps include improved software quality and reliability, faster time-to-market, increased efficiency and productivity, enhanced flexibility and scalability, and better collaboration and communication between teams. It is increasingly being adopted by organizations across a wide range of industries, from startups to large enterprises, as a key enabler of digital transformation and innovation.

With the need to accelerate development cycles in other domains as well like Advanced Driver-Assistance Systems (ADAS) or mechatronics it is crucial to carry over DevOps practices to computer simulations as well. However, there are several difficulties arising when transferring these methods from pure software environments into the world of computer aided engineering (CAE).

The first difficulty arising is that most people working in engineering and scientific fields are not software engineers. This means it is important to democratize DevOps practices with several tools which allow to easily implement, abstract, and reuse the setup of build pipelines to enable better automatic testing. The next problem that arises is the simulation of specific needs, especially when dealing with simulation coupling from different domains. Furthermore, testing and evaluating the simulation quality is much more difficult than regular software applications due to norms and safety requirements which often are physical in nature. Although, there are assessment processes for dealing with those issues (see e.g. the UPSIM project described in **ahmann2022towards**) it is still necessary to seamlessly integrate proper tooling and methodology into the DevOps cycle for simulations.

For that matter a graph-based meta-data model was developed in order to provide a data structure which easily can be stored into modern database systems and is also able to represent dependencies, the topology of co-simulations and is able to be easily mapped from and onto pre-existing standards. In Stefan H Reiterer, Schiffer, and Benedikt (2022) an overview of the topic is given. Additionally, it must be abstract enough to describe a whole range of use cases, but still concrete enough to generate process descriptions out of it to make it accessible for automation.

DOI
10.3384/ecp204159

In this work we will further investigate how to properly leverage the graph-based approach to harmonize it with established standards and will demonstrate its viability on a real-life use case. Furthermore, we will analyze potential shortcomings of the current state of affairs and will discuss potential extensions of the DCP standard to come by with these limitations.

For that matter we first start with a description of the goal we want to achieve with a specific ADAS use case as motivation, following a description of the established standards. We will then continue with a short summary of our graph-based approach and how to create mappings from high level system descriptions (e.g. SysML 2) to more concrete simulation descriptions (e.g. SSP) and how those are rolled out and integrated. In the end we will have a closer look at the DCP standard and what tools and extensions could be helpful in the future to support the proposed workflow.

2 Motivation and ADAS Use Case

In the development of complex systems there is a huge gap between the systems engineering point of view, the practical implementation of software, setup of simulations (Developers) and setting up the of the infrastructure (DevOps) as those require very different sets of skills. Especially, DevOps as the third pillar becomes much harder to perform with raising complexity as the automation pipelines need constant maintenance by experts. Expert knowledge of networking and understanding of containerization and virtualization technologies like Docker or Podman and services like Kubernetes. For that reason, it is important to provide abstraction between those layers to be able to seamlessly transfer the information from one end to the other without being confronted with to much detail. In order to lay out our approach we will describe in this section an example and how its automation will be handled.

2.1 ADAS Use Case Description

In this simple scenario we have 3 roles. The first role is a systems engineer which provides us with a system description and requirements provided in SysML 2.0 (see Figure 2). As the systems engineer does not know all aspects of the simulation the simulation engineer has to set up the simulation from the description and has to come up with a script to compute the desired Key Performance Indicator (KPI). Finally in order to improve on the workflow and raise re-usability the DevOps engineer has to automate the necessary steps to run (and potentially re-run) the simulation under different parameters and provide a pipeline which starts the simulation.

The ADAS use case consists of the following models: A simple vehicle dynamics written in C, a scenario player (EsMini) an FMU with an sensor perception, a simple ADAS function (in this case an FMU with an ACC implemented in C) a transform block for signal mappings. See Figure 1. Furthermore, a post processing script written in Python was used to analyse the driving comfort where we used the methodology described in de Winkel et al. (2023) to compute a suitable key performance indicator.

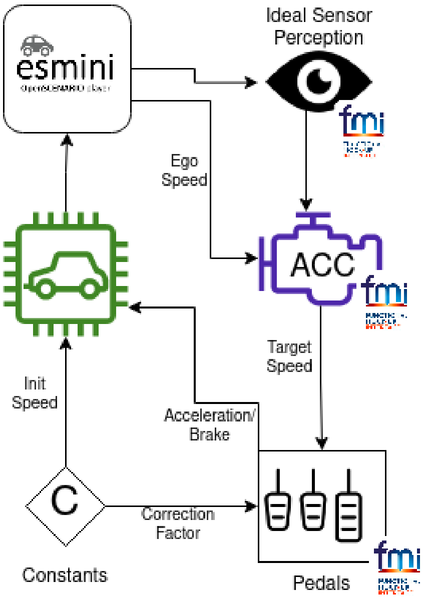


Figure 1. ADAS simulation architecture

2.2 Challenges

However, considering the complexity which arises with variations of the models when changing parameters or adding or removing models of the (co-)simulation this adds a layer of additional complexity to the problem which should not be underestimated. Even small changes to the model can be tedious to apply if they occur often. Hence, it is of utmost importance that these procedures are automated and made traceable as one could easily lose the oversight of the many steps that were taken.

In order to ease the load of the developer it is beneficial that the changes on parameters and architecture could be automatically mapped onto the different models so that the simulation engineers and DevOps engineers can focus on their main tasks. In the next sections we will discuss the standards and concepts which will be used to achieve that goal and how a potential implementation looks like.

3 Established Standards

We use this section to shortly introduce some of the more common standards we want to look at to tackle the arising challenges described in the previous section. We chose these standards due to their open nature and availability, which enables a broad spectrum of use cases. We also will have a short look at the not yet released SysML2.0 standard for system modeling, which is a successor to the widely accepted SysML standard.

3.1 SSP Standard

The Modelica SSP (System Structure and Parameterization) standard described in Hällqvist et al. (2021) is a comprehensive framework for developing cyber-physical sys-

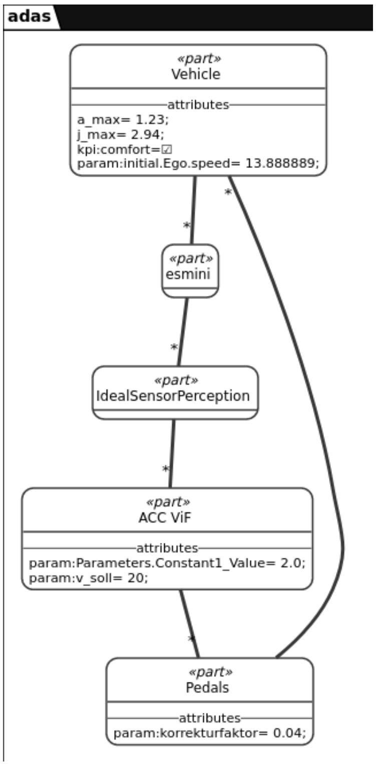


Figure 2. SysML2 model description with requirement

tems that enables the modeling and simulation of complex systems across various domains, including automotive, aerospace, and energy systems. It was designed to be compatible with major Modelica standards like Functional Mock-up Interface (FMI) and is based on the XML standard. The SSP standard provides a systematic approach to structuring and parameterizing system models, which facilitates model exchange and reuse, enhances interoperability, and enables the development of more accurate and efficient simulations.

The SSP standard includes a set of guidelines and conventions for modeling the structural and physical aspects of systems, such as components, connections, and parameterization. The standard also provides a well-defined syntax and semantics for describing the behavior and interactions of system components, which allows for the development of executable models that can be simulated using a range of simulation tools. In addition, it supports the integration of models with other software tools and platforms, such as control systems and optimization tools.

3.2 The SysML Standard 2.0

The widely used System Modeling Language (SysML) is a general-purpose modeling language for developing complex systems and SysML 2.0 is the latest version (which is under current development see OMG (2023)). It is designed to support model-based systems engineering (MBSE) and provides an integrated set of modeling concepts, notation, and semantics that are optimized for the specification, analysis, design, verification, and validation of complex systems. While its predecessor SysML is an extension of the Unified Modeling Language (UML), SysML 2.0 is based on the KerML metamodel.

The main objective of SysML 2.0 is to provide a comprehensive language for MBSE, which can be used throughout the entire system development life cycle. The language provides a standardized way of representing different aspects of the system, including its structure, behavior, requirements, constraints, and interfaces. Furthermore, support for the integration of different domains and perspectives, such as mechanical, electrical, and software, in a single model is provided.

The standard is open and is developed and maintained by the Object Management Group (OMG) with input from a wide range of stakeholders, including industry experts, academics, and users. The language is supported by a growing ecosystem of tools and frameworks, which enable users to create, analyze, and manage SysML models efficiently, e.g. plugins for Eclipse.

The standard can be used by a system engineer to represent the system in concise manner which can be used to generate graphical representations of the system (see Figure 2 for an example). Furthermore, this can be used to describe dependencies between different components of the system and the respective requirements which in return can be leveraged to extract the system architecture and its signal flows like we demonstrate in Section 5.1.

4 The Co-Simulation Process Graph

The Co-Simulation Process Graph concept was originally introduced in Stefan H. Reiterer, Balci, et al. (2020) and is an extension of the classical Process Graph Concept Tick (2007) which allows to not only map process steps (for e.g. a build and deploy pipeline) but also to map the structure of a co-simulation with inputs and outputs and necessary information of the setup steps. In this section we will give a brief introduction of the concept for reader which are unfamiliar with it and also will discuss methodologies to version changes of the Co-Simulation Process Graph which is important for many applications in engineering as traceability is a hard requirement in that sector.

4.1 Definition and an Example

The main problem when trying to map simulation configurations within a process graph is that the moment closed loop simulations are included this introduces cycles within the graph structure. However, this violates the main condition to compute execution orders namely that a process graph is cycle free. In order to solve the problem of cycles introduced by closed loop simulations and models without the need of separating the workflow sequence and the topology of the simulations the Co-Simulation Process is defined with the following properties:

- The set of nodes consists of data nodes, transformation nodes, master nodes, signal nodes and communication (or gateway) nodes.

- To represent the instantiation of a process or the usage of a signal inside a simulation, copies of the nodes which represent these instances are made. Instances must be directly connected to their originals.

- Instead of using the bi-partite structure to represent data transformations, only instances of processes can connect to data nodes to perform operations. In this way, the nodes which perform operations and their instantiation can be determined with a suitable algorithm, which determines a different partition of the graph with help of the defined structure, to provide the correct order of executions. This is necessary since it is allowed that transformation nodes are neighboring, e.g., a Docker container which is built and then used for executing a program.

- An information node can never be the successor or predecessor of another information node. A process must be placed in between. However, neighboring process nodes are allowed. This may happen if a program-performing transformation at a later stage is modified beforehand by another process (e.g., parameterization of tools).

- A simulation is a sub-graph with the following properties: a) It contains the instance of a master node. b) The instance of the master node is connected to all instances of signal nodes that belong to the simulation. c) All the other nodes inside the simulation (i.e., the simulation participants and communication gateways) neighbor a signal instance. d) Each instance of a signal is only allowed to appear once inside a simulation.

- Cycles are only allowed inside a simulation subgraph.

A more detailed description of the data structure and analysis of the used algorithms can be found in Stefan H. Reiterer and Kalab (2021), which was recently accepted in the International Journal of Simulation and Process Modelling. An example is shown in Figure 3. In this example the nodes (of type **Node**) c_1 and c_2 represent software sources (e.g., source code of a model) b represents a build tool like CMAKE and b_1 and b_2 (of type **Bridge**) represent two processes of this build tool which are started, which leads to the simulation units P_1 and P_2 (nodes of type **Bridge**) while the node M represents a simulation master (a node of type **Master/Bridge**). After the build in stage 1) the simulation is executed and the master is configured by the information contained in the node M and gets additional parameters from node I, while the node O represents the output of the simulation. The **Signal** nodes i_j and o_j represent in- and outgoing signals like velocity or acceleration, while **Gateway** node g_j represents the communication protocols (e.g., a network protocol like IP) for $j = 1,2$.

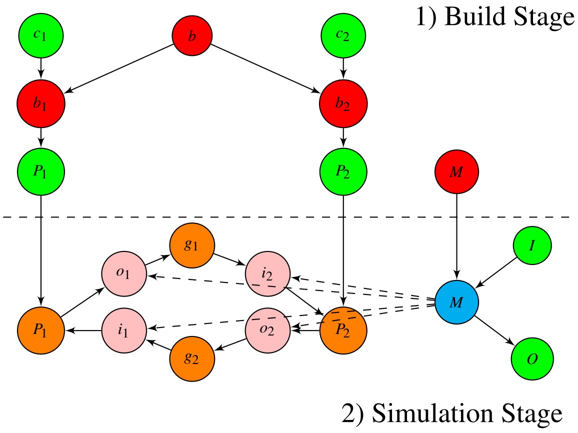


Figure 3. Simple example of a co-simulation process graph

4.2 Versioning Aspects

Graph Databases are to some degree able to use versioning, however this feature is in general not implemented (ArangoDB 2023). In stark contrast to relational databases, where several add-ons exist. Oracle Flashback or Postgres Time Travel are two notable examples. As the structure of a graph differs from the relational model a different approach has to be used. It is also required to take into consideration that not every backup-and-storage technology might be fitting to solve the challenge of versioning. The database used in this work is ArangoDB as it allows handling data more flexibly than its competitors.

When backup is discussed, there are mainly three strategies: Full vs. Incremental vs. Differential. The use case that we consider should cover the following aspects when handling the versioning of graphs which undergo many incremental changes:

- The current/latest graph has to be loaded the fastest as it is used in production.

- Fast loading of the previous versions which were recently added.

- Storage-benefits over full-backup as a lot of redundant data is created.

This is motivated by the fact that a model will undergo a lot of small incremental changes during the development process and hence will create a lot of redundant data. With regards to the solution is an inverse differential storage which has the latest version of the graph saved in an unmodified state. The previous versions only save the data changed over each iteration. For the sake of simplicity, currently this is done by using a signal-character which marks the unchanged data. One of the benefits of this model is the hybrid approach that still allows to implement a partial-full model to jump back to any given fully backed up version.

Almost any given record can be persisted in a collection of the database. After adding another collection of a different version of the data the versioning starts. All the fields are checked whether or not they are equal to the previous version. In case of unchanged data, the previous records will be overwritten by the signal character. The differing fields remain unchanged. By doing this version 1 and version 2 are obtained. On any other given data set the same algorithm is applied to create the next iteration.

In order to return back to a previous version either a full-backup-milestone or the most recent version are used to start from. All the signal-characters are reverted to the latest state. A prototype is currently under development in the scope of a bachelor thesis at Virtual Vehicle Research. First tests with a simple co-simulation process graph which underwent some changes over time already showed roughly a memory saving between 20-30% in contrast to full back ups of the different versions. See Table 1 for an overview of different cases and the saves for the file sizes. However, there is a lot of potential for optimization which will be explored further in the future.

Table 1. Difference in file sizes (FS) given in bytes

Records	FS Orig.	FS Compressed	Saving [%]
10	7094	5348	24.6
50	35156	24942	29.1
100	73455	49377	32.8
123	52964	43360	18.1
125	53822	44010	18.2
250	108036	88184	18.4
500	215884	176470	18.3
750	397946	315564	20.7
1000	668678	425724	36.3
2500	1394335	981182	29.6
5000	4168767	2938219	29.5
10000	15442067	12299693	20.4

5 Mappings between the Graph and the Standards

In this section we will discuss shortly the mappings between the Co-Simulation Process Graph and the SysML 2.0 and SSP standard and their connection to the graph database.

5.1 Mappings between the Graph Database and the SysML 2.0 Standard

Since the SysML 2.0 standard is not finalized yet tools are not completely available yet. Since the new standard is based on the KernML meta language and not XML like its predecessor it was necessary to write a simple Python parser which parses the SysML file. We used the freely available Pyparsing module for this task. As example we use the simple SysML 2.0 model described in Listing 1 which refers to the model depicted in Figure 2 and is linked to the use case we described in Section 2.1. To modularize the SysML 2.0 document it is hierarchically stored into the graph database (where the hierarchy is with respect to nesting of the KernML blocks) with dependencies within the hierarchy stored as edges. See Figure 4 for the structure in the database.

Listing 1. "SysML 2.0. model"

```
package adas
{
  part Vehicle {
    attribute 'param:initial.Ego.speed'
        =13.888889;
    attribute 'kpi:comfort';
    attribute 'j_max'=2.94;
    attribute 'a_max'=1.23;
  }
  part esmini {
  }
  part IdealSensorPerception{
  }
  part 'ACC ViF' {
    attribute 'param:Parameters.
        Constant1_Value'=2.0;
    attribute 'param:v_soll'=20;
  }
  part Pedals {
    attribute 'param:korrekturfaktor'=0.04;
  }
  connect Vehicle to esmini;
  connect Pedals to Vehicle;
  connect esmini to IdealSensorPerception;
  connect IdealSensorPerception to 'ACC ViF
      ';
  connect 'ACC ViF' to Pedals;
}
```

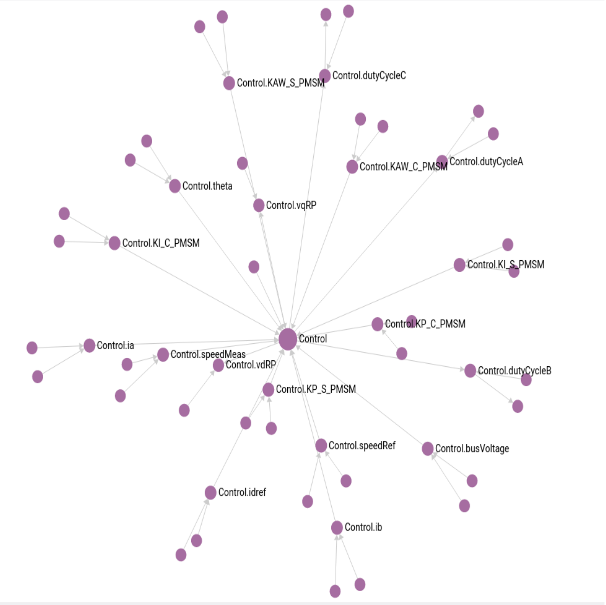


Figure 4. SysML2.0 model stored within the graph database

With help of this representation, we are able to extract building blocks and modules. We are also able to easily search and extract information by recursive search methods. We can use this to inject or update information of the co-simulation process graph which is used for setting up, configuration and start of the test simulation. The co-simulation process graph for this model can be seen in Figure 5. It should be noted that in the graph the simulation with all its sub-nodes (participants, signals, and communication) is contracted within the "Simulation" node for the sake of simplicity. With help of the Arango Query Language (AQL) it is then possible to locate possible parameter changes within the SysML 2.0 model and map it onto the graph. After the computation is finished and the post-processing tools evaluated KPIs and performed quality checks the information can then stored back into the graph database. With an inverse mapping it can even be written onto the abstract SysML 2.0 model as information for the system developer. While the transfer of parameter changes in this example is rather trivial, we already demonstrated in Stefan H Reiterer, Schiffer, and Benedikt (2022) that also changes of participants, simulation settings or even the topology of the co-simulation is easily possible. We will also discuss in the next section how this can directly applied to SSP files.

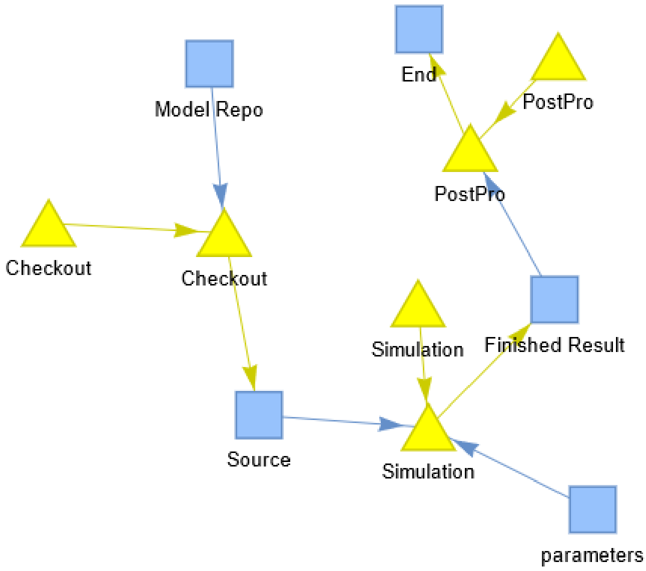


Figure 5. Co-Simulation process graph of the simulation model

5.2 Mappings between the Graph Database and the SSP 2.0 Standard

Since the SSP standard is designed to represent a co-simulation graph its mapping into the co-simulation process graph and the graph database is rather straight forward. We use the following transformation:

- The System itself and its parameters can be represented by the **Master** node of the co-simulation process graph, i.e., the **SystemStructureDescription** and its meta-data can be directly written into the (JSON-)dictionary representing the master node. Information like `ssd:DefaultExperiment` with start and stopping time directly go there.
- **Components** and their meta-data are directly mapped onto **Bridges** representing the simulation participants.
- The **Connectors** of the **Components** and their respective meta-data are mapped onto **Signal** nodes. The **kind** parameter which denotes if it is an input or output connector is indirectly mapped by the direction of the edge of between the participant and the signal node.
- Moreover, the metadata of **Connections** is directly mapped onto (communication) **Gateway** nodes, while the direction of the connection (**startConnector** to **endConnector**) is represented by the edges between the signals and the gateway connections. It should be noted that the keys **startElement** and **endElement** of the **Connections** is implicitly provided by the connectivity of the component, signal and gateway nodes and thus has not to be explicitly stored.
- Last but not least, necessary edges can be added afterwards as well,

With these mapping rules the inverse mapping is also rather clear. It may be necessary to use a tool for the geometry information if the graph comes from a different source than an SSP file, but such tools are vastly available. The mappings could be done either directly in XST or any programming language like Python. In Figure 6 we see the graph database view of the transformed SSP file representing the simulation architecture in Figure 1. Note that a lot of signals were filtered out in the view for better clarity in the representation as the simulation has a lot of signals which were not actively used during the simulation. Furthermore, it should be noted that the process graph is the master model in this scenario which collects the key information that is necessary to let the simulation run. It only has to deal with a portion of the SysML 2.0 description as the SysML 2.0 model may contain information which are not relevant for the simulation run or the DevOps processes (e.g. business relevant information), although, the graph based model is flexible enough to store information outside the scope of automation as well. On the other hand the graph model stores information about the setup of the simulation, hence, it contains more information than a regular SSP file. This means it may be necessary to enrich an extracted graph from an SSP file with additional information, except the simulation only consists of FMUs and the FMU master is predefined.

6 The Role of the DCP Standard in the Workflow and How to Enable Broader Adoption

The Distributed Co-simulation Protocol (DCP) is a Modelica standard (Modelica Association Project DCP 2019)

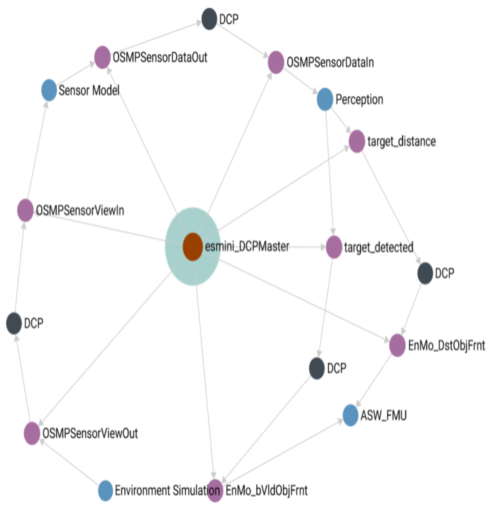


Figure 6. Graph database view

for real-time and non-real time system integration and simulation. It aims to augment the Modelica eco-system of FMI and SSP by adding distributed inter-operable simulation units, thus enabling the simulation of cyber-physical systems (CPS). This standardization is achieved by defining a common state machine and configuration together with the use of a transport protocol, either UDP, TCP are implemented in the reference implementation, while Bluetooth and CAN are specified but not yet implemented. The capabilities of a simulation unit (a DCP slave) are described in an XML-document known as a slave-description, which is intended to be shared with a DCP master before the simulation. The DCP master can then integrate a simulation scenario by configuring the simulation units according to their capabilities. The exchange of simulation data itself is achieved by protocol data units (PDUs) that are sent in a defined manner according to the transport protocol used. It aims to close the gap between software in the loop (SIL) and hardware in the loop (HIL) as DCP allows for a drop-in replacement of each component. The industry need for distributed simulation and standardized interfaces is in part covered by OPC UA (Schwarz and Börcsök 2013), ROS2 (Macenski et al. 2022) and similar technologies, however, only DCP specifically focuses on co-simulation. Recent use cases of DCP include (Rautenberg et al. 2023) which provides valuable input for possible extensions and use of DCP on coupled hardware test benches, while Segura, Poggi, and Barcena (2023) describe a generic interface using DCP in Simulink. Generally, in many applications where a distributed simulation is needed, it is implemented by either using proprietary technology or established standards with a different focus. Having a simplified – yet standard compliant – version of DCP available, such that developers only need to implement a minimal set of features, e.g., sending data via a TCP/IP port, would help in establishing DCP as a widely adopted protocol.

We identified DCP as a core technology for the proposed workflow as DCP provides us with a proper co-simulation standard for which configurations can be auto generated. While FMI has become the de-facto standard for the integration of co-simulation units and SSP for the description of systems of FMUs, there remain some issues: Either code for FMUs is generated resulting in a static artifact or the FMU has dependencies – such as installed programs, libraries or licenses. In the future we expect the development of the concept of on-line simulation platforms that enables the direct coupling of models using DCP without sharing the underlying model, as is partly discussed in Ahmann et al. (2022). The model needs to be available for a standardized distributed system simulation with a description available beforehand and the ability to be started remotely, which is essential for easy deployment on a big pool of workers either in the cloud or on premise.

However, during our work the high complexity of the DCP standard became more visible. While it is desirable that the standard covers a lot of use cases the vast range of options can become a hindrance when we want to provide basic tooling. Thus, we propose the idea of a reduced DCP core standard, which is able to cover most use cases, but enables the creation of easy-to-use tooling based on this minimal viable set of rules to accelerate the distribution of DCP. While we already proposed the use of an FMI to DCP wrapper to leverage the broad availability of FMI in our last work, we observed several times that packing FMUs can confront developers with several challenges to pack third party tools like open-source driving simulations such as esMini or Carla. A minimal standard could help in developing simple deployable DCP nodes but also a simplified master which covers a lot of use cases.

Ideally, the build and dependency of the model should be made explicit to allow for traceability as well as the ability to trigger a build to use the most recent version. This also would benefit the proposed workflow. The system description needs to be formulated in a standardized way, that allows for the description of the system architecture as well as the integration of the co-simulation system.

7 Summary

We have extended the methodology outlined in Stefan H Reiterer, Schiffer, and Benedikt (2022) how to make use of graph-based automation using dynamically generated build pipelines for co-Simulations by making use of mappings between standards for system description (SysML2.0) and system structure description (SSP) which can be used to configure a co-simulation master with a more practical example. Additionally, we demonstrated how to decompose SysML2.0 and SSP descriptions to properly store them into graph databases and how to map them properly into co-simulation process graphs to enable a more seamless workflow and proposed a method for graph versioning tailored for development workflows. Furthermore, we identified some shortcomings of the current state DCP standard and proposed a potential solution in the form of a DCP core standard to address these issues.

8 Outlook

While the proposed graph-based methodology already addresses several issues like making standardized formats available on graph databases and some methods for versioning them were discussed, there is still a lot of potential to improve on the existing algorithms and how to better organize the pool of data which is created. Further, we have to explore the potential of data driven testing and validating the running simulations with help of the generated data over time to foster a more automated continuous improvement process over longer development periods.

Furthermore, the proposal of a core DCP standard for easier tooling has to be explored and properly formulated and activities regarding discussions with partners from academia and industry have to be initiated.

Acknowledgements

This publication was written at Virtual Vehicle Research GmbH in Graz, Austria. The authors would like to acknowledge the financial support within the COMET K2 Competence Centers for Excellent Technologies from the Austrian Federal Ministry for Climate Action (BMK), the Austrian Federal Ministry for Labour and Economy (BMAW), the Province of Styria (Dept. 12) and the Styrian Business Promotion Agency (SFG). The Austrian Research Promotion Agency (FFG) has been authorized for the program management. They would furthermore like to express their thanks to Prof. Eugen Brenner and Georg Macher from the Institute of Industrial Informatics at TU Graz for their support.

References

Ahmann, Maurizio et al. (2022-11). “Towards Continuous Simulation Credibility Assessment”. In: *Modelica Conferences*, pp. 171–182. DOI: 10.3384/ecp193171.

ArangoDB (2023). *Data Modeling and Operational Factors*. URL: https://www.arangodb.com/docs/stable/data-modeling-operational-factors.html (visited on 2023-05-11).

Bass, Len, Ingo Weber, and Liming Zhu (2015). *DevOps: A software architect’s perspective*. Addison-Wesley Professional.

de Winkel, Ksander N. et al. (2023). “Standards for passenger comfort in automated vehicles: Acceleration and jerk”. In: *Applied Ergonomics* 106. DOI: 10.1016/j.apergo.2022.103881.

Hällqvist, Robert et al. (2021). “Engineering domain interoperability using the system structure and parameterization (SSP) standard”. In: *Modelica Conferences*, pp. 37–48. DOI: 10.3384/ecp2118137.

Macenski, Steven et al. (2022). “Robot Operating System 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66. DOI: 10.1126/scirobotics.abm6074.

Modelica Association Project DCP (2019). *DCP Specification Document, Version 1.0*. Linköping, Sweden: Modelica Association. URL: http://www.dcp-standard.org.

OMG (2023). *OMG Systems Modeling Language™ (SysML®) v2 Release*. https://github.com/Systems-Modeling/SysML-v2-Release. Accessed: 2023-05-13.

Rautenberg, Philip et al. (2023). “Electrified Powertrain Development: Distributed Co-Simulation Protocol Extension for Coupled Test Bench Operations”. In: *Applied Sciences* 13.4. ISSN: 2076-3417. DOI: 10.3390/app13042657.

Reiterer, Stefan H, Clemens Schiffer, and Martin Benedikt (2022). “A Graph-Based Metadata Model for DevOps in Simulation-Driven Development and Generation of DCP Configurations”. In: *Electronics* 11.20. DOI: 10.3390/electronics11203325.

Reiterer, Stefan H., Sinan Balci, et al. (2020). “Continuous Integration for Vehicle Simulations”. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. IEEE, pp. 1023–1026.

Reiterer, Stefan H. and Michael Kalab (2021). “Modelling deployment pipelines for co-simulations with graph-based metadata”. In: *International Journal of Simulation and Process Modelling* 16.4, pp. 333–342. DOI: 10.1504/IJSPM.2021.118852.

Schwarz, M. H. and J. Börcsök (2013-10). “A survey on OPC and OPC-UA: About the standard, developments and investigations”. In: *2013 XXIV International Conference on Information, Communication and Automation Technologies (ICAT)*, pp. 1–6. DOI: 10.1109/ICAT.2013.6684065.

Segura, Mikel, Tomaso Poggi, and Rafael Barcena (2023). “A Generic Interface for x-in-the-Loop Simulations Based on Distributed Co-Simulation Protocol”. In: *IEEE Access* 11, pp. 5578–5595. DOI: 10.1109/ACCESS.2023.3237075.

Tick, József (2007). “P-graph-based workflow modelling”. In: *Acta Polytechnica Hungarica* 4.1, pp. 75–88.

Introducing Dialectic Mechanics

Dirk Zimmer   Carsten Oldemeyer

Institute of System Dynamics and Control,
German Aerospace Center (DLR),
{dirk.zimmer, carsten.oldemeyer}@dlr.de

Abstract

This paper introduces a new method for mechanical systems with its own interface that enables the object-oriented formulation of very stiff contacts. It thereby suppresses high frequencies and yields stable replacement dynamics leading to an equivalent steady-state. Potential applications are the efficient modeling and simulation of robotic manipulation or the easier handling of what formerly have been variable-structure systems.

Keywords: multi-body systems, Mechanical contacts and limitations, Robotics

1 Motivation

Libraries for classic multibody simulation have been among the first Modelica libraries ever published. The Modelica Standard Library supports the 3D solution of multibody systems (Otter 2003) with special support for kinematic loops. There are also 1D rotational and translational libraries and a planar mechanical library has been developed that proved its value for teaching purposes (Zimmer 2012) and advanced modeling of gearwheels (van der Linden 2016).

Yet there are modeling tasks that have remained very difficult to master throughout all the years such as:

- The modeling of limited joints
- The modeling of breaking objects
- The modeling of stiction and friction
- The modeling of kinematic loops when reaching maximal extension
- Real-time simulation of hard contacts
- etc.

Our impression is that at least for the Modelica community, progress in these areas has been underwhelming, especially given the high relevance of these issues. For instance, when modeling the manipulation of an object using a robot hand on a robot arm, a combination of any of the above problem may occur.

Many attempts in solving this problem were focused on improving the tooling. Tasks like the modeling of limited joints were identified as variable structure problems or Multi-mode DAEs (Benveniste 2019) and tackled correspondingly by new tools (Mehlhase 2013) or even new languages (Zimmer 2010, Neumayr 2023). The underlying model equations were practically never questioned. After all, classic Newton mechanics is more than two centuries old (Szabo 1987), and seems hardly worth revisiting.

Au contraire, mon capitan! It is worth revisiting the way we idealize mechanical systems. After all, object-oriented modeling and computers are much younger. We may be able to find a reformulation that enables a better expression of modeler's intent than what was previously conceived. This is the exact aspiration of this paper.

2 On the Idealization of Rigid Body Mechanics

We easily forget that when we model the mechanics of rigid bodies, we model the mechanics of imaginary objects: rigid bodies.

In our real, physical world, there exist no rigid bodies. Everything is elastic and deformable. It is just a matter of degree. If a bullet out of a gun will not convince you, certainly a small piece of space debris as in Figure 1 will:

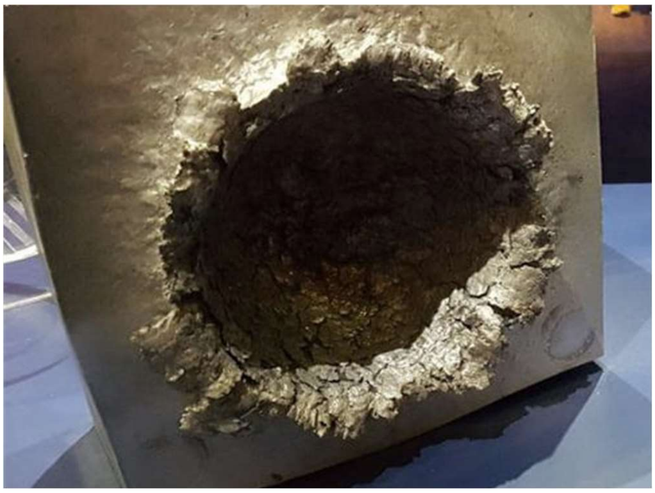


Figure 1: Impact of a 15g piece of plastic on a block of aluminum with a speed of 24140km/h in public display at NASA Johnson Space Center, Houston, TX, USA.

Rigid bodies thus represent an entirely hypothetical idea, but also a very useful idea. Instead of modeling the pressure waves through an elastic material we can directly formulate non-holonomic constraints and assume an immediate transmission of impulse that upholds the conservation of energy and momentum, since none of these terms can get dissipated in a truly rigid body. We thereby exchange a process that typically operates above 10 kHz (micro-elastic motion within objects) with a process that may often be slower than 10 Hz (macro-motion of objects). Evidently this enables a much more efficient simulation of the kinematic system using far fewer states and much slower eigen-dynamics.

Rigid body mechanics is thus the preferred method to use when we deal with kinematic chains with a fixed number of degrees of freedom. Phenomena as limited joints or stiction can consequently be interpreted as varying the number of degrees of freedom. When regarding such problems as discrete configuration changes, this leads straight to the previously mentioned approaches (Zimmer 2010, Mehlhase 2013, Benveniste 2019, Neumayr 2023) for variable structure systems. Also discrete Dirac impulses then need to be considered as in (Zimmer 2006).

However, even if a (potentially very complex) solution for discrete configuration changes is available, it is often inappropriate to apply since it forces us to simplify by discretization the very thing we actually want to focus on. Whether a gripping mechanism is actually holding an object or not and when and up to what degree is a question that is not easily answered by yes or no. When going into detail, one may detect many transient states.

For such cases, the modeler is now forced to re-establish elastic bodies at least for the region of contact dynamics. Whereas he may succeed, to keep the set of state variables small, applying realistic constants for the elasticity will often yield high frequency behavior or other ill-suited eigen-dynamics that drastically lower the simulation efficiency. This is especially true when a stiff object is tightly gripped, and notably it is the very intent of gripping devices to grip things tightly in order to create a force-locked connection.

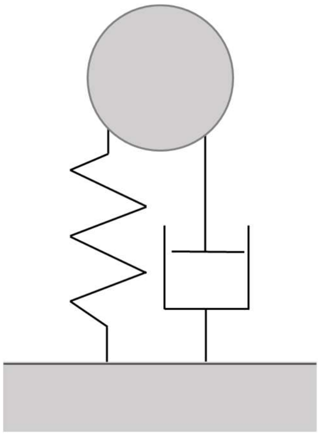


Figure 2: A one-dimensional spring-damper system modeling an elastic contact with ground.

For illustration, let us look at the simple 1D mechanics of a spring-damper system as in Figure 2.

$$v = \frac{ds}{dt} \tag{1a}$$

$$\frac{dv}{dt} = \frac{f}{m} + g \tag{1b}$$

$$f = -cs - dv \tag{1c}$$

where s is the position and v is velocity. The force f results out of the spring damper dynamics with their respective coefficients c and d . g is the gravity acceleration.

For $d > 0$ and $m > 0$, this system reaches a steady-state solution at:

$$s = \frac{mg}{c}; v = 0$$

The eigenvalues of the system are well known:

$$\lambda_{1,2} = -\frac{d}{2m} \pm \sqrt{\frac{d^2}{4m} - \frac{c}{m}}$$

Let us suppose, we as modelers are willing to sacrifice the precision of the transient dynamics for the sake of simulation efficiency. Since both m and c contribute to the steady-state solution, we may hence only modify the damping constant d.

Below critical damping we may move the eigenvalues only alongside a circle in the plane of imaginary numbers. This helps at least avoiding high frequencies and is often feasible for implicit ODE solvers. Going beyond critical damping makes matters even worse, causing one eigenvalue to become highly negative whereas the other starts to interfere with potentially other slow dynamics that may exist in extension of this system. The direct manipulation of d in a complex system is often cumbersome because a favorable choice depends on the values for spring constants and masses for the configuration.

Despite its tight limitations, this method is often applied and for real-time simulation, many simulation practitioners are desperate enough to even manipulate constants for masses or springs (Neves 2019, Reiser 2021), often leading to a virtual world of strangely wobbling objects.

3 The Idea of Dialectic Mechanics

When practitioners show such signs of desperation, it is mostly because their model does not match their original intent.

Indeed, it is not very intuitive for us why the gripping of an object is such a tough task to simulate, our brain simulates it all the time and it seems to do a pretty good job at it despite being a low-frequency computational device (albeit being massively parallel). We thereby intuitively decompose the macroscopic motion of our arm, hand, and object from the microscopic motion of the object in the tension-regime of the gripping hand. The first motion is dominated by the kinetic forces resulting from the acceleration of objects, the latter motion is dominated by the elastic forces resulting from the positional shift of the object.

Realizing such a decomposition in form of equations is unfortunately not intuitive at all but it can be achieved:

- We denote the velocity in the elastic regime: v_{el}
- We denote the velocity in the kinetic regime: v_{ki}

In an ideal world $v_{el} = v_{ki}$. However, to express the modeler's intent of splitting into two regimes, we formulate:

$$\frac{dv_{ki}}{dt} T_D = v_{el} - v_{ki} \tag{2a}$$

with T_D being denoted as dialectic time-constant. This represents a first-order filter for the kinetic motion. High-frequency motion in the elastic regime are therefore inhibited for their impact on the kinetic regime. Let us now restate the equations of our spring damper system:

We can compute the elastic force f_{el}:

$$f_{el} = -cs + mg \tag{2b}$$

With

$$\frac{ds}{dt} = v_{el} \tag{2c}$$

We can compute the kinetic force f_{ki}:

$$f_{ki} = -m\frac{dv_{ki}}{dt} - d \cdot v_{ki} \tag{2d}$$

Evidently, the decomposition of velocity led us to decompose also the forces and we now treat elastics and kinetics as separate phenomena. In order to rejoin them to a consistent solution, we remember our equation (1) of the first-order filter and enforce the balance of forces:

$$f_{el} + f_{ki} = 0 \tag{2e}$$

This is why we call this approach: dialectic mechanics. If we personify the phenomena of elastics and kinetics then both persons would argue for their regime by expressing their respective force. In the end, they have to reach a common conclusion that neutralizes their respective counterarguments.

In correspondence, this system of equations has two states: the position s belonging to the elastic domain and v_{ki}, belonging to the kinetic regime. We can plug in Equation (2b) and (2d) in Equation (2e) to eliminate the forces:

$$\frac{dv_{ki}}{dt} = g - \frac{c}{m}s - \frac{d}{m}v_{ki} \tag{3a}$$

and plugging in Equation (2a) in (2c) eliminates v_{el}:

$$\frac{ds}{dt} = gT_C - \frac{cT_D}{m}s + \left(1 - \frac{dT_D}{m}\right)v_{ki} \tag{3b}$$

We see that for $T_D \to 0$ this system becomes equivalent to the original system of Equations (1a-1c). Small values for T_D shall thus result in a small deviation. We also see that T_D has no impact on the steady-state solution, which is still:

$$s = \frac{mg}{C}; v_{ki} = 0$$

But the eigen-dynamics are now manipulated so that we have new eigenvalues:

$$\lambda_{1,2} = -\frac{d + cT_D}{2m} \pm \sqrt{\frac{(d + cT_D)^2}{4m^2} - \frac{c}{m}} \tag{4}$$

The term in the square root is now a quadratic function on c/m with a minimum at:

$$\left(\frac{c}{m}\right)_{min} = \frac{2}{T_D^2} - \frac{d}{mT_D}$$

and the minimum value of:

$$-\frac{1}{T_D^2} + \frac{d}{mT_D}$$

For an undamped system with $d = 0$, this simplifies to:

$$-\frac{1}{T_D^2}$$

which limits the imaginary part of the eigenvalues to not exceed $\pm iT_C^{-1}$, corresponding to a maximum rotation of

$$\omega_{max} = T_D^{-1}$$

or a frequency limitation of $\frac{1}{2\pi T_D}$. In the original undamped system, the angular velocity was simply:

$$\omega_S^2 = \frac{c}{m}$$

The dialectic undamped system yields a different rotation:

$$\omega_D^2 = \frac{c}{m} - \left(\frac{c}{m}\frac{T_D}{2}\right)^2$$

which (for $\omega_D > 0$) can be expressed in terms of ω_S:

$$\omega_D^2 = \omega_S^2 \left(1 - \omega_S^2 \frac{T_D^2}{4}\right)$$

We see that the deviation from the original system is small for low frequencies but keeps rising quadratically up to and beyond the frequency limitation. From equation (4) we can also see that there is an additional damping term added with the strength of $\omega_S^2 T_D/2$.

In terms of eigenvalue manipulation: what is subtracted on the imaginary axes is added on the left side of the real axis (for an undamped system). This means that our error is of stabilizing (or dissipative) nature. Indeed, we can see from Equation (4) that working with T_D is equivalent to manipulating the damping constant. The time-constant however offers a systematic approach to perform this: eigenvalues near the center are only little influenced, eigenvalues close to the frequency limitations are drastically manipulated. Also, we still have the original damping coefficient d available for further manipulation of the eigenvalues in case this is needed.

If the slow dynamics of interest is well below the imposed frequency limitation, we can expect our error to be within an acceptable range for many practical applications, especially those applications where the model uncertainty is quite high like gripping little known objects. We shall also remember that the steady-state solution is not manipulated.

4 Object-Oriented Formulation

4.1 1D Translational Systems

All what has been discussed in the previous section has just been the eigenvalue manipulation of a small system with two states. This would not deserve our attention, if the conclusion remains restricted to this problem class. Fortunately, the idea of dialectic mechanics is very well suited for an object-oriented formulation, which allows its application to larger and more complex kinetic constructs.

To this end, let us review the core idea and devise a 1D library for translational mechanics. The first key idea was to split the mechanics into two regimes:

- The elastic regime, taking care about position and storage of potential energy such as springs or gravity.
- The kinetic regime, taking care about dissipation and storage of kinetic energy

We can represent these two regimes, by two corresponding pairs of effort and flow:

Listing 1. 1D-connector implementation

```
connector Flange
  SI.Position s;
  flow SI.Force f_el;

  SI.Velocity v;
  flow SI.Force f_ki;
end Flange;
```

We also define that $v_{el} := \frac{ds}{dt}$ and if not stated explicitly otherwise $v_{ki} \coloneqq v$ and the acceleration is $a = dv_{ki}/dt$. When we implement the components, we simply do so in a dialectic manner. We set up the equations for each of the regimes independently.

The fixation is boring as usual:

Listing 2. Component for a fixed position

```
model Fixed
  Interfaces.Flange_b flange_b;
  parameter SI.Position s;

equation
  flange_b.s = s;
  flange_b.v = 0;
end Fixed;
```

The element for translation now has to contain the derivative of the non-holonomic constraint in the kinetic domain. Kinetic and elastic forces are independently transferred.

Here is the implementation of a body component:

Listing 3. Component representing a 1D mass

```
model Body
  Interfaces.Flange_a flange_a;

  parameter SI.Mass m;
  parameter SI.Acceleration g = -9.81;

  SI.Acceleration a;
  SI.Velocity v(stateSelect= ...avoid);
  SI.Position s(stateSelect= ...avoid);

equation
  a = der(v);
  s = flange_a.s;
  v = flange_a.v;

  flange_a.f_ki  = m*a;
  flange_a.f_el  = -m*g;
end Body;
```

Please note that the gravity is attributed to the elastic domain since it represents a potential force depending on position (albeit not in this particular example). Also, the body component does not state that the velocity is derivative of the position. Other than a typical body component, it does not define states.

To finally join the two regimes and reach a common conclusion, we have to apply the filter equation that relates v_{el} and v_{ki} and also enforce the balance of forces: $f_{el} + f_{ki} = 0$. This has to happen where we define our degrees of freedom for the motion of the system. These are the joint elements. In 1D mechanics there is only 1 degree of freedom and hence only one type of joint: the prismatic joint.

Listing 4. The prismatic joint in 1D

```
model Joint
  Interfaces.Flange_a flange_a;
  Interfaces.Flange_b flange_b;
  RealInput f_ext;
  parameter SI.Time TD;
  SI.Position s(stateSelect = …prefer);
  SI.Velocity v(stateSelect = …prefer);
  SI.Velocity v_el(start = 0);

equation
  flange_a.s + s = flange_b.s;
  flange_a.f_el + flange_b.f_el = 0;
  flange_a.v + v = flange_b.v;
  flange_a.f_kin + flange_b.f_kin = 0;
  flange_a.f_el + flange_a.f_kin =f_ext;
  v_el = der(s);
  der(v)*TD = (v_el - v);
end Joint;
```

In dialectic mechanics, typically s and v_{ki} are chosen as states of the system. A linear system has then to be solved, in order to solve for v_{el} with the balance of forces $f_{el} + f_{ki}$ forming the corresponding residual. In this particular component model, this sum adds up not to zero but to an external force f_{ext} that can be used to actuate the joint.

Following the same spirit, we can model an asymmetric spring-damper to model a mechanical stop element.

Listing 5. ElastoGap model

```
model ElastoGap
  Interfaces.Flange_a flange_a;
  Interfaces.Flange_b flange_b;
  parameter SI.Position l;
  parameter SI…SpringConst. c;
  parameter SI…DampingConstant d;
  SI.Position ds( start = 0);
  SI.Velocity dv( start = 0);

equation
  flange_a.s + l + ds = flange_b.s;
  flange_a.f_el + flange_b.f_el = 0;
  flange_b.f_el = if ds < 0 then ds*c
                  else 0;
  flange_a.v + dv = flange_b.v;
  flange_a.f_kin + flange_b.f_kin = 0;
  flange_b.f_kin = if ds < 0 then dv*d
                   else 0;
end ElastoGap;
```

The following model uses two of such elasto-gaps to model a 500g ball clamped into two pieces of hard wood with an indentation of 0.1mm resulting in a spring constant of roughly 2MN/m. The system is modelled without any damping (which is totally unrealistic). The whole construction is then moved by two subsequent and counteracting force impulses. The corresponding setup of Figure 3 can be regarded as a very simplistic model of a robotic grip holding and moving an object.

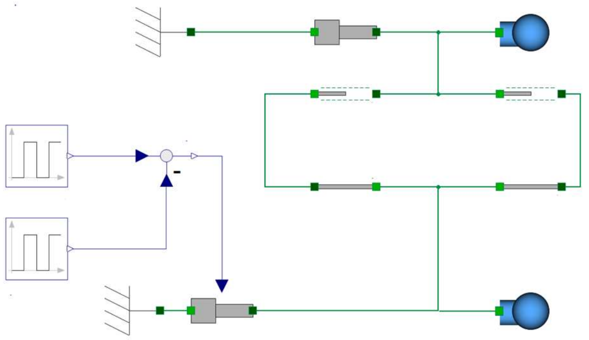


Figure 3: Modelica Diagram of a clamp on a fixed actuator. The upper body is squeezed between to elasto-gap models. The lower body represents the cartridge that is being moved by two force impulses.

The simulation plot in ***Figure 1***Figure 4 below shows the result of the corresponding simulation using two different time constants 1 microsecond and 1 millisecond. The system has been simulated in both cases with Runge-Kutta of 3rd order, using the corresponding step-width.

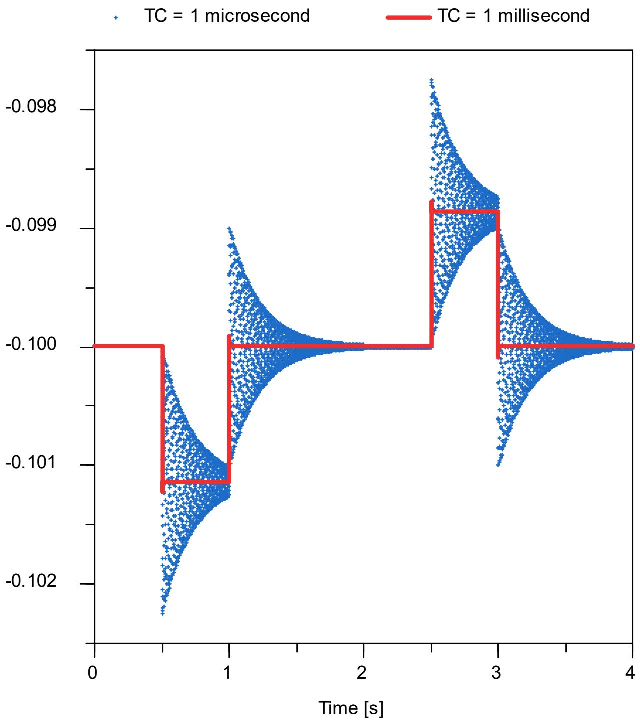


Figure 4: Penetration depth [mm] into the left clamp component represented by an elasto-gap, for the choice of two different time constants (TC = T_D). Both agree on the time-averaged solution.

Using a microsecond as time constant, we can see the resulting high-frequency solution in the contact region of the idealized hard-wood. Ideally, the oscillation should last forever (since no damping is assumed) but the small added damping lets the oscillation decay roughly within a second.

Using a millisecond as time constant, the system is almost perfectly damped artificially but exhibits the same shift in its quasi-equilibrium. The minute changes in penetration depth due the acceleration of the body are correctly assessed (on time-average basis).

4.2 1D Rotational Systems

Using strict analogy, a 1D-rotational library can be created. Here we use two angular velocities: ω_{el} and ω_{ki} to establish the dialectic regimes where again a balance of torque $\tau_{el} + \tau_{ki}$ forms the root of the equation system.

5 Complex Kinematics

To demonstrate the suitability of dialectic mechanics for complex kinematics, we have developed a planar mechanical library, similar to (Zimmer 2012).

As connector we use 2x3 pairs of potential and flow variables.

Listing 6. Planar mechanical connector

```
connector Frame
  //elastic regime
  SI.Position x;
  SI.Position y;
  SI.Angle phi;
  flow SI.Force fx_el;
  flow SI.Force fy_el;
  flow SI.Torque t_el;

  //kinetic regime
  SI.Velocity vx;
  SI.Velocity vy;
  SI.AngularVelocity w;
  flow SI.Force fx_ki;
  flow SI.Force fy_ki;
  flow SI.Torque t_ki;
end Frame;
```

The implementation is in strong correspondence, with the 1D translational library. For the sake of brevity, the code of the prismatic joint, is to be regarded as exemplary and provides sufficient insight into the general dialectic modeling style:

Listing 7. A prismatic joint in a planar world

```
model Prismatic "A prismatic joint"
  extends DialecticPlanarMechanics.Inter
faces.PartialTwoFrames;

  parameter Boolean useFlange=false;
  parameter SI.Time TD;
  parameter SI.Position r[2]
  final parameter SI.Length l=sqrt(r*r);
  final parameter SI.Distance e[2]= r/l

  Translational1D…Flange_a flange_a(
    s=s,v=v,
    f_el=f_el,f_kin=f_kin) if useFlange;

  SI.Position s(stateSelect = …prefer);
  SI.Velocity v(stateSelect = …prefer);
  SI.Velocity v_el;
  SI.Force f_el;
  SI.Force f_kin ;
  Real e0[2] ;
  SI.Position r0[2];
  Real R[2,2];

equation
  R={{cos(frame_a.phi),-sin(frame_a.phi)},
     {sin(frame_a.phi),cos(frame_a.phi)}};
  e0 = R*e;
  r0 = e0*s;

  //elastic regime
  frame_a.x + r0[1] = frame_b.x;
  frame_a.y + r0[2] = frame_b.y;
  frame_a.phi = frame_b.phi;
  frame_a.fx_el + frame_b.fx_el = 0;
  frame_a.fy_el + frame_b.fy_el = 0;
  frame_a.t_el  + frame_b.t_el
  + r0*{frame_b.fy_el,-frame_b.fx_el}
  = 0;

  //kinetic regime
  frame_a.vx - r0[2]*frame_a.w + v*e0[1]
  = frame_b.vx;
  frame_a.vy + r0[1]*frame_a.w + v*e0[2]
  = frame_b.vy;
  frame_a.w = frame_b.w;
  frame_a.fx_kin + frame_b.fx_kin = 0;
  frame_a.fy_kin + frame_b.fy_kin = 0;
  frame_a.t_kin  + frame_b.t_kin
  + r0*{frame_b.fy_kin,-frame_b.fx_kin}
  = 0;

  //synergy
  v_el= der(s);
  der(v)*TD = (v_el- v);
  {frame_b.fx_el,frame_b.fy_el}*e0
   + {frame_b.fx_kin,frame_b.fy_kin}*e0
   + f_el + f_kin = 0;


  //actuation force
  if not useFlange then
    f_el = 0;
    f_kin = 0;
  end if;

end Prismatic;
```

DOI
10.3384/ecp204167

Please note also, that the prismatic joint contains 1D-flange for actuation. Combing this joint with the 1D-elasto gap model provides for instance the opportunity to model limited joints in a natural way, without needing any extra components.

Indeed, when we combine this prismatic joint with a 1D-Elasto Gap model, we get a limited prismatic joint. We can then use then this joint to create the simple model of a thread pendulum as presented in Figure 5.

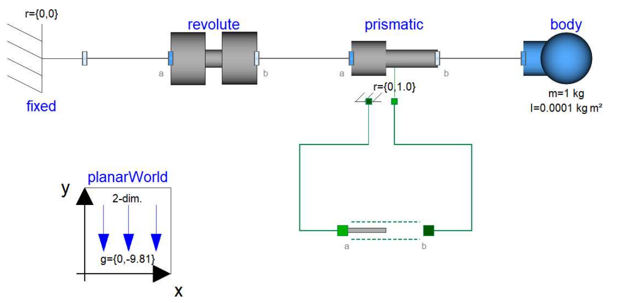


Figure 5: Model of thread pendulum. An elasto-gap is used to model the maximum extension of the thread.

Under the chosen initial conditions, the pendulum first swings through the lower hemicircle before it reaches its apogee and entering free fall conditions as in Figure 6. From then on, it sharply drops into its own thread, continually bouncing off the confining circle of the pendulum. This is because the thread is quite stiff with a spring constant of 1MN/m but only lightly damped with a damping constant of 1kNs/m

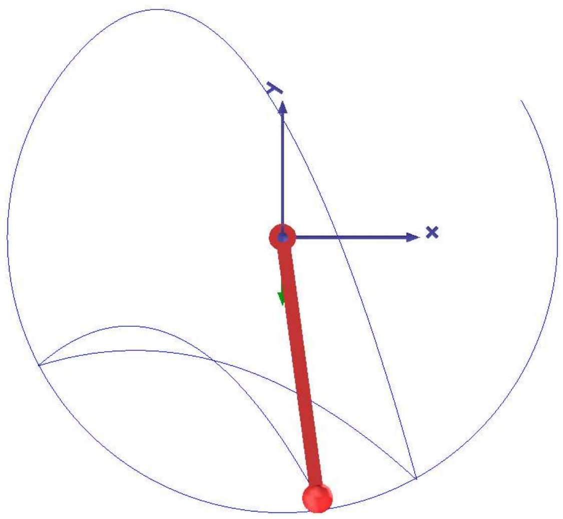


Figure 6: Trajectory of the thread pendulum for the first 2.6 seconds.

The simulation thus exhibits both slow mode and fast mode behavior. The first second with its swing through the hemicircle represents a slow mode behavior. Independent of the time constant for T_D all simulations agree on the elongation length of the thread due to the centrifugal and gravitational forces acting on the mass. There is only a slight phase shift depending on T_C visible in Figure 7.

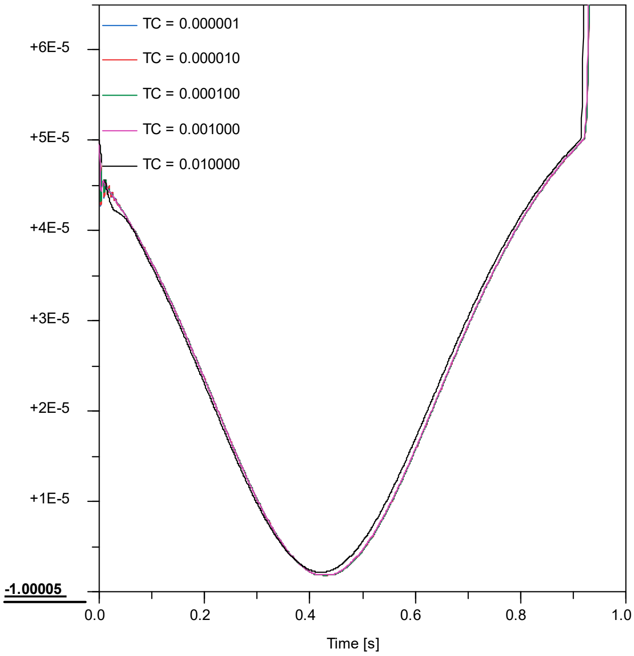


Figure 7: Extension of thread in meter through the first hemicycle due to gravity and centrifugal forces. In the slow-mode, the agreement of models with different time constants (TC $= T_D$) is high.

The bounce off its own thread represents a fast mode behavior. Here significant differences become visible in Figure 8 with respect to the choice of T_D. Low values for T_D lead to an artificially dampened system that dissipates its energy much quicker. This is exactly what is expected from the previous eigenvalue analysis.

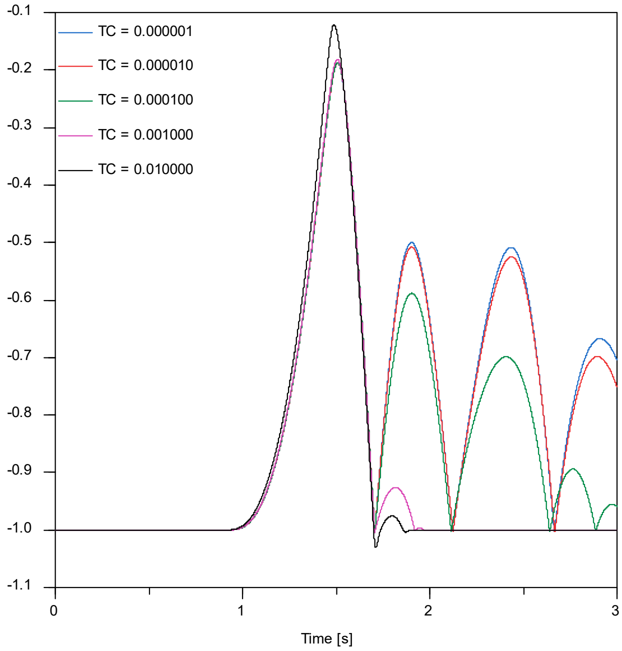


Figure 8: Center distance of pendulum body in meter during the overall trajectory. The artificial dampening with increased time constants (TC $= T_D$) impacts the elasticity of the bounce.

5.1 Kinematic Loops

When using dialectic mechanics, all joints always express state variables. Kinematic loops are closed using an elastic element (which however can be very stiff). The system therefore has more states than the classic set-up of kinematic loops, but avoids the formulation of a non-linear equation system. To solve for the balance of forces, still only a linear system of equations needs to be solved. Figure 9 presents is a simple 2D-kinematic loop for the extension of a landing gear.

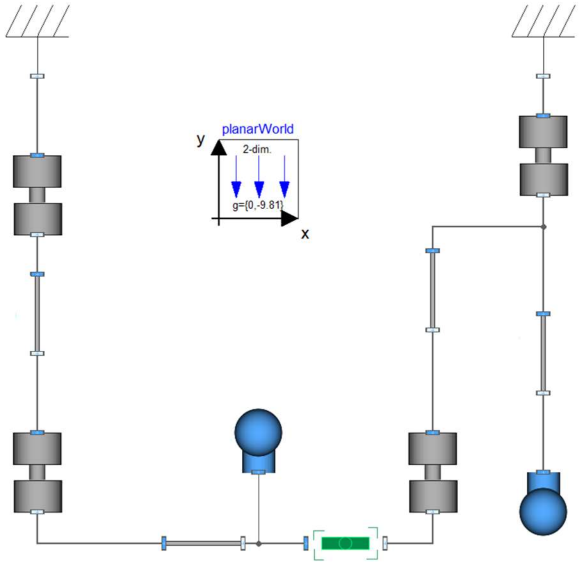


Figure 9: Model diagram of a simple unfolding kinematic of a landing gear. The loop is closed by the green component representing a spring-damper element (translational and rotational) with high stiffness.

This example has 8 state variables (the angles and the kinetic angular velocities of the revolute joints) and there is one linear implicit equation system of size that can be torn by 4 iteration variables: the 4 elastic angular velocities

One advantage of using an elastic element for closing loops is that typical singular points of maximal extension can now be properly handled. A fully rigid formulation exhibits a singular point at its point of maximum extension as depicted in Figure 10 because the kinetic energy at this point has nowhere to go. Using the elastic element for loop closure avoids this problem and the elastic elements can take the impulse from the kinematic reaching its limits.

As this example shows, even impulses on kinematic loops can be handled by dialectic mechanics. Because of the suppression of high frequencies, stiff springs can be used for closing kinematic loops without creating high frequencies. As with the example of the thread pendulum, the applied time-constant matters for the fast-mode behavior of the impulse but not for the slow unfolding dynamics.

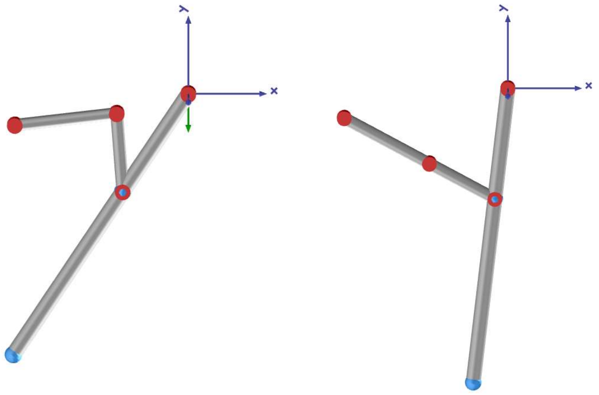


Figure 10: Visualization of the kinematic loop in two different states: unfolded on the right and partially folded on the left.

6 Conclusions

Let us now recapitulate on what we have actually implemented. Usually for a mechanical library, one of the first equations to write down would be:

$$a = \frac{d^2 s}{dt^2}$$

The acceleration is the second time-derivative of the position. What else should it be? Remarkably, this equation is not fulfilled in dialectic mechanics. Here we only make an approximation for lower frequencies.

$$a \approx \frac{d^2 s}{dt^2}$$

Effective modeling always represents an effective (and thereby lossy) compression of reality. It is hence all about doing an error on purpose where it is the most helpful. First of all, the rigid body assumption only holds up for low frequencies. At high frequency excitation, all bodies increasingly appear to be elastic. Limiting the frequency bandwidth is hence simply a consequential alignment to the rigid body assumption.

By making the acceleration only an approximation of the second derivative of position, we enable a disassociation of the regime for kinetic energy from the regime of potential energy. This disassociation enables the direct transfer of energy within these domains, especially within elastic elements.

Figure 11 illustrates a light-weighted, weakly damped body in the center of two very stiff springs and dampers.

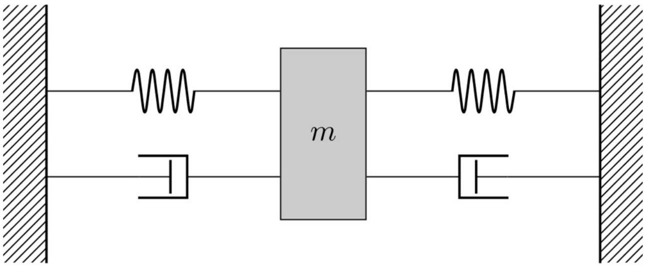


Figure 11: Classic representation of a body clamped in by two spring-damper elements. Any change of the potential energy stored in the springs has to go through the kinetic energy.

In the classic formulation any transfer of potential energy between the two springs has to go through the kinetic energy of the body component. This enforces very high frequencies.

In dialectic mechanics, we split the two regimes and couple them by a low-pass filter. This is illustrated in Figure 12.

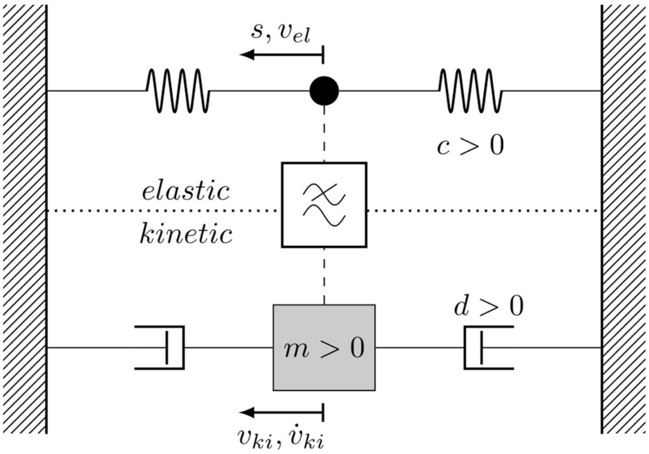


Figure 12: Dialectic view of the same system. On the elastic side, the body is now represented by a massless-point where only its low-frequency motion passes through the mass-holding body on the kinetic side. This enables an independent energy exchange of the potential energy stored in the springs. In this way, the elasto-static equilibrium can be found without transferring all energy through the body, instead it is (to a various degree) dissipated in the filter.

The dot connecting the two springs is now massless and only connected to the original mass by the first-order low-pass filter. In this way, a direct (dissipative) energy transfer between the springs is enabled and also the filter equation furthermore ensures that the springs are always undergoing a continues motion. Hence, the steady-state solution can be reliably and consistently found while avoiding higher frequencies.

This is useful because a lot of mechanical phenomena can be quite conveniently modelled using very stiff springs:

- Limited joints
- Contact dynamics
- Stiction
- etc…

The reason modelers learn to avoid very stiff springs is that they typically yield highly unfavorable eigendynamics. Using dialectic mechanics, the modeler can now use realistic spring constants without a bad conscience for many applications since the manipulation of eigenvalues for high frequencies keeps the dynamics in check without modifying the steady-state solution and only causing small errors for the dynamics of the slow modes. This greatly eases modeling of all of the above phenomena.

Regarding impulses: dialectic mechanics works fine for inelastic contacts. Fortunately, many gripping mechanisms are designed to provoke inelastic contacts having multiple layers of material with high damping constants (like human hands). For purely elastic contacts, there is a substantial error and the conservation of energy and momentum is disregarded. The error gets worse, the harder the material. With being too dissipative, the error is at least benevolent, meaning that it does not destabilize the system and enables a robust solution nonetheless.

The robustness and the avoidance of non-linear equation system in implicit form makes dialectic mechanics especially suited for the hard real-time simulation using explicit solvers. To this end, the presented manipulation of eigenvalues is however not sufficient and a further manipulation needs to be applied. Together with an extensive error analysis these are presented in the corresponding follow-up paper (Oldemeyer 2023). Interestingly also other approaches for explicit solvers split the time-domain and use a two-fold model approach such as (Peiret 2020).

Two remarks regarding the interface of dialectic mechanics. First remark: it is of course possible to model the ideal case where $v_{el} = v_{ki}$ using this interface as well. The interface would then be partly redundant which in consequence simply yields a slightly bloated formulation of classic multibody mechanics. In principal, mixing of approaches is hence possible. For the example of on-orbit servicing of satellites, the satellite trajectories could be modeled with ideal equations ensuring the conservation of momentum in space. The robotic interaction between satellites could then be modeled using a dialectic approach.

Second remark: Dialectic Mechanics is part of a larger modeling class denoted as Linear Implicit Equilibrium Dynamics (Zimmer, 2023). This class of models has originally been conceived to enable robust modeling of thermofluid systems but it revealed application potential outside this domain as well. Linear Implicit Equilibrium Dynamics is also a class of models whose compilation scheme is comparable simple and enables a generation of simulation code per component. This could be useful for mechanical libraries in a more dynamics run-time setting.

References

Benveniste, A., Caillaud, B., Elmqvist, H., Ghorbal, K., Otter, M., Pouzet, M. (2019). "Multi-Mode DAE Models - Challenges, Theory and Implementation". In: *Computing and Software Science. Lecture Notes in Computer Science*, vol 10000. Springer, Cham. https://doi.org/10.1007/978-3-319-91908-9_16

van der Linden, Franciscus L. J. (2016) *Gear contact modeling for system simulations and experimental investigation of gear contacts*. Dissertation, Technische Universität München.

Mehlhase, A. (2013) A Python framework to create and simulate models with variable structure in common simulation environments. *Mathematical and Computer Modelling of Dynamical Systems*, Vol. 20(6), pp. 566—583

Neumayr, Andrea & Otter, Martin. (2023). "Modelling and Simulation of Physical Systems with Dynamically Changing Degrees of Freedom." *Electronics*. 12. 500. 10.3390/electronics12030500.

Neves, Miguel (2019) *Human-In-The-Loop Controlled Lunar Landing Simulator*. Master Thesis, Technical University of Munich.

Oldemeyer, C., D. Zimmer (2023), "Hard Real-Time Simulation using Dialectic Mechanics", *Proceedings of 15th Modelica Conference,* Aachen, Germany

Otter, M., H. Elmqvist and S.E. Mattsson (2003), "The New Modelica MultiBody Library," *Proc. 3rd International Modelica Conference*, Linköping, Sweden, pp.311-330.

Peiret, A., González, F., Kövecses, J., and Teichmann, M. (2020). "Co-Simulation of Multibody Systems With Contact Using Reduced Interface Models." ASME. J. Computational. Nonlinear Dynamics. 15(4):

Reiser, Robert (2021) Object Manipulation and Assembly in Modelica. In: *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden*, pp 433-441. doi: 10.3384/ecp21181433.

Istvan Szabo (1986): *Geschichte der mechanischen Prinzipien*, 3. Auflage. Birkhäuser Verlag Basel.

Zimmer, D. (2012), A Planar Mechanical Library for Teaching Modelica *Proceedings of the 9th International Modelica Conference* , Munich, Germany

Zimmer, D. (2010), *Equation-Based Modeling of Variable Structure Systems*, PhD Dissertation, ETH Zürich, 219 pages

Zimmer, D. (2023), "Object-Oriented Formulation and Simulation of Models using Linear Implicit Equilibrium Dynamics" *Proceedings of 15th Modelica Conference,* Aachen, Germany

Pseudo Array Causalization

Karim Abdelhak[1] Francesco Casella[2] Bernhard Bachmann[1]

[1]Faculty of Engineering and Mathematics, University of Applied Sciences Bielefeld, Germany
{karim.abdelhak,bernhard.bachmann}@hsbi.de
[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy
francesco.casella@polimi.it

Abstract

In the current state-of-the-art modeling tools for simulation, it is common to describe system behavior symbolically using mixed continuous and discrete differential-algebraic equations, so called **hybrid DAEs**. To correctly resolve higher index problems, hybrid systems and to efficiently use ODE solvers, a matching and sorting problem has to be solved, commonly referred to as **Causalization**. Typically multidimensional equations and variables are scalarized, which leads to excessive build time and generated code size in the case of large systems. In the following paper an algorithm will be presented, that preserves array structures as much as possible while still solving the problem of causalization in scalar fashion. Test results carried out in the OpenModelica tool show a reduction in build time of one/two orders of magnitude and a reduction by a factor of two/three in the simulation run time for models of the ScalableTestSuite library.

Keywords: array preservation, causalization, matching, sorting, large scale

1 Introduction

The simulation of complex physical systems typically requires the handling of large systems of hybrid differential-algebraic equations. To model such systems the object-oriented equation based language *Modelica* was developed. The development of *Modelica* drastically decreased the amount of work necessary to simulate a model based on these so called ***hybrid DAEs***. Necessary steps such as causalization, index reduction and consistent initialization have been automated using symbolic transformation.

The results of this publication have been implemented in the *OpenModelica Compiler* (Fritzson et al. 2020), which is able to compile and simulate models from different domains, such as mechanics, electrics, fluids (Braun et al. n.d.) biology (Proß and Bachmann 2011; Kofránek et al. 2010) or power systems (Casella, Leva, and Bartolini 2017; Qi 2014; Viruez et al. 2017). Generally, it is designed to simulate any model that can be described with a system of hybrid DAEs. Theoretical background and definitions for differential-algebraic equations can be found in (Mattheij and Molenaar 2002). The goal, is to create a holistic environment for modeling and simulation to be used for teaching, research and in the industry.

2 State of the Art

Current simulation tools based on the modeling language *Modelica* scalarize the equations and variables of a system to apply scalar methods of causalization and symbolic manipulations. This has major drawbacks, mainly revolving around computation time and memory usage. Besides the approach of scalarization, there has been work published with similar intentions to this paper (Otter and Elmqvist 2017; Neumayr and Otter 2023; Zimmermann, Fernández, and Kofman 2020).

The work of (Otter and Elmqvist 2017) focusses around reducing a model containing array equations to index-1 form using index-reduction methods, without having to scalarize. Although index reduction will not be covered in this paper, it is expected to be able to apply scalar methods for index reduction using pseudo array causalization.

Methods presented in (Neumayr and Otter 2023) allow the size of generated arrays to be changed after code generation. These ideas are not part of this paper, but the idea of generalized for-equations will be expanded in future work to for-equations of variable size.

The approach of (Zimmermann, Fernández, and Kofman 2020) presents a new algorithm that adapts the idea of scalarized matching and expands it to set-based graphs. Array structures are preserved as much as possible and only split up during the process of matching if no other solution can be found.

This paper focusses on providing a solution that is applicable without language restrictions while retaining the most compact array form possible. The core algorithms of matching and sorting are not expected to be bottlenecks in computation time and are performed using scalar methods, while having all other symbolic manipulation methods operate on array structures. The test results shown in section 6 support this assumption.

3 Causalization

Solving hybrid differential algebraic systems of equations requires the process of causalization, also known as BLT-Transformation[1], to ensure that

a. high differential index problems (index > 1) can be resolved

[1] BLT: Block-Lower-Triangular

DOI
10.3384/ecp204177

b. the dependencies involving discrete equations and variables are found.

c. causalized systems can be simulated far more efficiently due to explicit assignments instead of a large implicit system. Exceptions prove the rule of course (Henningsson, Olsson, and Vanfretti 2019).

BLT-Transformation mainly consists of three steps, *Matching*, *Index-Reduction* and *Sorting*.

3.1 Scalarization

Before being able to perform *scalar* BLT-Transformation on a system of variables and equations, they need to be scalarized. **Modelica** offers comfortable ways of defining multiple equations at once, such as *for-equations*:

```
for i in 1:N loop
  der(x[i]) = i * sin(time);
end for;
```

Scalarizing this *for-equation* leads to following equations

```
  der(x[1]) = 1 * sin(time);
  der(x[2]) = 2 * sin(time);
  ...
  der(x[N]) = N * sin(time);
```

which scales with the size of N. This currently is common practice among **Modelica** tools. Scalarization increases the computation time unnecessarily, as symbolic manipulation on the body is done N times instead of only once on the body equation. However, for current methods of *Causalization* this step is necessary.

Instead of scalarization, for-equations and array-equations will be converted into canonical form, which is further explained in 5.

3.2 Matching

The first step to the process of causalization is *matching*. The goal is to find an equation for each variable in which it can be solved. Note that this is only a theoretical assignment, in the sense that these assignments can also be either ambiguous, resulting in an algebraic loop, or only implicitly solvable in the first place.

A system of equations can be understood as a bipartite graph, where one set of nodes represents the equations and the other set represents the unknowns for which the system has to be solved. Edges in that graph show variable incidences in equations. The goal of *matching* algorithms is to find a *perfect matching*, which is achieved by assigning each variable uniquely to an equation such that each variable and each equation is assigned only once. This *matching* problem was analyzed thoroughly and the most commonly used algorithm to solve it is the Ford-Fulkerson (or maximum flow) algorithm, first described in (Ford and Fulkerson 1956). The OpenModelica-Compiler has a large selection of different matching algorithms, of which Pothen and Fan's algorithm (PF+) was selected to be the default (Duff, Kaya, and Uçcar 2012; Kaya et al. 2011).

3.3 Sorting

After a perfect matching has been found, the process of sorting determines the order in which to execute those assignments. The order, once again, can be ambiguous, depending on the system. Furthermore, sets of equations which have to be solved at the same time, so called *algebraic loops* are identified. Tarjan's algorithm (Tarjan 1972) is implemented in OpenModelica and the most commonly used sorting algorithm.

4 Pseudo Array Causalization

The main idea of **P**seudo **A**rray **C**ausalization (**PAC**) revolves around doing as few scalarization steps as needed, while still using scalar causalization methods. Previous tests have shown that the graph-based causalization algorithms scale linearly with the size of well posed and reasonable models, even though the theoretical computational complexity is nonlinear (Kaya et al. 2011). Far more time is spend on symbolic manipulation or generating code.

In the following an algorithm will be presented, that keeps all equations and variables in their array form only creating a scalarized graph for causalization (a different approach to (Zimmermann, Fernández, and Kofman 2020), where a set-based graph is used). Known matching (see section 3.2) and sorting (see section 3.3) algorithms can be used on the scalar graph to resolve causalization. A three-step sorting as described in section 4.3 is applied, to ensure that the result contains as few slicing steps as possible. Array-based strong components can be derived using information about the underlying array structures and the result of causalization.

Due to the array equations and variables never being scalarized symbolically, all optimization methods only scale with the number of array components rather than the number of scalarized components. Furthermore, the created simulation code is far smaller, due to the fact that compact array structures were preserved.

4.1 Preparation for Causalization

To recover the array structures after causalization with scalar methods, three structures have to be created beforehand:

Mapping: Maps array indices to the list of their scalar children and vice versa.

Matrix: Represents the scalar graph as an adjacency matrix.

Modes: Compact way of tracking which equation is solved for what variable instance.

4.1.1 Mapping

The goal of this section is to obtain functions $\mathscr{M}_V$ and $\mathscr{M}_E$ that map a variable name and its indices or an equation and its iterator values (if it is a for-equation) to a unique scalar index. Furthermore, the four inverse functions $\mathscr{M}_V^{-1}$ and $\mathscr{M}_E^{-1}$ to recover the indices or iterator values and $\hat{\mathscr{M}}_V^{-1}, \hat{\mathscr{M}}_E^{-1}$ to recover the original variables and equations from the unique scalar indices, have to be defined.

First, there needs to be a mapping for variable and equation names to their respective array indices $\mathbb{I}_a$. These are trivial, but necessary for further explanations:

$$N_V : V \rightarrow \mathbb{I}_a \tag{1}$$

$$N_E : E \rightarrow \mathbb{I}_a \tag{2}$$

with V and E as the set of variable and equation names and $\mathbb{I}_a$ being the set of array indices. Since these have to be uniquely indexed, they are bijective and have inverse functions N_V^{-1}, N_E^{-1}.

Furthermore, an index mapping for variables and equation has to be defined. The main restriction is that all scalar variables that belong to the same array variable need to have consecutive indices, the same is true for equations. This restriction allows more predictable outcomes from the causalization methods such that reasonable recollection and slicing is possible. In the following the variables will be indexed in such a way, that the innermost dimension is iterated first, then the second etc. The same is true for equations and the innermost iterator, second to innermost iterator and so on. It is important to differ between iterator *value* and *normalized index* for this indexing method. Considering an iterator with a range of $10 : -2 : 2$, the *value* 10 corresponds to *index* 0, *value* 8 corresponds to *index* 1 and so on. Even though the modeling language *Modelica* has 1-based indices, all indexing will be considered to be 0-based. This allows easier computation of index mappings. The mapping of a single variable index s_v can be done by subtracting 1. The index s_e representing an iterator value i of an equation on the other hand, has to be computed:

$$s_v(i) = i - 1 \tag{3}$$

$$s_e(i) = \frac{i - r_{start}}{r_{step}} \tag{4}$$

with r_{start} as the start and r_{step} as the step of the range. This mapping is only defined for reachable iterator values with $i \equiv r_{start} \mod r_{step}$. The function applying this index shift on all indices of a variable or iterator values of an equation will be called $S_v(indices, v)$ and $S_e(indices, e)$ respectively. For scalar variables and equations these functions return 0 if *indices* is an empty list. Furthermore these functions are bijective and therefore invertible ($\exists S_v^{-1} \wedge \exists S_e^{-1}$).

The *local* mapping of an equation or variable with n dimensions can be described as a function

$$m : \Pi_{i=1}^{n} \mathbb{I}_i \rightarrow \mathbb{I} \tag{5}$$

where $\mathbb{I}_i = \{x \in \mathbb{N}_0 \mid x < d_i\}$ and d_i being the size of the i-th dimension. $\mathbb{I} = \{x \in \mathbb{N}_0 \mid x < d\}$ with $d = \Pi_{i=1}^{n} d_i$ as the set of all flattened indices. The inverse *local* mapping

$$m^{-1} : \mathbb{I} \rightarrow \Pi_{i=1}^{n} \mathbb{I}_i \tag{6}$$

is also needed to recover the original multi-dimensional indices for slicing. Each equation and variable has their own *local* mapping m and inverse *local* mapping m^{-1}.

The two *global* mappings M_E and M_V each map all array indices to the indices of their scalar members. They are derived by creating the pseudo inverse[2] maps $M_E^{\dagger}$ and $M_V^{\dagger}$ through enumeration of all array equations ($\mathbb{I}_E^a$) and variables ($\mathbb{I}_V^a$) and all (hypothetical) scalar equations ($\mathbb{I}_E^s$) and variables ($\mathbb{I}_V^s$) and letting the scalar indices point to the index of the original array equation.

$$M_E^{\dagger} : \mathbb{I}_E^s \rightarrow \mathbb{I}_E^a \tag{7}$$

$$M_V^{\dagger} : \mathbb{I}_V^s \rightarrow \mathbb{I}_V^a \tag{8}$$

Since scalar indices have to be consecutive, the *global* mapping M can be stored more efficiently, by only storing the start index and its length. For further explanations the length is irrelevant and therefore omitted. These *global* mappings have to be created for equations ($M_E, M_E^{\dagger}$) and variables ($M_V, M_V^{\dagger}$). An example for these mappings can be found in figure 1.

In the following, variable and equation names will be used instead of their indices. Using a variable as v_{ind} with *ind* as the indices will be used as a representor of the variable index in scalar context. It is found using the following full map $\mathscr{M}_V$ which converts a variable name and its list of subscript indices to the scalar variable index.

$$v_{ind} = \mathscr{M}_V(v, ind) = M_V(N_V(v)) + m_v(S_v(ind, v)). \tag{9}$$

The inverse of this function is split up into two parts, one recovering the variable name and one recovering the indices. It will be necessary in chapter 5 to recover the original variable representations in equations.

$$v = \hat{\mathscr{M}}_V^{-1}(v_{ind}) = N_V^{-1}(M_V^{\dagger}(v_{ind})) \tag{10}$$

$$ind = \mathscr{M}_V^{-1}(v_{ind}) = S_v^{-1}(m_v^{-1}(v_{ind} - M_V(N_V(v))), v) \tag{11}$$

Likewise an equation as e_{val} with *val* as the iterator values implies the following operations:

$$e_{val} = \mathscr{M}_E(e, val) = M_E(N_E(e)) + m_e(S_e(val, e)). \tag{12}$$

and has similarly formulated inverse mappings:

$$e = \hat{\mathscr{M}}_E^{-1}(e_{val}) = N_E^{-1}(M_E^{\dagger}(e_{val})) \tag{13}$$

$$val = \mathscr{M}_E^{-1}(e_{val}) = S_e^{-1}(m_e^{-1}(e_{val} - M_E(N_E(e))), e). \tag{14}$$

[2] These pseudo inverse maps have the property $M^{\dagger}M = \text{id}$, however in general $MM^{\dagger} \neq \text{id}$, which is similar to the Moore-Penrose pseudo inverse matrix definition from (Penrose 1955).

```
model mapping_example
  parameter Integer n = 3;
  Real x[n+1];
  Real y[n,n];
equation
  x[1] = sin(time) "Scalar equation e";
  for i in 1:n loop
    x[i] = y[i,i] + x[i+1];
  end for "For-equation f";
  for i in 1:n, j in 1:n loop
    y[i,j] = i*cos(j*time);
  end for "For-equation g";
end mapping_example;
```

Local Mapping

m_x :
$[0] \mapsto 0$
$[1] \mapsto 1$
$[2] \mapsto 2$
$[3] \mapsto 3$

m_y :
$[0,0] \mapsto 0$
$[0,1] \mapsto 1$
$[0,2] \mapsto 2$
$[1,0] \mapsto 3$
$[1,1] \mapsto 4$
$[1,2] \mapsto 5$
$[1,3] \mapsto 6$
$[2,1] \mapsto 7$
$[2,2] \mapsto 8$

m_e :
$[0] \mapsto 0$

m_f :
$[0] \mapsto 0$
$[1] \mapsto 1$
$[2] \mapsto 2$

m_g :
$[0,0] \mapsto 0$
$[0,1] \mapsto 1$
$[0,2] \mapsto 2$
$[1,0] \mapsto 3$
$[1,1] \mapsto 4$
$[1,2] \mapsto 5$
$[2,0] \mapsto 6$
$[2,1] \mapsto 7$
$[2,2] \mapsto 8$

Global Mapping

M_V :
$0 \mapsto 0$
$1 \mapsto 4$

$M_V^\dagger$:
$0,1,2,3 \mapsto 0$
$4,5,\ldots,11,12 \mapsto 1$

M_E :
$0 \mapsto 0$
$1 \mapsto 1$
$2 \mapsto 4$

$M_E^\dagger$:
$0 \mapsto 0$
$1,2,3 \mapsto 1$
$4,5,\ldots,11,12 \mapsto 2$

Figure 1. Example for index mapping.

4.1.2 Matrix

A scalar adjacency matrix A has to be created while respecting the index mappings $\mathscr{M}_E$ and $\mathscr{M}_V$ from (7). The rows belonging to for-equations can be created by first extracting all occurring variable instances and afterwards iterating over the ranges of the iterators, replacing every instance of an iterator in the variable instance indices with their local values. For each possible iterator configuration a list of occuring variable instances is created. The multi-dimensional indices are mapped to their respective scalar index using mapping $\mathscr{M}_V$. Each iterator configuration results in the i-th row of the scalar adjacency matrix, where i is that configuration mapped using the mapping $\mathscr{M}_E$. The pseudo code for this procedure is shown in algorithm 1.

In all further explanations a bipartite graph representation of the adjacency matrix will be used where the nodes are enumerated from top to bottom. The bipartite digraph for the example from figure 1 can be seen in figure 2. The causalization modes derived from algorithm 1 are represented as edge markings and will be explained in the following section 4.1.3.

4.1.3 Modes

Each equation can be solved in n different ways, where n is the number of different variable instances in that equation. It is important to note here, that the occurence of the same variable indexed differently, has to be counted as two distinct variable instances. A causalization mode for solving an array equation e for variable v will be denominated as

$$e \rightarrowtriangle v. \tag{15}$$

Example 4.1. There are three instances `x[i]`, `y[i]` and `x[i+1]` in the following for-equation e:

```
for i in 1:10 loop
  x[i] = y[i] + x[i+1];
end for;
```

It has three causalization modes, $e \rightarrowtriangle x[i]$, $e \rightarrowtriangle y[i]$ and $e \rightarrowtriangle x[i+1]$.

For each scalar equation a mode mapping will be created as a function

$$c : \mathbb{V} \to \mathbb{M}_i \tag{16}$$

with $\mathbb{V}$ being the set of all scalar variable indices and $\mathbb{M}_i$ as the set of all modes (variable instances) for the corresponding array equation i. These mode mappings can be created alongside with the adjacency matrix A while replacing the iterators. If variable indices are used it is more efficient to create it as an inverse mapping

$$c^{-1} : \mathbb{M}_i \to \mathbb{V} \tag{17}$$

to not create large arrays when only a few variables actually occur. Since the number of modes is usually very low, one can easily traverse all modes in search for the correct scalar variable to determine in which mode the scalar equation was solved. These causalization modes can be used to correctly slice an array equation after causalization by determining the slices that have been solved for the same variable instance.

For the example from figure 1 and its digraph shown in figure 2, one can see that there are five different causalization modes in total. These causalization modes are shown in figure 3 in greater detail.

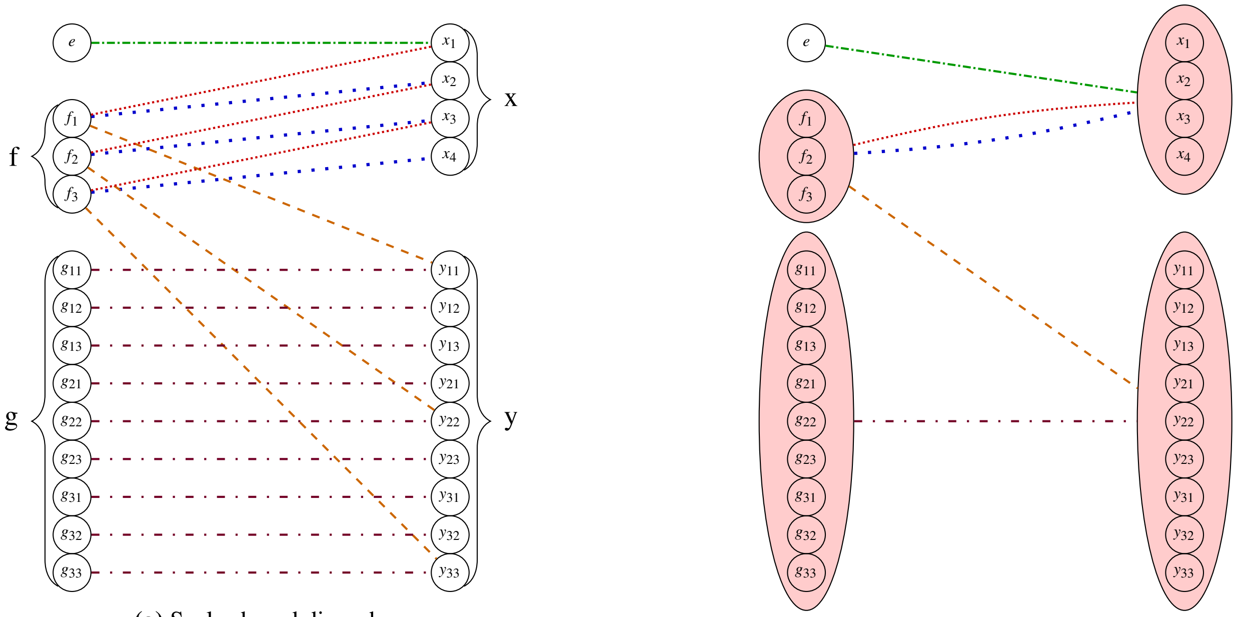
4.2 Pseudo-Array Matching

The Pseudo-Array Matching algorithm requires all the preparation described in section 4.1 and further explanations are based on the example in figure 1.

Based on the digraph shown in figure 2(a) the scalar matching algorithm described in section 3.2 is applied. The resulting perfect matching uses the causalization modes $e \rightarrowtriangle x[1], f \rightarrowtriangle x[i]$ and $g \rightarrowtriangle y[i,j]$ in its entirety. This convenient solution requires no slicing, harder problems requiring more sophisticated methods are shown in section 5. The matching solution is represented as an array Ω that maps a scalar equation index to the matched

Algorithm 1 Adjacency Matrix and Causalization Modes

```
Input: equation array $E$
Input: variable array $V$
Output: adjacency matrix $A$                                ▷ efficient structure for matching
Output: causalization modes $C$ (inverse map)               ▷ modes to recover after matching
Output: mode to variable instance map $N$
$A, C, N \leftarrow$ initialize as empty arrays of size $|\mathbb{I}_E^s|$
for $eq$ in $E$ do
    $vars \leftarrow$ find all variable instances in $eq$ using $V$
    $modes \leftarrow$ create unique identifiers for $eq$ being solved for each $var$ in $vars$
    $N[eq][modes] \leftarrow vars$                          ▷ array assignment
    if $eq$ is a for-equation then
        for all iterator combinations $iter$ in $eq$ do
            $row \leftarrow$ apply function (9) on each $var$ in $vars$ using $iter$
            $i \leftarrow$ apply function (12) on $eq$ name using $iter$
            $A[i] \leftarrow$ row                           ▷ list assignment
            $C[i][modes] \leftarrow$ row                    ▷ array assignment
        end for
    else
        $row \leftarrow$ apply function (9) on each $var$ in $vars$
        $i \leftarrow$ apply function (12) on $eq$
        $A[i] \leftarrow$ row                               ▷ list assignment
        $C[i][modes] \leftarrow$ row                        ▷ array assignment
    end if
end for
```



(a) Scalar-based digraph.

(b) Array-based digraph.

Figure 2. Digraph for model 1. The different causalization modes are as follows:
$e \multimap x[1]$ (green), $f \multimap x[i]$ (red), $f \multimap x[i+1]$ (blue), $f \multimap y[i,i]$ (orange), $g \multimap y[i,j]$ (purple).

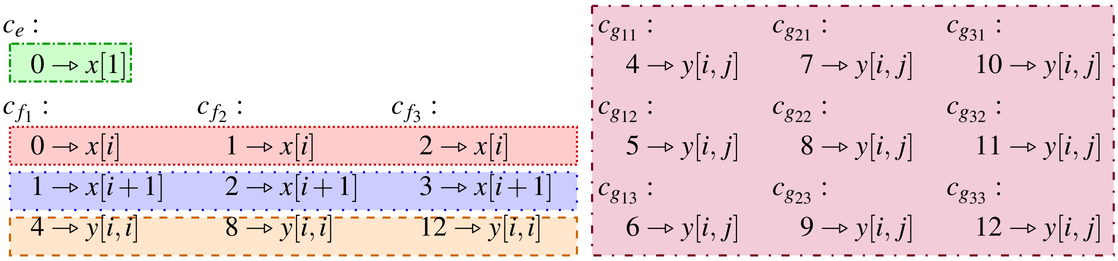

c_e :
$0 \rightarrowtriangle x[1]$
c_{f_1} :
c_{f_2} :
c_{f_3} :
$0 \rightarrowtriangle x[i]$
$1 \rightarrowtriangle x[i]$
$2 \rightarrowtriangle x[i]$
$1 \rightarrowtriangle x[i+1]$
$2 \rightarrowtriangle x[i+1]$
$3 \rightarrowtriangle x[i+1]$
$4 \rightarrowtriangle y[i,i]$
$8 \rightarrowtriangle y[i,i]$
$12 \rightarrowtriangle y[i,i]$
$c_{g_{11}}$:
$4 \rightarrowtriangle y[i,j]$
$c_{g_{21}}$:
$7 \rightarrowtriangle y[i,j]$
$c_{g_{31}}$:
$10 \rightarrowtriangle y[i,j]$
$c_{g_{12}}$:
$5 \rightarrowtriangle y[i,j]$
$c_{g_{22}}$:
$8 \rightarrowtriangle y[i,j]$
$c_{g_{32}}$:
$11 \rightarrowtriangle y[i,j]$
$c_{g_{13}}$:
$6 \rightarrowtriangle y[i,j]$
$c_{g_{23}}$:
$9 \rightarrowtriangle y[i,j]$
$c_{g_{33}}$:
$12 \rightarrowtriangle y[i,j]$

Figure 3. Scalar equation mode mapping. The colors indicate the corresponding causalization mode: $e \rightarrowtriangle x[1]$ (green), $f \rightarrowtriangle x[i]$ (red), $f \rightarrowtriangle x[i+1]$ (blue), $f \rightarrowtriangle y[i,i]$ (orange), $g \rightarrowtriangle y[i,j]$ (purple).

scalar variable index. To correctly interpret this matching in the context of array recovery, the causalization modes (see section 4.1.3) have to be taken into account. The causalization modes for the given example are shown in figure 3 but are stored as an inverse mapping for more efficient lookup by avoiding large empty arrays. The basic idea is to iterate over all modes of a given equation until the matched variable is found to determine the correct mode. A mapping, further called buckets, or in short B, that collects scalar equations that belong to the same array equation and are solved for the same variable instance (in the same causalization mode), is found by the procedure shown in algorithm 2. The bucket structure returns a list of scalar equations when provided with an array equation and a causalization mode identifier. For this trivial case, the arrays could be split up as shown in the array based digraph from figure 2(b). More complicated cases require the Three Step Sorting presented in the following chapter 4.3.

Algorithm 2 Recover Causalization Modes

```
Input: matching $\Omega$                          ▷ $eqn \rightarrow var$
Input: mapping $M_E^{-1}$                       ▷ $scalar \rightarrow array$
Input: causalization modes $C$
Output: buckets B
$B \leftarrow$ empty lists for all entries
for $e$ in $0 : length(\Omega) - 1$ do
    $m \leftarrow -1$
    do
        $m \leftarrow m + 1$
        $var \leftarrow C[e][m]$
    while $var \neq \Omega[e]$
    append $e$ to $B(M_E^{-1}(e), m)$
end for
```

4.3 Three Step Sorting

The result of the sorting process presented in section 3.3 does not have a unique solution and rather strongly depends on the ordering of variables and equations and even more so on the chosen mapping (see section 4.1.1). Since the result is ambiguous it can be hard to recover the most compact way of representing arrays if fragments of arrays are scattered instead of consecutive, if possible. The Three Step Sorting was implemented to ensure that the resulting sorted strong components respect the original array structures.

The three steps are *scalar sorting, array sorting* and *internal sorting*. A schematic outline for this process is shown in example 4.2 and the basic outline is as follows:

1. **Pseudo-Array Matching** Perform scalar matching while collecting all necessary information to recover arrays, as described in section 4.2.

2. **Scalar Sorting** The first step of sorting using Tarjan's algorithm (see section 3.3).

3. **Merge algebraic loop nodes** Merge all equation nodes that belong to the same strong component in the result of step 2 and do the same for variable nodes.

4. **Merge array nodes** Merge all equation nodes that belong to the same array and are solved for the same variable instance, using the information preserved in step 1. Do the same for variables. Equations and variables that were already merged in step 3 do not get merged in this step.

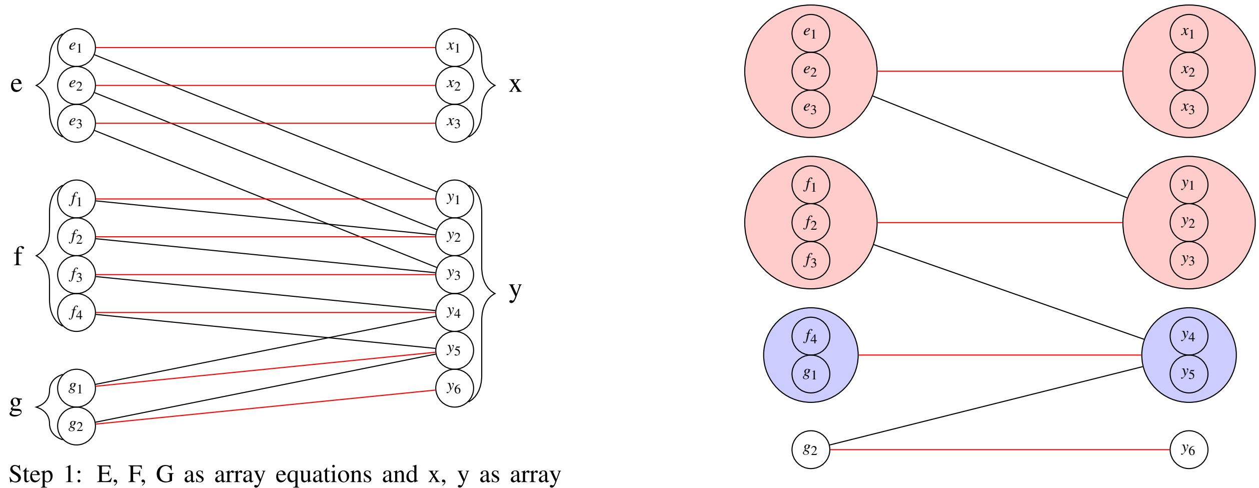
5. **Array sorting** Apply Tarjan's algorithm again on the new graph.[3]

6. **Internal sorting** Strong components of size greater than one that are a result of step 5 are not algebraic loops, because these would have been found in step 2. They are equations that have to be executed sequentially, but alternate between different arrays (and/or scalar equations). These strong components will be called *entwined equations* in further explanations. Furthermore all *array* and *entwined equations* have to be sorted internally using Tarjan's algorithm.

Example 4.2 (Sliced Arrays).
As a first example for hard to solve slicing problems, we consider the following model.

```
model sliced_arrays
  Real x[3];
  Real y[6];
```

[3] By construction each super node has scalar matching edges only to one other super node, therefore the matching for the new graph does not have to be computed.



(a) Step 1: E, F, G as array equations and x, y as array variables.

(b) Step 2: Blue super nodes represent algebraic loops and red super nodes represent arrays.

Figure 4. The process of Three Step Sorting.

```
equation
  for i in 1:3 loop
    x[i] = y[i]*cos(time);
  end for "For-equation e";
  for j in 1:4 loop
    y[j] = y[j+1]*2;
  end for "For-equation f";
  for k in 5:6 loop
    y[k] = y[k-1] + sin(time);
  end for "For-equation g";
end sliced_arrays;
```

The expected outcome of this model for the process of causalization is as follows:

- The first for-equation will be matched to all of x.
- The second for-equation will be matched to index 1 to 4 of y.
- The third for-equation will be matched to index 5 and 6 of y.
- The last equation f_4 of the second for-equation forms an algebraic loop with the first equation g_1 of the third for-equation.

As can be seen in figure 4(a) the matching turns out as expected. Furthermore, one can see that there is an algebraic loop connecting the mentioned equations f_4 and g_1 and $y[4]$, $y[5]$. To efficiently resolve this loop, it is desirable to slice the for-equations in such a way, that only the two relevant equations end up in the algebraic loop and the rest is recovered as for-equations. After the first step of scalar sorting one can combine all the equations of an algebraic loop to a singular super node and do the same with variables of each algebraic loop. Only after this is done, the array super nodes should be created by merging all the remaining equations and variables to super nodes, while respecting the information gathered in section 4.1. To achieve the desired result of minimal algebraic loops while retaining as much array structure as possible, it is required to do array node merging after algebraic loop merging. The result of node merging can be seen in figure 4(b).

5 Generalized For-Equations

Before processing for-equations and array-equations they have to be converted into canonical form. Any for equation that contains $n > 1$ body equations can be split into n for-equations that each contain one single body equation. If the solution requires the body equations to be evaluated in alternating order, it will be processed as an entwined for-equation, which is explained in the following. Furthermore, array-equations can be converted into canonical for-equations using simple transformations, allthough this is not sufficiently tested yet.[4]

After the process of sorting, described in section 4.3, there are four general types of strong components.

Explicit Strong Component. An explicit strong component is a single assignment that can be solved explicitly for the chosen variable. These don't necessarily have to be scalar, they can be array assignments.

Implicit Strong Component. Implicit strong components can be single equations that cannot be solved symbolically, as well as algebraic loops, consisting of multiple equations that have to be solved simultaneously.

Simple For-Equation. A simple for-equation is a section of a for-equation that can be executed without the need to perform other assignments in between.

Entwined For-Equation. Entwined for-equations are for-equations that have mutual dependencies and

[4] Some array-equation to for-equation transformations are not efficient e.g. if they contain a function call. These will be handled as algorithms.

need to be executed in alternating order. They can also contain explicit or implicit strong components that have to be executed once.

The first two types of strong components pose no further challenge an can be processed using techniques described in section 3. The latter two for-equation based strong components require further analysis. *Simple For-Equations* have to undergo the third step of inner sorting, described in chapter 4.3, which results in a specific order of the for-equation body. This order might not be in a order that can be represented using the original for-equation iterator ranges because of slices being solved differently. An example for this is shown in the following example 5.1.

Example 5.1 (Diagonal Slice).
As an example for slices of for-equations that cannot be recovered using the original iterator range, consider a model where the diagonal of a matrix has to be solved in a different equation than the rest of it. The results for following model are shown in the two digraphs of figure 5 and confirm the expected results:

- The first for-equation will be solved for the diagonal elements of x
- The second for-equation will be split up into two for-equations:
    1. $i \neq j$ solves the remaining non-diagonal elements of x
    2. $i = j$ solves y

```
model diagonal_slice
  Real x[3,3];
  Real y[3];
equation
  for i in 1:3 loop
    x[i,i] = i*cos(time);
  end for "For-equation e";
  for i in 1:3, j in 1:3 loop
    x[i,j] = y[j] + i*sin(j*time);
  end for "For-equation f";
end diagonal_slice;
```

As can be seen in figure 5(a), the expected matching is found. No algebraic loop super nodes are created, but three different equation and variable super node pairs.

1. The first for-equation (e_1, e_2, e_3) solved for the variable instance $x[i,i]$.
2. The second for-equation $(f_{12}, f_{13}, f_{21}, f_{23}, f_{31}, f_{32})$ solved for the variable instance $x[i,j]$.
3. The second for-equation (f_{11}, f_{22}, f_{33}) solved for the variable instance $y[j]$.

The three resulting for-equations are **Simple For-Equation** strong components and need to be sorted internally. The first for-equation does not pose a problem since it is not sliced at all and has no structurally forced order. Furthermore, one can safely assume that, if nothing is forced, the sorting algorithm will act index-first and keep the equation as it is:

```
for i in 1:3 loop
  x[i,i] = i*cos(time);
end for;
```

The second for-equation poses the problem that it does not use the entirety of the original for loop. Furthermore, it cannot be represented by a single for-equation, without using an additional element, like an if-condition to strip it off its diagonal. Even though this technique could be used in this specific case, a general solution is desirable. To achieve a procedure that can be applied on for-equations sliced and ordered in any way, the list of scalar equation indices is iterated by applying algorithm 3. Trivially, the same can be done for the third for-equation.

Algorithm 3 Evaluate Generic Body

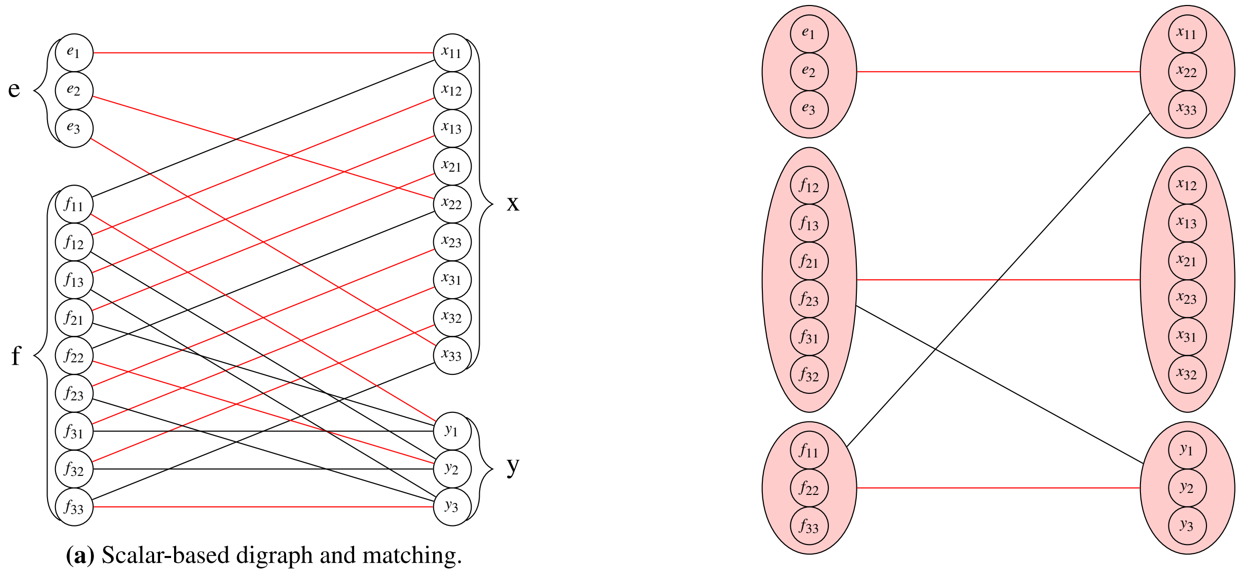
```
Input: Scalar index $e_{val}$
$e \leftarrow \hat{\mathscr{M}}_E^{-1}(e_{val})$        ▷ get equation body, see (13)
$val \leftarrow \mathscr{M}_E^{-1}(e_{val})$      ▷ get iterator values, see (14)
evaluate equation $e$ with iterator values $val$
```

The following code is representative for the code OpenModelica generates compiling the example model `diagonal_slice`, simplified for readability. It shows the non-diagonal section of for-equation f.

```
void diagonal_slice_eq_1(DATA *data)
{
  const int idx_lst[6] = {5,2,7,1,6,3};
  for(int i=0; i<6; i++)
    genericCall_0(data, idx_lst[i]);
}
```

The `idx_lst` represents the order in which the body equations have to be solved, which is arbitrary in this specific example. This solution is provided by the sorting algorithm and no further optimization is done since this order might be enforced structurally, which is the case for other examples. The function `genericCall_0` represents the body of the function, which maps the scalar index to the iterator values, as described in algorithm 3. Allthough the theory speaks of a global mapping, in practice one uses the local mapping and the local index to evaluate the body equations of a for-equation.

```
void genericCall_0(DATA *data, int idx)
{
  int tmp = idx;
  int i_loc = tmp % 3;
  int i = 1 * i_loc + 1;
  tmp /= 3;
  int j_loc = tmp % 3;
  int j = 1 * j_loc + 1;
  tmp /= 3;
  &data->realVars[(i-1)*3+(j-1)] /*x[i,j]*/
      = &data->realVars[9+(j-1)] /*y[j]*/
      + i * sin(j * data->timeValue);
}
```



(a) Scalar-based digraph and matching.

(b) Array-based digraph and matching.

Figure 5. Causalization of the example model 5.1 requiring diagonal slicing.

Example 5.2 (Entwined Loops).
As a second example for hard to solve slicing problems, we consider the following model.

```
model entwined_loops
  Real x[7];
  Real y[7];
equation
  x[1] = 1;
  y[1] = 2;
  for j in 2:7 loop
    x[j] = y[j-1] * sin(time);
  end for "For-equation e";
  for i in 2:4 loop
    y[i] = x[i-1];
  end for "For-equation f";
  for i in 5:7 loop
    y[i] = x[i-1] * 2;
  end for "For-equation g";
end entwined_loops;
```

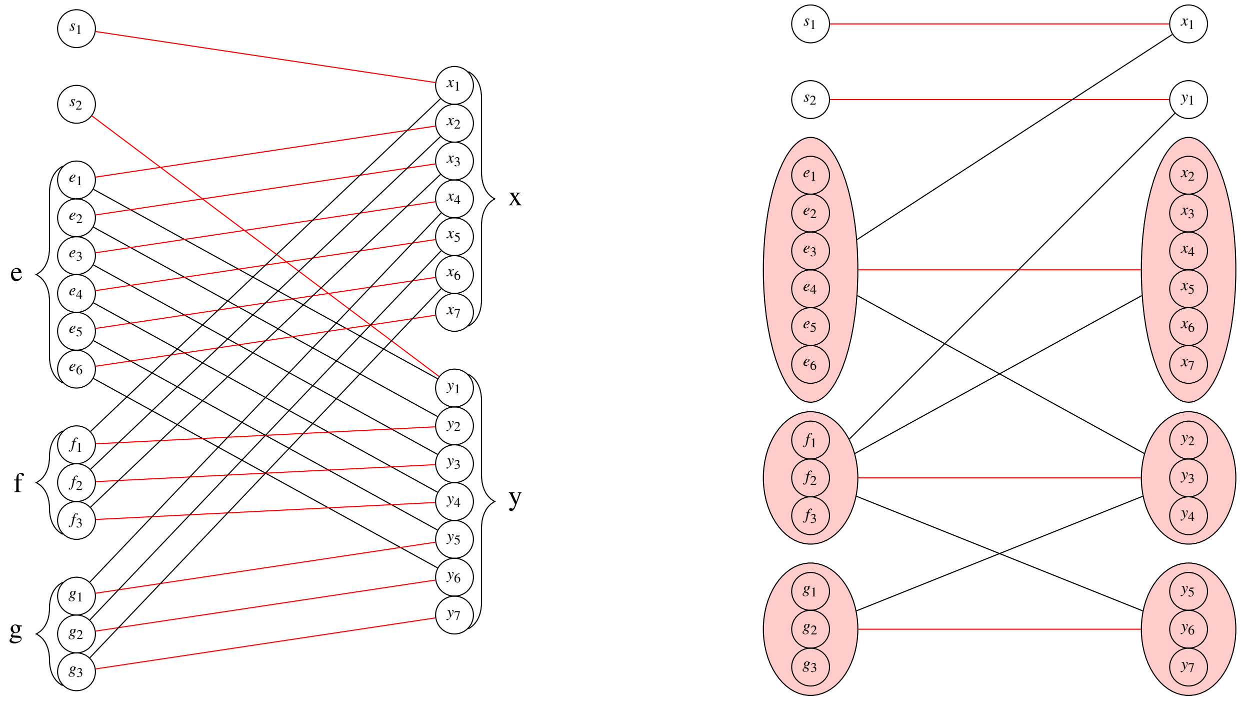
The expected results for this model are as follows:

- The first two scalar equations will be solved for $x[1]$ and $y[1]$
- The three for loops will be solved as follows:
  1. alternating between the first and the second for $i = 2:4$
  2. alternating between the first and the third for $i = 5:7$

Although this example seems more intricate, the same general solution as presented in example 5.1 can be used. The three for loops are accumulated to an entwined for-equation and its full list of alternating scalar equation indices are iterated while applying algorithm 3.

The following (simplified) code is generated by OpenModelica for the `entwined_loops` model. The body equations `genericCall_X` are similar to the one provided in example 5.1 and the alternating call order is represented by the array `call_order`.

```
void entwined_loops_eq_4(DATA *data)
{
  int call_indices[3] = {0,0,0};
  const int call_order[12] =
      {2,1,2,1,2,1,2,0,2,0,2,0};
  const int idx_lst_2[6] = {0,1,2,3,4,5};
  const int idx_lst_1[3] = {0,1,2};
  const int idx_lst_0[3] = {0,1,2};
  for(int i=0; i<12; i++)
  {
    switch(call_order[i])
    {
      case 2:
        genericCall_2(data, idx_lst_2[
            call_indices[0]]);
        call_indices[0]++;
        break;
      case 1:
        genericCall_1(data, idx_lst_1[
            call_indices[1]]);
        call_indices[1]++;
        break;
      case 0:
        genericCall_0(data, idx_lst_0[
            call_indices[2]]);
        call_indices[2]++;
        break;
      default:
        throwStreamPrint(NULL, "Call index
            %d at pos %d unknown for: ",
            call_order[i], i);
        break;
    }
  }
}
```



(a) Scalar-based digraph and matching.

(b) Array-based digraph and matching.

Figure 6. Causalization of the example model 5.2 requiring entwining.

6 Performance Test Results

The algorithms discussed in the previous Sections have been implemented in the new backend of the OpenModelica compiler. Their input is the result of the flattening process of an object-oriented Modelica model, where variable arrays and for-loop equations are *not* expanded into their scalar constituents. This output can be obtained from the new OpenModelica frontend (Pop et al. 2019), which preserves arrays during the flattening process, by skipping the final scalarization phase.

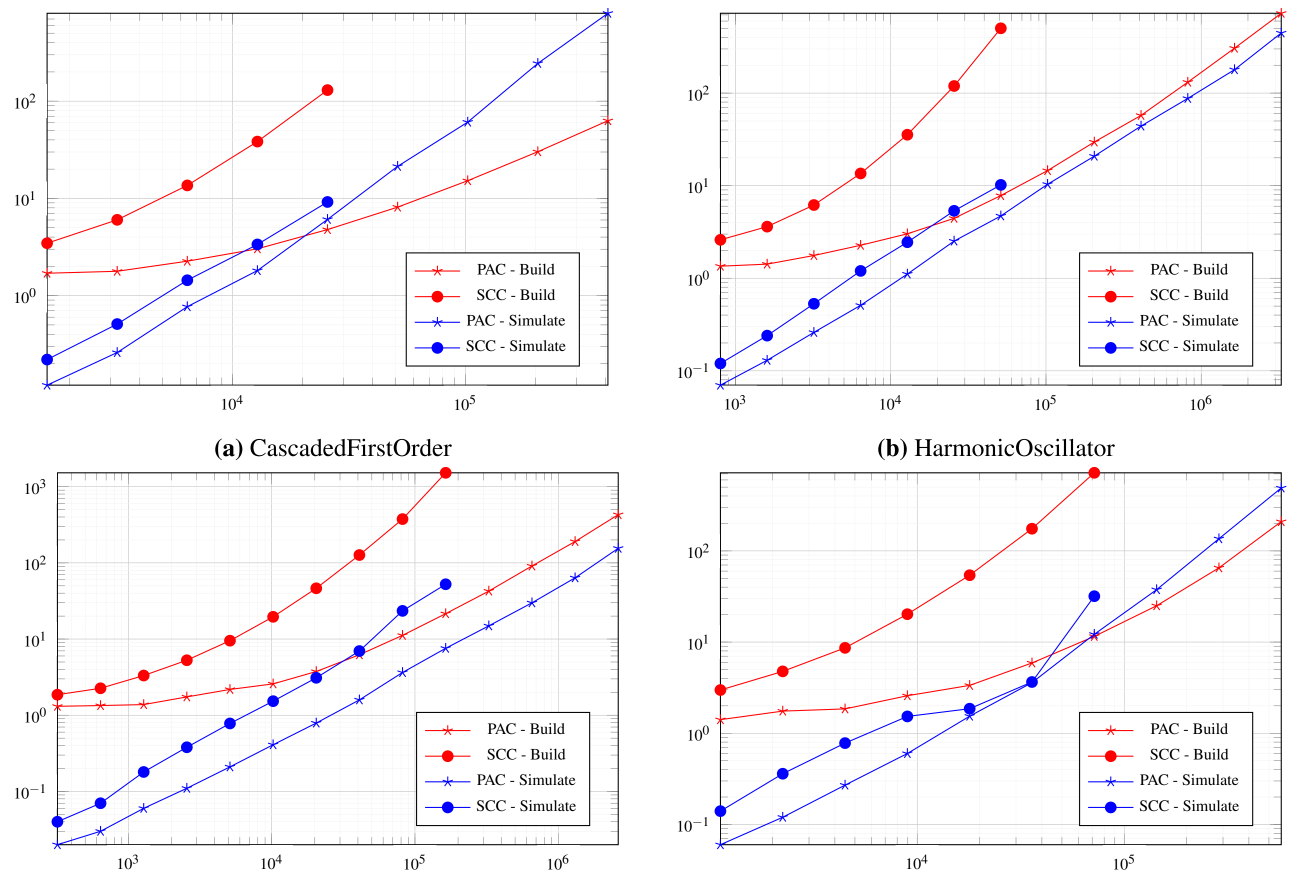
This Section reports the results obtained with large instances of some models of the ScalableTestSuite (Casella 2015), that can be run with the currently available implementation.

The tests were run on an AMD Ryzen 9 5950X 16-Core Processor, 63 GB RAM, running Ubuntu 22.04.2 LTS. The tests are run one at a time, so they can exploit parallelism on the 16 cores for garbage collection, code generation and C compilation. Also, simulations run at full speed, because they are not hindered by other processes competing for DMA channels. All simulations use variable step-size algorithms with error control.

The following models were run:

- *CascadedFirstOrder*: the model is a cascaded connection of N first-order linear systems, approximating a delay line. The response to a smooth increase of the system input is simulated with the sparse stiff solver IDA.
- *HarmonicOscillator*: the model describes the sequential connection of N masses with $N-1$ springs. It has $2N$ state variables and equations, where the initial position of the first mass is set off the equilibrium value, which initiates the propagation of an elastic wave through the system. As the system is only moderately stiff, the transient is simulated using the explicit DOPRI45 Runge-Kutta solver, whose execution time scales more favorably with system size.
- *OneDHeatTransferTT_FD*: this model contains the finite-volume discretization of 1D Fourier's equation, describing heat conduction in a rod, with N volumes and prescribed temperatures at the two ends. It has N states and about $2N$ equations. We simulate a transient with the sparse IDA solver, increasing the two boundary temperatures and observing how that change propagates through the length of the rod.
- *CounterCurrentHeatExchangerEquations*: this model contains a finite-volume discretization of a 1D counter-current heat exchanger model, considering the thermal inertia of the primary fluid, secondary fluid, and separating wall. Fluids are assumed to be incompressible and with constant specific heat capacity. The model has $3N$ states and $7N$ equations. We simulate a transient where we apply a step increase of the inlet temperature of one of the two fluids, using the sparse IDA solver.

Results are shown in Figure 7. On the x-axis, the number of model equations is shown; on the y-axis, build time (in red) and simulation time (in blue) are shown, compar-



(a) CascadedFirstOrder

(b) HarmonicOscillator

(c) OneDHeatTransferTT_FD

(d) CounterCurrentHeatExchangerEquations

Figure 7. Comparison of Pseudo Array Causalization (PAC) to Scalarized Causalization (SCC) on a logarithmic scale. The x-axis shows the number of equations and the y-axis shows the time spent in seconds. The largest tests could not be run with SCC methods on the provided machine, due to timeout or memory overflow.

ing the pseudo-array-causalization algorithm (PAC) with the scalar causalization algorithm (SCC).

The first interesting result is that the build time with PAC is nearly constant below 1,000 equations, and only starts growing linearly for substantially large sizes, where the time spent for the scalarized causalization dominates all other phases of code generation and compilation. As a consequence, the build time is one/two orders of magnitude smaller with PAC than with SCC, with increasing advantage as the size of the model grows. In fact, PAC allows to build the code of models with size exceeding one million equations, which are simply not manageable with the SCC algorithm. The time is saved mostly by avoiding to re-run code optimizations (e.g. alias elimination or CSE) on each instance of array equation, as well as avoiding to compile huge amounts of nearly identical C-code.

The second interesting result is that simulation time is also appreciably lower with PAC, though by a constant factor of about two-three. This is probably due to the execution of for-loops in the simulation code being more efficient than the execution of similar lines of code.

Last, but not least, we observe that build time, which used to be much larger than simulation time with SCC, is now comparable or possibly shorter than simulation time. This is a key usability improvement in the model development process, which is characterized by an iterative build-simulate-analyze-modify workflow.

These results were obtained with simple models written by array variables and for-loop equations. However, large number of components of the same type can be collected into arrays, eventually leading to the same kind of array-based structure once flattened, provided that arrays are preserved during flattening.

Hence, we can claim that the PAC algorithm unlocks the possibility of handling Modelica models in the million-equations range, characterized by large arrays of variables and/or components, which was not previously practically possible with state-of-the-art Modelica tools.

7 Conclusions

In this paper, a new Pseudo-Array-Causalization (PAC) algorithm was presented. When dealing with equation-based models using arrays of variables and equations, PAC allows to first carry out the causalization process on the fully flattened bipartite graph, as it is currently done in Modelica tools, without ever scalarizing them symbolically. This allows to generate much more compact simulation code that exploits for-loops, and to do so much faster.

The presented algorithm was demonstrated in a few simple cases on the paper, but also successfully implemented and tested in the OpenModelica compiler. Early results obtained on simple but large-sized models from the ScalableTestSuite show improvements of one/two orders of magnitude in the simulation code build time, and of a factor two/three in the simulation run time, thus unlocking the possibility of simulating models with over a million equations within reasonable amounts of time.

Future work includes improving the implementation so it can be tested in realistic use cases (e.g., large transmission or distribution power system models), and most importantly handling static and dynamic index reduction.

Furthermore, generating large integer lists for the generic for-equations might become a bottle neck in the future, therefore sequence compression methods will be used to reduce the used disk space and access time.

References

Braun, Willi et al. (n.d.). “Fast Simulation of Fluid Models with Colored Jacobians”. In: *Proceedings of the 9th International Modelica Conference*. DOI: 10.3384/ecp12076247. PDF.

Casella, Francesco (2015-09). “Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives”. In: *Proceedings of the 11th International Modelica Conference*, pp. 459–468. DOI: 10.3384/ecp15118459.

Casella, Francesco, Alberto Leva, and Andrea Bartolini (2017). “Simulation of Large Grids in OpenModelica: reflections and perspectives”. In: *Proceedings of the 12th International Modelica Conference*.

Duff, Iain S., Kamer Kaya, and Bora Uçcar (2012). “Design, Implementation, and Analysis of Maximum Transversal Algorithms”. In: *ACM Trans. Math. Softw.* 38.2. ISSN: 0098-3500. DOI: 10.1145/2049673.2049677. URL: https://doi.org/10.1145/2049673.2049677.

Ford, L. R. and D. R. Fulkerson (1956). “Maximal Flow Through a Network”. In: *Canadian Journal of Mathematics* 8, pp. 399–404. DOI: 10.4153/CJM-1956-045-5.

Fritzson, Peter et al. (2020-10). “The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development”. In: *Modeling, Identification and Control: A Norwegian Research Bulletin* 41, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Henningsson, Erik, Hans Olsson, and Luigi Vanfretti (2019-02). “DAE Solvers for Large-Scale Hybrid Models”. In: pp. 491–502. DOI: 10.3384/ecp19157491.

Kaya, Kamer et al. (2011-01). “Experiments on Push-Relabel-based Maximum Cardinality Matching Algorithms for Bipartite Graphs”. In: *Technical Report TR/PA/11/33, CER-FACS*.

Kofránek, Jiří et al. (2010). “Modelica – a language for integrative and system physiology modelling”. In: *Institute of Pathophysiology, First Faculty of Medicine, Charles University, Prague*.

Mattheij, R. and J. Molenaar (2002). *Ordinary Differential Equations in Theory and Practice*. Society for Industrial and Applied Mathematics. URL: https://epubs.siam.org/doi/pdf/10.1137/1.9780898719178.

Neumayr, Andrea and Martin Otter (2023). “Modelling and Simulation of Physical Systems with Dynamically Changing Degrees of Freedom”. In: *Electronics* 12.3. ISSN: 2079-9292. DOI: 10.3390/electronics12030500. URL: https://www.mdpi.com/2079-9292/12/3/500.

Otter, Martin and Hilding Elmqvist (2017). “Transformation of Differential Algebraic Array Equations to Index One Form”. In: *Proceedings of the 11th International Modelica Conference*.

Penrose, R. (1955). “A generalized inverse for matrices”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 51.3, pp. 406–413. DOI: 10.1017/S0305004100030401.

Pop, Adrian et al. (2019-02). “A New OpenModelica Compiler High Performance Frontend”. In: *Proceedings of the 13th International Modelica Conference*, pp. 689–698. DOI: 10.3384/ecp19157689.

Proß, Sabrina and Bernhard Bachmann (2011). “An Advanced Environment for Hybrid Modeling of Biological Systems Based on Modelica”. In: *J. Integrative Bioinformatics* 8.1, pp. 1–34. DOI: 10.2390/biecoll-jib-2011-152. URL: http://dx.doi.org/10.2390/biecoll-jib-2011-152. PDF.

Qi, Le (2014). “Modelica Driven Power System: Modeling, Simulation and Validation”. MA thesis. KTH Institute of Technology.

Tarjan, Robert (1972). “Depth-First Search and Linear Graph Algorithms”. In: *SIAM Journal on Computing* 1.2, pp. 146–160. DOI: 10.1137/0201010.

Viruez, Raul et al. (2017). “A Tool to ease Modelica-based Dynamic Power System Simulations”. In: *Proceedings of the 12th International Modelica Conference*.

Zimmermann, Pablo, Joaquín Fernández, and Ernesto Kofman (2020). “Set-Based Graph Methods for Fast Equation Sorting in Large DAE Systems”. In: *Proceedings of the 9th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. EOOLT ’19. Berlin, Germany: Association for Computing Machinery, pp. 45–54. ISBN: 9781450377133. DOI: 10.1145/3365984.3365991. URL: https://doi.org/10.1145/3365984.3365991.

Understanding and Improving Model Performance at Small Mass Flow Rates in Fluid System Models

Robert Flesch[1] Annika Kuhlmann[1] Johannes Brunnemann[1] Jörg Eiden[1]

[1]XRG Simulation GmbH, Germany, {flesch,kuhlmann,brunnemann,eiden}@xrg-simulation.de

Abstract

This paper provides a detailed analysis of the reasons behind the poor simulation performance observed when mass flow rates become very small, commonly referred to as zero mass flow issues. By using simple example models, we effectively demonstrate the underlying causes of these simulation performance issues. We highlight various contributing factors that play a significant role in exacerbating the problem.

Furthermore, we propose and examine countermeasures to mitigate these challenges. These countermeasures include modifications to the model itself, utilization of available settings in simulation tools, and adjustments to the solver. By implementing and evaluating these countermeasures, we illustrate their impact on improving simulation performance in scenarios involving low mass flow rates.

Keywords: zero mass flow issue, fluid dynamics, ODE integration, non-linear modeling

1 Introduction

Modelica is a great option to easily create models to simulate complex fluid systems. The created models can be simulated in one of the available tools. Thanks to advancements in algorithms and computational power, it is now possible to model these intricate systems within a short period of time.However, there is a challenge when it comes to systems that contain branches with no flow during certain periods or when the model is used to simulate ramp-up or shut-down sequences. In such cases, simulation time can drastically increase, resulting in what is commonly known as zero mass flow problems. From our experience in supporting several customers in creating and simulating models using different libraries, this issue causes large problems. Nevertheless, there is limited literature available on this subject. Dermont et al. (2016) presented an analysis of measures to improve robustness of models used to simulate air condition cycles. The demonstrated the impact of different measures including regularization of the mass flow pressure correlation at low mass flow rates, heat transfer modeling and choice of solver. The study (Li et al. 2020) highlighted the impact of an accurate and fast calculation of the system Jacobian matrix as part of the solution process. Qiao and Laughman (2022) analyzed the impact of different measures to improve performance of air conditioning models at low mass flow rates. They came up with a new regulation scheme for the pressure drop mass flow correlation around zero mass flows and an analysis of heat transfer handling methods.

In our opinion these article only scratch the surface of the numerical reasons which cause the slow simulations at zero mass flow rates. We strongly believe that gaining a deeper understanding of this phenomenon is crucial for developing effective strategies to improve simulation time in scenarios involving low mass flow rates. Therefore, the primary objective of this paper is to elucidate the underlying causes behind the sluggish performance observed during simulations with low mass flow rates.

To achieve this goal, we begin by examining and analyzing the solution process of a highly simplified fluid dynamic model. This analysis serves as a starting point to unveil the root causes of zero mass flow issues. Furthermore, we expand this model to incorporate additional complexities, thereby showcasing the impact of increased intricacy on simulation performance. By employing all these models, we illustrate the application of various countermeasures aimed at mitigating the challenges posed by zero mass flow problems.

2 Analysis for simple model

For demonstration of the underlying phenomenon we use a simple model of an isothermal ideal gas in a volume with the variable pressure p connected over a flow resistance to a boundary with fixed pressure p_b

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \frac{V}{R \cdot T}\frac{\mathrm{d}p}{\mathrm{d}t} = m_\mathrm{flow} \,. \tag{1}$$

When including a quadratic flow model with the parameter c but neglecting the dynamics of the momentum flow as often done for gas flows

$$m_\mathrm{flow} = c \cdot \mathrm{sign}(p_\mathrm{b} - p) \cdot \sqrt{|p_\mathrm{b} - p|} \tag{2}$$

we can derive the single ordinary differential equation

$$\frac{V}{R \cdot T} \cdot \frac{\mathrm{d}p}{\mathrm{d}t} = c \cdot \mathrm{sign}(p_\mathrm{b} - p) \cdot \sqrt{|p_\mathrm{b} - p|} \tag{3}$$

$$\frac{\mathrm{d}p}{\mathrm{d}t} = \frac{1}{\tau} \cdot \mathrm{sign}(p_\mathrm{b} - p) \cdot \sqrt{|p_\mathrm{b} - p|} = \frac{1}{\tau} \cdot F(\Delta p) \,. \tag{4}$$

In order simplify the equation we introduced $\Delta p = p_\mathrm{b} - p$ and $\tau = \frac{V}{R \cdot T \cdot c}$. Equation (4) could be integrated using an implicit Euler scheme

$$\frac{p_{t_i} - p_{t_{i-1}}}{\Delta t} = \frac{1}{\tau} \cdot F(\Delta p_{t_i}) \tag{5}$$

$$R(p_{t_i}) = p_{t_i} - p_{t_{i-1}} - \frac{\Delta t}{\tau} \cdot F(\Delta p_{t_i}) = 0 \tag{6}$$

in which p_{t_i} denotes the solution at the current and $p_{t_{i-1}}$ the solution at the previous time step. Equation (6) is a nonlinear equation which has to be solved to determine the pressure at the current step t_i. We have brought the equation to residual form. So we have to find the root of the residual function $R(p)$. For the solution solvers usually apply a Newton method, in which the linearized equation is repeatedly solved

$$p_{t_i}^j = p_{t_i}^{j-1} - \left(R'\left(p_{t_i}^{j-1}\right)\right)^{-1} \cdot R(p_{t_i}^{j-1}). \tag{7}$$

Here the prime denotes the derivative and $p_{t_i}^j$ is the updated solution for pressure at time step t_i in iteration j. The iteration is performed starting with an initial guess $p_{t_i}^0$ for the solution. Solving the equation requires solving the linear system $R'\left(p_{t_i}^{j-1}\right) \cdot (p_{t_i}^{j-1} - p_{t_i}^j) = R\left(p_{t_i}^{j-1}\right)$. For a single equation this can be done without much effort, but for a system of differential equations, the derivative of the residual equation system becomes a matrix (which is closely related to the Jacobian matrix of the system function F). Calculation of the derivative/Jacobian matrix and calculation of a decomposed form for solution comes at high computational effort. Additionally, in many systems the matrix does not change too much while advancing in time. Therefore, solvers usually never update the derivative during the iterative solution process for a time step and use $(R'(p_{t_i}^j))^{-1} = (R'(p_{t_i}^0))^{-1}$ (known as Chord method (Kelley 1995)). Additionally, solvers try to use the same derivative $(R'(\tilde{p}))^{-1}$ for solving multiple time steps. With this assumption the following iteration scheme is used with a constant $R'(\tilde{p}))^{-1}$

$$p_{t_i}^j = \Phi(p_{t_i}^{j-1}) = p_{t_i}^{j-1} - (R'(\tilde{p}))^{-1} \cdot R(p_{t_i}^{j-1})). \tag{8}$$

Figure 1 visualizes an exemplary successful iteration for the example problem from equation (4). The slope of the linearized approximations (red lines) do not match the actual local slope of the residual function (blue curve). Nevertheless, the iteration schemes converges, but it has lost its quadratic convergence due to the constant derivative. But as shown in Figure 2 the iteration might fail. The iterations schemes drifts off from the actual root of the residual equation and finally circles around the solution. In the following we will analyze this phenomenon in more detail and demonstrate that the iteration with not-up-to-date derivative becomes more difficult to solve when mass flow rate becomes small. If the solution diverges, as shown in Figure 2 or if due to the slower convergence no solution is found within a given number of iterations the step will be rejected. As reaction the solver will request an update of the Jacobian matrix and repeat the solution process. If the solution still fails, it might be necessary (as we will show later on) to reduce the step size. Then a solution with the outlined method can only be achieved for very small time steps and if the Jacobian matrix is updated at every time step. In this setting the time step is not chosen to fulfill the chosen tolerance anymore, but to make the system solvable with the approximated Jacobian matrix from the initial guess. This slows down the solution process and is the cause of slow models performance what is named zero mass flow issues.

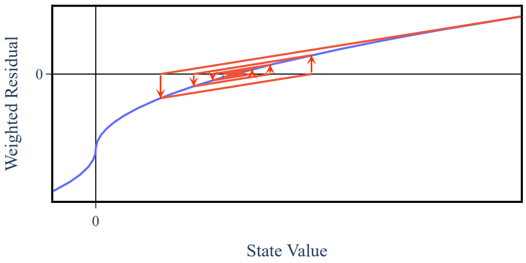


Figure 1. Example of a converging iterative solution for the example problem.

Let us take a closer look on the iterative solution process of Equation 4. The Banach fixed-point theorem states that an iteration scheme $\Phi(p_{t_i}^j)$ will converge if it is contractive. In that case a contraction factor $\lambda < 1$ exists and the iterative schemes fulfills

$$\left|\Phi(p_{t_i}^j) - \Phi(p_{t_i}^{j-1})\right| \leq \lambda \cdot \left|p_{t_i}^j - p_{t_i}^{j-1}\right|. \tag{9}$$

For the Chord method we can calculate the contraction factor with

$$\lambda = \frac{\left|\Phi(p_{t_i}^j) - \Phi(p_{t_i}^{j-1})\right|}{\left|p_{t_i}^j - p_{t_i}^{j-1}\right|} = \left|\frac{\Phi(p_{t_i}^j) - \Phi(p_{t_i}^{j-1})}{p_{t_i}^j - p_{t_i}^{j-1}}\right|. \tag{10}$$

By applying the mean value theorem there exists a $p_\xi \in (p_{t_i}^{j-1}, p_{t_i}^j)$ such that

$$\lambda = |\Phi'(p_\xi)| = |1 - R'(\tilde{p}))^{-1} \cdot R'(p_\xi)|. \tag{11}$$

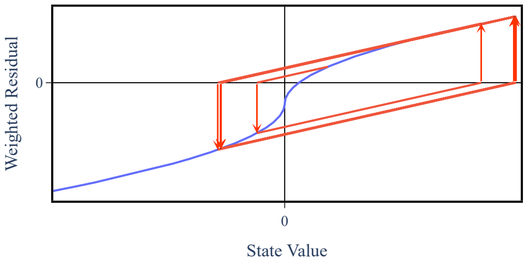


Figure 2. Example of a failing iterative solution attempt for the example problem.

For the second step we included Equation 8. As convergence requires $\lambda < 1$ we can obtain with Equation 11

$$0 < R'(\tilde{p}))^{-1} \cdot R'(p_\xi) < 2\,. \tag{12}$$

This criterion can be used for a more detailed analysis of the cause of zero mass flow issues and how it can be avoided, or its effects can be reduced.

But before applying this criterion to our demonstration problem Equation 1, we want to take a general look on the significance of Equation 12. For the reasons mentioned above a solver will try to reuse the derivative $R'(\tilde{p}))$ for multiple steps. But the actual derivative in the region of the solution $R'(p_\xi)$ might differ from the used derivative $R'(\tilde{p}))$ by a factor of two as demonstrated in Figure 2. In that case the iterative solution of the nonlinear system will fail. The solver will reject the step and repeat the solution with a reduced step size and an updated value for $R'(\tilde{p}))$. Unfortunately, the actual criterion for convergence is even stricter than that from Equation 12. If the condition is fulfilled, but with values close to two, the convergence is very slow. The integrator demand convergence within a few iteration, e.g. Hindmarsh et al. (2023), otherwise the step is rejected and the simulation performance decreases. But as the ratio $|R'(\tilde{p}))^{-1} \cdot R(p_{t_i}^j)|$ represent somehow the convergence speed of the method, we can use it for an analysis. Smaller values for $|R'(\tilde{p}))^{-1} \cdot R(p_{t_i}^j)|$ will lead to convergence within fewer steps. As convergence within few steps is demanded, a good initial guess is of great importance. The closer this initial guess is to the actual solution, the more likely is convergence of the method to the actual solution within the demanded solution tolerance. This has three consequences:

1. Methods which have a good approximation or forecast used for the initial guess will not be affected as much by the zero mass flow issue like method with no good forecast.

2. Reducing the time step improves the approximation of the initial guess. Therefore, reducing the step size has two positive aspects: it improves the initial guess and the ratio $|R'(p_{t_i}^0))^{-1} \cdot R(p_{t_i}^j)|$ will be closer to one. But obviously this comes as the price of a slower simulation.

3. The stricter the tolerance demanded for the solution, the harder it becomes to reach the tolerance within the desired number of iteration. So decreasing the tolerance might cause more problems at low mass flow rates.

After that general analysis, we will take a closer look on that criterion for our demonstration problem.

2.1 Analysis of Convergence for Demonstration Problem

If we apply Equation 12 to our simple problem in Equation 1, we get

$$\frac{1+\frac{\Delta t}{2\tau\sqrt{p_\mathrm{b}-p_\xi}}}{R'(\tilde{p})} < 2\,. \tag{13}$$

First of all, one can see that as $p \to p_\mathrm{b}$ results in $p \to p_\xi$ the method has only a chance to convergence for $\Delta t \to 0$. Therefore, in almost all libraries the square-root in the mass flow pressure relation is replaced by an approximation which has a finite derivative when crossing zero. For the regulation named regRoot from the Modelica Standard Library (*Modelica Standard Library 4.0.0* 2023) eq. 13 becomes

$$\frac{1+\frac{\Delta t}{\tau}\frac{0.5\cdot(p_\mathrm{b}-p_\xi)^2+\Delta p_\mathrm{small}{}^2}{((p_\mathrm{b}-p_\xi)^2+\Delta p_\mathrm{small}{}^2)^{1.25}}}{R'(\tilde{p})} < 2\,. \tag{14}$$

But still convergence for $p \to p_\mathrm{b}$ can become problematic for small values of τ and if Δp_small is not chosen large enough.The solver will find a solution within the demanded number of iteration but only for small time steps. Additionally, the simple demonstration model can be used to explain some general phenomena which can be seen in more complex models concerning if zero mass flow issues occur or not. Therefore, the simple model has been implemented as Modelica model and was solved with Dymola 2022x (Dassault Systemes AB 2023) with the regulated form of the square root mentioned above. Two different solvers from Dymola were used: Dassl and Radau. Dassl as a general purpose solver with good performance and Radau, as it was recommend by Dermont et al. (2016), when encountering zero mass flow problems. Figure 3 shows the number of Jacobian matrix evaluation when solving the problem while varying the pressure level of the boundary p_b. The pressure was initialized at the same level as the boundary. After one second the pressure in the boundary was increased or decreased by 10 Pa and the system was simulated for further 10 s to settle. One can see that after exceeding a certain threshold in pressure

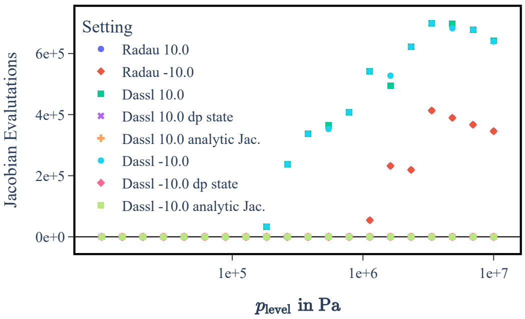


Figure 3. Number of Jacobian matrix evaluations for the example problem for different settings.

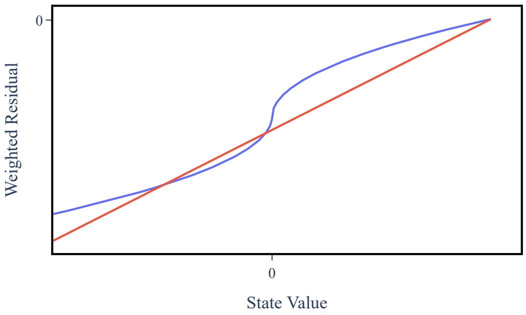


Figure 4. Demonstration of the approximate calculation of the derivative as secant (red) to the residual function (blue). If the two supporting points have different signs, the slope of the derivative approximation is much low than the slope in the region around $\dot{m} = 0$.

level the number of Jacobian matrix evaluation dramatically increases. This indicates presence of a zero mass flow problem as the integration steps are rejected, and a new Jacobian matrix must be calculated. This pressure level threshold depends on the chosen solver. Radau generally performs better in this case. But in general, it is surprising that the pressure level has an impact on the solution of the problem. The pressure level of p_b is only a constant offset for the solution and therefore on first look it should not have an impact. Additionally, if Dassl is chosen as solver, the number of Jacobian matrix evaluations depends on the sign of the pressure step. When Radau is used, the same number of Jacobian matrix evaluation is required independent of the sign of the pressure step.

Both phenomena can be explained if one focuses on how the derivative $R'(\tilde{p})$ is calculated in the solution process. If no special settings are applied the derivative is calculated numerically presumably with a given relative variation $\Delta p_\varepsilon = \varepsilon \cdot \tilde{p}$ with a small number ε. Typical methods are

$$R'(\tilde{p}) \approx \frac{R(\tilde{p}+\Delta p_\varepsilon) - R(\tilde{p})}{\Delta p_\varepsilon} \tag{15}$$

or

$$R'(\tilde{p}) \approx \frac{R(\tilde{p}+\Delta p_\varepsilon) - R(\tilde{p}-\Delta p_\varepsilon)}{2\cdot\Delta p_\varepsilon}\,. \tag{16}$$

Firstly, if the pressure level increases, the absolute variation for calculation of the derivative increases. But the physical meaning of a changed pressure difference has not changed. For a solution around the steady solution $\tilde{p} \approx p_\text{b}$ the variation Δp_ε used to approximate the derivative might flip the sign of $\tilde{p} - p_\text{b}$. In that case the absolute value of approximation of $R'(\tilde{p})$ becomes smaller than the absolute value of the actual derivative (see Figure 4). As Equation 13 shows this decreases the speed of convergence. Secondly, the problem itself is symmetric to an increasing or decreasing step of the pressure. But using the approximation Equation 15 to calculate the derivative introduces an asymmetry. As only the Dassl solver is susceptible to the sign of the variation it suggests itself that Dassl uses a forward differencing scheme while Radau uses a central difference scheme like Equation 16. We created an external external function which is called in the model to track the value of states during all models calls: by analyzing the state variation one can identify the times at which the Jacobian matrix is updated. When using Radau as solver the perturbation is applied in both directions while for Dassl only a perturbation in one direction is applied - therefore we can verify the assumption. The central scheme is is more accurate and symmetric, and therefore the results to not show a dependence on the sign of the pressure step.

There are two options to improve model performance for zero mass flow rate. If possible one could use an analytic calculation of the Jacobian matrix to get rid of the underestimation of the derivative. Additionally, or solely the problem could be reformulated to use $\Delta p = p - p_\text{b}$ as state. This measure improves the accuracy of the numeric approximation of the Jacobian matrix. Pressure differences are driving the mass flow rates and therefore the changes in pressure. If the pressure becomes large, the perturbation applied $\varepsilon \cdot \Delta p$ to calculated the derivative might be in the range of the driving pressure differences and the approximate Jacobian matrix is not accurately enough to solve the system with large steps. Choosing the pressure difference as state, leads to a perturbation much smaller used to calculate the derivative and the applied solution method is much more robust. In both cases the model performs much better as one can see in Figure 3. If one of the tweaks is applied, no issues occur even if pressure level is increased. Additionally, the number of evaluations is independent of the step sign.

2.2 Non-linearity as root of the problem

Furthermore, we want to use the simple model to emphasize the importance of the non-linearity of the problem. Dermont et al. (2016) gave the time constant of the problem which increases when decreasing the mass flow rate as reason for the poor model performance for small mass flow rates. We want to demonstrate that the time constant itself is not the root of the problem. For this test we run the model with Radau as solver and the pressure level of 1×10^6 Pa with a step of 10 Pa. For the second run the regRoot mass flow pressure relation was replaced with a linear relation which has the same slope for vanishing mass flow rates (Figure 5). Due to the linearity this slope remains the same for all mass flow rates. The slope of the mass flow pressure relation determines the time constant of the problem. The comparison is designed in a way that the time constant of the non-linear problem is at maximum the same as the time constant of the linear problem. Figure 6 shows the solver step when both models are simulated with the same settings. Though the linear problem should be tougher to solver from a simple perspective of the time constant, it is actually the other way round. A linear model cannot face zero mass flow

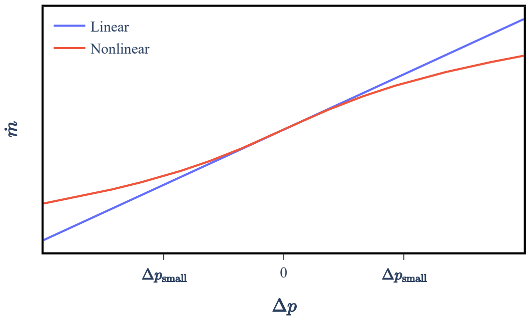


Figure 5. Linearized and actual non-linear mass flow pressure relation used in the example model to demonstrate the impact of the non-linearity.

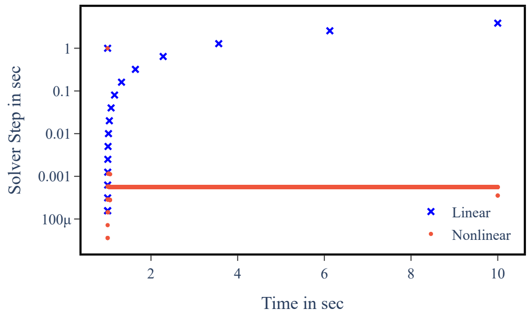


Figure 6. Solver step size for example model with linear and non-linear mass flow pressure relation as shown in Figure 5.

issues as the Jacobian matrix calculated at any position is valid for the whole domain. As a result the linear problem can be solved with much higher time steps, though it actually has the higher dynamics. Actually, the presented findings could have been derived from Equation 12. But the results are presented as they impressively show which solver steps are possible coming from the pure dynamic of the problem. The non-linearity of the problem which results in a change of the time constant, causes the Chord method to fail and as a result the possible time steps of the integration method cannot be exploited.

3 Extended model with transported scalar

Obviously, models are normally much more complex. One additional level of complexity are additional equations to be solved e.g. balances for balance, mass, composition etc. The form of these equations has a crucial impact on the resilience of the model against zero mass flow issues. In order to demonstrate and analyze this impact we extend our simple models with an additional equation

$$\frac{\mathrm{d}\psi}{\mathrm{d}t} = \begin{cases} \frac{m_{\text{flow}}}{m} \cdot (\psi_{\text{b}} - \psi), & \text{if } m_{\text{flow}} \geq 0 \\ 0, & \text{otherwise} \end{cases} . \tag{17}$$

Now we introduce the abbreviation $\Delta\psi = \psi_{\text{b}} - \psi$ and observe that $\text{sign}(m_{\text{flow}}) = \text{sign}(\Delta p)$ in order to use the Heaviside step function $\text{H}(\Delta p)$. Moreover from Equation 4 and Equation 2 we can replace $m_{\text{flow}} = c \cdot F(\Delta p)$ and $m = \frac{V}{R \cdot T} \cdot p$. Then

$$\frac{\mathrm{d}\psi}{\mathrm{d}t} = c^2 \tau \cdot \frac{G(\Delta p)}{p} \cdot \Delta\psi \tag{18}$$

where we have introduced $G(\Delta p) = \text{H}(\Delta p) \cdot F(\Delta p))$ and again use $\tau = \frac{V}{R \cdot T \cdot c}$.

The residual equation becomes the form

$$R(\psi_t) = \psi_t - \psi_{t-1} - \Delta t \cdot c^2 \tau \cdot \frac{G(\Delta p)}{p_t} \cdot \Delta\psi_t \tag{19}$$

The solution of equation $R(\psi_t) = 0$ depends on $G(\Delta p) = \text{H}(\Delta p) \cdot F(\Delta p)$ and hence on the flow direction through $\text{H}(\Delta p)$ and on the actual mass flow rate through $F(\Delta p)$. It is given by

$$\psi_t = \frac{\psi_{t-1} + \Delta t \cdot c^2 \tau \cdot \frac{G(\Delta p_t)}{p_t} \psi_{\text{b}}}{1 + \Delta t \cdot c^2 \tau \cdot \frac{G(\Delta p_t)}{p_t}} \tag{20}$$

Though in our model it would be possible to solve the equation for pressure and the equation for the passive scalar sequentially, in a Modelica tool these equations are solved simultaneously. So Equation 17 is added to the model from the previous section and solved again while tracking the number of Jacobian matrix evaluations as it was plotted in Figure 3. This time only Radau and a step of 10 Pa was used. The results are show in Figure 7. For reference purposes the results for only the pressure equations are shown additionally. Adding the equation for the transported scalar causes the zero mass flow issue to occur at lower pressure levels and from that pressure level additional Jacobian matrix evaluations are caused compared to the reference case. When integrating the solution of Equation 17 depends on the actual mass flow rate. Therefore, the iteration can only converge after the Equation 1 has converged reasonably close to the actual solution. If Equation 1 alone converges only just within the desired number of iteration, the system will not converge, causing the zero mass flow issue occurring at lower pressure levels. Figure 8 shows an exemplary course of the residuals

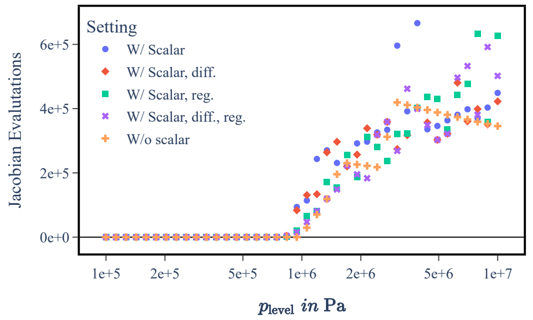


Figure 7. Number of Jacobian matrix evaluations for example model with (w/) and without (w/o) transported scalar and different settings.

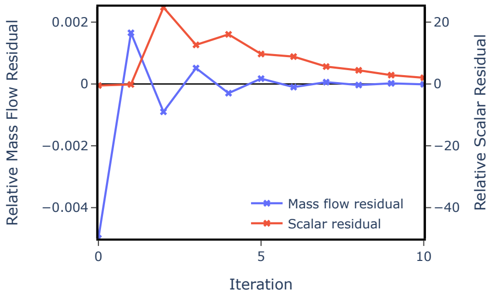


Figure 8. Residuals of pressure and transported scalar during iterative solution.

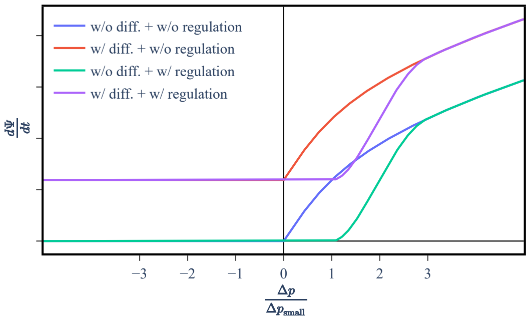


Figure 9. Transported scalar flow rate for different modeling approaches: all combinations of with (w/) and without (w/o) diffusion and convection with (w/) and without (w/o) regulation.

of the two equations over the iterations. One can clearly see that the residual of the scalar (Equation 17) even increases before it starts to converge. But convergence rate is slow, while the mass flow rates oscillates around its final value. Figure 7 contains measure which are intended to improve convergence around zero mass flow rate. One idea is to introduce diffusion into Equation 17 by adding a transport term which is independent of the mass flow rate

$$\frac{\mathrm{d}\psi}{\mathrm{d}t} = \left(c^2\tau\cdot\frac{G(\Delta p)}{p}+\mu\right)\cdot\Delta\psi \tag{21}$$

We use a artificial diffusion constant μ.

The second idea is to introduce a (nonphysical) regulation of the convective transport as depicted in Figure 9. The convective transport with is set to be zero if the mass flow is regulated. After the mass flow has left the regulation the convective flow is ramp up to its actual value.

From Figure 7 we can see that the pure diffusion does not improve the system convergence around zero. No improvement was observed for reasonable big values of the artificial diffusion constant μ. The problem is that the solution for ψ still depends on the mass flow rate, though it is shifted. Nevertheless, the scalar converges only after the mass flow has converged. For an improvement the solution should be independent of the mass flow rate for very small mass flow rates. That is the aim of the second measure, with the additional regulation of the convective flow. This method has to be implemented carefully to avoid a violation of conservation. When this measure is applied the system is more robust against zero mass flow issues, but it still performs worse than the model with only mass conservation.

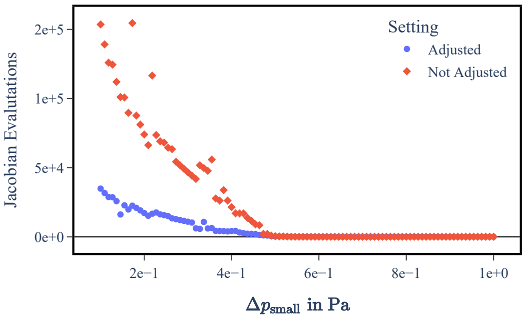


Figure 10. Number of Jacobian matrix evaluations for a multi-volume model with and without adjusted linearization interval.

4 Extended model with multiple volumes

In this step we want to extend the model from section 2 to a model with multiple connected control volumes.

4.1 Pressures

For the pressures we obtain

$$\frac{\mathrm{d}p[k]}{\mathrm{d}t} = \frac{1}{\tau[k]}\cdot\Big(F(\Delta p[k]) - F(\Delta p[k+1])\Big) \tag{22}$$

where we have extended the notation from Equation 4 to the control volume label $k = 1 \ldots N$ with $\Delta p[1] = p_b - p[1]$, $\Delta p[k] = p[k-1] - p[k]$, $\Delta p[N+1] = 0$ and control volume dependent time constant $\tau[k] = \frac{V[k]}{R\cdot T\cdot c}$.

Following the procedure in Equation 14, we now approximate

$$\begin{aligned} F(\Delta p[k]) &= \mathrm{sign}(\Delta p[k])\cdot\sqrt{|\Delta p[k]|} \\ &\approx \mathrm{regRoot}(\Delta p[k], \Delta p_{\mathrm{small}}[k]) \end{aligned} \tag{23}$$

In a numerical experiment with $n = 9$ the control volume $V[k]$ was chosen uniformly $V[k] = V$ except for volume five, in which it was chosen to be a tenth of the other values $V[5] = 0.1\cdot V$, such that $\tau[5] = \tau/10$. The other parameter varied is $\Delta p_{\mathrm{small}}[k]$. Two strategies are applied: in the first strategy all regularization intervals have the same size and in the second strategy all have the same size except for that of $k = 5$. The value is chosen to be $\Delta p_{\mathrm{small}}[5] = 10\cdot\Delta p_{\mathrm{small}}$. The results are given in Figure 10. It is important to mention that the given level of $\Delta p_{\mathrm{small}}$ corresponds to the value applied for most of the equations. The first important thing which one can see is that increasing the regularization interval will improve the performance until a certain threshold. After that one could say that the zero mass flow problem is not present anymore. But the increase in Δp_{small} comes at a certain price, the pressure drop at low mass flow rates will be computed incorrectly. If this is an issue, the second strategy might be an interesting option. It is only required to apply a larger linearization for flow models which are connected to pressure state with the large values of τ. So the overall accuracy of the model will be better.

4.2 Passive Scalars

Using the same notation as in the previous section we can extend Equation 18 for the passive scalar to obtain

$$\begin{aligned}\frac{\mathrm{d}\psi[k]}{\mathrm{d}t} &= \left(c^2\tau[k]\cdot\frac{G(\Delta p[k])}{p[k]}+\mu\right)\cdot\Delta\psi[k] \\ &\quad -\left(c^2\tau[k]\cdot\frac{\widetilde{G}(\Delta p[k+1])}{p[k]}+\mu\right)\cdot\Delta\psi[k+1]\end{aligned} \tag{24}$$

with $\Delta\psi[1]=\psi_b-\psi[1]$ and $\Delta\psi[k]=\psi[k-1]-\psi[k]$ and $\widetilde{G}(\Delta p)=G(-\Delta p)=\big(H(\Delta p)-1\big)\cdot F(\Delta p)$, where we have used the symmetry properties of the Heaviside step function H and the function F due to Equation 4. Finally we approximate

$$\mathrm{H}(\Delta p)\approx \mathrm{SM}(\Delta p,\mathrm{func},\mathrm{nofunc}) \tag{25}$$

with the `stepsmoother` function SM(.) from `Modelica.Fluid.Dissipation.Utilities...` as regularization for the Heaviside step function. Here the difference $|\mathrm{func}-\mathrm{nofunc}|$ denotes the width of the rgulated step.

4.3 Generalized convergence criterium

In this section we would like to extend the computation of λ in Equation 11 and the resulting convergence criterium in Equation 12 to general state space systems with N states $(x[1],\cdots,x[N])=:\vec{x}$. For a given time step t_i we have

$$\frac{d}{dt}\vec{x}_{t_i}=\vec{f}(\vec{x}_{t_i}) \tag{26}$$

and for an implicit integration algorithm it holds that

$$\vec{x}_{t_i}=\vec{x}_{t_{i-1}}+\Delta t\cdot\vec{f}(\vec{x}_{t_i}) \tag{27}$$

with the non linear residuum equation

$$\vec{R}(\vec{x}_{t_i})\overset{!}{=}\vec{0}=\vec{x}_{t_i}-\vec{x}_{t_{i-1}}-\Delta t\cdot\vec{f}(\vec{x}_{t_i}) \tag{28}$$

which can be solved iteratively by a Newton-Raphson scheme

$$\vec{x}^{j+1}_{t_i}=\vec{\Phi}(\vec{x}^{j}_{t_i})=\vec{x}^{j}_{t_i}-\left[\mathrm{J}_{\vec{R}}(\vec{x}^{j}_{t_i})\right]^{-1}\vec{R}(\vec{x}^{j}_{t_i})\,. \tag{29}$$

Here $\mathrm{J}_{\vec{R}}$ denotes the Jacobian of the residuum vector $\vec{R}(\vec{x}^{j}_{t_i})$. In analogy to Equation 8 the Chord method attempts to solve Equation 29 with the inverse Jacobian fixed at some state $\widetilde{\vec{x}}$:

$$\vec{x}^{j+1}_{t_i}=\vec{\Phi}(\vec{x}^{j}_{t_i})=\vec{x}^{j}_{t_i}-\left[\mathrm{J}_{\vec{R}}(\widetilde{\vec{x}})\right]^{-1}\vec{R}(\vec{x}^{j}_{t_i})\,. \tag{30}$$

Now in analogy to Equation 9 this iteration converges if it is contractive, that is for some $\lambda<1$ it holds that

$$\left\|\vec{\Phi}(\vec{x}^{j}_{t_i})-\vec{\Phi}(\vec{x}^{j-1}_{t_i})\right\|\le\lambda\cdot\left\|\vec{x}^{j}_{t_i}-\vec{x}^{j-1}_{t_i}\right\| \tag{31}$$

In order to further develop this expression, notice that we re-write the difference on the left as the result of an integration along a straight line $\vec{x}(s)=\vec{x}^{j-1}_{t_i}+s\cdot(\vec{x}^{j}_{t_i}-\vec{x}^{j-1}_{t_i})$ in state space with curve parameter $s\in[0,1]$:

$$\vec{\Phi}(\vec{x}^{j}_{t_i})-\vec{\Phi}(\vec{x}^{j-1}_{t_i})=\int\limits_0^1 ds\left\{\mathrm{J}_{\vec{\Phi}}\big(\vec{x}(s)\big)\cdot\frac{\mathrm{d}\vec{x}(s)}{\mathrm{d}s}\right\}\quad, \tag{32}$$

where $\mathrm{J}_{\vec{\Phi}}\big(\vec{x}(s)\big)$ denotes the Jacobian of $\vec{\Phi}$ at position $\vec{x}(s)$. Now clearly the tangent vector $\mathrm{d}\vec{x}(s)/\mathrm{d}s$ is constant along the line and given by

$$\frac{\mathrm{d}\vec{x}(s)}{\mathrm{d}s}=\vec{x}^{j}_{t_i}-\vec{x}^{j-1}_{t_i}\quad. \tag{33}$$

Moreover from Equation 30 it follows that

$$\mathrm{J}_{\vec{\Phi}}\big(\vec{x}(s)\big)=\mathbb{1}-\left[\mathrm{J}_{\vec{R}}(\widetilde{\vec{x}})\right]^{-1}\mathrm{J}_{\vec{R}}\big(\vec{x}(s)\big) \tag{34}$$

with $\mathbb{1}$ denoting the N-dimensional identity matrix. So we can formally re-write Equation 32 as

$$\vec{\Phi}(\vec{x}^{j}_{t_i})-\vec{\Phi}(\vec{x}^{j-1}_{t_i})=A\cdot\left(\vec{x}^{j}_{t_i}-\vec{x}^{j-1}_{t_i}\right) \tag{35}$$

with the matrix A given by

$$A=\int\limits_0^1 ds\left\{\mathbb{1}-\left[\mathrm{J}_{\vec{R}}(\widetilde{\vec{x}})\right]^{-1}\mathrm{J}_{\vec{R}}\big(\vec{x}(s)\big)\right\} \tag{36}$$

With these ingredients we can replace Equation 31 similarly to Equation 11 by

$$\|A\|\le\lambda<1 \tag{37}$$

Here $\|A\|=\|A\|_2$ denotes the spectral norm of the matrix A as induced from the L^2 vector norm $\|\vec{x}\|$. The spectral norm is defined as the square root of the largest eigenvalue of A^TA, with A^T the conjugate transpose of A. We can further estimate $\|A\|$ as follows

$$\begin{aligned}\|A\| &= \left\|\int_0^1 ds\left\{\mathbb{1}-\left[\mathrm{J}_{\vec{R}}(\widetilde{\vec{x}})\right]^{-1}\mathrm{J}_{\vec{R}}\big(\vec{x}(s)\big)\right\}\right\| \\ &\le \max_{s\in[0,1]}\left\|\mathbb{1}-\left[\mathrm{J}_{\vec{R}}(\widetilde{\vec{x}})\right]^{-1}\mathrm{J}_{\vec{R}}\big(\vec{x}(s)\big)\right\| \\ &\le \left\|\left[\mathrm{J}_{\vec{R}}(\widetilde{\vec{x}})\right]^{-1}\right\|\cdot\max_{s\in[0,1]}\left\|\mathrm{J}_{\vec{R}}(\widetilde{\vec{x}})-\mathrm{J}_{\vec{R}}\big(\vec{x}(s)\big)\right\| \\ &\overset{!}{<} 1\end{aligned} \tag{38}$$

Although Equation 38 holds for any matrix norm, for practical application in the context of Equation 31 one has to use the spectral norm $\|\mathrm{J}\|_2$.

4.4 Explicit application

The bound obtained in Equation 38 can be used in order to give conditions for convergence and also recommendations for the regularization parameters of the regRoot and SM functions contained in $F(\Delta p)$ and $G(\Delta p)$.

4.4.1 General case

The matrix elements $\mathrm{J}_{\vec{R}}(\vec{x})[I,J]$of the Jacobian $\mathrm{J}_{\vec{R}}(\vec{x})$ can be computed from Equation 28 as follows:

$$\mathrm{J}_{\vec{R}}(\vec{x})[I,J] = \frac{\partial R[I](\vec{x})}{\partial x[J]} = \delta[I,J] - \Delta t \cdot \frac{\partial f[I](\vec{x})}{\partial x[J]} \qquad (39)$$

Here $\delta[I,J]$ denotes the Kronecker delta. For the coupled pressure scalar system we have the time derivatives $f[I]$ given from Equation 22 and Equation 24. Let $(x[1],\ldots,x[N]) = (p[1],\ldots,p[N])$ and $(x[N+1],\ldots,x[2N]) = (\psi[1],\ldots,\psi[N])$. Then

$$\boxed{1 \le I \le N} \quad f[I] = f\big(x[I-1],x[I],x[I+1]\big)$$

$$\boxed{N < K \le 2N} \quad f[K] = f\big(x[K-1],x[K],x[K+1], x[K-N-1],x[K-N],x[K-N+1]\big)$$

Hence the resulting incidence matrix is tri-diagonal for the pressures and tri-diagonal for the scalars with an additional tri-band between pressures and scalars. This property may be used in order to arrive at an estimate to the spectral norms of Equation 38. An explicit computation of the eigenvalues can in principle be avoided by approximating the spectral norm by the general property:

$$\|A\|_2 \le \sqrt{\|A\|_1 \cdot \|A\|_\infty} \qquad (40)$$

where $\|A\|_1 = \max_j \sum_{i=1}^N |a_{ij}|$ is the maximum absolute column sum norm of A and $\|A\|_\infty = \max_i \sum_{j=1}^N |a_{ij}|$ is the the maximum absolute row sum norm of A. This also avoids computation of A^TA. An explicit symbolic analysis for the pressure system in the fashion of section 7 in J. Brunnemann (2008) will be subject to future work.

4.4.2 Single pressure

For the case of a single pressure $p[1]$ and boundary pressure p_b the Jacobian $\mathrm{J}_{\vec{R}}(\vec{x})$ has only one single element:

$$\mathrm{J}_{\vec{R}}(\vec{x})[1,1] \quad = \quad \frac{\partial R[1](\vec{x})}{\partial x[1]} =: R'(p) \qquad (41)$$

For simplicity of notation we have dropped the discretization index on the right hand side. Plugging this into Equation 38 we obtain

$$\max_{s\in[0,1]} \big\|1 - [R'(\widetilde{p})]^{-1} \cdot R'(p(s))\big\| < 1c \qquad (42)$$

In order to fulfill this inequality it must hold that

$$0 \le \frac{R'\big(p(s)\big)}{R'(\widetilde{p})} \le 2 \quad \forall s \in [0,1] \qquad (43)$$

This re-produces the result of Equation 12.

4.4.3 Single pressure and passive scalar

For the case of a single pressure $p[1]$ with boundary pressure p_b acoupled to a single passive scalar $\psi[1]$ with boundary ψ_b the Jacobian $\mathrm{J}_{\vec{R}}(\vec{x})$ is a $2x2$ matrix with three non-zero elements:

$$\begin{aligned} \mathrm{J}_{\vec{R}}(\vec{x})[1,1] &= \frac{\partial R[1](\vec{x})}{\partial x[1]} = \frac{\partial R[1](p)}{\partial p} =: a \\ \mathrm{J}_{\vec{R}}(\vec{x})[2,1] &= \frac{\partial R[2](\vec{x})}{\partial x[1]} = \frac{\partial R[2](p,\psi)}{\partial p} =: c \\ \mathrm{J}_{\vec{R}}(\vec{x})[2,2] &= \frac{\partial R[2](\vec{x})}{\partial x[2]} = \frac{\partial R[2](p,\psi)}{\partial \psi} =: d \end{aligned}$$

Here we have left out the discretization indices for p,ψ on the left hand side for simplicity of notation.

$$\mathrm{J}_{\vec{R}}\big(p(s),\psi(s)\big) = \begin{pmatrix} a & 0 \\ c & d \end{pmatrix} \text{ and } \mathrm{J}_{\vec{R}}\big(\widetilde{p},\widetilde{\psi}\big) = \begin{pmatrix} a_0 & 0 \\ c_0 & d_0 \end{pmatrix}$$

In the sequel we will suppress the (s)-dependence of (a,c,d) for simplicity of notation. The above setting implies

$$\begin{aligned} M &:= \mathbb{1} - \big[\mathrm{J}_{\vec{R}}(\widetilde{\vec{x}})\big]^{-1} \mathrm{J}_{\vec{R}}\big(\vec{x}(s)\big) \\ &= \begin{pmatrix} m_{11} & 0 \\ m_{21} & m_{22} \end{pmatrix} = \begin{pmatrix} -\frac{a}{a_0}+1 & 0 \\ -\frac{c}{d_0}+\frac{ac_0}{a_0d_0} & -\frac{d}{d_0}+1 \end{pmatrix} \end{aligned}$$

And finally

$$M^TM = \begin{pmatrix} {m_{11}}^2 & m_{11}m_{21} \\ m_{11}m_{21} & {m_{21}}^2 + {m_{22}}^2 \end{pmatrix} \qquad (44)$$

From the symmetry of M^TM it follows that the two eigenvalues λ_1,λ_2 are real. For the 2D case we can explicitly compute them. However one may also apply Gershgorins circle theorem (Gershgorin (1931)) $|\lambda_1 - {m_{11}}^2| \le |m_{11}m_{21}|$ and $|\lambda_2 - {m_{21}}^2 - {m_{22}}^2| \le |m_{11}m_{21}|$ for an upper bound of the eigenvalues:

$$\|A\|_2 \le \sqrt{\max_{s\in[0,1]} \big(|\lambda_1(s)|,|\lambda_2(s)|\big)} \overset{!}{<} 1 \qquad (45)$$

This is of particular use for higher dimensional cases of Equation 38.

5 Solver Modification

The adaptions shown which should lead to an improved simulation of the model where all on the model side. These modification can be applied by the simulation engineer or library developer and they work independently of the used tool. But these modifications cause deviations between model results and the physical correct or the expected results. These deviations might be tolerable in many cases, but in some cases they are not. Therefore, it would be interesting to have a solution process which completely avoids zero mass flow issues or reduces the impact.

Table 1. Performance key figures for the solution process for the presented models with and without damping strategy.

	Jacobian Evaluations		Function Calls		Integrator Steps	
	Damped	Normal	Damped	Normal	Damped	Normal
Simple Mass Flow (section 2)	7.1×10^1	2.0×10^4	7.0×10^3	1.4×10^5	2.1×10^3	3.4×10^4
Mass Flow + Scalar (section 3)	5.3×10^2	6.7×10^3	3.0×10^4	7.7×10^5	1.0×10^4	3.0×10^4
Multi volume (section 4)	2.2×10^1	1.1×10^4	2.7×10^3	1.3×10^5	1.1×10^3	4.6×10^4
BranchingDynamicPipes	7.3×10^1	7.1×10^1	4.3×10^3	4.3×10^3	3.4×10^3	3.4×10^3
PID	1.9×10^1	1.9×10^1	9.5×10^2	9.5×10^2	8.4×10^2	8.4×10^2
BatchPlant_StandardWater	1.7×10^2	1.9×10^2	6.8×10^3	6.7×10^3	5.1×10^3	5.2×10^3

Overshooting during the iteration process is a known problem even for normal Newton methods, if the starting point is far off from the actual solution and/or if the equation to solve is highly nonlinear. In that case damping of the solution can reduce the required number of iteration or even avoid divergence of the method. As shown above overshooting is as well the problem which causes the zero mass flow issue. Therefore, we tried to apply a damping strategy to the solution process. For this test we used OpenModelica and the Cvode solver as for both the full code is available and can easily be modified. The algorithm is taken from Dahmen and Reusken (2008) and modified to be usable for the Chord method. For a general residual equation

$$R(x_{t_i}) = 0 \tag{46}$$

a Chord step is done to calculate an update the solution for all states

$$\Delta x_{t_i}^{j} = -(R'(\tilde{x}))^{-1} \cdot R(x_{t_i}^{j-1})\,. \tag{47}$$

The update is not applied directly. Instead the condition

$$\begin{aligned} &||(R'(\tilde{x}))^{-1} \cdot R\left(x_{t_i}^{j-1} + \lambda \cdot \Delta x_{t_i}^{j}\right)|| \\ &\quad \leq C_\lambda \cdot ||(R'(\tilde{x}))^{-1} \cdot R(x_{t_i}^{j-1})||\,, \end{aligned} \tag{48}$$

is checked until it is fulfilled with the series $\lambda = 1, 0.5, 0.25, ...$ using $C_\lambda = 1 - \frac{\lambda}{4}$. When the condition is fulfilled the step is applied and the same procedure is repeated for the next step $\Delta x_{t_i}^{j+1}$. The right hand side of Equation 48 is (a fraction) of the norm of the full step calculated from Equation 47. The left hand side is the next full step which is calculated if the current damped step is applied. Therefore, by using this method we damp the calculated step until the norm of the next step is smaller than the current. As the size of the step depends of the residual $R(x_{t_i}^{j}))$, we enforce the residual to reduce. Checking the condition comes with no relevant extra computational effort. The price of a recalculation after a rejected Chord update costs the same as a normal step. So if the rejected steps are included in the total number of Chord updates, one could design a method which comes at almost no extra costs.

The method was applied to the example problems described above in configuration in which the zero mass flow issue occurs. The performance of the damped strategy is compared to the normal algorithm. The results are summarized in Table 1. Damping the steps causes a significant reduction in Jacobian matrix evaluations and functions calls. The number of integrator time steps is reduced as well, though in case of the problem from section 3 the reduction of integrator steps is not a high as in the other. But function calls and the decomposition of the Jacobian matrix are the main expenses during integration. Therefore, using the damped solving approach significantly reduces the solution time in all the cases. Beside the some models from the Modelica Standard Library were simulated with both methods, though zero mass flow rate was not a particular issues for these models. The performance indicator numbers are very similar. The slight difference can result from differences in the solution and the fact that with the used settings damping steps were not counted in the total step count for an iteration. Therefore, the number of function with damping exceed the number of calls without damping though less steps were taken and less Jacobian updates were required. These minor deviation could be tolerated. All in all the method looks promising to reduce the impact of zero mass flow issues.

6 Summary and Outlook

In this paper we demonstrated the reason for the slow integration process of fluid system close to vanishing mass flow rates: Due to the high non-linearity of the problem the solution of the non-linear system for calculating an integration step fails, as the simplified chord method is used. The system can only be solved for steps smaller than the steps required from the dynamics of the system itself. And additionally the Jacobian has to be updated almost every step. Therefore, more step which are itself even more computational expensive are required. The problem can be aggravated due to inaccuracies in the calculation of the Jacobian matrix depending on the choice of the states. Though we focused on models of fluid systems, this phenomenon is not restricted to this kind of problem. It occurs for any model with high non-linearity in state where a system should come to rest.

Some approaches which help to avoid the issues were given: If possible the non linearity of the model should be reduced e.g. by linearization close to the zero mass flow rate. The linearization interval can be adapted to the local model behavior. Additionally, it is helpful to make other variables independent of the solution of the mass flow rate if the mass flow rate becomes small. Furthermore, it is advisable to improve the accuracy of the Jacobian matrix calculation e.g. by modified state choice or by a analytic calculation of the Jacobian matrix. With the knowledge of the exact cause of the problem it might be possible to develop additional improvements or enhance the existing once. For example it might be possible to develop an automatic calculation of the linearization interval for the relation of mass flow rate and pressure drop.

As the zero mass flow issue comes from the non-linear solution process, a modification of that process was suggested as well. By including a damping procedure in the solution process, the solution process for small mass flow rates can be improved. Unfortunately, this method only improves solution process if the iterative solution "overshoots" the actual solution. Non-linearity might also lead to a slow iteration progress without overshooting: for example when calculating the root of $\dot{m}^2 = 0$. If a dynamic momentum balance is used, the problem to solve is in that nature and a damping of the steps is not helpful anymore. This problem should be analyzed as well as it is done here for models without mass flow rate state. It might be possible to improve the solution process as well, e.g. by including factors or solution processes which do an approximated update of the (inverted) Jacobian matrix during iteration (e.g. Broyden method). The usage of these methods might be useful in the presented cases as well.

References

Dahmen, W. and A. Reusken (2008). *Numerik für Ingenieure und Naturwissenschaftler*. Springer-Lehrbuch. Springer Berlin Heidelberg. ISBN: 9783540764939.

Dassault Systemes AB (2023). *Dymola Release Notes*.

Dermont, Pieter et al. (2016-05). “Advances of Zero Flow Simulation of Air Conditioning Systems using Modelica”. In: *Proceedings of the 1st Japanese Modelica Conference, Tokyo, Japan*. Vol. 144, pp. 139–144. DOI: 10.3384/ecp16124139.

Gershgorin, S. (1931). “Über die Abgrenzung der Eigenwerte einer Matrix”. In: *Bulletin de l’Académie des Sciences de l’URSS. Classe des sciences mathématiques et na* (6), pp. 749–754.

Hindmarsh, Alan C. et al. (2023). *User Documentation for CVODE*. https://sundials.readthedocs.io/en/latest/cvode. v6.5.1. URL: https://sundials.readthedocs.io/en/latest/cvode.

J. Brunnemann, D. Rideout (2008). “Properties of the Volume Operator in Loop Quantum Gravity II: Detailed Presentation”. In: *Class.Quant.Grav.* (25:065002).

Kelley, Carl T (1995). *Iterative methods for linear and nonlinear equations*. SIAM.

Li, Lixiang et al. (2020-11). “Fast Simulations of Air Conditioning Systems Using Spline-Based Table Look-Up Method (SBTL) with Analytic Jacobians”. In: *American Modelica Conference*. DOI: 10.3384/ECP2016964.

Modelica Standard Library 4.0.0 (2023-04-18). URL: https://github.com/modelica/ModelicaStandardLibrary.

Qiao, Hongtao and Christopher R Laughman (2022). “Performance Enhancements for Zero-Flow Simulation of Vapor Compression Cycles”. In: *Modelica Conferences*, pp. 128–135. DOI: 10.3384/ECP21186128.

Hybrid data driven/thermal simulation model for comfort assessment

Romain Barbedienne[1] Sara Yasmine Ouerk[1] Mouadh Yagoubi[1] Hassan Bouia[2] Aurélie Kaemmerlen[2] Benoit Charrier[2]

[1]IRT-SystemX, France, {romain.barbedienne,sara-yasmine.ouerk,mouadh.yagoubi}@irt-systemx.fr
[2]EDF Lab Les Renardières, EDF, France, {hassan.bouia,aurelie.kaemmerlen,benoit.charrier}@edf.fr

Abstract

Machine learning models improve the speed and quality of physical models. However, they require a large amount of data, which is often difficult and costly to acquire. Predicting thermal comfort, for example, requires a controlled environment, with participants presenting various characteristics (age, gender, ...). This paper proposes a method for hybridizing real data with simulated data for thermal comfort prediction. The simulations are performed using Modelica Language. A benchmarking study is realized to compare different machine learning methods. Obtained results look very promising with an F1 score of 0.999 obtained using the random forest model.

Keywords: machine learning, hybridization, simulation, thermal comfort

1 Introduction

1.1 Context and problematic

Nowadays, numerical simulation represents an essential tool in designing and managing real-world systems, thanks to its lower cost compared to direct experimental testing on the system to be designed. Many industrial applications have benefited from the contributions of numerical simulation to improve the performance of systems. Thermal comfort is considered as a important topic the field of numerical simulation and several studies have been conducted but the results are often far from reality (Feng et al. 2022). One of the main difficulties is the lack of reliable data. Indeed, the acquisition of data on thermal comfort is very expensive. It requires to place the subjects in an environment where the temperature, the hygrometry rate and the thermal radiation are controlled. It also requires testing over a long period of time to avoid transient phenomena, and on a wide variety of subjects (age and gender). In this paper, we address the following problem: how can we increase the quantity of data to improve thermal comfort prediction?

1.2 State of the art

Time series data augmentation is a technique that aims to increase the size of the dataset using synthetic data generation or data transformation methods. This technique is used to improve the performance of time series prediction models by increasing the diversity of the training data and reducing the risk of overfitting. In the area of time series, the increase in data is particularly important because data are often scarce and expensive to collect.

1.2.1 Data generation approaches

One of the most well-known approaches in the field of data augmentation is synthetic data generation. Synthetic data generation approaches aim to increase the size of the dataset by generating synthetic data that resembles the real data. Some of the most common approaches include Markov processes, Gaussian mixture models and generative adversarial neural networks (GANs).

Since their inception, GANs have gained a lot of traction in the deep learning research community. Their ability to generate and manipulate data in multiple domains has contributed to their success.

A GAN is a generative model composed of a generator and a discriminator, typically two neural network (NN) models. GANs have demonstrated their ability to produce high-quality images and videos, transfer styles, and complete images. They have also been successfully used for audio generation, sequence prediction and imputation. Jinsung Yoon et al. (Yoon, Jarrett, and Van der Schaar 2019) proposed Time-series GAN (TGAN), a novel version of GAN for generating realistic time-series data. They introduced the concept of supervised loss; the model is encouraged to capture time conditional distribution within the data by using the original data as a supervision. They obtained significant improvements over state-of-the-art benchmarks in generating realistic time-series of multiple datasets.

One of the advantages of these techniques is their power to greatly increase the size of the dataset and help to model extreme situations that may not be observed in real data.

And, one of the main limitations of TGANs is the restriction of the specified sequence length that the architecture can handle. In addition, generated data may not accurately reflect real data and may require significant computational resources.

1.2.2 Data transformation approaches

Data transformation approaches aim to increase the size of the data set by applying transformations to the existing

DOI
10.3384/ecp204199

time series. Some common approaches include normalization, Fourier transform, time warping and interpolation.

Time warping technique consists in applying random guided transformations to existing time series to generate new training series. The time warping transformations are applied using a cost function that measures the similarity between two series. The experiments conducted in the paper (Iwana and Uchida 2021) show that the proposed data augmentation technique significantly improves the performance of neural networks for various time series related tasks, such as energy consumption prediction and human activity recognition.

The Fourier Transform method (Yang, Yuan, and X. Wang 2023) involves dividing the training data into multiple sets and then applying the Fourier Transform to each set. The Fourier coefficients of each set are then combined in a stratified manner to generate new training series. The newly generated series are used to train a time series classification model.

The paper also describes a method for selecting the data sets to be used for data augmentation. This method involves using a clustering algorithm to group the training data into similar sets and then selecting the data sets that are most different from each other.

Experiments conducted in the paper show that the proposed data augmentation method significantly improves the classification performance for various time series datasets.

Interpolation (Oh, Han, and Jeong 2020) is a method of estimating unknown values in the time series using the known values based on a specific interpolation function like cubic splines. This method may greatly improve the score on the generated data especially when the interpolation function is well suited to the problem.

Other approaches using data transformation include time slicing window which consists of cutting a portion of each data sample, to generate a different new sample. Adding noise to time series, flipping by inverting a time series, scaling by changing the magnitude of a certain step in the time series, rotation and permutation. Some of these techniques can only be used for specific datasets. Indeed, it does not make sense to apply flipping for a time series describing a temperature variable for example.

One of the advantages of these techniques is their simplicity to implement and the fact that they allow to control the generated time series.

On the other hand, if the data have complex patterns the generated data may not accurately reflect the real data.

1.2.3 Simulation approaches

Another way for data augmentation is to use a simulation model to generate synthetic data.

For example, in the autonomous driving (Cao and Ramezani 2022) field, simulators such as DeepGTA-V and CARLA (Car Learning to Act) can be used to generate large amounts of synthetic data that can complement the existing real-world dataset in training autonomous car perception. These models allow to generate several scenario configurations (bad weather conditions, road accidents, obstacles...etc.), which gives different driving environments.

One potential downside of data augmentation using simulation is that the simulated data may not perfectly represent the real-world data, which can affect the performance of machine learning models trained on the data.

On the other hand, simulation also provides more diverse information. For example, for a time series describing the operative temperature of a housing, it is possible to simulate the operative temperature in several seasons.

Simulation enrichment allow to reduce the gap between the training dataset and the dataset used for inference and evaluation. the comfort models are typically learned from real data but evaluated of these models are performs from simulation results. It is important to note that simulation results may deviate from the actual models, thus impacting the accuracy of the learning process based on simulated data. To address this issue, it is crucial to conduct the inference of the learned model using simulated variables.

However, in order to prevent any biases in machine learning (ML) models, simulations must accurately reflect the real-world context. Creating a simulation model that closely resembles the actual environment poses a significant challenge.

2 Methodology

The aim of this approach is to complete each observation with environment variables generated by simulation. For each observation a simulation model will act as a digital twin. The methodology is divided into four main stages Figure 1.

The first step is the data preprocessing. This is a classical step in preparing data for machine learning. The main objectives of this step are, to verify the data format, to ensure the data consistency, complete missing data and remove outliers.

The second step is the data adaptation. The prepossessed data may differ from the parameters of the model to be generated. for example, if the data consists of questionnaires sent to a sample of people. The person answering the questionnaire may not know the parameter of a simulation model. But this parameter can be deduced from another question. For example, in the case of thermal simulation of buildings, it is easier for respondents to enter the year of construction of their house than the thickness of its insulation. It is then possible to approximate the thickness of the insulation with the norms for the year of construction. The application section 3.1 will contains more details on this step. As well as surveys.

The third step is the model generation. Model generation is performed for each observation. It requires the creation of rules to generate the simulation model, or the creation of several simulation templates whose parameters are filled in according to the adapted observations.

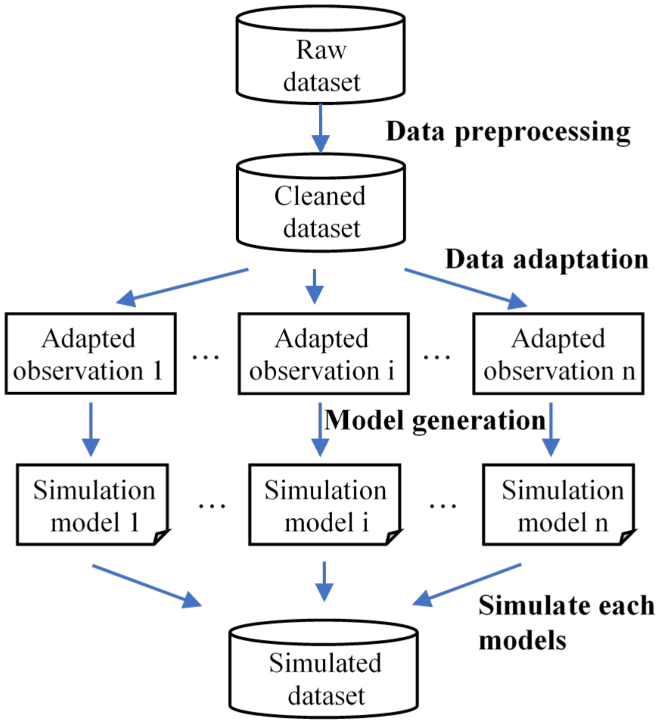


Figure 1. Description of data generation process

The last step is the simulation of each model, and the post-processing of the results. The objective of this step is to prepare simulation results for learning.

3 Application

3.1 Description

3.1.1 Survey Analysis

The aim of this study is to predict household thermal comfort. A survey was sent to 4000 French households. The sample was selected to be as representative as possible of the French population.

The survey is composed of 240 questions divided into 5 categories; building geometry, building insulation, heating systems, heating habits and comfort perception. As descried in Figure 2, building geometry, building insulation and heating systems questions are adapted in order to generate the thermal simulation of the housing.

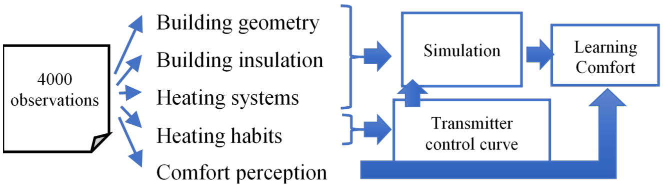


Figure 2. Synopsis of approach

Heating habits answers are used in order to reproduce household heating curve, including, Transmitter control curve and the opening and closing curves of the shutters and windows. Finally, questions about perceived comfort are adapted to feed the learning model.

3.1.2 Survey Validation And Preprocessing

The reliability of questionnaire responses was validated by an external organization (IPSOS). Prior to the survey, 200 homes were instrumented with power and temperature sensors for each emitter. The technicians who visited the homes filled in the necessary information. When the survey was completed, the results from these 200 homes were compared with the instrumented data to validate the approach. These results, and the comparison with simulation results, will be the subject of a future paper. The present paper deals with the methodology Process.

The first step was to pre-process the data. Dwellings containing outliers were removed. For each variable the Interquartile Rule were applied in order to identify outliers. For example, dwellings with surface too large or too small are removed. After this step, 3 529 dwellings contained statistically acceptable variables.

3.2 Completion of data with simulation

3.2.1 Model hypothesis

The simulation model used for the model generation is created thanks to buildSysPro (Plessis, Kaemmerlen, and Lindsay 2014). This opensource modelica library contains parametric models for different building parts, including wall, windows, roof and floor. The parametrization is simplified by grouping all the characteristic parameters in records. A record contains the parameters of the different materials used for the structure, insulation and interior cladding. It also includes geometrical parameters, such as the thickness of the different materials. The records are established according to the years of construction of the buildings and the different standards.

For every dwelling, each room is modeled as a thermal node. The temperature is considered uniform at each point of the room volume. The conductive exchanges between the walls of the rooms and the external walls are modeled using the heat transfer in one spatial dimension. Transient phenomena are considered. Walls are discretized every time materials are changed or every five cm (this is done automatically by the buildSysPro library). It allows to linearize the heat equations for each wall.

The air flow exchanges between individual rooms are neglected. We have assumed that all the interior doors are closed. Convection exchange between the outside air and the wall is calculated using the newton law. Heat transfer coefficient is given by the record according to the exterior materials.

The solar radiation is calculated using the model from Hay Davies Klucher Reindl (HDKR) (Padovan and Del Col 2010). The environment variables (external temperature, humidity, wind speed and direction, variables needed to calculate the incident radiative flux, etc.) are loaded from a file.

3.2.2 Description of model templates

Two generic building models are created with buildSysPro library; a 2-3 bedroom house (Mozart house) and a 1-2 bedroom apartment (Matisse apartment). 35% of the households interviewed in the survey correspond to these two types of housing which corresponds to about 1400 dwellings. The plans of these dwellings are described in Figure 3. For each template, the Bedroom 2 can be empty.

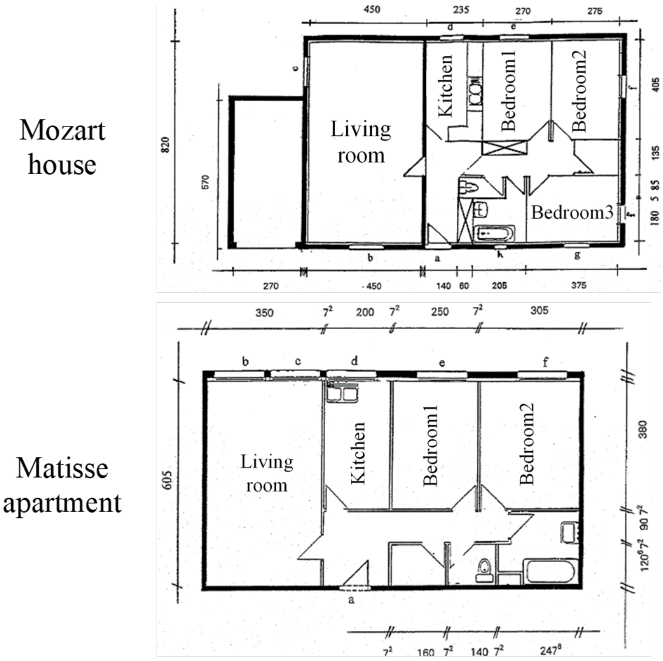


Figure 3. Plan of different model templates

A simulation model template has been created by dwelling category. An example of simulation template for Mozart house is described in Figure 4. In this figure, each variable is a vector. Each coordinate of the vector corresponds to a room. The dimension of the vector corresponds to the number of rooms of the dwelling. This template is composed of 4 parts; one part is the model that generates environment variables previously described. The second part is the thermal model of the building. Hypothesis of this model have also been previously described.

The window model controls the opening and closing of windows room by room. The opening of the windows is controlled by an external file (generated from the survey answers). At each time step, the model allows to open a window if its window state variable is set to true and the set temperature is reached. This model also closes the window if the difference between the set temperature and the air temperature is below a certain threshold. This threshold has been set at 3°C by default.

The heating models are composed of two models. One model controls the heat flow injected room by room by the fixed heaters. The second one controls the heat flow injected room by room by the mobile heaters. The model controlling the heat flow for mobile heaters consists in two heat flow injections; one by convection and a second one by radiation. Total heat flow injection is defined by an input csv file (generated from the survey answers). The ratio between convective and radiative heat flow is considered constant and is established according to the type of heating system entered in the questionnaire.

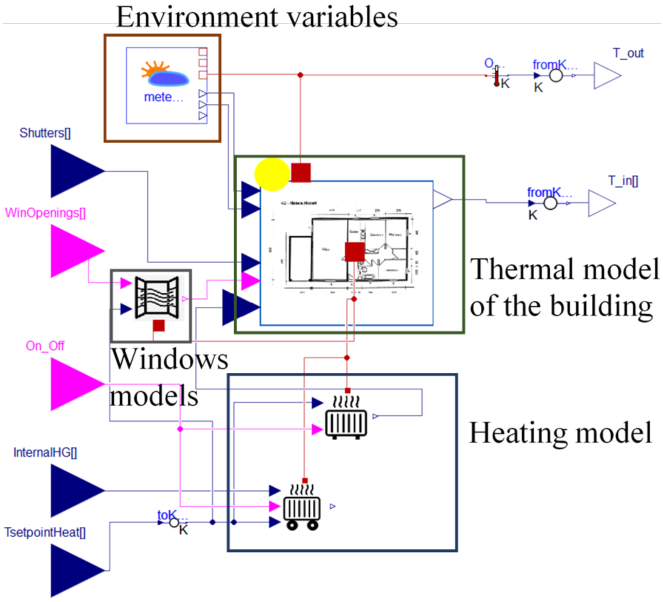


Figure 4. Example of simulation template for Mozart House

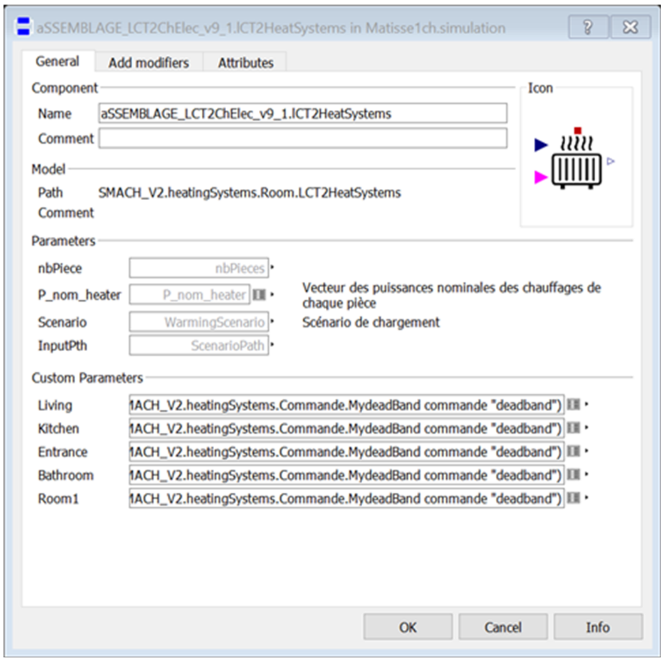
The parametrization of fixed heaters for Matisse apartment with one bedroom is described in Figure 5. For each room, it is possible to set the heater type and heater controller. Variable P_nom_heater is a vector that contains at each coordinate the sum of the nominal power of all the heaters in a room. Scenario and InputPath parameters indicate the path for the file describing the wood reloading hours. This scenario is specific for the inhabitants using a fireplace for heating. It depends mainly on the activity of the inhabitants composing the household.

Three control models have been implemented. A Proportional integral differential (PID) model, a dead band model and a model of absence of control, when the inhabitants declared not to have heating in the room. Six heating models have been implemented: electric heating type convector, radiant panel, soft heat, accumulation, water heating or wood heating.

Models have been implemented using method Th-BCE (écologique 2023) except for wood heating model that have been instantiated from the buildSysPro library. Th-BCE method is a French regulation. The two hypothesis concerning heaters are that: the thermal inertia of heaters is neglected and the ratio of heat transfer radiated flow and convective flow is considered constant.

3.2.3 Description of data used for simulation

The parameters used to fill the simulation model are described in Table 1. This table also describes the questions in the survey, and the method used to calculate simulation


aSSEMBLAGE_LCT2ChElec_v9_1.lCT2HeatSystems in Matisse1ch.simulation
General
Add modifiers
Attributes
Icon
OK
Cancel
Info

Component
Name: aSSEMBLAGE_LCT2ChElec_v9_1.lCT2HeatSystems
Comment:

Model
Path: SMACH_V2.heatingSystems.Room.LCT2HeatSystems
Comment

Parameters
nbPiece: nbPieces
P_nom_heater: P_nom_heater — Vecteur des puissances nominales des chauffages de chaque pièce
Scenario: WarmingScenario — Scénario de chargement
InputPth: ScenarioPath

Custom Parameters
Living: 1ACH_V2.heatingSystems.Commande.MydeadBand commande "deadband")
Kitchen: 1ACH_V2.heatingSystems.Commande.MydeadBand commande "deadband")
Entrance: 1ACH_V2.heatingSystems.Commande.MydeadBand commande "deadband")
Bathroom: 1ACH_V2.heatingSystems.Commande.MydeadBand commande "deadband")
Room1: 1ACH_V2.heatingSystems.Commande.MydeadBand commande "deadband")

Figure 5. Parametrization of fixed heaters room by room

model parameters from the questions. The temporal variables were filled from the survey by choosing three typical days per week. For each day, the variables are filled in hour by hour for each room. These typical days are assigned to each day of the week.

The temperature measured in the questionnaire is based on a typical week. It does not give details of the temperature hour by hour over a year. Also, room temperatures are likely to vary according to the insulation of the dwelling and the power of the radiators. The simulation shows this variation. In the simulation, the temperature measured over a typical week is approximated as the set temperature and repeated every week.

Two algorithms were used; a first one allows to compute the orientation variables and a second one to control the opening and closing of the shutters.

Concerning orientation variable, the chosen algorithm is a decision tree. The decision node was manually implemented. The survey contains for each room whether the windows are predominantly south-facing. Thus, orientation is calculated thanks to different plans (Matisse and Mozart) in order to maximize the surface of the windows facing to the south according to the survey. 1 describes the decision tree for Mozart house. By convention, the Orientation variable is null when north is oriented at the top of the plan in Figure 3. A similar algorithm was built for the Matisse template.

In this algorithm, $IsSouth_{room}$ is a binary list where *room* belongs to each available room of Mozart house, $Orientation$ designates the variable in degree of house Orientation.

The instruction for opening and closing the shutters is calculated from the presence variable room by room. The time at which the shutters open is determined by the time at which a room becomes empty.

Table 1. links between questions and variables used to complete the simulation models

Question	*Variable in simulation*	*Calculation method*
Number of rooms	Number of rooms	assignment
The total floor area of the dwelling	Floor total area	assignment
Room with south-facing windows	orientation	deductive algorithm
Heating power for each room	Nominal power of heater for each room	assignment
Year of construction of the dwelling	House record	assignment
Temperature measured hour by hour over a week	setpoint temperature	assignment
Ignition time of auxiliary heaters and power of auxiliary heaters	auxiliary heating power	assignment
Hour of opening of the windows and duration of the opening	instruction for opening and closing the windows	assignment
Presence in the rooms	instruction for opening and closing the shutters	deductive algorithm
Date of switching on the heating	instructions for switching the heater on and off	assignment

The closing time of the shutters has been calculated in relation to the sunset time. The sunset time depends on the day and the location of the dwelling. The python library suntime (Stopa 2019) has been used for the calculation of the sunset time according to the department in which the dwelling is located and the simulated day. The closing time corresponds to the times of presence in the dwelling closest to the sunset.

3.2.4 Description of input files

Thermal regulation 2012 (RT 2012) (écologique 2023) is a French regulation. it separates France into 8 thermal zones. RT2012 provides for each thermal area, an average environment file. This file provides each 30 minutes a value for weather variables, including, wind speed and orientation, different temperatures variables, relative humidity, atmospheric pressure and solar irradiance direct

Algorithm 1 Calculation of orientation for Mozart House

```
Start
  if $IsSouth_{living}$ then
    if $IsSouth_{bedroom3}$ then
      $Orientation \leftarrow 0°$
    else
      $Orientation \leftarrow 90°$
    end if
  else
    if $IsSouth_{bedroom2}$ & $IsSouth_{bedroom3}$ then
      $Orientation \leftarrow 270°$
    else
      $Orientation \leftarrow 180°$
    end if
  end if
End
```

and indirect. These files were used in order to estimate environment variables. The thermal area of a dwelling is determined according to its department.

3.2.5 Generation of modelica models

Each observation is instantiated using the class diagram of the algorithms as described in Figure 6. Modelica files were generated from instantiation of this class diagram using model transformation process. The implementation was carried out using the Modelica Language.

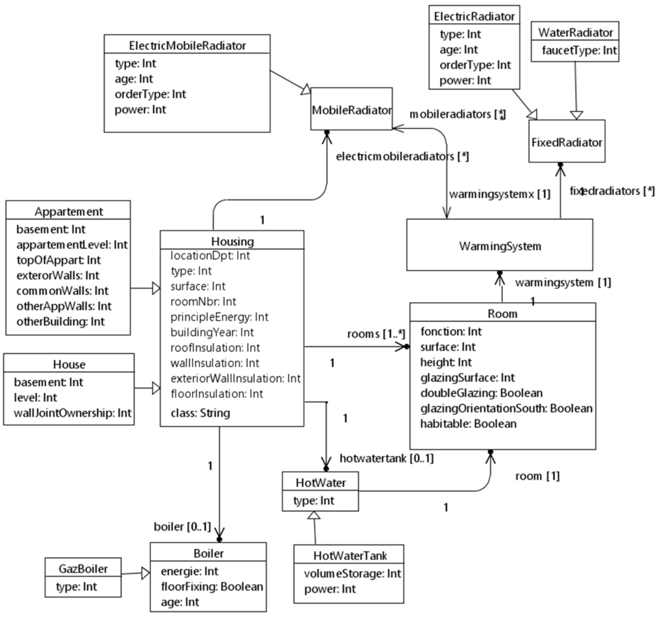


Figure 6. Class Diagram of python models

The simulations were launched using Dymola-Python application programming interface (API). The solver used is Differential/Algebraic System Solver (DASSL) with a time step of 1800s. The simulations were launched from October 1st to April 30th, results files are stored as CSV files. The simulations were run on a laptop with a quad core processor with 16GB of RAM. It took 72 hours to run 1,400 simulations. Simulations were parallelized on 4 cores.

In order to facilitate the training of recurrent machine learning models, it is frequent to use an invariant time step for all the time series. Therefore, the results were post-processed to have a constant time step of 1800s, because DASSL is a variable time steps solver.

3.2.6 Simulation results

The simulation results are given in Figure 7. The average air temperature in the living room is $19°C$ for an apartment or a house. The main difference concerns the maximum and minimum values. Indeed, the maximum and minimum temperatures are higher for apartments than for houses. The difference concerns the minimum temperature, which is explained by the proximity of an apartment to other apartments. Thus, an unheated apartment will be heated by convection by the other surrounding apartments, avoiding too low temperatures. This is not the case for houses. As regards the increase of the temperature between October 1st and November 1st, it concerns only one apartment, this apartment combines an early ignition of heating managed by a collective boiler with a high outside temperature due to the localization of the apartment (in the department of Corsica). Note that weather variables are approximate, and local weather conditions may be different, which explains this difference. Finally, the generalized fall of temperature after the 15th April is due to the stop of the heating at this date.

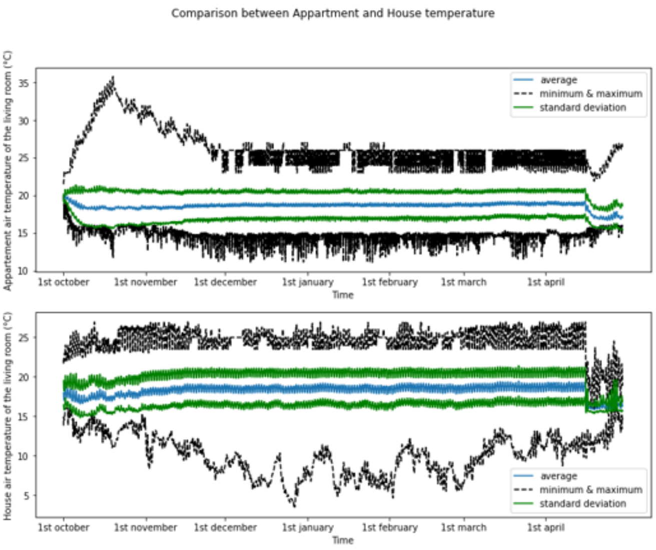


Figure 7. Comparison of simulation results between House and Apartements

The inhabitants surveyed were able to fill in the thermal discomfort time in the questionnaire. 5 choices were available: being comfortable (84.6% of total observations), being cold for at least 24 hours (6.9%), being cold for a few days (5.2%), being cold almost all the time (1.9%) or all the time (1.4%). Figure 8 illustrates the average liv-

DOI
10.3384/ecp204199

ing room temperature of households according to their response to the comfort question.

This figure highlights that air temperature is not a good indicator. It does not allow to differentiate between households that are cold all the time and those that are cold some days.

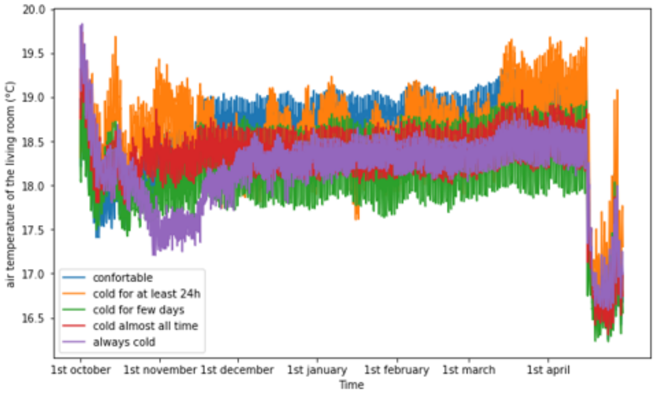


Figure 8. Air temperature of living room according to comfort clusters

The operating temperature is a good but not sufficient indicator of comfort. In fact, it considers the air temperature and the radiated temperature. However, this temperature does not represent the temperature felt by an inhabitant, because the inhabitant does not always occupy the cold room. For example, even if the operating temperature of the bathroom is $10°C$, if there is never anyone in the bathroom it is useless to take this variable into account. Thus, we have introduced the operating temperature of presence. This variable is the operating temperature averaged by the presence of inhabitant per room. This variable is calculated at each time step when there is at least one inhabitant in the house ($\sum_i^{n_{room}} Pres_{room_i} > 0$), as follow:

$$T_{op\ pres} = \frac{\sum_i^{n_{room}} T_{op\ room_i} . Pres_{room_i}}{\sum_i^{n_{room}} Pres_{room_i}} \tag{1}$$

Where; n_{room} is the number of rooms for one dwelling, $T_{op\ room_i}$ is the operating temperature of the $room_i$ with $i \in 1; nroom$, $Pres_{room_i}$ is a Boolean variable for each room of one dwelling with $i \in 1; nroom$. This variable is equal to 1 if there is a presence in the room and null if there is nobody in the room and $T_{op\ pres}$ is a list composed of the operating temperatures. As illustrated in Figure 9, the presence operating temperature allows to recover the comfort trend established in the questionnaire. Therefore, we focused on this variable for learning. Note, however, that inhabitants that are cold for a few days and those that are cold almost all the time are difficult to differentiate.

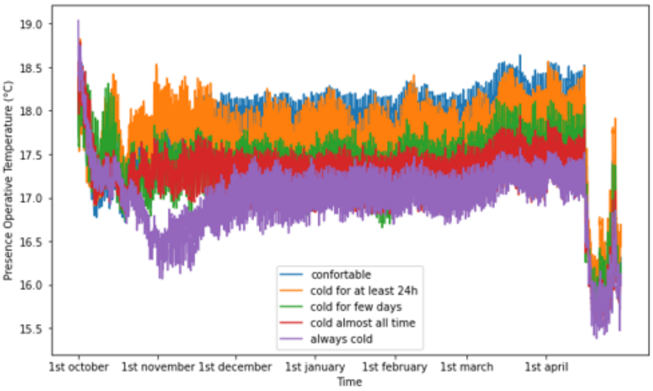


Figure 9. Air temperature of living room according to comfort clusters

3.2.7 Preparation of data for learning

Simulation model computes for each time step the thermal environment of the inhabitants for a dwelling. However, in order to train the ML model of thermal comfort, it is required to calculate the thermal comfort at each time step.

The first algorithm implemented is a simple threshold on the operating temperature of presence. Indeed, for each dwelling a threshold was calculated to respect the discomfort period. If operating temperature is below this threshold, the inhabitants are considered as uncomfortable, otherwise, they are considered as comfortable. The problem with this model is that there were some comfort/discomfort switches between two successive time step(s). The second issue of this approximation is that it does not consider the inertia of thermal comfort.

The improved algorithm computes for each dwelling two thresholds. When the first threshold is reached, the inhabitant is in a discomfort state. It is then required to wait for the presence operative temperature to rise to the second threshold before the inhabitant will be again considered to be comfortable. The calculation of the thresholds is performed by minimizing the number of comfort/discomfort switches under the constraint of respecting the discomfort time indicated in the survey.

For one dwelling, the problem is stated as the following optimization problem:

$$argmin(\varepsilon_{max}, n_{switch}) \tag{2}$$

The constraints are:

$$\begin{cases} \varepsilon_{max} > \varepsilon_{min} \\ max\,(\Delta t_1, \ldots,\ \Delta t_n) \geq t_{discomfort_survey} \end{cases} \tag{3}$$

Where; $(\varepsilon_{max}, \varepsilon_{min})$ is the couple of thresholds to calculate, with $\varepsilon_{max} \in \mathbb{R}^+$ and $\varepsilon_{min} \in \mathbb{R}^+$, n_{switch} is the number of comfort/discomfort switches, with $n_{switch} \in \mathbb{N}$, $(\Delta t_1,\ \ldots, \Delta t_{n_{switch}})$ is the list of discomfort times with $\Delta t_k \in \mathbb{R}^+$ with $k \in 1, nswitch$ and $t_{discomfort_survey} \in \mathbb{R}^+$ is the discomfort time indicated in the survey.

The estimation of these two thresholds is performed using the following algorithm. This algorithm is an heuristic which is divided into 3 main steps. The first step is to calculate a first value of ε_{max}, named ε_0 (2). The calculation of ε_0 is similar to the calculation of a single threshold for comfort/discomfort switching.

In this algorithm, the following variables are used; $T_{op\ pres}[\ ]$ designates a real list composed of the operating temperature at each time step, $T_{asc\_op\ pres}[\ ]$ is a real list composed of the operating temperature ordered in an ascending order, n is an integer , $n_{consecutive}$ is an integer representing the number of consecutive time steps in $T_{asc\_op\ pres}$ between 0 and n, $t_{discomfort\_survey}$ a real representing the discomfort time indicated in the survey, $\Delta tstep$ is a real representing the duration of one simulation time step, ε_0 is a real representing the minimum threshold to have $n_{consecutive}$ discomfort time steps.

The function used are : $list \leftarrow sorting\_asc(list)$ is a function that orders a list in an ascending order, $int \leftarrow get\_consecutive\_tstep(int : n, list)$ is a function that returns the number of consecutive time steps in a list between 1 and n.

The second step is to define a set of $\varepsilon_{max,i}$ and $\varepsilon_{min,i}$ pairs that are close to the optimal solution $\varepsilon_{max,1}, \varepsilon_{min,1}, \ldots, \varepsilon_{max,m}, \varepsilon_{min,m}$ (3). The third step is to select the optimal couple $\varepsilon_{max,k}, \varepsilon_{min,k}$ that minimizes the objective function and respects the constraints described above.

In this algorithm, the following variables are used : $T_{op\ pres}[\ ]$ a real list composed of the operating temperature at each time step; ε_0 is the first value of the threshold calculated by the first algorithm; $T_{threshold_op\ pres}[\ ]$ is a real list composed of the operating temperature bellow the threshold ε_0; $t_{discomfort_survey}$ is a real representing the discomfort time indicated in the survey; $\Delta tstep$ is a real representing the duration of one simulation time step; n_{clu} designates an integer defining the number of time steps of discomfort; $id_min[\ ]$ a list of indexes for local minimums; *i*,*minID* ,*maxID* are integers, $\varepsilon_{min}[\ ]$ is a list of candidates of minimal threshold value; $\varepsilon_{max}[\ ]$ is a list of candidates of maximal threshold value.

The function used are : $list \leftarrow keep\_value\_below(list, real : threshold)$ a function that keeps the values of a list below the threshold value; $list \leftarrow local\_minium\_list(list)$ a function allowing to calculate the local minimums of a list; $int \leftarrow len(list)$ a function that calculates the length of the list.

The algorithm was implemented in python language and executed on the 1400 dwellings. Figure 10 shows an example for an inhabitant that reported to be cold almost all the time. In this figure, the x axis defines the time step. The y axis corresponds to the operating temperature in degrees Kelvin.

Algorithm 2 First step Find a first value for ε_{max}

```
Initialization
    $T_{asc_op\ pres} \leftarrow sorting_asc(T_{op\ pres})$
    $n \leftarrow \frac{t_{discomfort\_survey}}{\Delta t_{step}}$
    $n_{consecutive} \leftarrow get\_consecutive\_tstep(n, T_{asc\_op\ pres})$
Start
    while $n_{consecutive} \cdot \Delta t_{step} < t_{discomfort_survey}$ do
        $n_{consecutive} \leftarrow get\_consecutive\_tstep(n, T_{asc_op\ pres})$
        $n \leftarrow n + 1$
    end while
    $\varepsilon_0 \leftarrow T_{asc\_op\ pres}[n]$
end
```

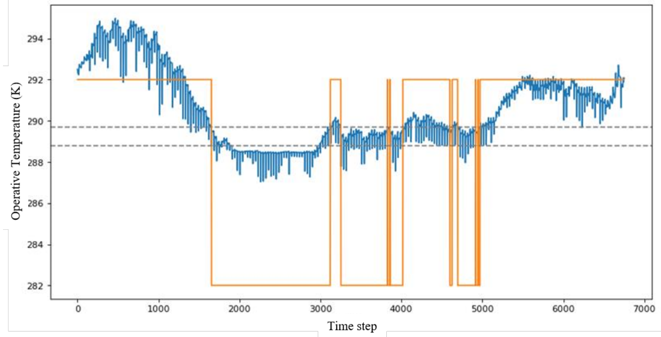


Figure 10. Results of the change from comfort to discomfort in a household that is almost always cold

Algorithm 3 Find candidate couples ε_{max}

```
Initialization
    $n_{clu} \leftarrow t_{discomfort_survey}/over\Delta t_{step}$
    $i \leftarrow 0$
Start
    $T_{threshold\_op\ pres} \leftarrow keep\_value\_below(T\_op\ pres\ , \varepsilon_0)$
    $id\_min \leftarrow local\_minium\_list(T_{threshold\_op\ pres})$
    while $i < len(id\_min)$ do
        $minID \leftarrow id\_min[i]$
        $\varepsilon_{min}[i] \leftarrow T_{op\ pres}[minID]$
        $maxID \leftarrow min(minID + n_{clu}, len(T_{op\ pres}))$
        $Epsi \leftarrow max(T_{op\ pres}[minID : maxID\ ])$
        $\varepsilon_{max}[i] \leftarrow Epsi$
        $i \leftarrow i + 1$
    end while
End
```

3.3 Machine learning results

A data-driven modeling was performed in order to learn the comfort based on the both real and simulated data. The inputs considered are the simulations output at each time step (Radiation temperature, Convective temperature, presence operative temperature, heat flux emitted by radiators for each rooms, Outdoor temperature) and sociological data from survey (average age of household, average gender of household). The output of the machine learning model is a prediction of occupant comfort and discomfort states.

The dataset was divided into 3 sets (60% on Train, 20% on Validation and 20% on Test). The training set is used to train the model and calibrate its parameters, the validation set is used to prevent the model from overfitting during the training phase; by monitoring the evolution of the cost function on both sets. Finally, the test set is used the evaluate the model performances once the training is done.

A first version was built, considering the time steps independent between them. That is, the comfort at a given time step depends only on the simulated temperatures and the characteristics of the housing, and does not depend on the comfort at the time step that precedes it.

Using this configuration, several Machine Learning (ML) models were trained and tested; including Ensemble models like Random Forest and XGBoost, neural networks: Multi-Layer-Perceptron (MLP) (Singh et al. 2017).

In a second step, a new version of modeling was built. It would allow to consider that each comfort value at a given time step depends on its previous values, in addition to exogenous variables (temperatures...etc.). For this configuration, a multi-horizon model (Wen et al. 2018) was tested, consisting of a past horizon and a prediction horizon. This model is particularly well suited for time series prediction.

For this model, two configurations were compared. A first one was built using the real values in the past horizon of the model. The second supposes not to know these real comfort values, and therefore uses only the predicted values to feed the past horizon comfort values.

In order to train these different models, the CrossEntropy loss (CE) (Q. Wang et al. 2020) was used as a cost function to minimize during training, it is defined for one sample as follow:

$$CE = \sum_{i=1}^{C} y_i \times log(p_i) \tag{4}$$

Where C is the total number of classes, y_i is the truth value of the label, and p_i the softmax probability for the i^{th} class.

This loss penalizes the probabilities far from the truth label. The logarithm gives a large score for large differences close to 1 and small score for the ones tending to 0. The total cost is then calculated by averaging the individual costs obtained for the different samples.

And, to evaluate and compare the different models' performances, many classification scores were used; including precision, recall and F1 score (Erickson and Kitamura 2021) for each class of comfort. Precision represents the rate of correct predictions, recall represents the rate of positive samples detected, and the F1 score is a compromise of these two scores. These scores are defined as follow:

$$Precision = \frac{TP}{TP+FP} \tag{5}$$

$$Recall = \frac{TP}{TP+FN} \tag{6}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{7}$$

Where TP represents the number true positives, FP the number of false positives and FN the number of false negatives. Table 2 and Table 3 illustrate the different classification scores evaluated on the test dataset. Table 2 shows the results obtained using the first modeling configuration with the three models (RF, XGBoost and MLP).

Table 2. links between questions and variables used to complete the simulation models

Class	*Model*	*Precision*	*Recall*	*F1score*
Comfort	MLP	0.95	0.98	0.97
	XGBoost	**0.999**	**0.999**	**0.999**
	Random Forest	**0.999**	**0.999**	**0.999**
Discomfort	MLP	0.61	0.37	0.46
	XGBoost	0.97	0.95	0.96
	Random Forest	**0.999**	**0.999**	**0.999**
Unknown	MLP	**1.0**	**1.0**	**1.0**
	XGBoost	**1.0**	**1.0**	**1.0**
	Random Forest	**1.0**	**1.0**	**1.0**

Table 3 shows the results obtained with the second modeling configuration using the multi-horizons model with its two configurations. The support column represents the number of test examples used for each class of comfort.

From Table 2, with the first configuration, the random forest model performed very promisingly for all three comfort classes with the different evaluation metrics. This is likely due to the fit between how the comfort labels were defined using the thresholds and how a decision tree (unit of an RF model) works. In addition, a random forest model is composed of a set of simple decision trees, making it accurate and robust on small datasets.

On the other hand, Table 3 shows that with the second configuration, the multi-horizon model can also obtain very promising results when it is possible to feed its past horizon with the actual comfort values. Unfortunately, for this use case, and with the available data, this configuration cannot be applied because the comfort values in the past horizon cannot be available for each time step. Therefore, according to these benchmark results, the most sweated model for comfort modeling is the random forest model which is simple and demonstrated very accurate prediction results.

Table 3. Multi-horizons model evaluation

Class	*Prediction strategy*	*Precision*	*Recall*	*F1score*	*Support*
Discomfort	Real values in past horizon	**0.999**	**0.999**	**0.999**	33550
	Recursive prediction	0.88	0.25	0.39	
Comfort	Real values in past horizon	**0.999**	**0.999**	**0.999**	124390
	Recursive prediction	0.83	0.99	0.90	
Unknown	Real values in past horizon	**1.0**	**1.0**	**1.0**	44220
	Recursive prediction	1.0	1.0	1.0	

4 Conclusion

The hybridization of thermal simulation and data-based modeling addressed the problem of data scarcity and allowed for the inclusion of additional variables not captured in the survey. Various machine learning models were trained and tested, with the random forest model performing best.

This first study considers temperature, convective, and radiative flux variables. To improve the accuracy and realism of the approach, humidity and air speed parameter shall be considered. In this context, implementing a Stolwijk model (Stolwijk 1971) instead of calculating thresholds on operating temperature would significantly improve the realism of simulated data. Additionally, integrating a multi-agent model like the SMACH model (Albouys et al. 2019) developed by EDF would help for making more accurate predictions of comfort. Finally, a more complete simulation model, modeling air exchanges between each room, could improve the precision of results.

Lastly, although this approach is promising, it has taken a long time to develop. A comparison of the cost, quality, development time and repeatability of the different approaches would allow to assess which approach is best suited to the need. The LIPS platform (LEYLI ABADI et al. 2022) will be used to perform such a benchmark.

5 Acknowledgements

This work was supported by funds from the French Program ”Investissements d’Avenir”.

References

Albouys, Jérémy et al. (2019). “SMACH: Multi-agent Simulation of Human Activity in the Household”. In: *Advances in Practical Applications of Survivable Agents and Multi-Agent Systems: The PAAMS Collection: 17th International Conference, PAAMS 2019, Ávila, Spain, June 26–28, 2019, Proceedings 17*. Springer, pp. 227–231.

Cao, Minh and Ramin Ramezani (2022). “Data generation using simulation technology to improve perception mechanism of autonomous vehicles”. In: *arXiv preprint arXiv:2207.00191*.

écologique, Ministère de la transition (2023). *Gestion des versions du Moteur de calcul “ th-BCE 2020 ” et du RSEE*. Publication Title: RT-RE-bâtiment. URL: https://rt-re-batiment.developpement-durable.gouv.fr/gestion-des-versions-du-moteur-de-calcul-th-bce-a688.html.

Erickson, Bradley J and Felipe Kitamura (2021). *Magician’s corner: 9. Performance metrics for machine learning models*. Issue: 3 Pages: e200126 Publication Title: Radiology: Artificial Intelligence Volume: 3.

Feng, Yanxiao et al. (2022). “Data-driven personal thermal comfort prediction: A literature review”. In: *Renewable and Sustainable Energy Reviews* 161. Publisher: Elsevier, p. 112357.

Iwana, Brian Kenji and Seiichi Uchida (2021). “Time series data augmentation for neural networks by time warping with a discriminative teacher”. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, pp. 3558–3565.

LEYLI ABADI, Milad et al. (2022). “LIPS-Learning Industrial Physical Simulation benchmark suite”. In: *Advances in Neural Information Processing Systems* 35, pp. 28095–28109.

Oh, Cheolhwan, Seungmin Han, and Jongpil Jeong (2020). “Time-series data augmentation based on interpolation”. In: *Procedia Computer Science* 175. Publisher: Elsevier, pp. 64–71.

Padovan, Andrea and Davide Del Col (2010). “Measurement and modeling of solar irradiance components on horizontal and tilted planes”. In: *Solar Energy* 84.12. Publisher: Elsevier, pp. 2068–2084.

Plessis, Gilles, Aurélie Kaemmerlen, and Amy Lindsay (2014). “BuildSysPro: a Modelica library for modelling buildings and energy systems”. In: *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. Issue: 096. Linköping University Electronic Press, pp. 1161–1169.

Singh, Sonali et al. (2017). “Modeling the spatial dynamics of deforestation and fragmentation using Multi-Layer Perceptron neural network and landscape fragmentation tool”. In: *Ecological Engineering* 99. Publisher: Elsevier, pp. 543–551.

Stolwijk, Jan AJ (1971). *A mathematical model of physiological temperature regulation in man*. Tech. rep. NASA.

Stopa, Krzysztof (2019). *Suntime*. Publication Title: PyPI. URL: https://pypi.org/project/suntime/ (visited on 2023-06-09).

Wang, Qi et al. (2020). “A comprehensive survey of loss functions in machine learning”. In: *Annals of Data Science*. Publisher: Springer, pp. 1–26.

Wen, Ruofeng et al. (2018). “A Multi-Horizon Quantile Recurrent Forecaster”. In: _eprint: 1711.11053.

Yang, Wenbo, Jidong Yuan, and Xiaokang Wang (2023). “SFCC: Data Augmentation with Stratified Fourier Coefficients Combination for Time Series Classification”. In: *Neural Processing Letters* 55.2. Publisher: Springer, pp. 1833–1846.

Yoon, Jinsung, Daniel Jarrett, and Mihaela Van der Schaar (2019). “Time-series generative adversarial networks”. In: *Advances in neural information processing systems* 32.

ThermalSystemsControlLibrary: A Modelica Library for Developing Control Strategies of Industrial Energy Systems

Fabian Borst, Michael Georg Frank, Lukas Theisinger,
Matthias Weigold

Institute for Production Management, Technology and Machine Tools,
Technical University of Darmstadt, Germany
f.borst@ptw.tu-darmstadt.de

Abstract

The transformation of energy-intensive industries towards greenhouse gas neutrality leads to increasing complexity of industrial energy supply systems. This affects particularly thermal energy systems due to waste heat utilization measures as well as the integration of renewable energy sources and further storage capacities. This complexity is also reflected in the control strategies of such systems, which makes the development of dynamic simulation models for testing them a research field of growing interest.

The *ThermalSystemsControlLibrary* is a novel Modelica library, which aims at standardized modeling of industrial energy supply systems for control strategy development. Based on a generic data model, all components cover physical as well as control modeling and are particularly suitable for testing supervisory control strategies within external frameworks using the FMI standard. The library is validated for an exemplary use case of an industrial energy supply system comparing two different supervisory control strategies.

Keywords: supervisory control, HVAC, dynamic simulation

1 Introduction

Global aspirations towards greenhouse gas neutrality as well as consequences of geopolitical developments force industrial companies to increasingly consider energy related aspects. In addition to sustainability, also affordability as well as resilience must be addressed from an industrial perspective. Therefore, measures like diversification, redundancy as well as de-centrality will be prevalent in future energy supply systems (Fridgen, Keller, Körner, & Schöpf, 2020; Lund et al., 2021). Apart from the positive effects of these measures, they tend to increase the complexity of the underlying system.

To master this complexity throughout the operation of the system, control strategies must be developed and implemented. Here, not only local control functions (e.g., temperature control loop) but also supervisory control functions (e.g., converter sequencing control) must be considered (Wang & Ma, 2008). In research, many different approaches exist for the latter. Those can be differentiated for example by the nature of the underlying model (e.g., reinforcement-learning) as well as the degree of centralisation (e.g., multi-agent system) (Yao, Hu, & Varga, 2023). Here, detailed simulation models can still be beneficial for validation of the developed control strategies.

For that, we present a Modelica library which enables developers and users to model the physical systems' behavior as well as the corresponding control strategy. We primarily focus on fluid-bound thermal energy supply systems, due to the relevance of heating and cooling supply in industry.

1.1 State of research

The following section focuses on existing Modelica libraries and modeling approaches, as the multi-domain capabilities of Modelica are particularly suitable for the described application. The research field can be separated in two major fields: modeling of *physical system behavior* and *control functions*.

Regarding the physical system behavior, a profound research base already exists. Here, Modelica-libraries such as *AixLib*, *Buildings* and *BuildingSystems* are state-of-the-art and can be applied depending on the users' needs (Müller, Remmen, Constantin, Lauster, & Fuchs; Nytsch-Geusen, Huber, Ljubijankic, & Rädler, 2013; Wetter, Zuo, Nouidui, & Pang, 2014). However, these Modelica-libraries inherit only basic functionalities for system control.

An approach which focuses on the development of automation and control programs is presented in (Wetter et al., 2022). Here, a workflow is presented for the design, verification, and deployment of control sequences. An exemplary implementation is included in the Modelica Buildings Library 7.0.0. Modelica-aspects are more intensively addressed in (Schneider, Pessler, & Steiger, 2017). Here, the *BuildingControlLib* is presented, which allows the modeling and simulation of standardized control functions. Focusing more on supervisory control,

DOI
10.3384/ecp204209

Blum et al. presents a framework for simulation-based testing of control-strategies in buildings (Blum et al., 2021). Therefore, Modelica blocks are developed, which enable overwriting local control-functions. By that, different control approaches can be tested and compared in a robust manner. Wüllhorst et al. also present with BESMod a library for the development of supervisory control functions for building energy systems (Wüllhorst et al., 2022). Furthermore, *Modelon Impact* provides a web-based tool for modeling and virtual testing of industrial energy systems (Modelon, 2023).

1.2 Research gap and requirements

The approaches described beforehand outline, that the modeling of physical systems as well as control functions are often addressed separately within the research community. However, Modelica already offers standard models for the development of rule-based operating strategies, which are still necessary as a fallback strategy in combination with intelligent approaches when applied to real-world systems. By the integration of an additional interface, which allows for switching between the fallback strategy in Modelica and an intelligent strategy within an external framework (e.g., reinforcement learning in Python), a Modelica library would represent the automation architecture of such a system in more accurate way and accounts for the simulation of operation permissions of the external strategy.

In addition, the modeling of local-control loops in Modelica is often regarded only on a small scale. Requirements caused by more complex system, such as cascading of multiple local control-loops as well as sequencing of multiple components, are not addressed in detail so far.

The requirements for a Modelica library in the given research field can be summarized as follows:

- Standardized development of physical and control models through provision of base classes
- Provision of base methods for more complex control tasks
- Hierarchical package structure for development and testing of (system) models
- Consistent variable declaration and data model representing the automation architecture of real-world systems

1.3 Automation data model for industrial energy systems

To develop a Modelica library that meets the above mentioned requirements, we use our previously published automation data model for the energy-flexible cyber-physical production systems (Fuhrländer-Völker, Borst, Theisinger, Ranzau, & Weigold, 2022). The properties and methods of this model form the basis for the developed model library and are briefly summarized in the following. The model consists of three base classes to abstract the control functions of single actuators and systems of multiple actuators:

- *Actuator2Point*: Actuator with discrete behavior (e.g., uncontrolled pump)
- *ActuatorContinuous*: Actuator with continuous behavior (e.g., speed-controlled pump)
- *SystemContinuous*: System consisting of several actuators and sensors with continuous behavior (e.g., boiler with distribution pump and mixing valve).

These base classes implement two essential key methods, which will be reused within the Modelica libary:

- *SelectControlMode*: Enables switching between the automatic control (e.g., fallback strategy) and algorithm mode (e.g., reinforcement learning)
- *SystemFlowControl*: Sequential component control of the actuators within a system.

Furthermore, we propose standard data structures to ensure a high comprehensibility of the data model. Therefore, we define the following structures holding the related variables.

- *control*: access and discrete control variables
- *controlState*: current component state
- *setSetPoint*: component setpoints
- *setPointState*: operating point, setpoint limits
- *systemState*: current system state.

2 Library concept

In the following, the structure of the developed *ThermalSystemsControlLibrary* (TSCL) and the underlying base classes for component as well as system modeling are explained. After introducing the main structure as well as the base classes, we describe the general procedure for system modeling. All components are based on the *Modelica Standard Library* (MSL, version 3.2.3).

2.1 Overall structure

The library follows in its basic structure existing Modelica libraries (Modelica Association, 2020). Therefore, we introduce the packages *User's Guide*, *BaseClasses*, *Components* and *Applications* (see Figure 1).

While the *User's Guide* contains basic license and usage information, the *BaseClasse*s package consists of the sub-packages *AutomationBaseClasses*, *FluidBaseClasses*, *Utilities*, *Media* and *Icons*. *AutomationBaseClasses* holds all classes, functions and interfaces of the underlying data model, which must be used for component and system modeling with the TSCL. A component model represents an individual system, whereas a system model consists of multiple component models. The physical model part uses *Modelica.Fluid* connectors, whose basic properties are defined within the package *FluidBaseClasses*. Their use is also mandatory for TSCL-based models. The other base class packages hold utility functions, media declaration and icon models. The *Components* package contains sub-packages for each technology. These always contain a control method package, a package for physical models and the component itself, consisting of the control and physical model. The *Applications* package holds exemplary use cases, which demonstrate possible applications of the TSCL. Here, a package structure consisting of records holding use case specific parameters, system models, thermal networks, operating strategies, and the main model may be used.

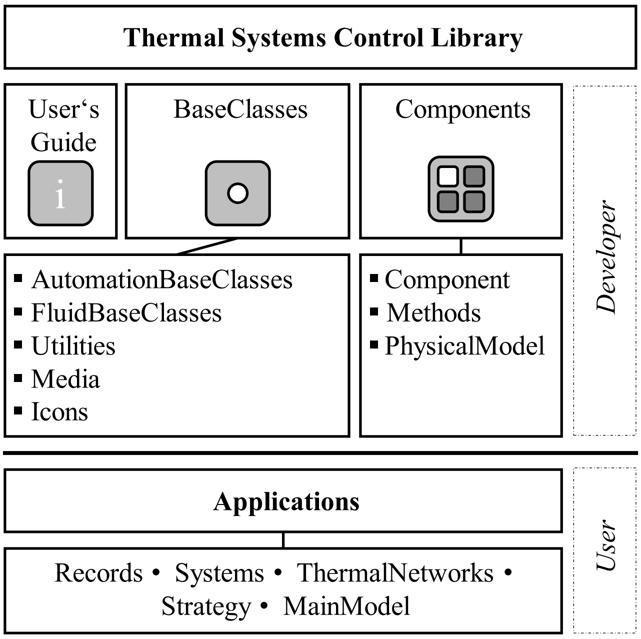


Figure 1. Library structure.

2.2 Base classes

One of the main features of the TSCL lies in the implementation of the *AutomationBaseClasses* package for standardized component and system control. The package is structured in the base classes and packages for interfaces, methods, and tests. The implementation follows largely the introduced data model, but is extended for the implementation of local control functions (Fuhrländer-Völker et al., 2022).

The *ActuatorContinuousLocalControlMode* class enables the standardized implementation of multiple control modes for one actuator. This feature is also applied to the *SystemContinuousLocalControlMode*, which considers the implementation of control modes concerning multiple components within a system (e.g., thermal storage with loading and unloading pump). Figure 2 shows the overall architecture of the *AutomationBaseClasses.*

Within the *Interfaces* package, several connectors implement the hierarchical data structure introduced in (Fuhrländer-Völker et al., 2022). In addition, we provide FMI connectors, giving the user reading and writing permission on sub-level variables within Functional Mock-Up Units (FMU) based on the underlying data model (Fuhrländer-Völker et al., 2022). Usually, input variables of a FMU can only be set when propagated to the top level of the model, which would be not in accordance with our data model. Using the FMI connectors results in a fully compatible implementation of the hierarchical data model. Following this, name strings for accessing the variables of the FMU are fully compatible to the Open Platform Communications Unified Architecture (OPC UA) identifiers from a Programmable Logic Controller (PLC). Thus, OPC UA nodes for controlling the real-world device can be accessed the same way like for the FMU, which makes time-consuming variable mapping obsolete. In summary, this enables the virtual commissioning of the implemented systems.

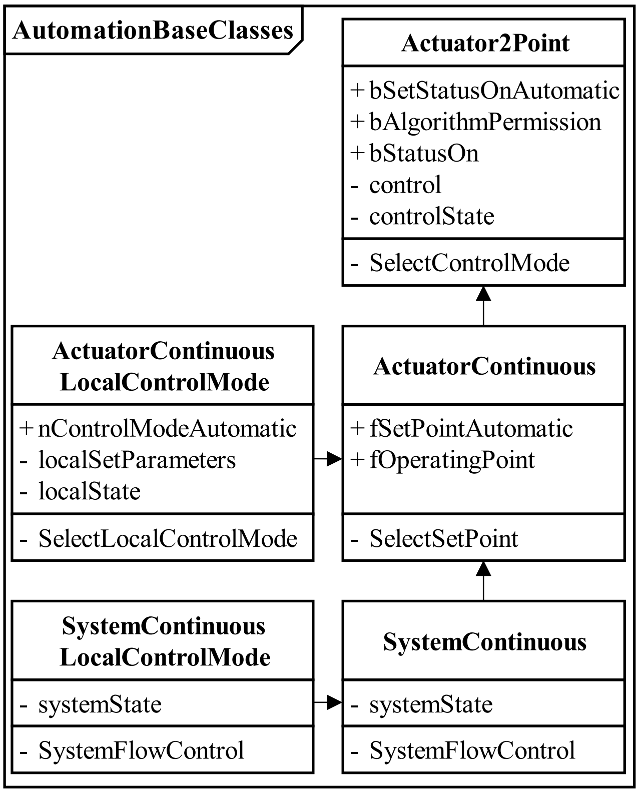


Figure 2. Class diagram of *AutomationBaseClasses* package.

Regarding the key methods of the data model, the sequence control enabling method *SystemFlowControl*, is the only method, which is extensively revised against (Fuhrländer-Völker et al., 2022). The application of the data model to numerous systems has shown that the machine states defined in (ISO, 2017) are suitable for the implementation of machine tools, but not for the implementation of energy supply systems due to the limited number of states. Therefore, this limitation of system states is removed, resulting in the basic sequence control, which is shown in Figure ***3***.

```
for component in system loop
    if run system then
        if not in standby state then
            if previous component runs then
                switch on component
            end if
        else
            switch on component
        end if
        set system state
    else
        if not in working state then
            if not previous component runs then
                switch off component
            end if
        else
            switch off component
        end if
        set system state
    end if
end for
```

Figure 3. Procedure of *SystemFlowControl* method.

2.3 Components

Component models follow the structure shown in Figure *4* and consist of a physical model part based on *Modelica.FluidPorts*, the base class control part, and a local, component-specific control unit. Furthermore, each component has standardized control and state interfaces, which are implemented by the underlying automation base class. Based on this modeling structure, the TSCL provides the following component models: pipes, valves, buffer storages, heat exchangers, pumps, condensing boilers, combined heat power units (CHP), compression chillers, dry coolers, and generic heat consumers. All component models are parameterized by records.

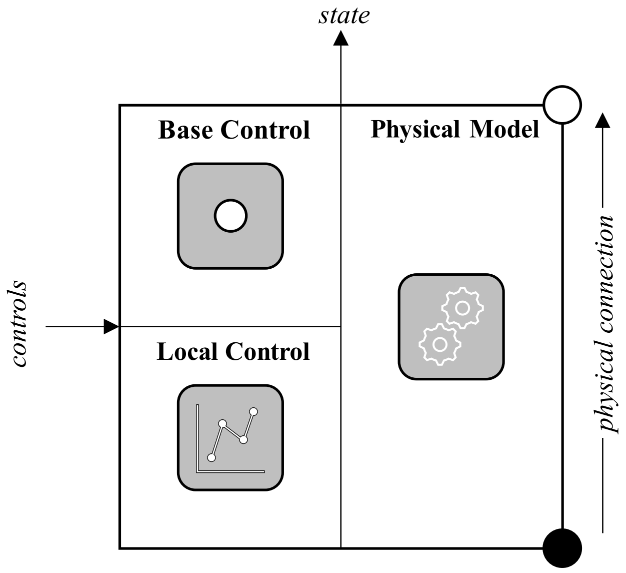


Figure 4. Component and system model structure.

2.4 Modeling procedure

Modeling with the TSCL follows a three-step procedure consisting of technology, system, and scenario modeling (see Figure *5*).

During the first step, the generic component records are extended by use case-specific parameters. After that, subsystems implementing the physical interactions of component models and especially their sequence control are modeled.

Secondly, several subsystems may be combined in a supply system model. This is especially useful for complex structures, e.g. multiple networks of different supply temperatures. The control variables of the subsystems within a supply system should be defined by use case-specific connectors which allows using the same physical model with different control strategy models. Afterwards, the control strategy, which enables the top-level system control, is implemented. Here, supply temperatures and prioritization of energy supply as well as storage systems are implemented.

Thirdly, the developed strategy models are combined with the supply system model to set up different scenarios. Finally, the scenarios may be evaluated in simulation studies.

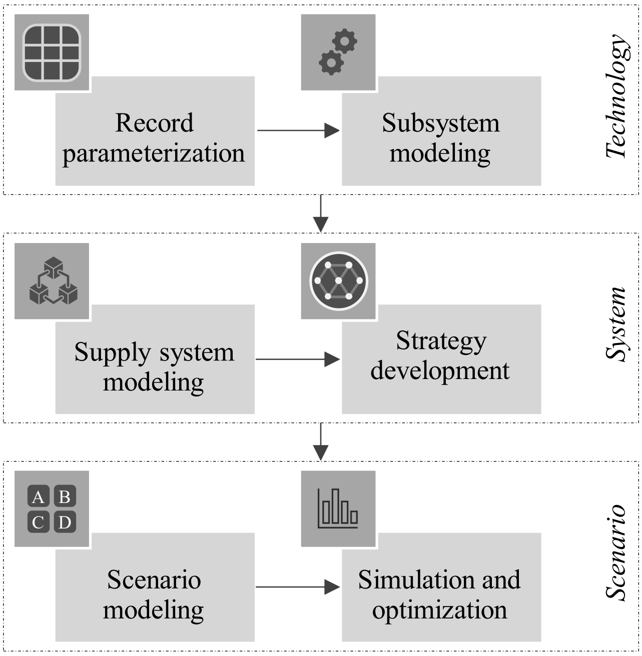


Figure 5. TSCL modeling procedure.

DOI
10.3384/ecp204209

3 Use case - ETA Research Factory

According to the described modeling procedure, the heat supply system of the ETA Research Factory of the Technical University of Darmstadt is partly modelled as a validation use case. In the following section the overall system topology, the parameterization of network components and the basic operating strategy is described.

3.1 Overall system topology

The use case comprises of two connected heating networks, operating on a high and low temperature level. The high temperature heating network (HNHT) includes two combined heat and power units and a condensing boiler. For active heat storage, a vacuum super insulated (VSI) heat storage is integrated in the HNHT. For the decoupling of production and consumption, a buffer storage is used as a hydraulic separator. A static heating represents the consumer in the HNHT.

Furthermore, a counterflow heat exchanger connects the HNHT to the heating network low temperature (HNLT) and acts as a heat producer in the HNLT. The consumer side comprises an underfloor heating, which is connected to the producer side through a buffer storage.

3.2 Supervisory control strategy

For the supervisory control strategy of the thermal supply systems, temperature limits for both networks are defined. The prioritization of the producers is tuned by defining different off sets for the producers' set points, which are controlled by hysteresis controllers. Following the nomenclature of the data model, the *fSetPointAutomatic* of the producers is defined by the target temperatures of the HNHT and the HNLT. The hysteresis controllers prioritize the condensing boiler against the CHP units and control the operation variable *bSetStatusOn* for all systems.

The *fSetPointAutomatic* of the consumers, static heating in the HNHT and underfloor heating in the HNLT, is set according to the heating characteristic, determined by the outdoor temperature.

4 Application

In the following section, we validate the TSCL for the use case of the ETA Research Factory. We first demonstrate the system modeling procedure for an exemplary CHP system and then present the results from comparing two operating strategies using the TSCL.

4.1 Subsystem modeling for a CHP system

Figure ***6*** shows the implementation of an exemplary CHP system. For this, the base class *SystemContinuous* is extended to provide basic control functions. The system consists of a discrete valve, a rotational pump, a mixing valve, a heat meter and the CHP unit. For the physical part of the model, all components are connected by *Modelica.FluidPorts*. All components have identical control interfaces due to the use of the same base classes (see Figure ***2***). The system control enables the state dependent start and stop process of the components. In this case, discrete and mixing valve, pump and CHP are started in sequence. The time delay is thereby modeled by component-internal PT_1-elements. The component control (*nControlMode*) implements a flow temperature control for the CHP, a differential temperature control of 15 K for the mixing valve and constant speed control for the pump.

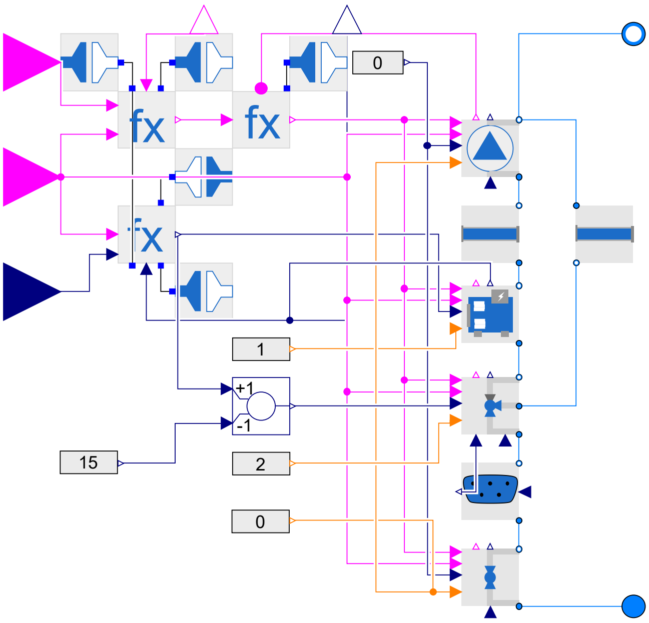
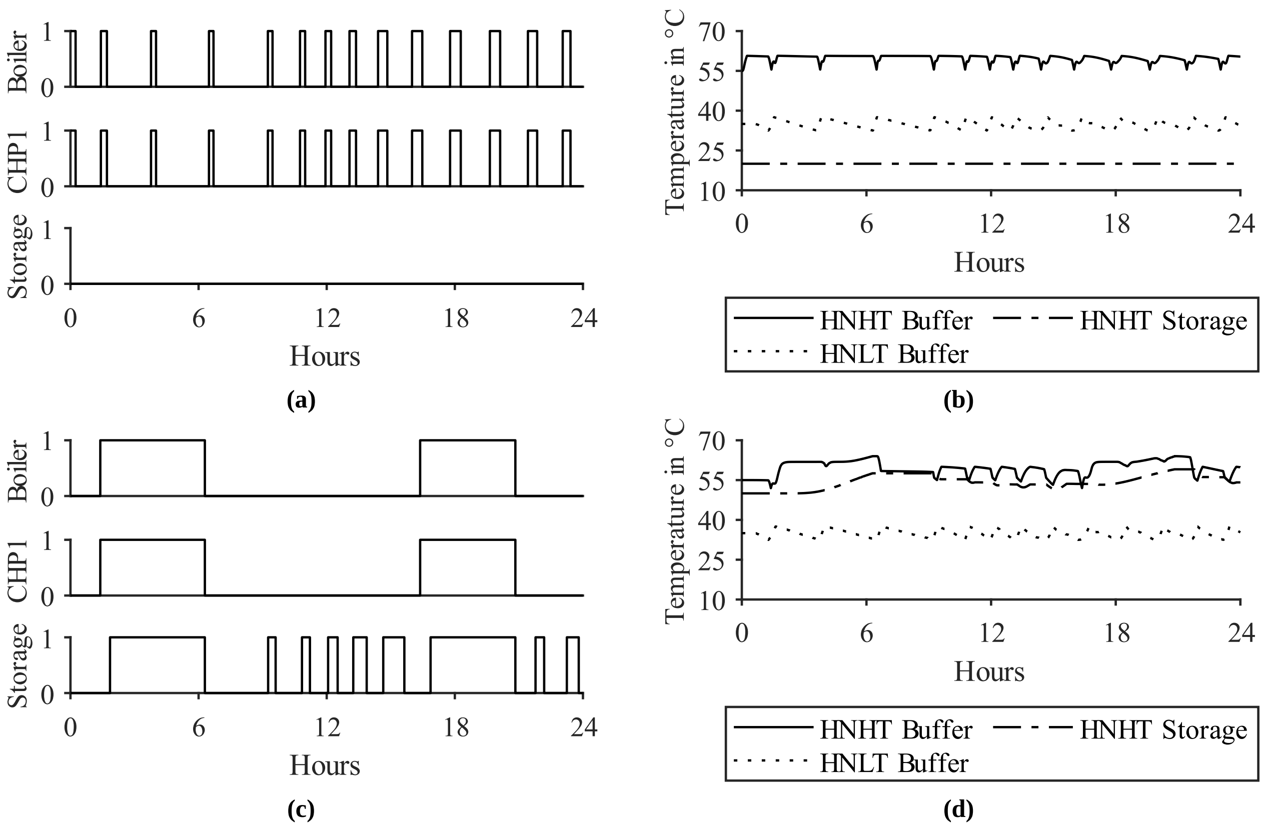


Figure 6. Modelica model of exemplary combined heat power system.

All other systems described in section 3 follow this modeling approach and are instantiated in two supply system models for HNHT and HNLT. Both supply systems are completed by specific control connectors forwarding the control signals from the supervisory control. The supervisory control strategy models consist of several sub-models for the implementation of supply system-specific prioritization of the subsystems. The complete modeling example is located within the *Applications* package of the TSCL repository.



(a)

(b)

(c)

(d)

Figure 7. Simulation results for the control strategy without (a-b) and with an active storage (c-d).

4.2 Results

Figure 7 shows the simulation results of the supply system model of the ETA Research Factory considering two different supervisory control strategy models. In the baseline scenario, it is assumed that the temporal decoupling of heat production and consumption is exclusively enabled by a buffer storage. To reduce the number of operation cycles of heat producers, the VSI heat storage is used as an additional, controllable heat storage. The simulation duration for both scenarios is set to 24 hours.

For the baseline scenario, the strongly fluctuating demand within the HNLT leads to many operation switching cycles of the condensing boiler and CHP (see Figure 7a). In addition, the allowed temperature range in the HNHT is very small (see Figure 7b).

By considering the VSI heat storage in the supervisory control strategy, the operating cycle of the heat producers can be significantly reduced (see Figure 7c). The additional storage is used after switching on the CHP and enables a longer operation time of the condensing boiler and the CHP. From 6:30 a.m. to 5 p.m., the heat demand is exclusively covered from the previously loaded VSI heat storage. This leads to a continuously decreasing storage temperature while the buffer storage temperature remains within the allowable range. As soon as the temperature of the VSI heat storage is too low to meet the supply requirements, the condensing boiler and the CHP are started to heat up the buffer storage and then the additional storage. Finally, the simulation study validates that the integration of an additional heat storage including an optimized supervisory control strategy, leads to a reduction of switching cycles from 14 to 2 within 24 hours.

5 Conclusion

The *ThermalSystemsControlLibrary* is the consistent implementation of our previous published data model for the energy-optimized control of industrial energy supply systems. Because of its flexible base classes, which already implement lots of control functions that are necessary for using the model in combination with intelligent control strategies within external frameworks, it allows the rapid modeling of complex infrastructure and enables virtual testing of supervisory control strategies. Moreover, all models implement a data model conform FMI interface, so they can be easily used to validate control strategies operating in external frameworks.

Up to now the TSCL has only been validated for the heat supply systems of the ETA Research Factory. Future work should therefore address modeling other use cases, especially cooling supply systems consisting of more complex supply technologies (e.g. absorption chillers). Furthermore, it should be noted that the library is limited to the modeling of fluid systems. In the future, additional base classes to model sector coupling technologies could be also addressed. Finally, the performance of co-simulations in combination with different optimization approaches should be investigated. This may also include the real-time capability of the modeling approach to enable the implementation of digital twins.

Acknowledgements

The authors gratefully acknowledge financial support of the project *ETA im Bestand* (grant agreement No. 03EN2048A-I) by the Federal Ministry of Economic Affairs and Climate Action (BMWK) and project supervision by the Projektträger Jülich (PtJ).

Data availability

The *ThermalSystemContolLibrary* (TSCL) including the presented use case within the application package is available at Github:

github.com/PTW-TUDa/ThermalSystemsControlLib

References

Blum, D., Arroyo, J., Huang, S., Drgoňa, J., Jorissen, F., Walnum, H. T., . . . Helsen, L. (2021). Building optimization testing framework (BOPTEST) for simulation-based benchmarking of control strategies in buildings. *Journal of Building Performance Simulation*, *14*(5), 586–610. https://doi.org/10.1080/19401493.2021.1986574

Fridgen, G., Keller, R., Körner, M.-F., & Schöpf, M. (2020). A holistic view on sector coupling. *Energy Policy*, *147*, 111913. https://doi.org/10.1016/j.enpol.2020.111913

Fuhrländer-Völker, D., Borst, F., Theisinger, L., Ranzau, H., & Weigold, M. (2022). Modular data model for energy-flexible cyber-physical production systems. *Procedia CIRP*, *107*, 215–220. https://doi.org/10.1016/j.procir.2022.04.036

ISO (11/2017). *ISO 14955-1:2017-11*. (14955).

Lund, H., Østergaard, P. A., Nielsen, T. B., Werner, S., Thorsen, J. E., Gudmundsson, O., . . . Mathiesen, B. V. (2021). Perspectives on fourth and fifth generation district heating. *Energy*, *227*, 120520. https://doi.org/10.1016/j.energy.2021.120520

Modelica Association (2020). ModelicaStandardLibrary: Free (standard conforming) library to model mechanical (1D/3D), electrical (analog, digital, machines), magnetic, thermal, fluid, control systems and hierarchical state machines. Retrieved from https://github.com/modelica/ModelicaStandardLibrary

Modelon (2023, June 27). Modelon Impact Platform: Turn simulation results into business decisions with confidence. Retrieved from https://modelon.com/modelon-impact/

Müller, D., Remmen, P., Constantin, A., Lauster, M. R., & Fuchs, M. *AixLib - An Open-Source Modelica Library within the IEA-EBC Annex60 Framework* (Lehrstuhl für Gebäude- und Raumklimatechnik No. RWTH-2017-00476). Retrieved from Fraunhofer IRB Verlag website: http://publications.rwth-aachen.de/record/681852

Nytsch-Geusen, C., Huber, J., Ljubijankic, M., & Rädler, J. (2013). Modelica BuildingSystems − eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme. *Bauphysik*, *35*(1), 21–29. https://doi.org/10.1002/bapi.201310045

Schneider, G. F., Pessler, G. A., & Steiger, S. (2017). Modelling and Simulation of Standardised Control Functions from Building Automation. In *Linköping Electronic Conference Proceedings, Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017* (pp. 209–218). Linköping University Electronic Press. https://doi.org/10.3384/ecp17132209

Wang, S., & Ma, Z. (2008). Supervisory and Optimal Control of Building HVAC Systems: A Review. *HVAC&R Research*, *14*(1), 3–32. https://doi.org/10.1080/10789669.2008.10390991

Wetter, M., Ehrlich, P., Gautier, A., Grahovac, M., Haves, P., Hu, J., . . . Zhang, K. (2022). OpenBuildingControl: Digitizing the control delivery from building energy modeling to specification, implementation and formal verification. *Energy*, *238*, 121501. https://doi.org/10.1016/j.energy.2021.121501

Wetter, M., Zuo, W., Nouidui, T. S., & Pang, X. (2014). Modelica Buildings library. *Journal of Building Performance Simulation*, *7*(4), 253–270. https://doi.org/10.1080/19401493.2013.765506

Wüllhorst, F., Maier, L., Jansen, D., Kühn, L., Hering, D., & Müller, D. (2022). Besmod - A Modelica Library providing Building Energy System Modules. *Modelica Conferences*, 9–18. https://doi.org/10.3384/ECP211869

Yao, R., Hu, Y., & Varga, L. (2023). Applications of Agent-Based Methods in Multi-Energy Systems—A Systematic Literature Review. *Energies*, *16*(5), 2456. https://doi.org/10.3390/en16052456

HVAC and Control Templates for the Modelica Buildings Library

Antoine Gautier[1] Michael Wetter[2] Jianjun Hu[2] Hubertus Tummescheit[3]

[1]Solamen, France, agautier@solamen.fr
[2]Lawrence Berkeley National Laboratory, Berkeley, CA
[3]Modelon, Hartford, CT

Abstract

This article reports on our experience in creating Modelica classes that serve as templates for modeling HVAC systems with thousands of configurations and closed-loop controls. Our motivation is to reduce model creation and parameterization time, provide access to state-of-the-art control sequences, while limiting the risk of error and enforcing modeling best practices. The development of such templates required exploration of class parameterization techniques and data structures for handling large sets of equipment parameters. By describing these issues and the approach taken, we show how the Modelica language can support advanced templating logic. The main limitation we encountered relates to parameter assignment and propagation. The interpretation of parameter attributes at user interface runtime, or the handling of non-trivial constructs involving record classes at compile time is not consistently supported by Modelica tools. This leads to choices that are difficult to make when looking for a generic implementation.

Keywords: Modelica Buildings Library, template, class parameterization

1 Introduction

Modeling Heating, Ventilation and Air-Conditioning (HVAC) systems with the Modelica Buildings Library (Wetter, Zuo, et al. 2014) usually relies on a component-by-component approach that is both time-consuming and error-prone, requiring expertise in configuring HVAC systems and designing and implementing the appropriate feedback control logic. This motivates the development of pre-built Modelica models that can be easily reconfigured and serve as templates for a variety of HVAC systems.

In our experience, developing Modelica-based templates for complex systems requires not only advanced knowledge of the language, but also experience in template development in particular. To circumvent a high level of complexity, or to allow for a more straightforward development process, some tool developers instead resorted to external templating engines, such as Mako (Nytsch-Geusen et al. 2017) or Jinja (Long et al. 2021). In the latter application, the highly variable network topology of district heating and cooling systems is captured by a GeoJSON parameter schema, from which a Python-based templating layer generates Modelica code. For use of templating engines, in addition to the underlying Modelica models that serve as building blocks for such a workflow, a parameter schema must be developed, along with template files and the software that does the translation into Modelica. This leads to more dependencies to manage, and we believe it also increases the maintenance overhead compared to Modelica-based templating. For example, a change in the underlying Modelica library may require an update of the templates or the translator itself. Moreover, such tools usually implement a one-way trip from the parameter schema to Modelica. Any subsequent change to the Modelica model will therefore result in the configuration workflow no longer being applicable. If the configuration workflow involves programmatic creation of `connect` statements, this will most likely affect the graphical aspect of the model. Finally, a lack of reusability becomes apparent. There are as many template schemas and translators as there are development projects, with no clear way for other applications—even from the same domain—to leverage the existing work.

Alternatively, some developments rely exclusively on the Modelica language and its parametric polymorphism (Broman, Fritzson, and Furic 2006). For example, the Vehicle Dynamics Library developed by Modelon achieves a high degree of configurability through the use of class parameterization techniques, which are described in more detail in this paper. The Vehicle Dynamics Library also provides the ability to specify system parameters via XML, JSON, MAT, or Adams data properties files. The library covers both the chassis and the powertrain configurations of all current architectures of passenger vehicles, as well as many classes of trucks. The hierarchical use of class parameterization gives a high flexibility to include new technologies, and also allows to use different levels of modeling details during the design process. Another example is given by Greenwood et al. (2017) for modeling power plants. The approach uses replaceable elements to configure subsystems and controls, and a record class for parameter assignment within each subsystem. More recently, Wüllhorst et al. (2023) introduced BESMod, an open-source Modelica library for research and teaching purposes that provides a modular approach to domain-coupled simulations of building energy systems. The library is structured with modules that represent the various systems, e.g., demand, ventilation, hydraulic system. Each module is built using expandable connectors,

DOI
10.3384/ecp204217

vector-sized ports, and a unified parameterization framework based on Modelica records. The modules are agnostic of the component models, which can come from various open source libraries such as Buildings or IBPSA (Wetter, Blum, et al. 2019). An example illustrates how building models are interchangeable from one library to another. However, for each module, the configuration options are limited to those of the underlying libraries, and there is still a need for system-level templates. For example, it is not possible to change the system layout and control options of an air handling unit if those features are not present in the library that provides that component.

In this paper we will go over the advantages and disadvantages of Modelica-based templating, which is the method we use. In addition to providing insights to help future template developers, we discuss the main constructs of the Modelica language that serve the purpose of templating, and we point out the limitations and possible language extensions or tool improvements that could make the task easier. We start with some important definitions in section 2 and the key requirements guiding our development in section 3. The core concepts that support templating are introduced in section 4. Some implementation choices related to connecting signal variables, structuring system parameters and integrating graphical elements for control diagrams are then presented in section 5, section 6 and section 7, respectively. Finally, an overview of the test workflow we use to validate the numerous configurations covered by the templates is given in section 8.

2 Definitions

Throughout this article, the following terms are used according to the definitions given here.

Configuration. A system configuration corresponds to the specification of the type and layout of the equipment and the corresponding control logic. Systems with different capacities may have the same configuration, provided they have the same control software and hardware type.

Parameterization. By parameterization we mean all possible class modifications, such as changing parameter values and redeclaring components or classes, which we refer to as class parameterization (Zimmer 2010).

Structural and value parameters. We use the term structural parameters if a parameter affects the number and structure of the equations, and value parameters if they do not. An example of a structural parameter is a parameter used to specify an array size. The use of these terms is consistent with Kågedal and Fritzson (1998).

System. By system we mean a set of components that "share a load in common, i.e., collectively act as a source to downstream equipment, such as a set of chillers in a lead/lag relationship serving air handlers", whereas "each air handler constitutes its own separate system because it does not share a load (terminal unit) in common with the other air handlers". Our use of the term "system" is adopted from ASHRAE (2021).

Template. A template, or template class, is defined as a Modelica model that can be parameterized (as defined above) *to represent a particular system configuration.*

3 Requirements

We will now present key requirements that guided the development of the templates to provide the necessary context for understanding the main implementation choices.

3.1 Tool Compatibility

Our main requirement is that the language constructs used to create the templates are supported by various Modelica compilers. This appeared particularly constraining when dealing with nested expandable connections (see section 5) or choosing the right data structure for system parameters (see section 6). Our test workflow (see section 8) currently includes Dymola (Dassault Systèmes AB 2023), Modelon Impact (Modelon AB 2023b; Modelon AB 2023a), and we are working on support with OpenModelica. In addition, the graphical primitives used for icons and diagrams (see section 7) should also be supported by various Modelica tools, especially if they include a `visible` attribute that requires the evaluation of Boolean expressions at user interface (UI) runtime.

3.2 Diversity of Equipment and Controls

To illustrate the diversity that must be represented, it should be noted that a simple air handling unit can have thousands of possible combinations of equipment, not counting the various control options and the type and placement of sensors required for them.

In addition, there is a strong dependency between the different types of equipment and control logic. For instance, ASHRAE (2021) specifies that the primary hot water flow sensor in a boiler plant is "required for primary-only plants", that the sensor is "optional for variable primary-variable secondary plants" and "not required nor recommended for constant primary-variable secondary plants." In this case, the specification of some equipment (the primary and secondary hot water pumps) together with a control option (the type of sensors used to control the primary recirculation in variable primary-variable secondary systems) constrains the possible options for another piece of equipment (the primary flow sensor), which, if present, can be located either in the supply or in the return pipe.

Conceptually, this means that the user's choices can affect the possible options that are exposed at another level of the model's composition. We will see in subsection 4.3, with a concrete example, the language constructs that are used to support this process.

3.3 System Parameters

The data structure containing the design and operating parameters should allow parameter values to be assigned via a unique object at the top level of the simulation model. Such an object can be viewed as a digital avatar of the manufacturer's data sheets for a complete HVAC system, from plant to zone equipment.

System parameters usually run into the hundreds and are highly dependent from one device to another, so it should be possible to express these relationships in binding equations. In addition, a mechanism should be available to expose only the parameters required for a project's specific system configurations.

Finally, it should be possible to reuse existing equipment datasets implemented as Modelica `record` classes from the library on which the templates are based, e.g., pumps, fans, chillers and boilers for the Buildings library.

We will see in section 6 the resulting implementation choices.

3.4 System Level Templates

The templates need to be provided at the system level, e.g., a central plant or a air handler. Therefore, creating a simulation model for a complete HVAC system involves multiple instances of templates and multiple connections between physical connectors (for fluid circuits) and input/output connectors (for controls). This task should be achievable without need of an automation tool. In practice, this leads to the use of expandable connectors (Modelica Association 2021) to connect control inputs and outputs between systems (see section 5), as otherwise a large set of connections would be required, and this set would vary with each system configuration.

3.5 Scalability

Any number of identical devices must be supported. In practice, this leads to use of array instances for models that can represent multiple units, such as pumps, chillers or zone equipment. The main difficulty then lies in managing this dimensionality for non-trivial constructs such as nested expandable connectors (see section 5) or record classes (see subsection 6.2).

3.6 Integration With OpenBuildingControl

The OpenBuildingControl project aims to digitize the control delivery process based on control specifications that are a subset of Modelica and now being standardized through ASHRAE Standard 231P (Wetter, Grahovac, and Hu 2018; Wetter, Ehrlich, et al. 2022). To support this workflow, the templates shall contain the information necessary to prepare the documents required for the bidding and project execution of HVAC systems.

The ability to generate a control diagram is of particular importance to our development, see section 7. The main requirement is that the data used to create control diagrams be provided as graphical annotations that a Modelica tool with a graphical user interface (GUI) can interpret. This way, when a template is configured in a Modelica tool, the user can get direct graphical feedback on the system layout.

Although outside the scope of this work, additional resources can be exported by dedicated tools, all of which use a JSON representation of Modelica templates as a central digital resource (Wetter, Hu, et al. 2021). These resources include documentation of the sequence of operation, the control point list, or an executable version of the control sequence that control vendors can translate into their product line and commissioning agents can use to verify implementation of the control logic.

4 Structural Changes to a Model

In this section, we describe the mechanism by which we enable structural changes to a model directly through the parameter dialog, i.e., without manual user intervention on the model components. These structural changes allow representing a variety of system layouts and control options—and corresponding sensors and actuators—with a single pre-built model. This mechanism is thus at the core of template design and is based on the concept of class parameterization, which we first introduce in subsection 4.1. To support class parameterization in practice, template components typically need to be derived from interface classes designed to ensure *plug-compatibitility* as described in subsection 4.2. In subsection 4.3, we then present a concrete example of how parameterization constructs are used in conjunction with element annotations to represent dependencies between options for different components of a model and, more broadly, to cover the diversity of equipment and controls as first described in subsection 3.2.

4.1 Class Parameterization

The concept of class parameterization is central to template development. Class parameterization allows a class or component to be used as a parameter of another class. Zimmer (2010) gives an overview of the main language constructs supporting class parameterization in Modelica. Class parameterization can be accomplished via the following alternative approaches:

Container class. A container class, also called a wrapper class, is a class that contains structural parameters that are used to conditionally instantiate the components of the class. The main advantage is that a simple parameter binding with possible expressions can be used to reconfigure the class when instantiating or extending it. The main disadvantage is that the instance tree becomes more complex with additional nesting levels and instance names that vary depending on the system configuration.

Replaceable elements. Replaceable elements, either instances or classes, provide an alternative in which the instance tree is preserved, at least down to the level of the object being replaced. However, as pointed out by Zimmer (2010), these elements cannot be manipulated as standard parameters and require specific syntax (using the keywords `replaceable`, `constrainedby` and `redeclare`) that precludes conditional redeclarations involving expressions and parameters.

There is no single way to achieve the same goals in creating templates, and our developments use a variety of the

System option → Equipment ↓	Mixed air system	100 % outdoor air system
Outdoor air dampers	• Separate dampers for ventilation and economizer • Single damper for ventilation and economizer	• Shut-off damper
Heat recovery	• No heat recovery	• Flat plate heat exchanger • Rotary air-to-air heat exchanger • Run-around coil

Figure 1. Example of equipment options depending on a high-level configuration option for an air handling unit air intake.

above constructs. In this section, we will try to illustrate how these concepts can be put into practice to solve some challenging use cases.

4.2 Interface Class

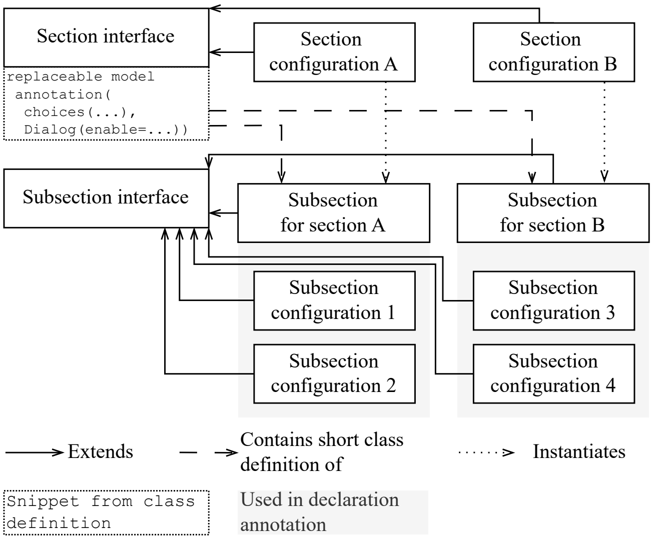
Designing appropriate interface classes is of paramount importance when creating templates. The main goal is to achieve *plug-compatibility* (Modelica Association 2021) for each component model created by extending such interface classes. Applying this concept ensures that all possible connections and parameter assignments can be specified in advance in a template class, so that each time a component is redeclared, no change to the `connect` clauses or binding equations is required.

This differs from the usual practice where interface classes typically contain the minimum common set of elements (e.g., outside connectors and parameters) required by all derived classes, which then extend this set as needed and are thus *type compatible*. In our templates, all outside connectors are declared within the interface class, with the appropriate conditional instance statements. Any class that extends an interface class does not declare any outside connector, but rather conditionally removes inherited connectors. Similarly, the interface class instantiates a record containing the full set of system parameters covering all possible configurations, see section 6.

4.3 Redeclarations and Choices Annotations

4.3.1 Concrete Example

To illustrate the use of class parameterization, let us consider the possible equipment options when specifying the air intake section of a multiple-zone air handling unit. Figure 1 shows these options: In a mixed air system, there are several options for the outdoor air dampers, but usually no heat recovery. In contrast, a 100 % outdoor air system offers several heat recovery options, but should be equipped with a shut-off outdoor air damper. The challenge is to fully cover these options and their constraints, and to select appropriate control logic while minimizing code duplication and maintenance overhead.


Section interface
Section configuration A
Section configuration B
Subsection interface
Subsection for section A
Subsection for section B
Subsection configuration 1
Subsection configuration 3
Subsection configuration 2
Subsection configuration 4
Extends
Contains short class definition of
Instantiates
Snippet from class definition
Used in declaration annotation

```
replaceable model
 annotation(
  choices(...),
  Dialog(enable=...))
```

Figure 2. Class diagram for implementing the air intake section for an air handling unit template.

One might see the lack of conditional class definitions (or redeclarations) in the Modelica language as a limitation when trying to represent dependencies between options for different components of a model. For example, the following construct, although appealing, is not a valid short class definition.

```
class-prefixes IDENT "=" if expression
then type-specifier [ class-modification ]
else type-specifier [ class-modification ]
```

Similarly, conditional `choices` annotations, in which the exposed options depend on a Boolean expression, are not allowed. However, by using different levels of inheritance we can achieve almost the same intent, at the price of a more complicated class structure though.

Figure 2 gives an overview of the classes developed for such purpose, focusing on the options for one type of equipment (e.g., the outdoor air dampers) and considering the most generic case where multiple options exist for each system configuration, i.e., options `1` and `2` for system configuration `A` (mixed air) and options `2` and `3` for system configuration `B` (100 % outdoor air). At the top level of the template, a so-called "section" is declared, which contains all the interdependent components (labelled "subsection" in the figure), e.g., in our case the outdoor air dampers and the heat recovery (not shown in the figure for conciseness). This section derives from an interface containing short class definitions, optionally as replaceable models, and uses these definitions as constructors for its components. As illustrated in Listing 1 the choices specified in the annotation of the replaceable model `Subsection_A` (resp. `Subsection_B`) are only exposed for configuration `"A"` (resp. `"B"`) due to the `enable` attribute in the dialog annotation. Then the model selected from the given choices is used by the derived class `Section_A` (resp. `Section_B`) to create the actual object representing the subsection `subSec`.

Listing 1. Minimal working example illustrating the structure of a template class.

```
model Template "Template"
 replaceable Section_A sec
  constrainedby PartialSection "Section"
  annotation(choices(
   choice(redeclare replaceable Section_A
       sec),
   choice(redeclare replaceable Section_B
       sec)));
end Template;

partial model PartialSection
 "Section interface"
 parameter String config;

 replaceable model Subsection_A = Config1
  constrainedby PartialSubsection
  "Subsection"  annotation(choices(
   choice(redeclare replaceable model
       Subsection_A = Config1),
   choice(redeclare replaceable model
       Subsection_A = Config2)),
   Dialog(enable=config=="A"));

 replaceable model Subsection_B = Config3
  constrainedby PartialSubsection
  "Subsection"   annotation(choices(
   choice(redeclare replaceable model
       Subsection_B = Config3),
   choice(redeclare replaceable model
       Subsection_B = Config4)),
   Dialog(enable=config=="B"));
end PartialSection;

model Section_A "Section config A"
 extends PartialSection(final config="A");
 Subsection_A subSec "Subsection";
end Section_A;

model Section_B "Section config B"
 extends PartialSection(final config="B");
 Subsection_B subSec "Subsection";
end Section_B;

partial model PartialSubsection
 "Subsection interface"
 // Instantiate all possible connectors and
      parameters.
end PartialSubsection;

model Config1 "Subsection config 1"
 // Set parameter values needed to remove
     non—needed connectors, and implement
     actual components.
end Config1;
model Config2 "Subsection config 2"
end Config2;
model Config3 "Subsection config 3"
end Config3;
model Config4 "Subsection config 4"
end Config4;

replaceable Template system;
```

Zimmer (2010) mentions that replaceable *models* are most commonly used instead of replaceable *components* when multiple instances are to be redeclared with a unique statement. Replaceable packages are typically used for medium models because access to enclosed elements (e.g., constants and functions) is required. Our use case differs and replaceable models are used here in conjunction with inheritance and "deferred" instantiation as a practical means of achieving conditional class parameterization and conditional choices for replaceable elements. From a user experience (UX) perspective, this is identical to manipulating a replaceable component. Note that we systematically use `choices` annotations with **`redeclare replaceable`** to support further editing of the template after a configuration workflow.

4.3.2 Caveats and Alternatives

Resorting to UI features does not provide the same degree of robustness as using pure language constructs for object manipulation. For example, with a single line of code, one could manually redeclare `Subsection_A` with any type compatible model and violate the constraints imposed by the `choices` annotation.

One could also argue that for the configurations described in Figure 1, where there is either an option list or a *unique* option, as opposed to another option list, a simpler construct is to declare a replaceable *component* in the air intake section to represent the outdoor air dampers, with an `enable` attribute that evaluates to `true` for the mixed air configuration, and to `false` for the 100 % outdoor air configuration. For the latter case, an additional **`redeclare`** `final` statement then enforces the unique option for the outdoor air dampers (i.e., shut-off damper). However, additional scrutiny on the configuration workflow is here necessary. Indeed, an issue appears if a system model is first created by extending the template with a class modification that pertains to the mixed air configuration. Any further modification of the air intake section, either during inheritance or instantiation, risks an error due to a final override in the merging of modifiers, as shown below.

```
partial model PartialSection
 "Section interface"
 parameter String config;

 replaceable Config1 subSec "Subsection"
 annotation(choices(
  choice(redeclare replaceable Config1
      subSec),
  choice(redeclare replaceable Config2
      subSec)),
  Dialog(enable=config=="A"));
end PartialSection;

model Section_A "Section config A"
 extends PartialSection(final config="A");
end Section_A;

model Section_B "Section config B"
 extends PartialSection(final config="B",
  redeclare final Config3 subSec);
end Section_B;
```

```
model System1
 extends Template(sec(redeclare replaceable
      Config2 subSec));
end System1;
// The following yields a final override
    error.
System1 system(redeclare Section_B sec);
```

Another limitation arises from the fact that no parameter dialog is generated for the subcomponent redeclared as final, so that other configuration options nested below it are not accessible to the user.

As an alternative, an annotation override may be considered. Indeed, Modelica Association (2021) specifies that a `description` is allowed as part of an `element-redeclaration`. So, if the original component is replaceable, the concrete syntax allows an `annotation-clause` when redeclaring the component. If a replaceable component `subSec` is declared inside `PartialSection` as before, the same configuration logic as in Listing 1 could be implemented as follows.

```
model Section_B "Section config B"
 extends PartialSection(
  redeclare replaceable Config3 subSec
  annotation(choices(
   choice(redeclare replaceable Config3
       subSec),
   choice(redeclare replaceable Config4
       subSec))));
end Section_B;
```

However, although it conforms with Modelica Association (2021), the above syntax is not interpreted by any of the Modelica tools we tested.

5 Connections

The `connect` equations for signal variables in the templates use expandable connectors, also called control busses, which have the following useful properties. The set of variables in an expandable connector is augmented whenever a new variable is connected to an instance of the class. Thus, there is no requirement to pre-declare variables, and in fact we do not pre-declare any variable within the control bus. Variables that are potentially present but not connected are eventually considered as undefined, i.e., a tool may remove them or set them to a default value. Not all variables need to be connected, and therefore the control bus does not need to be reconfigured depending on the model structure.

Like any other Modelica type, expandable connectors can be used in array instances. A typical use case is to connect control signals from a set of terminal units to a supervisory controller of an air handling unit. In our experience, some care is required when handling such array instances to maximize support by different Modelica compilers, especially when nested expandable connectors are involved. We have opted for the pragmatic rule of limiting the number of nested expandable connectors to one. In other words, a control bus may have none or one sub-bus.

Also, we use local instances of array sub-busses to force the compilers to assign the dimensionality to the correct variable. For instance, let us consider the following model where the variables nested under the `bus` object are not pre-declared.

```
model Control
 parameter Integer nDim1 = 2, nDim2 = 1;
 Modelica.Blocks.Examples.
     BusUsage_Utilities.Interfaces.
     ControlBus bus;
 Modelica.Blocks.Sources.RealExpression exp
     [nDim1, nDim2](y=fill(fill(1, nDim2),
     nDim1));
equation
 connect(exp.y, bus.subbus.y);
end Control;
```

Some compilers assign the dimensionality of `exp.y` (that is equal to two) to `bus.subbus.y`, while the template developer may expect both `bus.subbus` and `bus.subbus.y` to have a dimensionality of one. Other compilers will reject such a model. Therefore, the following implementation is used instead.

```
model Control
 parameter Integer nDim1 = 2, nDim2 = 1;
 Modelica.Blocks.Examples.
     BusUsage_Utilities.Interfaces.
     ControlBus bus;
 Modelica.Blocks.Sources.RealExpression exp
     [nDim1, nDim2](y=fill(fill(1, nDim2),
     nDim1));
protected
 Modelica.Blocks.Examples.
     BusUsage_Utilities.Interfaces.
     SubControlBus subbus[nDim1];
equation
 connect(exp.y, subbus.y);
 connect(subbus, bus.subbus);
end Control;
```

Most compilers we have tested can handle the above implementation, and `bus.subbus` and `bus.subbus.y` are necessarily assigned a dimensionality of one. Furthermore, having the instance of the sub-bus as a protected element of the control block instead of a pre-declared variable inside the main control bus avoids binding equations for the dimension parameters wherever the control bus is used.

Finally, we use strict naming conventions for all components, including signal variables, which support a natural syntax for the `connect` equations. For example, connecting the measurement signal yielded by the supply air temperature sensor component to the bus variable used by the controller is done with: `connect(TAirSup.y, bus.TAirSup)`. Connecting the supply fan command and feedback signals to the corresponding sub-bus is done with: `connect(fanSup.bus, bus.fanSup)`. Connecting all signal variables for the air intake section described in subsection 4.3 is done with: `connect(secAirInt.bus, bus)`, where the main bus of the template is used in the section class, because the section itself contains nested components with sub-busses (such as dampers), so the number of nested levels is effectively limited to one.

6 Design and Operating Parameters

When trying to meet the requirements of subsection 3.3, the main difficulty is that conditional declarations cannot be used for parameters, since a "component declared with a condition attribute can only be modified and/or used in connections" (Modelica Association 2021). We therefore considered the use of an external data file and the use of record classes and chose the latter in our implementation for the reasons explained below.

6.1 External Data File

At first glance, using an external data source, such as a JSON parameter file, is promising because it eliminates the need for parameter propagation. Instead, each component can retrieve the required parameter values by invoking accessor functions. The Modelica library ExternData (Beutlich and Winkler 2021), which we used for this purpose, provides accessor functions for each predefined variable type (`Real`, `Integer`, `Boolean`, `String`) with dimensionality up to 2.

However, we had to scale back our original requirements from subsection 3.3 due to the inherent limitations of using literal constants from an external file. For instance, referencing existing equipment data records from the Modelica Buildings Library is not possible because a class cannot be instantiated by passing the class name as a string. Similarly, binding equations cannot be used to express relationships between parameters of different systems because there is no built-in function to interpret a string as Modelica code and evaluate it. Also left open is the question of how to create the structure of this external data file so that only the parameters required for the actual system configurations are exposed, although automation could address this problem.

More importantly, many structural parameters need to be stored in the external data file, for example, to specify the size of the parameter array for a multi-unit component. Ideally, the value for these parameters should be assigned in this file. However, as structural parameters, they must be evaluated at compile time. FMI-compliant compilers (such as Modelon Impact) can handle this well as the compiler flags that the accessor function must be executed at compile time. This is not the case with other compilers. Dymola, for example, requires the function annotation `__Dymola_translate = true` to force compile time function execution, even though the parameter declaration is already annotated with `Evaluate = true`. This raises concerns about the impact on the translation time, since *each* function call requires the creation of an external object and access to the external file with `fopen`, even though all function calls target the same external file.

It gets even worse when the compiler also stores the values for some variable attributes in the translated model, either by legacy (e.g., `nominal`) or because they are used in symbolic processing (e.g., `min` and `max`). Then all *value* parameters used in bindings of these attributes have to be evaluated as well, causing a significant translation overhead. This marked the end of our attempt to use an external parameter file.

6.2 Record Class

As an alternative, we resorted to record classes to handle parameter assignment from the top level and propagation throughout the instance tree. The use of record classes benefits from the following features of the Modelica language. Records are the only composite components allowed in bindings, and the only composite components of an instance that can be accessed by another instance. Thus, the following conforms with Modelica Association (2021) and records are the only specialized classes that support such constructs, which largely help reduce the number of binding equations when propagating parameters.

```
record Rt "Template record"
 parameter Rc c;
end Rt;

record Rc "Component record"
  parameter Real p=1;
end Rc;

model Template
 parameter Rt dat;
 Component c(final dat=dat.c);
end Template;

model Component "Component"
 parameter Rc dat;
end Component;
```

Specifically, in our case, two record classes are developed for each component model and instantiated within the model's interface class. The first record contains all configuration parameters (structural parameters). This can be considered as the "signature" for a given system configuration, accessible from any component and any template. The second record contains the *full set* of design and operating parameters (value parameters) covering all possible configurations, as well as an instance of the first configuration record.

The inclusion of configuration parameters makes it possible to disable input fields in the parameter dialog if the parameters are not needed for a particular configuration by using the annotation attribute `enable`. Also, if `enable = false`, no value can be assigned to this parameter (Modelica Association 2021) although this is a non-normative part of the language specification, and some compilers may issue warnings or errors if no value is assigned to a parameter, even with `enable = false`. In addition, some compilers require that the `start` attribute be assigned for parameters that have no assignment.

The inclusion of all value parameters is a requirement to ensure plug compatibility with the interface class, which serves as the constraining type and supports parameter propagation via a single binding equation.

Although simple by design, the implementation of this parameterization logic proves tricky, as shown in Listing 2.

Listing 2. Minimal working example illustrating the use of parameter records.

```
model Template
 final parameter ConfigTemplate config(
  comp=comp.config) "Configuration record";
 parameter DataTemplate data(
  config=config) "Data record";
 Component comp(final data=data.comp)
  "Component";
end Template;

model Component
 parameter Integer typ;
 final parameter ConfigComponent config(
  typ=typ);
 parameter DataComponent data(config=config
    );
 // Parameter myPar needed only if typ==1.
 SubComponent1 sub1(myPar=data.myPar)
  if typ==1;
end Component;

model SubComponent1
 parameter Real myPar;
end SubComponent1;

record DataTemplate
 // Annotation enable used in lieu of final
      binding in instance to avoid final
     override.
 parameter ConfigTemplate config
  annotation(Dialog(enable=false));
 parameter DataComponent comp(
  config=config.comp);
end DataTemplate;

record DataComponent
 parameter ConfigComponent config
  annotation(Dialog(enable=false));
 // Explicit start attribute is needed to
     avoid the warning: "The following
     parameters don't have any value"
 parameter Real myPar(start=0)
  annotation(Dialog(enable=config.typ==1));
end DataComponent;

record ConfigTemplate
 parameter ConfigComponent comp;
end ConfigTemplate;

record ConfigComponent
 // Annotation Evaluate is needed to avoid
     the warning: "The following parameters
      don't have any value"
 parameter Integer typ
  annotation(Evaluate=true);
end ConfigComponent;

// To use the template, they can be
    instantiated as follows:
// If typ=1, the data need to be set to
    assign parameter myPar.
replaceable Template system1(
 comp(typ=1), data(comp(myPar=1)));

// The following instance does not require
    assigning parameter myPar.
replaceable Template system2(comp(typ=2));
```

The comments inserted in this listing give an insight into our experience and show that some ad hoc rules from the various Modelica tool vendors make a generic implementation difficult and lead to a lengthy trial-and-error process. Also, a parameter with `enable = false` remains in the variable namespace, and compilers use the value of the `start` attribute to initialize the parameter when it is unassigned. So we still need to assign a value to this attribute and guard against corner cases, such as division by zero of another variable attribute that has a binding with this parameter, or zero-sized arrays of records. The situation gets worse when dealing with UI/UX features as, in the above example, the input field for the parameter `system.data.comp.m_flow_nominal` may remain enabled in the parameter dialog, even though the configuration parameter `system.data.comp.config.typ` evaluates to `2`. This is the case with many Modelica tools we tested, even for such a minimal example where class name lookup is kept as simple as possible. In practice, and as illustrated in subsection 4.3, templates require complex class structures with multiple levels of inheritance and composition, or the use of **`outer`** components that further limit the support by various tools.

Our ultimate goal entails even more demanding requirements for interpreting annotation attributes at UI runtime. Specifically, we want to read the configuration parameters of template instances from an object at the top level of the simulation model, such as with the following construct. Note that we use a generic **`class`** construct as opposed to the specialized class **`record`** because the latter does not allow for **`outer`** elements.

```
class DataAllSystems
 outer Template system;
 parameter DataTemplate data_system(
  config=system.config);
end DataAllSystems;

parameter DataAllSystems data;

inner replaceable Template system(
 comp(typ=2),
 final data=data.data_system);
```

The top-level component `data` can thus serve as a single object for storing all design and operating parameters, displaying only those required for the particular system configurations, thus satisfying the requirements of subsection 3.3. This structure is composed of records for each system and its components, so existing equipment data records from the library can be easily reused. Vectorized instances are also possible for systems with multiple units, and different *classes* can be used in them as long as they have the same *type*. Parameterization of multiple units with different properties is therefore straightforward and most compilers allow these arrays to be populated on-the-fly using record functions as shown below.

```
parameter ElectricReformulatedEIR.Generic
    data[2] = {
 ElectricReformulatedEIR.
     Carrier_19XR_1234kW_5_39COP_VSD(),
 ElectricReformulatedEIR.
     Carrier_19XR_1143kW_6_57COP_VSD()};
```

All in all, the use of Modelica records offers great potential, but suffers from uneven support from various compilers, especially when using array instances or evaluating the enable attribute at UI runtime, and for use cases requiring complex class structures. At the beginning of the development of system templates, we certainly did not expect that the handling of *value* parameters would be the most difficult part and take the most development time—estimated to be over 30 %—with a final result that we still judge to be far from optimal. We believe that the lack of conditional declarations of *value* parameters is the biggest obstacle to the development of complex system templates and makes the parametric polymorphism of the Modelica language cumbersome in practice.

7 Control Diagram

To meet the requirements of subsection 3.6, all data needed to create the diagram for a given system are included in the template class in the form of Modelica graphical annotations. Figure 3 shows the diagram view of the boiler plant template and gives an example of the direct graphical feedback that can be obtained about a particular system configuration. It also illustrates the capability of the template to adapt to different equipment and control specifications by programmatically generating the necessary objects for equipment, actuators and sensors, and resolving the hydronic routing of components and control signal connections without user intervention.

The template components include Bitmap primitives in the icon layer to reference equipment symbols provided as vector graphics in SVG format (W3C 2003). Although Modelica tools render them as raster images in the diagram view, a tool can use the referenced SVG files to create diagrams that conform to industry standards and are accurate at the pixel level. The visibility of these graphical objects is controlled with the annotation attribute visible and bindings to the template class configuration parameters. Due to the lack of a "group" element in the Modelica Language Specification—as opposed to the SVG specification (W3C 2003) which includes the 'g' element—dealing with complex graphical objects using only native graphical primitives would require many duplicates of the visible attribute and its binding equation, which is the main reason we rely on external SVG files. Text in equipment symbols is handled separately, so it can be flipped or rotated independently of the component symbol.

Piping systems are represented directly by the graphical annotations of the connect equations, with an explicit visible attribute added in the case of conditional components. This is necessary because Modelica tools, while removing the corresponding connect statements at translation, generally do not provide direct graphical feedback and the connection lines remain visible in the diagram view. In the case of air systems, we use separate graphical annotations in the diagram layer to represent the ductwork. The connection lines are then graphical artifacts that should be deleted when creating the final control diagram.

8 Validation

We have implemented a comprehensive test workflow to verify that all system configurations supported by a given template are implemented correctly. However, generating the list of these supported configurations was not straightforward, considering that elaborate class parameterization techniques are used together with choices annotations to implement the actual decision tree (see subsection 4.3). Thus, we recreated this set of options in a standalone script that first builds the list of all possible combinations of structural parameters and **redeclare** clauses, and then prunes this list based on exclusion patterns that must be manually specified by the template developer. Then, simulations are run for all the resulting class modifications. For example, this results in over 2000 simulations for the chiller plant template and nearly 1000 simulations for the boiler plant template. Due to the computation load, we only trigger these tests in our continuous integration workflow when the checksum computed for all classes in the Templates package changes.

Currently, the percentage of tool coverage is about 60 %,[1] but is steadily growing with the updates of the Modelica compilers released by various vendors.

9 Conclusion and Future Work

Our experience with relying entirely on the Modelica language to create user-friendly models for systems with thousands of configurations and closed-loop controls has shown that complexity is not necessarily where one expects it to be. Originally, we saw the lack of conditional element redeclaration as the main obstacle, especially since there is convincing work already pointing out this deficiency and proposing some changes to allow for better class parameterization (Zimmer 2010). Although these proposed changes would have made our task easier, it appeared to us that advanced model configuration could be achieved with the current syntax, provided that com-

[1] The percentage of tool coverage is calculated as the ratio between the number of successful tests and the number of tests, where the number of tests is equal to the number of templates times the number of compilers tested.

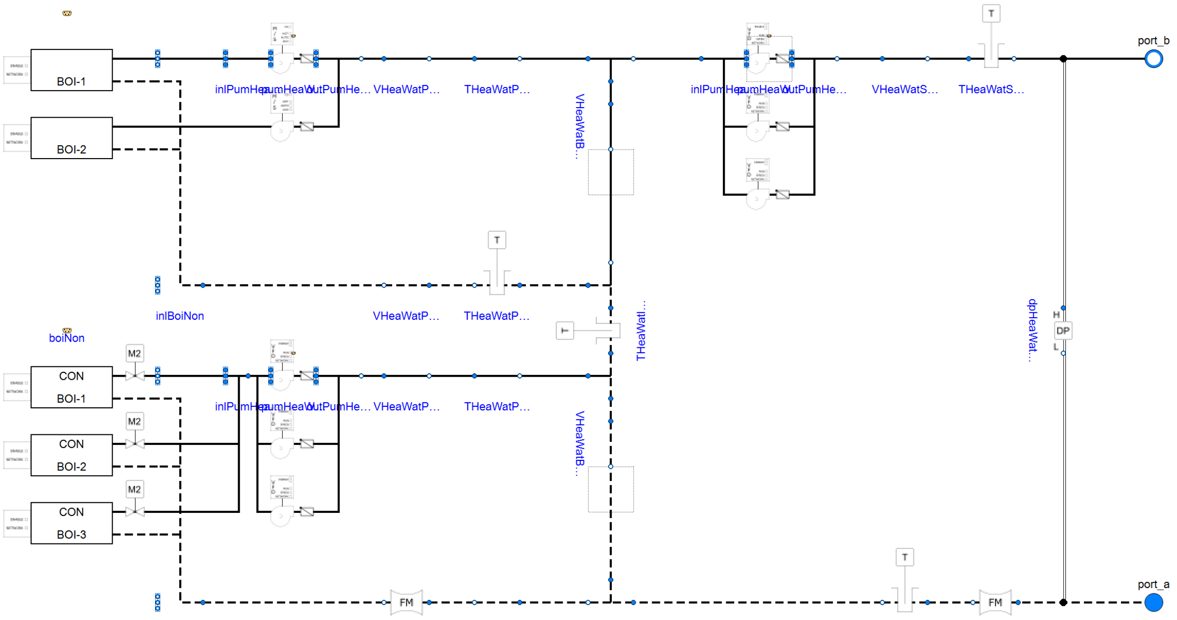


Figure 3. Diagram view of the boiler plant template (as rendered by Dymola) in a primary-secondary configuration, with three condensing boilers equipped with headered variable speed pumps and two non-condensing boilers equipped with dedicated constant speed pumps.

ponents have well-designed interface classes and that UI features are used in conjunction with pure language constructs for object manipulation.

However, the assignment of equipment parameters and their propagation throughout the instance tree proved to be the most problematic. The main reason is that the conditional declaration of parameters used in binding equations, as opposed to connectable components, is not allowed. We believe that this restriction is what most limits the templating capabilities of the Modelica language. In the absence of such a feature, we are left with only a non-normative part of the specification, namely the use of the attributes `enable` and `start` in parameter declarations. With this pattern, a parameter can remain unassigned if it is not needed for a given configuration where `enable` evaluates to `false`. But this approach is fraught with some issues. First, the fact that it is a non-normative feature limits the support of our templates by various Modelica compilers, which may issue a warning or error in case of unassigned parameters. Second, a disabled parameter remains in the variable namespace and compilers use the value of the `start` attribute to initialize the parameter if it is unassigned. So we still need to assign a value to this attribute, and several corner cases occur that we need to guard against. Finally, the behavior of the UI is almost unpredictable because the interpretation of the `enable` attribute at UI runtime to generate the parameter dialog is not specified. Thus, if the logic to disable a parameter input field fails, it is difficult to determine which constructs are the cause. We believe that a unified UX matters, especially for templating, and that it would be good for the Modelica community if more UI features were normative and much more clearly specified than currently. The uneven support by Modelica compilers of another fundamental structure for handling parameters further complicates the task. Indeed, we have observed and reported many compiler failures with record classes used in non-trivial constructs such as array instances, composite component bindings, or on-the-fly instantiation with record functions. Thus, development work often becomes a tricky navigation around the specific limitations of the various Modelica compilers.

Our next step is to implement templates for entire HVAC systems, from plant to zone equipment, with coupling to thermal zone models. This means assembling templates from templates, and for highly scalable systems. The main difficulty that we foresee arises from the constraint that array elements must be of the same type, which is rarely the case with class parameterization techniques that only aim at type or plug compatibility. Again, we think that the suggestions from Zimmer (2010) would be very valuable to use an array of model parameters for this purpose. We have some alternatives, such as building container classes around templates, or using multiple array instances for the same system type. This also means that our developments have not yet reached the highest level of complexity, even though it sometimes seems that they have already exceeded the complexity that most compilers can handle.

10 Data Availability

The templates discussed in this paper are available in the feature branch `issue3266_template_HW_plant` from commit `e15d845`, and are planned to be released in future versions of the Modelica Buildings Library.

Acknowledgements

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231.

References

ASHRAE (2021). *Guideline 36: High-Performance Sequences of Operation for HVAC Systems*. Guideline. Atlanta, GA.

Beutlich, Thomas and Dietmar Winkler (2021). “Efficient Parameterization of Modelica Models”. In: *Proceedings of 14th Modelica Conference*. Linköping, Sweden. DOI: 10.3384/ecp21181141.

Broman, David, Peter Fritzson, and Sébastien Furic (2006-09). “Types in the Modelica Language”. In: *Proceedings of the 5th International Modelica Conference*. Vienna, Austria: The Modelica Association and arsenal research. URL: https://modelica.org/events/modelica2006/Proceedings/sessions/Session3c3.pdf.

Dassault Systèmes AB (2023-03). *Dymola: Dynamic Modeling Laboratory. Full User Manual (Dymola 2023x Refresh 1)*. Lund, Sweden.

Greenwood, Michael Scott et al. (2017). “A Templated Approach for Multi-Physics Modeling of Hybrid Energy Systems in Modelica”. In: DOI: 10.2172/1427611.

Kågedal, David and Peter Fritzson (1998-07). “Generating a Modelica Compiler From Natural Semantics Specifications”. In: *Proceedings of the Summer Computer Simulation Conference*. Reno, Nevada.

Long, Nicholas et al. (2021). “Modeling District Heating and Cooling Systems With URBANopt, GeoJSON to Modelica Translator, and the Modelica Buildings Library”. In: *Proceedings of the Building Simulation Conference*. DOI: 10.26868/25222708.2021.30943.

Modelica Association (2021-02). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.5*. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.5/MLS.html.

Modelon AB (2023a). *Modelon Impact Helpcenter*. Lund, Sweden. URL: https://help.modelon.com/latest/.

Modelon AB (2023b). *OPTIMICA Compiler Toolkit. Version 1.43.4*. Lund, Sweden.

Nytsch-Geusen, Christoph et al. (2017). “Template Based Code Generation of Modelica Building Energy Simulation Models”. In: *Proceedings of the 12th International Modelica Conference*. Prague, Czech Republic: Linköping University Electronic Press. DOI: 10.3384/ecp17132199.

W3C (2003-01). *Scalable Vector Graphics (SVG) 1.1 Specification*. Tech. rep. URL: https://www.w3.org/TR/2003/REC-SVG11-20030114/REC-SVG11-20030114.pdf.

Wetter, Michael, David Blum, et al. (2019-05). *Modelica IBPSA Library v1*. DOI: 10.11578/dc.20190520.1.

Wetter, Michael, Paul Ehrlich, et al. (2022). “OpenBuildingControl: Digitizing the Control Delivery From Building Energy Modeling to Specification, Implementation and Formal Verification”. In: *Energy* 238, p. 121501. DOI: https://doi.org/10.1016/j.energy.2021.121501.

Wetter, Michael, Milica Grahovac, and Jianjun Hu (2018-08). “Control Description Language”. In: *1st American Modelica Conference*. Cambridge, MA, USA. DOI: 10.3384/ecp1815417.

Wetter, Michael, Jianjun Hu, et al. (2021). “Modelica-json: Transforming Energy Models to Digitize the Control Delivery Process”. In: *Proceedings of the IBPSA Building Simulation Conference*. Brugge, Belgium.

Wetter, Michael, Wangda Zuo, et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Wüllhorst, Fabian et al. (2023-02). “BESMod - A Modelica Library pProviding Building Energy System Modules”. In: pp. 9–18. DOI: 10.3384/ECP211869.

Zimmer, Dirk (2010). “Towards Improved Class Parameterization and Class Generation in Modelica”. In: *Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. Oslo, Norway: Linköping University Electronic Press. URL: https://ep.liu.se/ecp/047/004/ecp4710004.pdf.

Modeling and Simulation of the Hydrogen Value Chain with ThermoSysPro and Modelica

Sebastian Vallejo Jimenez[1] Luis Corona Mesa-Moles[2] Damien Faille[2] Dina Irofti[2]

[1]ENSTA Paris, France, sebastian.vallejo-jimenez@ensta-paris.fr
[2]EDF R&D, France, {luis.corona-mesa-moles, damien.faille, dina.irofti}@edf.fr

Abstract

Hydrogen will play a key role in the global energy transition if we are able to produce it with low-carbon emissions. However, clean hydrogen production today is mostly limited to demonstration projects ranging from 2 up to 20 MW. Modeling the way low-carbon hydrogen is produced, stored and used will allow us to significantly improve our understanding of how clean hydrogen could be produced and thus increase the production efficiency. In this paper, we show that the newest version of ThermoSysPro (TSP), an open-source Modelica library for modeling energy systems, provides a suitable framework to model and simulate the hydrogen production, storage and consumption. The model presented in this paper contains three electrolyzers, a storage station and a vehicle station. We present how the model was built, which components were adapted and how, and show that its simulation can be useful in the design phase, as well as for diagnosis purposes. In the future, a complete validation of these developments will be performed when experimental data is publicly available.

Keywords: Modelica, ThermoSysPro, Hydrogen, Real gas, Equation Of State

1 Introduction

The global energy system is undergoing major changes, (International Energy Agency 2022b). This transformation aims mostly at reducing the oil and other fossil fuels dependency in order to cut as much as possible CO_2 emissions and thus the consequences of global warming. In this context, hydrogen plays a crucial role in accelerating this energy transition by allowing the decarbonisation of key sectors like industry, aviation, shipping or heavy-duty transport, (International Energy Agency 2022a; *Green hydrogen cost reduction* 2020).

Hydrogen has already been identified as indispensable in the strategy to build a climate-neutral Europe (European Commission 2020). However, it is still mostly produced with natural gas, which represents 96% of the total production (European Commission 2023). For this reason, and in order to accomplish the ambitious climate objectives, it is extremely important to develop technologies able of producing clean hydrogen (i.e. with no or low CO_2 emissions) at a large scale and in a competitive way. To successfully develop these hydrogen related technologies and integrate them in the future energy mix, the development of flexible and adequate modeling tools is also fundamental. This will allow to contribute and accelerate the ongoing energy transition. Hydrogen based systems can be modeled with different types of modeling tools. Without attempting to be exhaustive, in the literature it is possible to find Matlab (Khan and Iqbal 2005; Eriksson and Gray 2017; Bhuyan, Hota, and Panda 2018), Python (Kuroki et al. 2021) and Modelica (Migoni et al. 2016) based models. These models aim at representing different aspects of a hydrogen based system (electrolyzers, vehicle hydrogen fueling, storage, etc.) and for different purposes (optimization of the management of hydrogen based technologies, better understanding of the physical behavior of such systems, etc.).

Hydrogen based systems usually require different types of physics to be described and this is why the Modelica language can be suitable to describe such systems. In particular, it can be interesting to adapt existing Modelica libraries that have already shown relevant and trustful results in similar domains. This is the case of the ThermoSysPro (TSP) library[1](El Hefni and Bouskela 2019), an open-source Modelica library developed by EDF R&D[2], which has already been used to successfully model other energy systems such as power plants, industrial processes, energy conversion systems etc. (El Hefni and Bouskela 2017). To achieve these results, different physics are already available in TSP such as thermal-hydraulics, combustion, solar radiation and neutronics. In addition, to these already existing features, the latest version of ThermoSysPro[3], the so-called V4, allows to easily modify the fluid considered for modeling in addition of giving the possibility to handle efficiently zero-mass flow-rates configurations (based on the principles explained in (Bouskela and El Hefni 2014b). For the reasons previously mentioned, the ThermoSysPro library appears as suitable starting point to model and simulate hydrogen based systems.

[1] https://thermosyspro.com

[2] ThermoSysPro is compatible with OpenModelica and many organizations and individuals use currently ThermoSysPro around the world to model different types of energy systems.

[3] ThermoSysPro is freely available here: https://github.com/ThermoSysPro/ThermoSysPro.

The goal of this paper is therefore to expose and show how the latest version of the ThermoSysPro library has been adapted to efficiently model and simulate hydrogen based systems that already exist or that are going to exist in the future. To illustrate the modification of ThermoSysPro previously described, this paper focus on the modeling and simulation of a low-carbon hydrogen platform in which the whole value chain of hydrogen is considered (production, storage and distribution, and consumption/application (TÜV SÜD 2023)).

This paper is structured as follows. Section 2 describes the model considered in this study as well as the equations implemented in the new developed modules. Section 3 presents the construction of the model as well as the simulations performed with this model. Finally, section 4 presents the discussion as well as the future work that can be initiated with the promising results presented in this paper.

2 Model Description and Equations

In this section, the proposed low-carbon hydrogen platform is presented as well as the components that constitute a library dedicated to hydrogen which is compatible with ThermoSysPro. As mentioned previously, TSP is already available and thus provides certain advantages like the possibility of reusing some of its components in this hydrogen platform.

2.1 Gas Plate Scheme

The proposed platform is denominated Gas Plate and it serves to connect the components of the system. Such components include mainly, as seen in the simplified model scheme on Figure 1, electrolyzers and a set of blocks as valves, compressors and heat exchangers to compose two branches: Storage and Station, each representing one of the considered final applications. First of all, the block corresponding to the electrolyzer allows to estimate hydrogen production through water electrolysis at given conditions. Such hydrogen gas will then go through a check valve, ensuring one-directional flow in the platform. Afterwards a compressor and a heat exchanger block will allow to reach the desired conditions in terms of temperature and pressure depending on each final use: either storing it directly or manipulating it in a filling station. Finally, to ensure the correct flow of hydrogen, a control valve will be used for each one of these so-called branches.

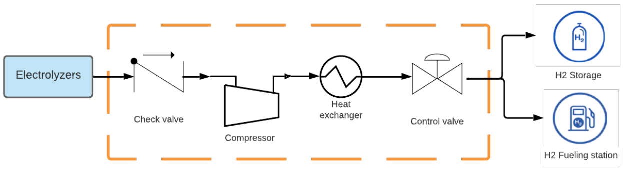


Figure 1. Case study model

2.2 Notations

C_p	$\frac{J}{kgK}$	Isobaric heat capacity
C_v	$\frac{m^4}{sN^5}$	Flow coefficient
C_V	$\frac{J}{kgK}$	Isochoric heat capacity
C_V^0	$\frac{J}{kgK}$	Ideal gas isochoric heat capacity
$\Delta_r g^0$	$\frac{J}{mol}$	Reference specific Gibbs free energy difference for the reaction
$\Delta_r h^0$	$\frac{J}{mol}$	Difference in specific enthalpy of reference for the reaction
ΔT_{LM}	K, °C	Logarithmic mean temperature difference
η	-	Energy efficiency
η_f	-	Faraday efficiency
$\eta_{is,comp}$	-	Isentropic compression efficiency
F	$96485\frac{C}{mol}$	Faraday constant
h	$\frac{J}{kg}$	Specific enthalpy
I	A	Current
$\dot{m}$	$\frac{kg}{s}$	Mass flow rate
LHV_{H_2}	$\frac{J}{kg}$	Low Heat Value for hydrogen
P	bar, Pa	Absolute pressure
P_c	bar, Pa	Critical pressure
P_{vH_2O}	bar, Pa	Vapor pressure
R	$8.31\frac{J}{molK}$	Ideal gas constant
s	$\frac{J}{kg}$	Specific entropy
T	K, °C	Temperature
T_c	K, °C	Critical temperature
T_r	-	Reduced temperature
u	$\frac{J}{kg}$	Specific internal energy
U	$\frac{W}{m^2K}$	Overall heat transfer coefficient
U_{act}	V	Activation over-potential
U_{cell}	V	Cell voltage
U_{res}	V	Resistive over-voltage
U_{rev}	V	Reversible voltage $:= \Delta E_{eq}$
μ_{JT}	$\frac{K}{bar}$	Joule-Thomson coefficient
v	$\frac{m^3}{mol}$	Molar volume
ω	-	Acentric factor

2.3 Pure Real Gas Properties

The hydrogen platform operates at a pressure ranging from a few decades to several hundred bars. At high pressure, the ideal gas assumption is not precise enough and the use of an Equation of State for real gas is preferable. The so-called Peng-Robinson Equation of State (Equation 1)[4] can be found in the classical literature of thermodynamics and process engineering (Zohuri 2018):

$$P = \frac{RT}{v-b} - \frac{\alpha a}{v^2 + 2bv - b^2} \tag{1}$$

$$a = 0.45723553\frac{R^2T_c^2}{P_c} \ ; \ b = 0.07779607\frac{RT_c}{P_c} \ ;$$

[4] From now on referred as PR-EOS

$$\alpha = \left(1 + k\left(1 - \sqrt{\frac{T}{T_c}}\right)\right)^2 \;;$$

$$k = 0.37464 + 1.54226\omega - 0.26993\omega^2 \;;$$

$$\omega = -1 - \log\left(\frac{P^{sat}}{P_c}\right)_{T_r=0.7},$$

where "a" accounts for inter-molecular attractive forces (it decreases the pressure that molecules exert on the reservoir (Zohuri 2018)), while "b" represents the volume occupied by the gas molecules. The molar volume is represented by v, P^{sat} is the saturation pressure of the fluid (to find the acentric factor ω we use the value of P^{sat} considering $T_r = \frac{T}{T_c} = 0.7$) and R is the universal gas constant.

Taking the total derivative of the internal energy ($u = f(T,v)$) and after development using Maxwell's third relation, we obtain the following expression verified by (Cengel and Boles 2015) where the pressure P is defined by the PR-EOS and the isochoric heat capacity C_V is unknown.

$$\Delta u = \int_{T_1}^{T_2} C_V dT + \int_{v_1}^{v_2}\left[T\left(\frac{\partial P}{\partial T}\right)_v - P\right]dv \quad (2)$$

Concerning Modelica, a function called *PR_InternalEnergy* (containing the Equation 4) is created after having developed the Equation 2 considering the state 1 as $v_0 = \infty$ and using the PR-EOS Equation 1 (Trujillo, O'Rourke, and Torres n.d.):

$$u(v,T) - u(v_0,T_0) = u(v_0,T) - u(v_0,T_0) + \int_{v_0}^{v}\left[T\left(\frac{\partial P}{\partial T}\right)_v - P\right]dv \quad (3)$$

$$u(v,T) = u^0 + a\left(\alpha - T\frac{\mathrm{d}\alpha}{\mathrm{d}T}\right)\frac{1}{2\sqrt{2}b}\ln\left[\frac{v + b(1-\sqrt{2})}{v + b(1+\sqrt{2})}\right] \quad (4)$$

With :

$$\frac{\mathrm{d}\alpha}{\mathrm{d}T} = \frac{-k}{\sqrt{T_c T}}\sqrt{\alpha} \quad (5)$$

Where $u^0 = C_V^0 T$, and C_V^0 corresponds to the isochoric heat capacity of perfect gases. Then, according to (Leachman et al. 2009) one can express such isochoric heat capacity according to the following expression:

$$\frac{C_V^0}{R} = 1.5 + \sum_{k=1}^{N} u_k\left(\frac{v_k}{T}\right)^2 \frac{\exp(v_k/T)}{(\exp(v_k/T) - 1)^2} \quad (6)$$

The parameters u_k and v_k are taken from (Leachman et al. 2009) for normal hydrogen (n-H_2) because in the temperature range of interest ($\approx 260 - 360K$) one will always have n-H_2. Moreover, n-H_2 is defined as hydrogen's equilibrium concentration at room temperature, which corresponds to 75% orthohydrogen and 25% parahydrogen.

Finally, as there is interest in estimating enthalpy and entropy at different states, we take advantage of previous developments to use the following expressions in order to calculate such properties:

$$h = u + Pv \quad (7)$$

$$s_2 - s_1 = \int_{T_1}^{T_2}\frac{C_V}{T}dT + \int_{v_1}^{v_2}\left(\frac{\partial P(v,T)}{\partial T}\right)_v dv \quad (8)$$

However, Equation 8 is expressed more specifically as follows, where the integrals are to be solved:

$$\Delta s = C_V^0 \ln\frac{T_2}{T_1} - \frac{a}{2\sqrt{2}b}\ln\left[\frac{v + b(1-\sqrt{2})}{v + b(1+\sqrt{2})}\right]\Bigg|_0^{v_2}\int_{T_1}^{T_2}\frac{\mathrm{d}^2\alpha}{\mathrm{d}T^2}dT + \int_{v_1}^{v_2}\left(\frac{\partial P(v,T)}{\partial T}\right)_v dv \quad (9)$$

The derived formulas for internal energy, enthalpy and entropy can be found in the Appendix subsection 4.1.

2.4 Electrolyzer

To model the electrolyzer we start from the classical expression (Kuroki et al. 2021; Ulleberg 1998)[5] which includes the over-potentials in the electrodes of the cell:

$$U_{cell} = U_{rev} + U_{res} + U_{act} \quad (10)$$

The term U_{rev} concerns the reversible tension expressed by the Nernst equation (Atkins and Paula 2006). It is obtained by considering that two opposite forces of equal magnitude act on the ion at equilibrium, electrical forces and diffusion forces are equal in magnitude but of opposite sign. From such expression, while making certain assumptions[6], the Equation 11 is obtained to approximate the value of the reversible voltage.

$$U_{rev} := \Delta E_{eq} \approx \frac{\Delta_r g^0(T)}{nF} + \frac{3RT}{4F}\ln\left(\frac{P - P_{vH_2O}}{P_0}\right), \quad (11)$$

where the vapor pressure P_{vH_2O} is defined by

$$P_{vH_2O} \approx e^{\left(37.04 - \frac{6276}{T} - 3.416\ln T\right)}, \quad (12)$$

with pressures expressed in bar and T in Kelvin. The approximation (12) is valid between 25-250°C (Patterson et al. 2019).

To determine the last two terms of Equation 10 (namely the over-potentials U_{res} and U_{act}, referring to the electrical and the activation resistance of the electrodes, respectively) we consider semi-empirical models found in the literature for alkaline electrolyzers. For the time being,

[5] It should be noted that there exists also a component called diffusion over-potential (U_{diff}) but it is negligible (Ulleberg 1998). It is created by slow diffusion of ions through the electrolyte making it harder for ions to reach active sites.

[6] Considering the isobaric heat capacity (C_p) to be constant, uniform acidity between the electrodes, water as an undiluted liquid, and oxygen and hydrogen to be ideal gases at the same pressure equal to the total pressure minus the vapor pressure of water (Patterson et al. 2019)

two semi-empirical models were coded and an ulterior and more detailed work will compare both models as it is beyond the scope of the present work. Therefore, only the one found in literature will be presented here. The model in question is taken from the work of (Ulleberg 1998; Ulleberg 2003) which models a stand-alone photovoltaic-hydrogen power plant (PHOEBUS) in Jülich (Germany) equipped with an alkaline electrolyzer with circular bipolar cells and a 30 wt% KOH solution as electrolyte.

$$U_{res} = (r_1 + r_2 T)\frac{I}{A_{cell}} \tag{13}$$

$$U_{act} = (s_1 + s_2 T + s_3 T^2)\log\left(\frac{t_1 + \frac{t_2}{T} + \frac{t_3}{T^2}}{A_{cell}} I + 1\right) \tag{14}$$

Where s_i, t_i, and r_i are known parameters [7], A_{cell} is the area of a cell in m^2, I is the current in amperes and the temperature T is expressed in °C. We can observe that Equation 14 is an expression derived from the Butler-Volmer relation. Then, for all that concerns the Faraday efficiency, we consider the model developed by (Ulleberg 1998) and already used in (Khan and Iqbal 2005) where the temperature T is expressed in °C:

$$\eta_f = f_2 \frac{\left(\frac{I}{A_{cell}}\right)^2}{f_1 + \left(\frac{I}{A_{cell}}\right)^2} \tag{15}$$

$$f_1 = 2.5T + 50 \;\; ; \;\; f_2 = -0.00075T + 1$$

This calculation allows to estimate the real production of hydrogen in the electrolyzer considering the current losses and the number of cells N_{cell} in the stack.

$$\dot{m}_{H_2} = \eta_f \frac{N_{cell} M_{H_2}}{2F} I \tag{16}$$

In addition, we want to estimate the heat losses since they could potentially be used as an energy source for the heating of the water inlet. This is obtained by an energy balance:

$$\dot{Q} = \dot{W}_{elec} - \Delta\dot{H} = \dot{W}_{elec} - \dot{m}_{H_2}\left(\Delta_r h^0_{T0} + \Delta_r C_p (T - T_0)\right) \tag{17}$$

Finally, we can also estimate the energy efficiency of our electrolyzer by using the lower heating value of hydrogen.

$$\eta = \frac{\dot{m}_{H_2} LHV_{H_2}}{\dot{W}_{elec}} \tag{18}$$

In Figure 2, we observe that there are inputs as working temperature (T_{ref}), the supplied current (I) and the water inlet at a given pressure (P_{ref}); as well as the outputs of oxygen and hydrogen flows on the top side of the electrolyzer model.

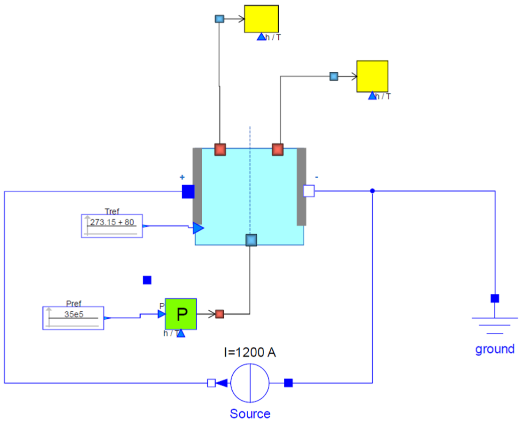


Figure 2. Model of Alkaline Electrolyzer with corresponding inputs and outputs

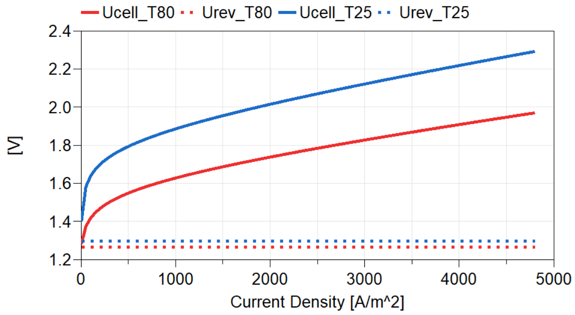


Figure 3. Polarization curves at two different temperatures (80°C in red lines, and 25°C in blue lines)

After having modeled the electrolyzer, it is pertinent to obtain the polarization curve (which represents the voltage as a function of current density) to verify that the cell voltage calculated using Ulleberg (1998) model has been well coded. From its shape and values, Figure 3 corresponds to the expected behaviour for alkaline electrolysis at two working temperatures (80°C in red line, and 25°C in blue line). The cell voltage U_{cell} is represented by the continuous lines and the reversible voltage U_{rev} is represented by the dotted lines. By observing the continuous lines (cell voltage), one could also specifically verify that when placed around low current densities, the activation over-potential (Equation 14) has a more important impact on the total voltage of the cell and at higher densities the behavior is rather linear due to the shape of the resistance over-potential (Equation 13). One can also verify that the reversible potential (Equation 11) is independent of the current density.

2.5 Compressor

The compressor is modeled using a isentropic efficiency coefficient. From Equation 8 isentropic processes may then be introduced. They refer to the assumption of equal entropy between two different states (1 → 2). Namely, there is not any change when moving from state 1 to state 2, and then the left side of Equation 8 becomes equal

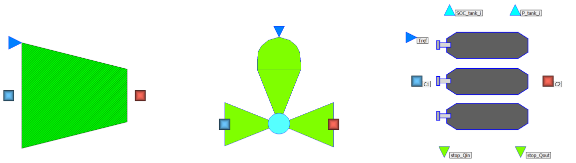
[7] The empirical parameters (s_i, t_i, and r_i) can be found numerically using non-linear regression techniques. Consequently one can fit the model for an alkaline electrolyzer in particular

to zero. Knowing the properties in such a state allows us to use the isentropic efficiency of the compressor defined by Equation 19. A graphical representation of the compressor model is given in Figure 4a.

$$\eta_{is,comp} = \frac{h_{2is} - h_1}{h_2 - h_1} \tag{19}$$

Starting from the fact that the state 1 and the value of $\eta_{is,comp}$ is known, and considering $\Delta s = 0$ in Equation 8, h_{2is}, the enthalpy in the state 2_{is} where $s_{2is} = s_1$ is calculated, and then h_2 for state 2 is obtained from Equation 19.

Moreover, when looking at the safety aspect, one must not forget that there are limits concerning the temperature and pressure that the gas should reach. For example, the SAE J2601 standard mentioned in the section "SAE2020" indicates these upper limits during hydrogen refueling. So for compression a maximum temperature of 135°C (Institute 2007) may be established (recommendation for high pressure hydrogen rich services) when using a piston compressor. This safety measure consequently sets a maximum compression ratio and therefore probably the need to compress in several stages.



(a) Representation of the compressor model

(b) Representation of the Pressure Control Valve model

(c) Representation of the gaseous storage model

Figure 4. Example of model components.

2.6 Heat Exchanger

For the heat exchanger used to cool the hydrogen, the concept of the Logarithmic Mean Temperature Difference (LMTD) should be introduced first. According to Newton's law of cooling, the rate of heat transfer is related to the instantaneous temperature difference between the hot and cold media (in a heat transfer process, the temperature difference varies with position and time). As a result, the temperature variation is non-linear and can best be represented by a logarithmic calculation, hence the LMTD (Engineering ToolBox 2003).

$$\dot{Q} = UA\Delta T_{LM} \quad , \quad \Delta T_{LM} \frac{\Delta T_b - \Delta T_a}{\ln \frac{\Delta T_b}{\Delta T_a}} \tag{20}$$

Where ΔT_a is the temperature difference between the two flows at the A end, and ΔT_b is the temperature difference between the two flows at the B end (sides of the exchanger whether if its design is that of a counter-current heat exchanger or not), $\dot{Q}$ $[W]$ is the heat service exchanged, U $[\frac{W}{m^2K}]$ is the overall heat transfer coefficient, and A $[m^2]$ is the area of exchange. After developing Equation 20 we obtain an expression according to the specific heat capacities of "cold" and "hot" fluids ($C_{p,f}$ and $C_{p,c}$ respectively):

$$\ln \frac{\Delta T_b}{\Delta T_a} = UA \left[\frac{1}{\dot{m}_f C_{p,f}} - \frac{1}{\dot{m}_c C_{p,c}} \right] \tag{21}$$

As the main interest of this first work is not particularly the heat transfer phenomenon, the heat exchanger model is quite simple which entails to having certain limitations that must be indicated to not make physical mistakes while using the component. Such limitations and assumptions will be discussed in Section 4

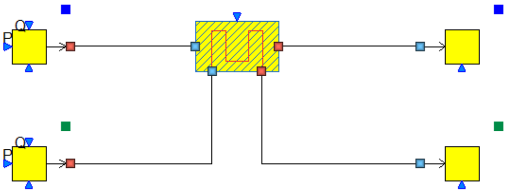


Figure 5. Test model of the heat exchanger component

2.7 Pressure Control Valve

The valve flow coefficient (C_v) is defined as the flow capacity of a control valve under fully open conditions relative to the pressure drop across the valve. It is defined as the volume of water (GPM in the U.S.) at 60°F that will flow through a fully open valve with a pressure differential of 1 psi across the valve. It is useful to know how to calculate C_v because it is the standard method of sizing and selecting control valves used in the industry. In addition, an oversized valve can lead to control problems such as flushing or poor heat transfer (ΔT) through a coil due to overflow. Conversely, an undersized valve may not provide sufficient flow and exceed the available ΔP. In general, the volume flow rate q and C_v are related by an expression of type

$$q = C_v f(x) \sqrt{\frac{\Delta P}{G}}$$

Where G is the specific gravity of the fluid ($G = \rho / \rho_{air}$). However, the functions to describe the behavior of the gas passing through the valve are generally more complex. According to (Swagelok 2007) they can be expressed with the following two equations, where $p = [bar]$, $T = [K]$, $q = \left[std \frac{L}{min}\right]$:

$$q^2 GT_1 = (0.471 N_2)^2 C_v^2 p_1^2 \quad , \quad p_2 \leq \frac{p_1}{2} \tag{22}$$

$$q^2 GT_1 = N_2^2 C_v^2 p_1 \left(1 - \frac{2\Delta P}{3p_1}\right)^2 \Delta P \quad , \quad p_2 > \frac{p_1}{2} \tag{23}$$

N_1 and N_2 are parameters provided in (Swagelok 2007) that depend on the units used and the volumetric flow rate q is expressed under standard conditions (25°C and 1 atm).

On the other hand, it is also necessary to illustrate this valve from a thermodynamic point of view to understand the physical process. Choke valves are generally small devices, and the flow through them can be assumed to be adiabatic ($Q \cong 0$) since there is not enough time or area for efficient heat transfer to occur. There is also no work done ($w \cong 0$), and the change in potential energy, if there is any, is very small ($e_p \cong 0$). Even though the output velocity is often considerably larger than the input velocity, in many cases the increase in kinetic energy is negligible (Cengel and Boles 2015). Then the energy conservation equation for this single-stream steady-state flow device reduces to:

$$h_1 \cong h_2$$

$$u_1 + P_1 v_1 \cong u_2 + P_2 v_2 \tag{24}$$

Thus, the end result of a throttling process depends on how much a certain property has increased during the process. If the flow of energy increases during the process ($P_2 v_2 > P_1 v_1$), this can be at the expense of the internal energy. As a result, the internal energy decreases, which is usually accompanied by a decrease in temperature. If the Pv of the product decreases, the internal energy and temperature of a fluid will increase during a throttling process[8]. In these cases, the magnitude of the ΔT is governed by a property called the Joule-Thomson coefficient :

$$\mu_{JT} = \left(\frac{\partial T}{\partial P}\right)_H \tag{25}$$

The Joule-Thomson coefficient is a measure of the variation of temperature versus pressure during a process with constant enthalpy, which corresponds to the situation of the valve. So if this coefficient is negative, the temperature increases during an expansion. The sign of the coefficient will therefore depend on the conditions in which the gas is, and the so-called inversion temperature (Figure 6) at these conditions.

Thus, considering such phenomenon, a valve model is adapted (Figure 4b) from TSP to take into account the effects on temperature of a throttling process with hydrogen as a fluid. From such development, and through a comparative simulation, one can observe the difference in properties at the end of the expansion process using the TSP valve[9] and the hydrogen-adapted Pressure Control Valve (PCV). As a simple study case, a simulation is carried out with the following parameters for both of the valves:

- Inlet pressure: $400 bar$
- Hydrogen flow rate: $0.0038885674 \frac{kg}{s}$
- Inlet temperature: 65°C

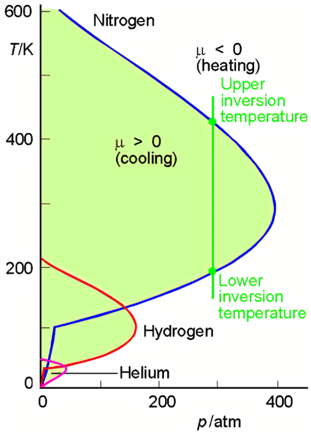


Figure 6. Inversion temperatures for three real gases: nitrogen, hydrogen and helium (Central Michigan University 2013)

- $Cvmax : 0.015 \frac{m^4}{sN^5}$
- Outlet pressure which varies from 150 to 350 *bar*

In this comparative simulation, the valve opening is controlled to obtain a certain outlet pressure that will vary from 150 to 350 *bar*. The goal is to analyze the change of the outlet temperature as a function of pressure and while maintaining the same inlet pressure, thus generating a different pressure drop value at each moment.

From the main result (Figure 7), it is observed that the outlet temperature of the PCV ($PCV\_realsgas.T$) evolves in function of the outlet pressure ($sensorP.C2.P$), which is not the case when using the TSP default valve ($PCV\_idealgas.T$) that considers an ideal fluid. Specifically, it is noted that for the operating conditions of temperature and pressure, the Joule-Thomson coefficient is always negative, causing a temperature rise during the isenthalpic expansion process, which can be verified in Figure 6. Therefore, it is proved that the modified valve does include the Joule-Thomson phenomenon through the use of PR-EOS which describes a real gas. Even though the results have not been compared to real values from experimentation, the PR-EOS is widely accepted as an approximation in the process engineering field, however the use of a hydrogen-dedicated equation of state could be envisaged (Sakoda et al. 2012).

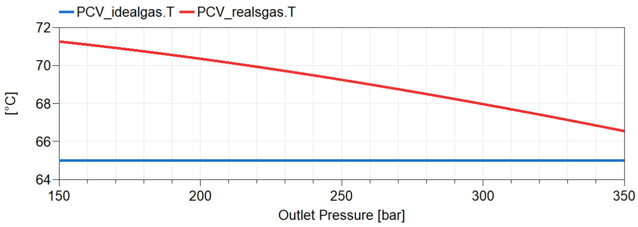


Figure 7. Outlet temperature comparison for two valve models at variable outlet pressure

[8] In the case of an ideal gas, $h = h(T)$; therefore, the temperature must remain constant during a throttling process, which is not the case for a real gas where $h = h(T,P)$

[9] The valve available on TSP was initially designed for water and steam flows applications

2.8 Gaseous Storage

The modeling of the pressurized gas storage is summarized in (Migoni et al. 2016) where an equation of state is implemented to estimate the pressure. However, unlike such work which uses the ideal gas equation, one can improve the estimates by using the PR-EOS. For that the Equation 1 which allows to model the pressure is used. Then it is necessary to know the mass stored in a given moment. For that we use a mass balance (Equation 26) which allows to calculate the quantity of hydrogen in each bottle and finally to estimate the pressure according to this mass and the volume of the tank. The number of bottles in the storage is a parameter that can be modified and the way in which the bottles are filled is directly managed by the model itself (a maximum and a minimum pressure in each bottle can be provided in order to manage the switches).

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \dot{m}_{in} - \dot{m}_{out} \tag{26}$$

3 Gas Plate modeling

3.1 Model construction

The case study model (Figure 1) was briefly presented in subsection 2.1 where the general idea of the so-called Gas Plate was introduced alongside the two branches (Storage and Station) which illustrate the final usages of the low-carbon hydrogen produced upstream through water electrolysis. At this time the theoretical fundamentals of each of the components (or blocks) in the platform has been settled and so a more detailed version of the model may be presented. Such is the case for Figure 8, where three electrolyzers are placed upstream on the left, and then a series of components such as pressure loss blocks at the outlet of each electrolyzer, valves[10], volumes, sensors (to read temperature, pressure and mass flow) compose the hydrogen pipeline towards two branches or usages.

On the right upper side (Storage Branch) a pressure control valve, a compressor, and counter-current heat exchanger are used to regulate the flow, to obtain a desired pressure and to refrigerate the hydrogen flow to a safe temperature, respectively. This, to finally arrive to the gaseous storage system at the end of the branch. On the other hand, the charging station branch in our chain is composed of a compressor to increase the pressure to the desired values according to the type of vehicle, a valve (PCV) that allows to regulate the output pressure according to the normative (SAE 2020), and a heat exchanger between those previous components in order to limit the temperature rise due to the Joule-Thomson effect in the valve.

Finally, it must be noted that some of the components used in this platform are taken from the TSP library. Such is the case for the Singular pressure loss component at the outlet of each electrolyzer, the volumes, the check valve and other common blocks as signals and sources. Components like the sensors, the compressor and the pressure control valve where adapted from TSP; other blocks like the electrolyzer and the storage tanks where developed for a new hydrogen-dedicated library compatible with TSP.

3.2 Model simulation

The model presented in Figure 8 and described in the previous subsection has been used to perform various dynamic simulations that have allowed to validate and better understand the physical behavior of the hydrogen platform. These simulations give as well an overview of the different types of predictions that can be performed with this model.

For this purpose, three different scenarios have been considered. These dynamic simulations in which several events occur are described below together with graphs showing the evolution of key variables for each scenario.

Scenario 1: the three electrolyzers are working during the whole simulation and at the beginning the hydrogen storage is being filled up. At 500s, the outlet mass flow rate of the storage is temporarily increased (see dotted red line in Figure 9). During this simulation it is possible to observe the filling of gaseous storage while there is no increase in the outlet mass flow rate, see continuous blue line in Figure 9 in which the evolution of the hydrogen mass in the first bottle (and only bottle in this simulation) of the gaseous storage is presented . When the outlet mass flow rate increases, it can be observed how the bottle is being emptied before resuming the filling at the end of the simulation.

Scenario 2: as in the previous scenario, the three electrolyzers are working during the whole simulation and at 1200 seconds the valve of the storage branch of the platform is partially closed (as shown by Figure 10a). The consequence of closing this valve is directly observed in the storage inlet mass flow-rate evolution, see Figure 10b where a remarkable decrease is shown after 1200s. In addition to this important decrease, it is also possible to notice in the previous graph a small perturbation of the inlet mass flow-rate at around 800s. This reduction can be explained by the graph plotted in Figure 10c and in which the evolution of the hydrogen mass in the storage is represented: the blue line corresponds to the total hydrogen mass of the first bottle and the red one to the total hydrogen mass in the second bottle. The above-mentioned perturbation of the inlet mass flow-rate occurs when the switch between the filling of the first and second bottle occurs (corresponding to the moment at which the maximum allowed pressure in the first bottle is reached). The valve closing occurs therefore when the second bottle is being filled and the consequences can be observed in the red curve where the increase of the hydrogen mass in the second bottle is reduced (slope decrease) after 1200s.

Scenario 3: contrary to the previous scenarios, in this

[10] The Check Valve right before the volume that serves to divide the flow into the two branches is found on TSP (El Hefni and Bouskela 2019) and the PCVs used once on the Storage branch and twice on the Station branch is adapted from the Control Valve model of the same TSP library

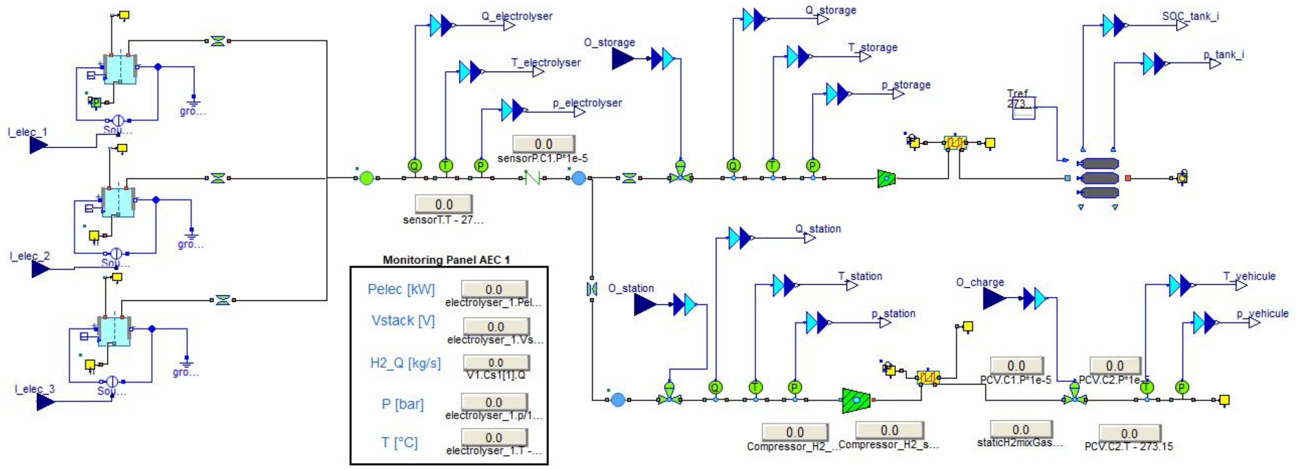


Figure 8. ThermoSysPro model of the hydrogen platform

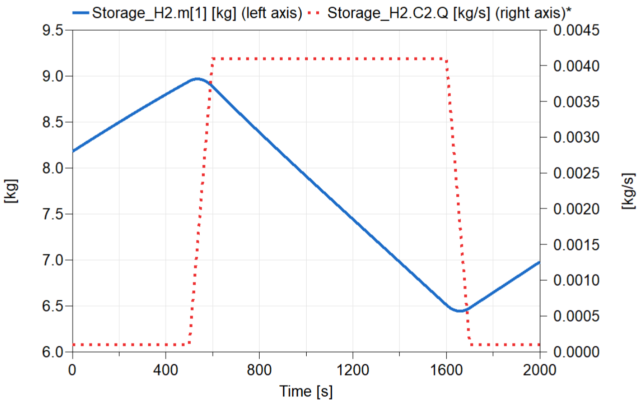


Figure 9. Scenario 1 simulation results

case, the three electrolyzers of the gas plate are not working during the whole simulation. In this scenario, one of the electrolyzers is completely shutdown at 200s as shown by the dotted red line in Figure 11 which represents the evolution of the current in this electrolyzer. When this shutdown occurs, the hydrogen produced by the three electrolyzers is reduced by a third. The corresponding evolution of the hydrogen mass flow rate leaving all the electrolyzers is plotted in continous blue line in Figure 11.

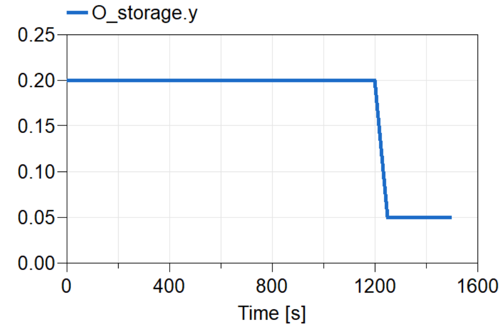
The three simulated scenarios correspond to smooth transients implying slow dynamics and to avoid numerical problems the valves are never completely closed.

4 Discussion and Further Work

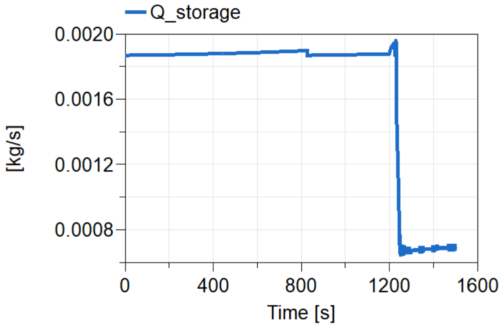
In this paper, we illustrated how to use the latest version of ThermoSysPro Modelica-based library, the so-called V-4, in order to model a hydrogen platform where hydrogen is produced by three electrolyzers, stored in a storage station and consumed by a vehicle station. It turns out that one advantage of using TSP is that some components of TSP library can be reused and easily adapted to hydrogen modeling. In our case study, we created two new components in TSP (the electrolyzer and the storage station) and adapted the other components (the compressor, the pressure control valve and the heat exchanger).

Even if the results obtained so far are already extremely encouraging and useful for the current studies, it is important to keep in mind that the modeled components have some limitations. For instance, the dynamics were neglected for the pipes, the heat exchanger and the electrolyzer. Also, the heat transfer equations might not be adapted for fast transients. Moreover, even though the last version of TSP can handle zero flow rate, this feature has not been tested in the present work and fast closing of valves may require solutions like multi-mode simulation for instance (Bouskela and El Hefni 2014a; Bouskela 2016). In addition, some simplifications have been made in the architecture of the gas plate with respect to a real installation. For example, here we only modeled the electrolyzer behaviour without considering the water purification or gas separation. We also merged the various compression and cooling stages into one single stage.

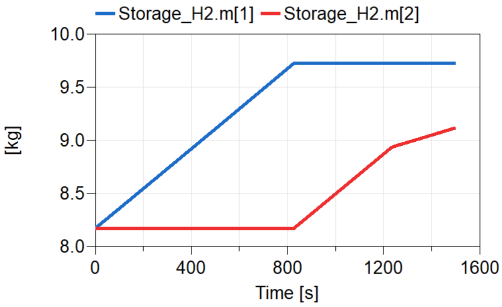
Considering the above mentioned limitations, we argue that the model presented in this paper is good enough for what it was meant for: model a hydrogen platform at a large-scale with Modelica and show its potential applications. The model presented in this work shows how this modeling approach can be used in the design phase of an hydrogen platform (determine an adequate architecture, study the dynamic response of the platform, etc.) and could be used as well in the operation phase of the platform. In this later case, the model developed would correspond to an existing platform, and the results provided by this model could be combined to on-site measurements in order to provide accurate diagnosis. This diagnosis could be performed using advanced mathematical methods that allow to combine simulated and measured and that have recently been adapted to Modelica models such as Data



(a) Valve opening command



(b) Hydrogen storage inlet mass flow rate



(c) Hydrogen mass in the pressurized gas storage composed of two bottles

Figure 10. Scenario 2 simulation results

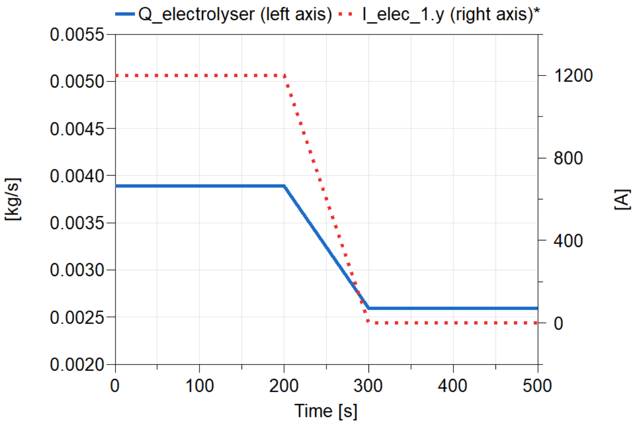


Figure 11. Scenario 3 simulation results

Assimilation (Corona Mesa-Moles, Argaud, et al. 2019; Corona Mesa-Moles, Henningsson, et al. 2021) or Data Reconciliation (Bouskela, Jardin, et al. 2021).

In addition of the above-mentioned advanced uses of the models, in future work, the model developed could be used to optimize other systems of the hydrogen platform such as the associated instrumentation and control (using a linearized version of the model) or to perform technical-economic optimization. This kind of approach could be used as well to optimize the overall architecture of the hydrogen platform (number of electrolyzers, storage volume, number of branches for the fuelling station, etc.). In addition, this model can be easily enriched to include for example the source of energy used for the electrolyzers (for instance wind or solar energy sources). It is however important to keep in mind the possible simulation difficulties that may occur when increasing the size of the model or combining different kind of physics. Furthermore, the TSP developments to correctly model hydrogen based systems could be continued. For example, the library can be completed to model more detailed components and physical phenomena as needed or required to correctly model the behaviour of the real system. To reach this goal it is important to validate the models with experimental data, however, as far as we know, there is no publicly available data on real experiments to validate the developments presented in this article as a whole. For now, only validated models in the literature can be considered, such as the filling stations models (Kuroki et al. 2021) which have been validated with experimental data (SAE 2020).

References

Atkins, Peter and Julio de Paula (2006). *Physical chemistry*. Ed. by W. H. Freeman and Company. 8th. 1. Oxford: Oxford University Press. ISBN: 0-7167-8759-8.

Bhuyan, Sujit Kumar, Prakash Kumar Hota, and Bhagabat Panda (2018-06). “Modeling, control and power management strategy of a grid connected hybrid energy system”. In: *International Journal of Electrical and Computer Engineering* 8 (3), pp. 1345–1356. ISSN: 20888708. DOI: 10.11591/ijece.v8i3.pp1345-1356.

Bouskela, Daniel (2016). “Multi-Mode Physical Modelling of a Drum Boiler”. In: *Procedia Computer Science* 95. Complex Adaptive Systems Los Angeles, CA November 2-4, 2016, pp. 516–523. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2016.09.331. URL: https://www.sciencedirect.com/science/article/pii/S1877050916325042.

Bouskela, Daniel and Baligh El Hefni (2014a). “A physical solution for solving the zero-flow singularity in static thermal-hydraulics mixing models”. In: *10th Modelica Conference 2014*. Linköping, Sweden, pp. 847–855. DOI: http://dx.doi.org/10.3384/ecp14096847. URL: https://ep.liu.se/ecp/096/088/ecp14096088.pdf.

Bouskela, Daniel and Baligh El Hefni (2014b). “Modeling and simulation of complex ThermoSysPro model with OpenModelica - Dynamic Modeling of a combined cycle power plant”. In: *10th Modelica Conference 2014*. Linköping, France, pp. 847–855. DOI: 10.3384/ecp14096847.

Bouskela, Daniel, Audrey Jardin, et al. (2021-09). “New Method to Perform Data Reconciliation with OpenModelica and ThermoSysPro”. In: *14th Modelica Conference 2021*. Linköping, France, pp. 453–462. DOI: 10.3384/ecp21181453.

Cengel, Yunus and Michael Boles (2015). *Termodinámica de Cengel*. 8th ed. The McGraw-Hill Companies, p. 1001. ISBN: 978-607-15-0743-3.

Central Michigan University (2013). URL: http://people.se.cmich.edu/teckl1mm/pchemi/chm351ch3bf01.htm.

Corona Mesa-Moles, Luis, Jean-Philippe Argaud, et al. (2019-03). “Robust Calibration of Complex ThermosysPro Models using Data Assimilation Techniques: Application on the Secondary System of a Pressurized Water Reactor”. In: *The 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. Regensburg, Germany, pp. 553–560. DOI: 10.3384/ecp19157553. URL: https://hal.science/hal-03540782.

Corona Mesa-Moles, Luis, Erik Henningsson, et al. (2021-09). “New Equation-based Method for Parameter and State Estimation”. In: *14th Modelica Conference 2021*. Linköping, France, pp. 129–139. DOI: 10.3384/ecp21181129. URL: https://hal.science/hal-03540777.

El Hefni, Baligh and Daniel Bouskela (2017-05). “Modeling and simulation of complex ThermoSysPro model with OpenModelica - Dynamic Modeling of a combined cycle power plant”. In: *12th Modelica Conference 2017*. Linköping, France, pp. 407–414. DOI: 10.3384/ecp17132407.

El Hefni, Baligh and Daniel Bouskela (2019). *Modeling and Simulation of Thermal Power Plants with ThermoSysPro A Theoretical Introduction and a Practical Guide*. Springer Nature Switzerland AG.

Engineering ToolBox (2003). URL: https://www.engineeringtoolbox.com/arithmetic-logarithmic-mean-temperature-d_436.html.

Eriksson, E. L.V. and E. Mac A. Gray (2017). “Optimization and integration of hybrid renewable energy hydrogen fuel cell energy systems – A critical review”. In: *Applied Energy* 202, pp. 348–364. ISSN: 03062619. DOI: 10.1016/j.apenergy.2017.03.132. URL: http://dx.doi.org/10.1016/j.apenergy.2017.03.132.

European Commission (2020). *A hydrogen strategy for a climate-neutral Europe*. Tech. rep. European Commission.

European Commission (2023). *Energy Systems Integration - Hydrogen*. https://energy.ec.europa.eu/topics/energy-systems-integration/hydrogen_en. Consultation date: 2023-03-02.

Green hydrogen cost reduction (2020). Tech. rep. International Renewable Energy Agency.

Institute, American Petroleum (2007). *API Standard 618*.

International Energy Agency (2022a). *Hydrogen*. Tech. rep. International Energy Agency.

International Energy Agency (2022b). *World Energy Outlook 2022*. Tech. rep. International Energy Agency.

Khan, M. J. and M. T. Iqbal (2005). “Dynamic modeling and simulation of a small wind-fuel cell hybrid energy system”. In: *Renewable Energy* 30.3, pp. 421–439. ISSN: 09601481. DOI: 10.1016/j.renene.2004.05.013.

Kuroki, Taichi et al. (2021-06). “Thermodynamic modeling of hydrogen fueling process from high-pressure storage tank to vehicle tank”. In: *International Journal of Hydrogen Energy* 46 (42), pp. 22004–22017. ISSN: 03603199. DOI: 10.1016/j.ijhydene.2021.04.037.

Leachman, J. W. et al. (2009). “Fundamental equations of state for parahydrogen, normal hydrogen, and orthohydrogen”. In: *Journal of Physical and Chemical Reference Data* 38 (3), pp. 721–748. ISSN: 00472689. DOI: 10.1063/1.3160306.

Migoni, G. et al. (2016-08). “Efficient simulation of Hybrid Renewable Energy Systems”. In: *International Journal of Hydrogen Energy* 41 (32), pp. 13934–13949. ISSN: 03603199. DOI: 10.1016/j.ijhydene.2016.06.019.

Patterson, Bruce D. et al. (2019). “Renewable CO2 recycling and synthetic fuel production in a marine environment”. In: *Proceedings of the National Academy of Sciences of the United States of America* 116.25, pp. 12212–12219. ISSN: 10916490. DOI: 10.1073/pnas.1902335116.

SAE (2020). *Fueling Protocols for Light Duty Gaseous Hydrogen Surface Vehicles J2601_202005*. SAE Mobilus.

Sakoda, N. et al. (2012). “Burnett PVT measurements of hydrogen and the development of a virial equation of state at pressures up to 100 MPa”. In: *International Journal of Thermophysics* 33 (3), pp. 381–395. ISSN: 0195928X. DOI: 10.1007/s10765-012-1168-2.

Swagelok (2007). *Valve Sizing*. DOI: 10.1016/b978-0-7506-2255-4.50070-2.

Trujillo, Mario, Peter O’Rourke, and David Torres (n.d.). *Generalizing the Thermodynamics State Relationships in KIVA-3V*.

TÜV SÜD (2023). *Explore the hydrogen value chain*. https://www.tuvsud.com/en/themes/hydrogen/explore-the-hydrogen-value-chain#:~:text=The%20hydrogen%20value%20chain%20is,%2C%20and%20standards%20(RCS). Consultation date: 2023-03-21.

Ulleberg, Øystein (1998). “Stand-alone power systems for the future: Optimal design, operation & control of solar-hydrogen energy systems”. PhD thesis. Norwegian University of Science and Technology Trondheim.

Ulleberg, Øystein (2003). “Modeling of advanced alkaline electrolyzers a system”. In: *Hydrogen Energy* 28, pp. 21–33.

Zohuri, Bahman (2018). *Physics of Cryogenics: An Ultralow Temperature Phenomenon*. Elsevier, pp. 59–70.

Appendices

4.1 Properties Equation Solutions

4.1.1 Specific Internal Energy (Equation 4)

$$u(v,T) = u^0 + a\left(\alpha - T\frac{\mathrm{d}\alpha}{\mathrm{d}T}\right)\frac{1}{2\sqrt{2}b}\ln\left[\frac{v+b(1-\sqrt{2})}{v+b(1+\sqrt{2})}\right],$$

with $\frac{\mathrm{d}\alpha}{\mathrm{d}T} = \frac{-k}{\sqrt{T_cT}}\sqrt{\alpha}$.

4.1.2 Specific Enthalpy (Equation 7)

$h(v,T) = u^0 + a\left(\alpha - T\frac{\mathrm{d}\alpha}{\mathrm{d}T}\right)\frac{1}{2\sqrt{2}b}\ln\left[\frac{v+b(1-\sqrt{2})}{v+b(1+\sqrt{2})}\right] + Pv$, with $\frac{\mathrm{d}\alpha}{\mathrm{d}T} = \frac{-k}{\sqrt{T_cT}}\sqrt{\alpha}$.

4.1.3 Specific Entropy (Equation 9)

$\Delta s = C_V^0 \ln\frac{T_2}{T_1} - \frac{a}{2\sqrt{2}b}\ln\left[\frac{v+b(1-\sqrt{2})}{v+b(1+\sqrt{2})}\right]\Big|_0^{v_2}\int_{T_1}^{T_2}\frac{\mathrm{d}^2\alpha}{\mathrm{d}T^2}dT + \int_{v_1}^{v_2}\left(\frac{\partial P(v,T)}{\partial T}\right)_v dv$ has the first integral solution $\int_{T_1}^{T_2}\frac{\mathrm{d}^2\alpha}{\mathrm{d}T^2}dT = \frac{k}{2}\left[\frac{k\ln T}{T_c}\left(1-\frac{1}{\sqrt{T_c}}\right) - \frac{2}{\sqrt{T_cT}}(k+1)\right]\Big|_{T_1}^{T_2}$, and the second integral solution $\int_{v_1}^{v_2}\left(\frac{\partial P(v,T)}{\partial T}\right)_v dv = \left[R\ln|v-b| - a\frac{\mathrm{d}\alpha}{\mathrm{d}T}\frac{1}{2\sqrt{2}b}\ln\left(\frac{v+b(1-\sqrt{2})}{v+b(1+\sqrt{2})}\right)\right]\Big|_{v_1}^{v_2}$.

Dialectic Mechanics: Extension for Real-Time Simulation

Carsten Oldemeyer   Dirk Zimmer

Institute of System Dynamics and Control (SR), German Aerospace Center (DLR), Germany,
{carsten.oldemeyer,dirk.zimmer}@dlr.de

Abstract

Dialectic mechanics was introduced as an approximative modeling alternative to the classic Newtonian formulation of mechanics. It allows for additional freedom in placing a systems eigenvalues to facilitate simulation of systems, that are not suitable for most integration methods, when modeled according to the classic approach. The original idea of dialectic mechanics enables the suppression of high frequencies, but may still yield very stiff systems unsuitable for explicit integration methods. An additional term is added to enable real-time simulation with explicit methods. The goal of this paper is an analysis of the resulting equations and a comparison to the classic Newtonian formulation, aiming for an understanding of which applications most benefit from using dialectic mechanics.

Keywords: simulation, stiff systems, dialectic mechanics, real-time

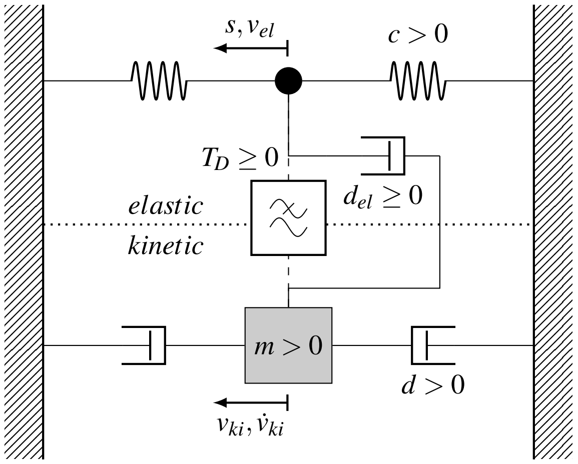


Figure 1. Concept drawing of damped dialectic mechanics. Additionally to the first order filter a damper connects elastic and kinetic domains.

1 Introduction

Industrial robots are designed to move precisely and repeatably in the presence of external forces, leading to systems that are built stiffly. These properties pose problems during simulation, since both within the robot itself as well as in its interaction with the environment stiff springs are an obvious choice for the modeler. While considering the robots gear stiffness does not require special care, simulation performance takes a significant hit, when the structural parts are explicitly modeled as flexible bodies with a stiffness a few orders of magnitude higher.

Another challenge is the simulation of a process, that requires the robot to touch a non-compliant object or environment. An example is the standard robot task of gripping an workpiece and moving it. One way to include the contact dynamics in the existing Modelica models would be to model the contact between gripper and object as a stiff spring and locking both parts together with friction. As the simulation performance with realistic parameters is unacceptable for most use cases of the model, approximations are usually necessary. Just reducing the stiffness might get the simulation to run well, but has unwanted side effects like larger oscillations and changed equilibriums.

Being able to simulate such processes in real-time would enable model use cases like model predictive control, virtual commissioning and Hardware-in-the-loop testing. In these applications the bandwidth of interest is often limited and well below the eigenfrequencies of very stiff objects. Removing high frequency oscillations with small amplitudes, that are not relevant to the models purpose, is the main goal of dialectic mechanics.

1.1 Real-Time simulation of stiff systems

Stability is an essential property in both system dynamics as well as solver methods. Even if a system is stable, simulating it stably require a careful choice of solver method and step size. For a lot of solver methods stability regions can be calculated, in which a systems eigenvalues have to be for a stable simulation. Although Higham and Trefethen (1993) indicate looking at eigenvalues is not enough in some cases, stability regions provide a useful tool for most systems in explaining problems with stiff systems. Higham and Trefethen (1993) summarized the essence of stiffness of Ordinary Differential Equations (ODE) as the case, when *"Stability is more of a constraint than accuracy"*. A remark earlier made by Hairer and Wanner (1991) states, that *"Stiff equations are problems for which explicit methods don't work"*. A remedy for this is the usage of implicit methods as pointed out by e.g. Dahmen and Reusken (2008). The authors mention backward differentiation formula (BDF) methods as especially useful, as well as higher order implicit Runge-Kutta methods.

When adding the requirements for real-time simulation, the number of suitable integration methods reduces significantly. The analysis in Cellier and Kofman (2006) comes

DOI
10.3384/ecp204239

to the conclusion, that mostly low-order explicit methods fulfill the need for predictable, bounded execution times. This creates a dilemma when planning to simulate a stiff system in real-time. Two ways around this fact proposed in Cellier and Kofman (2006) are linear implicit methods and multi-rate integration which requires slow and fast dynamics to be contained in discernible subsystems. Arnold, Burgermeister, and Eichberger (2007) mention the possibility of having dedicated real-time models neglecting stiff terms. This comes with the disadvantage of having to keep all models consistent.

1.2 Extending the equations of dialectic mechanics

Figure 1 shows a concept drawing of a dialectic mass-spring-damper-system. Although it cannot be used to derive the equations because of the way forces are split up between components it is helpful to explain the general idea of dialectic mechanics. The system is split into a massless elastic part on top and a kinetic part below. Both parts are connected by a first order filter that allows for low frequency interaction, but reduces high frequency effects. In addition to the system introduced in Zimmer and Oldemeyer (2023) a damper is placed between the elastic and kinetic parts. The resulting behavior, beneficial for real-time simulation as will be shown in the following sections, limits high-frequency energy transfer within the two domains.

The additional damper extends the equations presented in Zimmer and Oldemeyer (2023) and is marked with squared brackets in Equation 1b:

$$\frac{\mathrm{d}s}{\mathrm{d}t} = v_{el} \tag{1a}$$

$$f_{el} = -cs + mg\left[-d_{el}\left(v_{el} - v_{ki}\right)\right] \tag{1b}$$

$$\frac{\mathrm{d}v_{ki}}{\mathrm{d}t} = \frac{\left(v_{el} - v_{ki}\right)}{T_D} \tag{1c}$$

$$f_{ki} = -m\frac{\mathrm{d}v_{ki}}{\mathrm{d}t} - dv_{ki} \tag{1d}$$

$$f_{el} + f_{ki} = 0 \tag{1e}$$

1.3 Content

Based on the change to the dialectic equations this publication will analyze and illustrate the properties of damped dialectic mechanics. In section 2 the eigenvalues of the dialectic mass spring damper system are formulated and a choice for the added damping parameter is derived. Section 3 compares the properties of damped dialectic mechanics with the standard modeling approach, before section 4 illustrates the contents of the previous sections with simulation examples implemented in Modelica. In the end section 5 provides a summary and an outlook towards remaining work.

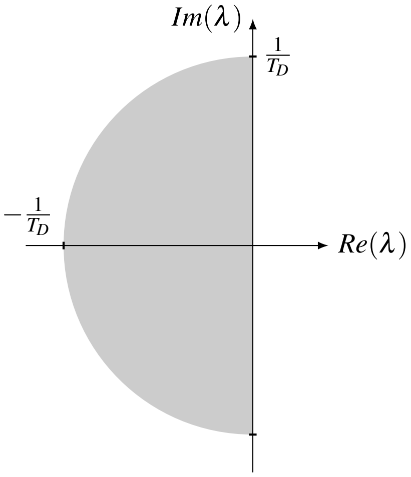


Figure 2. Illustration of the eigenvalue limitation of dialectic mechanics with added damping as shown by Equation 18. All eigenvalues are transformed into the gray area.

2 Properties of damped dialectic mechanics

For an understanding of what effect damped dialectic mechanics has on a system's dynamics Equation 1 can be combined to form a second order differential equation.

$$\left(m + d_{el}T_D\right)\ddot{v}_{ki} + \left(d + cT_D\right)\dot{v}_{ki} + cv_{ki} = 0 \tag{2}$$

The eigenvalues λ' of the dialectic system are:

$$\beta' = \frac{d + cT_D}{2\left(m + d_{el}T_D\right)} \tag{3a}$$

$$\omega_d' = \sqrt{\underbrace{\beta'^2 - \frac{c}{m + d_{el}T_D}}_{D'}} \tag{3b}$$

$$\lambda' = -\beta' \pm \omega_d' \tag{3c}$$

Based on these equations it is interesting to derive limits, within which the eigenvalues of the dialectic system lie. The maximum absolute value of the eigenvalues is derived differently depending on whether they are real-valued or complex-valued.

$$\left|\lambda'\right| = \begin{cases} \left|-\beta' - \sqrt{D'}\right|, & D' \geq 0 \qquad (4) \\ \left|-\beta' \pm i\sqrt{-D'}\right|, & D' < 0 \qquad (5) \end{cases}$$

Considering the assumptions made in Figure 1 the second term of D' is always positive, which in the first case leads to the upper bound

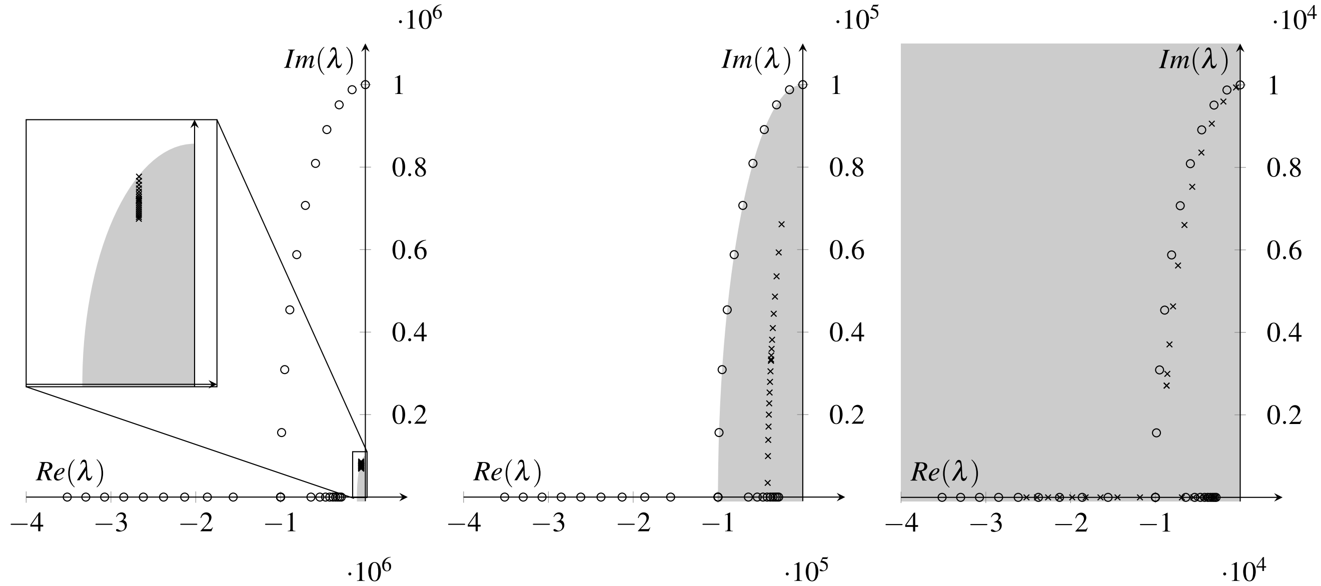
$$\beta'^2 \geq D' \tag{6}$$

With this upper bound Equation 4 is limited to

$$\left|-\beta' - \sqrt{D'}\right| = \left|\beta'\right| + \left|\sqrt{D'}\right| \tag{7}$$

$$\leq \left|2\beta'\right| \tag{8}$$

$$\left|2\beta'\right| = \frac{d + T_D c}{m + d_{el}T_D} \tag{9}$$



(a) Factor 10 **(b)** Factor 1 **(c)** Factor 0.1

Figure 3. Dialectic transformation of eigenvalues for three different magnitudes of undamped eigenfrequecies. Original eigenvalues (○) are created by increasing damping in a system with a chosen undamped eigenfrequency. For each pair of original eigenvalues the dialectic eigenvalues (×) are calculated based on Equation 3. The area to which transformed eigenvalues are constrained is marked by the same gray (cut-off) semicircle in all figures corresponding to the chosen $T_D = 1 \times 10^{-5}$ s. Each figure zooms in by a factor of 10 when going from 3a to 3c.

In order to come to a useful conclusion, a suitable choice for d_{el} is necessary. Setting $d_{el} = d + cT_D$ and using the assumption $m > 0$ again, results in

$$\frac{d+cT_D}{m+d_{el}T_D} = \frac{d+cT_D}{m+(d+cT_D)\,T_D} \tag{10}$$

$$< \frac{d+cT_D}{(d+cT_D)\,T_D} \tag{11}$$

$$\frac{d+cT_D}{(d+cT_D)\,T_D} = \frac{1}{T_D} \tag{12}$$

In the second case the absolute value simplifies to

$$\left|-\beta' \pm i\sqrt{-D'}\right| = \sqrt{(-\beta')^2 + \left(\sqrt{-D'}\right)^2} \tag{13}$$

$$= \sqrt{\frac{c}{m+d_{el}T_D}} \tag{14}$$

With the same choice for d_{el} as in the first case and the strict inequalities $m > 0$ and $dT_D > 0$ this results in an upper bound for the absolute value of the eigenvalues

$$\sqrt{\frac{c}{m+d_{el}T_D}} = \sqrt{\frac{c}{m+(d+cT_D)\,T_D}} \tag{15}$$

$$< \sqrt{\frac{c}{cT_D^2}} \tag{16}$$

$$\sqrt{\frac{c}{cT_D^2}} = \frac{1}{T_D} \tag{17}$$

As a result Equations 4 and 5 can be replaced by the simple limitation of the absolute value of the eigenvalues of the dialectic system:

$$\left|\lambda'\right| < \frac{1}{T_D} \qquad \forall\, m,d,c,T_D > 0 \tag{18}$$

Together with the fact that $\beta' > 0$, which can be seen from Equation 3a, Equation 18 enables the user to specify a half-circle in the left half-plane in which all eigenvalue of the dialectic system lies. The imaginary axis is not part of the half-circle. An illustration is presented in Figure 2. The radius depends only on one design parameter T_D, that can be adjusted to fit the stability requirements for a chosen solver-method and step size.

Figure 3 illustrates the qualitative difference in the dialectic modification of eigenvalues depending on the scale of the original eigenvalues. The original eigenvalues in Figure 3c, lying well within the gray area, are changed relatively little. In the middle figure, where the original eigenvalues are of the same magnitude as the limit, the transformation is considerably more obvious. Note that all original real-valued eigenvalues become complex-valued. For the highest magnitude, depicted on the left, the transformation is very aggressive and shoves all eigenvalues into the same region of the limit semicircle.

Another interesting observation is visible in the enlarged part of Figure 3a. The real part of the transformed eigenvalues are all very similar and appear to be bounded by

$$\beta' < \frac{1}{2T_D} \tag{19}$$

This is explained by the steps taken to get from Equation 8 to Equation 11. The omission of m goes from just leading to a valid inequality to being a decent approximation, if the mass is far smaller than the rest of the denominator. This is the case here, because of the fixed, high undamped eigenfrequencies used to create the original eigenvalues. The same is true for the inequality 18. Hence, when the undamped eigenfrequencies are increased even higher than in the left picture, all transformed eigenvalues trend more and more towards

$$\lambda^* = -\frac{1}{2T_D} \pm i\sqrt{\frac{3}{4}}\frac{1}{T_D} \tag{20}$$

For systems with low stiffness and high damping however this is not the case and there the transformed eigenvalues stay real-valued and move towards

$$\lambda^\dagger = -\frac{1}{2T_D} \pm \frac{1}{2T_D} \tag{21}$$

Generally the properties of damped dialectic mechanics discussed in this section are advantageous for real-time simulation with explicit solvers. The semicircle boundary provides a solid assumption about the placement of the eigenvalues even in complex systems and is easily configured by a single global parameter. A difference in magnitude of 100 is enough to have a strong reduction in the fast dynamics without changing the slow dynamics significantly. This behavior gets more pronounced, when there are more orders of magnitude between fast and slow dynamics. Of course the user has to check in every application, whether the model still fulfills its purpose with the changes made to the system. This check, however, can be conducted in an easy manner since the library enables the global setting of T_D. A corresponding sensitivity analysis is thus quickly performed.

3 Comparison to the classic Newtonian formulation

Formulating the eigenvalues of a regular mass spring damper system analogous to Equation 3 gives:

$$\beta = \frac{d}{2m} \tag{22a}$$

$$\omega_d = \sqrt{\beta^2 - \frac{c}{m}} \tag{22b}$$

$$\lambda = -\beta \pm \omega_d \tag{22c}$$

A comparison of the eigenvalue equations shows that the difference between Newton and dialectic mechanics can be viewed as a modification of damping and mass, while the stiffness remains untouched.

$$m' = m + d_{el}T_D \tag{23a}$$

$$d' = d + cT_D \tag{23b}$$

$$c' = c \tag{23c}$$

$$d_{el} = d + cT_D \tag{23d}$$

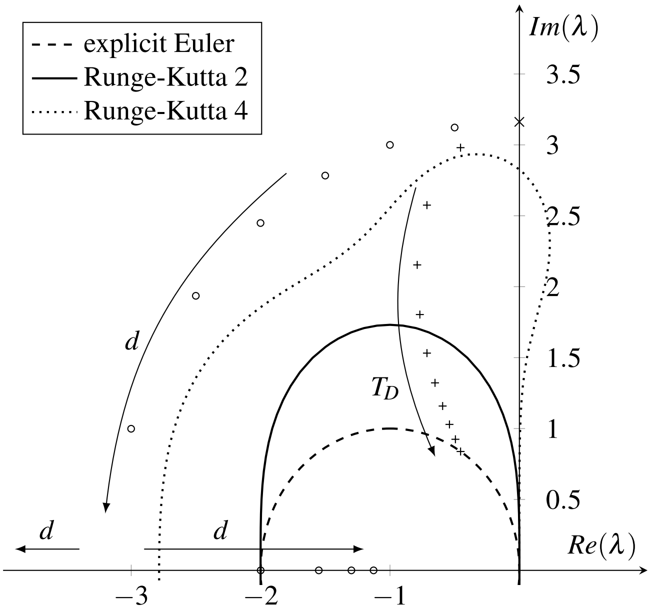


Figure 4. Changes in eigenvalues from original eigenvalues (×), when increasing d (○) or increasing T_D (+) and stability regions of selected integration methods. Systems are simulated stably, if their eigenvalues lie within the plotted contours.

Results of numerical experiments related to this observation will be presented in subsection 4.1. Notice that without the additional damper introduced in Equation 1b the modification is limited to the damping parameter.

The design parameters d_{el} and T_D allow a modification of the system dynamics without explicitly changing the values of m and d. Thus effects like gravity can still be calculated based on the original values. This is beneficial, as it prevents also changing the steady state of the system.

An advantageous choice of the additional damper parameter is $d_{el} = d + cT_D$, as shown in section 2. With this choice the eigenvalues of a stiff system with little damping are modified depending on the remaining design parameter T_D as shown in Figure 4. While the imaginary part of the eigenvalues becomes smaller with every increase in T_D, the real parts absolute value increases at first before decreasing again as the dialectic transformation gets more and more aggressive. In general the transformed eigenvalues show a strong trend towards the included stability regions of the solver methods.

Figure 4 also shows what happens, if just the damping of the original system is increased. As can be seen from Equation 23b for a single pair of eigenvalues this is equivalent to the modification done by dialectic mechanics as presented in Zimmer and Oldemeyer (2023) without the additional damper. The values of d were chosen based on the varied T_D according to Equation 23b to get a better comparison between the two approaches. Since the transformed eigenvalues travel along the semicircle with radius of the eigenfrequency of the original system, they circumvent the solver stability regions of the explicit solver methods. This modification however is sufficient for the use of

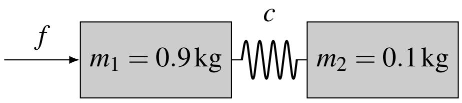


Figure 5. Two masses coupled by a spring as implemented by example model in subsection 4.2

implicit methods, because it effectively lower the imaginary part of the eigenvalues, thereby suppressing high frequencies.

4 Modelica examples

In order to verify the theoretical results derived above, example Modelica models are built with the dialectic planar mechanics library introduced in Zimmer and Oldemeyer (2023). The models are simulated using Dymola.

Equations 1b and 23d are added to the elastic components, for which they are relevant.

4.1 Spring Damper System

Section 3 showed, that dialectic mechanics can be viewed as a modification of the parameters of the Newton system, for a simple mass spring damper system. To verify this the system is built twice, once with the Modelica Standard Library (MSL) and once with the Dialectic Planar Mechanics Library (DPM) introduced in Zimmer and Oldemeyer (2023).

Table 1 lists the parameters used. The parameters on the right have been calculated according to Equation 23. Both models are simulated with the same solver settings using an explicit fixed step solver. Dymola's Linear Analysis feature confirms, that both models have the same eigenvalues. The masses return to their equilibrium from the same starting position.

When comparing the masses trajectory, a first order filter with time constant T_D is required for the position s in the dialectic model to receive the exact same output in both models. This raises the question of whether to include first order filters in sensor implementations in the DPM library.

Table 1. Parameterizations of Spring Damper models

DPM	*MSL*
$m = 1\,\mathrm{kg}$	$m = 2.1\,\mathrm{kg}$
$d = 100\,\frac{\mathrm{Ns}}{\mathrm{m}}$	$d = 1100\,\frac{\mathrm{Ns}}{\mathrm{m}}$
$c = 1\times10^6\,\frac{\mathrm{N}}{\mathrm{m}}$	$c = 1\times10^6\,\frac{\mathrm{N}}{\mathrm{m}}$
$T_D = 0.001\,\mathrm{s}$	-

4.2 Moving two stiffly coupled masses

An example showing the capability of dialectic mechanics to use stiff springs as a means for force transfer is shown in Figure 5. Two bodies are connected by a stiff spring. A force f of $10\,\mathrm{N}$ is applied to m_1 in such a way that it results in a point-to-point motion of both bodies. The forces acting between both bodies in the DPM model are compared with the corresponding forces in the MSL model. Since elastic and kinetic forces are separated in the DPM library, the spring forces are calculated within the spring component from the stiffness and the displacement. In contrast to the previous example the MSL model is parameterized with the same values as the DPM model. The stiffness c is chosen together with the step size of the 3rd order Runge-Kutta method to bring the MSL model close to the border of the stability region. Figure 6 shows an excerpt of the simulation results. The dialectic model reacts slower to the start of the movement and oscillations of the force values die down much quicker towards a value of zero than in the other model. Note that the oscillations in the MSL model are damped as well despite there being no damping modeled. The damping is introduced by the solver method itself as described by Cellier and Kofman (2006). Looking at the mean value of the difference between both signals, which is close to zero, verifies that the dialectic model approximates the MSL model well by filtering the high frequency oscillations. Although they have a high amplitude, these oscillations have little effect on the large movement of the two bodies. The oscillations are visible in the difference between both bodies' positions, but their final positions are exactly the same.

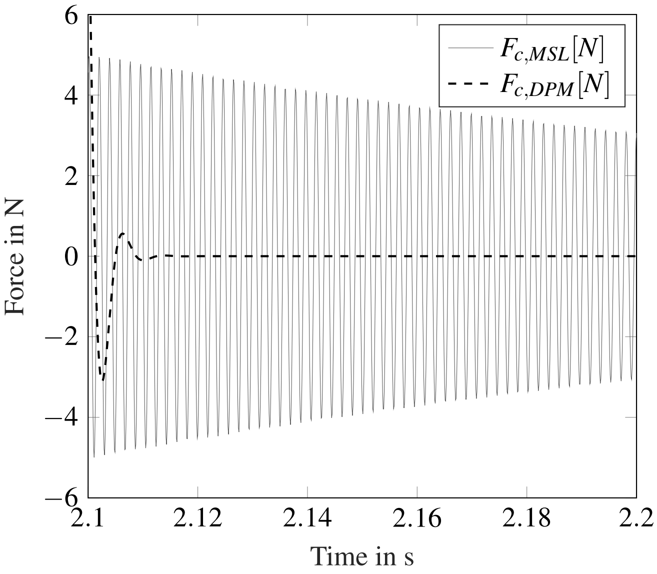


Figure 6. Simulation results for DPM and MSL model of the system in Figure 5. Plotted is an excerpt of the respective spring forces beginning with a step of the applied force f from $10\,\mathrm{N}$ to $0\,\mathrm{N}$. Under the assumption that the modeler is mainly interested in the macroscopic movement of the bodies, accuracy in the high-frequency oscillations is not necessary to fulfill the models purpose.

After establishing the difference between MSL and DPM models, it is time to make use of the advantages of the DPM model and increase the stiffness beyond what the MSL model could do without changing solver settings.

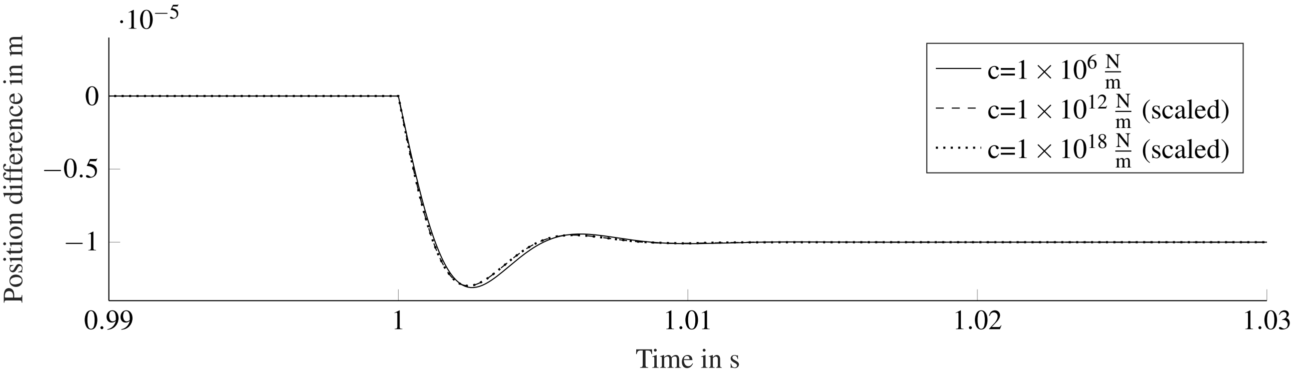


Figure 7. Simulation results for DPM model of two stiffly coupled masses for different values of the spring stiffness. Plotted are the differences of position between the two masses. The values have been scaled for the stiffer simulation runs by $1 \cdot 10^6$ and $1 \cdot 10^{12}$ to have all curves in the same order of magnitude.

The order of magnitude of the spring stiffness is doubled and tripled in successive simulation runs. While the MSL model becomes unstable with these settings, the dialectic simulates as expected as Figure 7 shows. The difference in the masses positions gets smaller as expected, when increasing the stiffness. In the plot this has been adjusted by applying scaling in order to focus on the transient behavior. The small change in dynamics fits the observation formulated in Equation 20. A T_D of 0.001 s leads to a limit period of 0.007 25 s, which is roughly the distance that can be measured between minima in Figure 7.

4.3 Performance considerations

The strength of damped dialectic mechanics mainly lies in enabling the simulation with settings, that would otherwise be unstable. More equations and variables lead to a higher computational cost for function evaluations. First observations indicate an increase of approximately 20% in the dialectic example models used in this section, when compared to the MSL ones. Hence using the dialectic model for systems suitable for simulation with classic MSL models might slow down simulation.

Variable step solvers in combination with stiff systems benefit from using the dialectic approach by enabling larger step sizes. Once the number of necessary function evaluations decreases significantly, a simulation speed-up is observed.

5 Discussion

Summing up the previous sections, damped dialectic mechanics appears to be a powerful tool for simulating models including challenging eigendynamics with explicit solvers. The central result is the guaranteed limit to the absolute value of the transformed eigenvalues. Since this limit is configured via a single parameter adaption to different use-cases should be straightforward. Heavily modifying fast dynamics, while largely keeping slow dynamics the same is another beneficial property. The integration into the Dialectic Planar Mechanics library enables further exploring example applications within existing tools. Additional attention should be dedicated to more complex systems and how dialectic mechanics works within higher order or nonlinear systems.

For robotic applications a 3D implementation of dialectic mechanics is necessary. Especially the inclusion of 3D rotations might introduce different challenges, that require addressing. Existing models will need to be rebuild for use with dialectic mechanics. To avoid this the feasibility of automatic model transformation or adapters between a MSL and a DPM part in one model should be checked.

References

Arnold, Martin, Bernhard Burgermeister, and Alexander Eichberger (2007-02). “Linearly implicit time integration methods in real-time applications: DAEs and stiff ODEs”. In: *Multibody System Dynamics* 17.2-3, pp. 99–117. DOI: 10.1007/s11044-007-9036-8.

Cellier, Francois E. and Ernesto Kofman (2006). *Continuous System Simulation*. Springer, p. 643. ISBN: 9780387261027.

Dahmen, Wolfgang and Arnold Reusken (2008-03). *Numerik für Ingenieure und Naturwissenschaftler*. Springer-Verlag GmbH. ISBN: 3540764925.

Hairer, E. and Gerhard Wanner (1991). *Solving Ordinary Differential Equations II: Stiff and Differential - Algebraic Problems*. Springer-Verlag, p. 601. ISBN: 3540537759.

Higham, Desmond J. and Lloyd N. Trefethen (1993). “Stiffness of ODEs”. In: *BIT*. DOI: 10.1.1.140.12.

Zimmer, Dirk and Carsten Oldemeyer (2023). “Introducing Dialectic Mechanics”. In: *Proceedings of the 15th International Modelica Conference 2023, Aachen, Germany*.

Testing the verification and validation capability of a DCP based interface for distributed real-time applications

Mikel Segura[1] Alejandro J. Calderón[1] Tomaso Poggi[2] Rafael Barcena[3]

[1]Dependable Embedded Systems Team, IKERLAN, José María Arizmendiarrieta, 2, Arrasate 20500, Basque Country, Spain, {msegura,ajcalderon}@ikerlan.es
[2]Mondragon Unibertsitatea, Loramendi Kalea, 4, Arrasate, 20500, Basque Country, Spain, tpoggi@mondragon.edu
[3]Department of Electronic Technology, University of the Basque Country (UPV-EHU), Torres Quevedo Ingeniariaren Enparantza, 1, Bilbao, 48013, Basque Country, Spain, rafa.barcena@ehu.es

Abstract

Cyber-physical systems are composed of a variety of elements developed by different vendors that are often geographically distributed. Therefore, its development process presents a double challenge: each element has to be developed individually and, at the same time, a correct interaction with the rest of the elements has to be ensured. In a previous work, we proposed and developed an interface, based on the non-proprietary Distributed Co-simulation Protocol standard, to ease the interaction between these elements. In this paper, we improve it to be applicable in a variety of hardware platforms and we test its applicability for the verification and validation process. To do so, firstly, we prove that our interface is hardware agnostic, demonstrating its easy implementation on different platforms. Secondly, we test its applicability in different X-in-the-Loop simulations. Finally, we also test its behaviour in distributed real-time executions, a necessary requirement for linking elements from different suppliers and helping to preserve their Intellectual Property.

Keywords: Simulation interface, Real-time, Intellectual Property protection, Distributed Co-Simulation Protocol, Verification and Validation

1 Introduction

Model-based design (MBD) is a commonly used practice for the development of cyber-physical systems (CPS) (Böhm et al. 2021). This process consists of developing virtual models that reproduce certain behaviours of a real system, thus avoiding the need to create costly physical prototypes and facilitating the process of validation and verification (Marwedel 2021). It is common that these models are located in different modelling and simulation (M&S) environments, either because they have been developed by different vendors and they want to preserve confidentiality (Falcone and Garro 2019), or because it is wanted to implement part of the model on a specific hardware platform in order to verify its performance in a specific environment (Alfalouji et al. 2023). Testing the correct interaction between these elements at an early stage of the development phase facilitates the process of validation and verification of them. However, as stated in (Attarzadeh-Niaki and Sander 2020), it is common to solve the challenge of linking such elements using ad-hoc methods. In (Segura, Poggi, and Barcena 2021) we argue the lack of a language and platform independent co-simulation architecture to address this problem and in (Segura, Poggi, and Barcena 2023) we propose a solution for it, presenting an architecture based on the non-proprietary Distributed Co-Simulation Protocol (DCP) standard. Nevertheless, we did not dive into how it could be deployed on different hardware platforms and thus, demonstrate how the verification and validation process can be simplified. It is worth mentioning that this implementation would save time, resources and money, as it facilitates the coupling of elements, saves displacements for integration testing and helps to preserve Intellectual Property.

Accordingly, this article discusses three points. First, we demonstrate the easy implementation of this interface on a variety of hardware platforms, deploying it in generic but different targeted hardware platforms such as, the Xilinx Zynq UltraScale+, Xilinx Zynq-7000 SoC ZC702, NVIDIA Jetson Nano, and Raspberry-Pi. Second, taking into account that the use of X-in-the-Loop (XIL) simulations is widely extended in the development of CPSs, we analyse the limitations of our interface in a real-time communication between a development software such as Simulink and the platforms mentioned above. Finally, in order to show how our interface can solve the challenge of communicating systems developed by geographically distributed suppliers, we link the Xilinx Zynq-7000 SoC ZC702 with the Raspberry-Pi in a real-time simulation using our interface via UDP communication.

The paper is structured as follows. Section 2 addresses the need for a generic architecture for co-simulation. Section 3 introduces the generic interface that enables performing co-simulations between a variety of simulation environments. Section 4 explains the tests that we executed to demonstrate the applicability of the interface. Section 5 exposes the results of the conducted tests. Section 6 analyses the results of the previous section. Finally,

DOI
10.3384/ecp204245

Section 7 presents the conclusions and future work.

2 Background and Related Work

Co-simulation is used to couple different simulation environments (e.g. a continuous and an event driven simulation environment) in order to use an appropriate simulation environment for each part of the system (Köhler 2011). Co-simulation can also be used to link spatially distributed models (Baumann et al. 2019). Additionally, in the development and verification process of control systems, different co-simulation techniques referred to as X-in-the-Loop (XIL) (Ivanov et al. 2019) are used, encompassing the well-known Model-in-the-Loop (MIL), Software-in-the-Loop (SIL), Processor-in-the-Loop (PIL), or Hardware-in-the-Loop (HIL) techniques. Nevertheless, despite being a widely used technique, coupling problems often arise and there is no a generic methodology for linking different simulation environments. This problem is detected in several works, where different co-simulation architectures are proposed to solve it. For instance, (Hatledal et al. 2019) presents a language- and platform-independent co-simulation framework based on the Functional Mock-up Interface (FMI). Its major drawback is that it has not considered the integration of real-time systems or hardware-in-the-loop simulations. In (Ivanov et al. 2019) the authors propose an architecture applicable in a variety of XIL approaches and it is intended to perform real-time and distributed co-simulations in different geographical locations. However, they use a proprietary architecture that is not enforced by any standard and there is no mention of how it could be implemented in different modelling and simulation environments, which suggests that its integration is not straightforward. The architecture proposed in (Attarzadeh-Niaki and Sander 2020) attempts to avoid ad-hoc approaches and it is focused on guaranteeing IP protection, however, no mention is made on real-time applications.

Some standards also aim to manage distributed co-simulations, such as Distributed Co-Simulation Protocol (DCP) (Modelica Association 2023b). DCP is a non-proprietary standard that aims to integrate real-time systems into co-simulation environments. It follows the master-slave principle and it is independent of the communication medium, as it works over common transport protocols such as Bluetooth, UDP, or CAN. However, as the DCP is a relatively new standard, it has a limited applicability in terms of simulation environments. Furthermore, its operation resides in encapsulating the systems to be communicated, thus a particular DCP slave must be created for each application.

In previous works, we propose (Segura, Poggi, and Barcena 2021) and present (Segura, Poggi, and Barcena 2023), a co-simulation architecture, based on non-proprietary standards, that facilitates coupling between different M&S environments and hardware platforms. By relying on a non-proprietary standard such as DCP, the architecture is implementable without any intellectual property restrictions and on top of that, it is compatible with any other DCP slave. In comparison with the DCP, we propose a generic co-simulation interface, i.e., the system to be communicated is independent to the interface and there is no need to develop a specific slave for each application. In (Segura, Poggi, and Barcena 2023) we extended the scope of the DCP by creating a Simulink library, allowing Simulink to be easily integrated not only into our architecture, but also into any DCP application. Moreover, our architecture is agnostic to the simulation platform and to the communication medium, which facilitates cross-platform migration, which is what we will demonstrate in this article. Additionally, it enables real-time co-simulation. This is boosted by the Simulink implementation, that is, thanks to the automatic code generation capability of Simulink, we can convert our interface into source code (e.g. C or C++) and execute it on a wide variety of platforms, facilitating the validation of the system to be developed. In this work we take advantage of this capability by adapting the Simulink code developed previously, so it can be directly implemented on different platforms. Consequently, we provide a means to perform real-time communication between the models executed on such platforms.

As this paper deals with real-time (RT) simulations, it is worth having a little background on the characteristics of these systems. First, it is worth mentioning that, in real-time systems, the instant in which the response occurs is as important as the response itself (Kopetz 2011). If the response does not arrive at a predefined time, called deadline, the response may be unusable and may have adverse consequences for the system. Another characteristic of these systems is the so-called wall-clock. All computational elements involved in a real-time simulation must have a common clock reference and be synchronised to it. The higher the accuracy of this synchronisation, the better the system will be able to carry out temporally more constrained simulations. Determinism is another characteristic of these systems, it indicates the reproducibility of the system. That is, if we run several simulations of the same system, where all its components start at the same time and with the same starting conditions, the determinism means ability of the system to replicate the results at the same time instants.

On the other hand there are non real-time (NRT) simulations. These are controlled simulations that usually repeat a read-compute-write sequence, where they first wait for receiving data, then process it, and finally write the result at the output port. Once this cycle is finished, the next cycle starts following the same sequence. Comparing with real-time systems, they do not have to provide a temporally accurate response. It is to say, the message transport latency can vary without affecting the behaviour of the system. Simulink, for example, is a tool that by default runs in NRT, however, it also has a tool called Simulink Desktop Real-Time (MathWorks 2023), which allows us to synchronise the simulation with the wall-clock. This tool has two modes of use: I/O mode and kernel mode. The first one synchronises the I/O drivers with the real-time clock and allows us to perform real-time executions up to 1 kHz (1 ms sampling time), this is the one that we use in this work.

3 Proposed interface

The interface we propose is based on the non-proprietary DCP standard, thus it must be configured as a DCP slave. Nevertheless, as depicted in Figure 1, its behaviour is not that of a conventional DCP slave. Our implementation focuses on transmitting information from one environment to another and it is completely model-independent. Whereas in conventional usage, the slave wraps the model (Krammer et al. 2020), having to link them internally by hand. From a practical point of view, there is a big difference, since in the original paradigm a specific DCP slave has to be developed for each application, whereas our proposal is designed to indicate only the number of input/outputs plus an easy configuration of them. To achieve this independence between the model and the DCP slave, apart from implementing a specific DCP slave, we also developed a series of peripheral modules, which are explained in (Segura, Poggi, and Barcena 2023). Thus, our interface is composed of these modules and a DCP slave. Our goal in developing this interface as a Simulink library, was to take advantage of its tools so that we could generate C/C++ code for our interface and implement it on a variety of hardware platforms without additional modifications.

3.1 Configuration of the interface

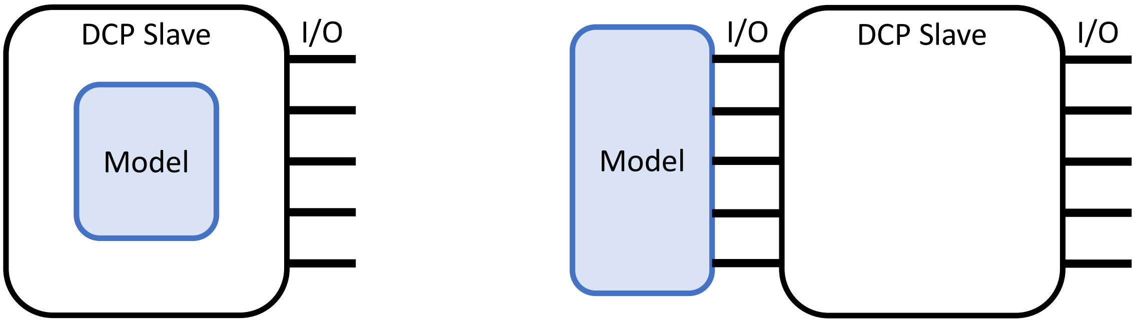
We have advanced that the configuration of the interface is based on the DCP standard, therefore, we will use the DCP standard specification document (Modelica Association 2023a) for the explanations of this section. The references to specific clauses of the standard will be in italics to better guide the reader. In this section we will only focus on the essential parameters (in monospace font) to define our interface, however, there are also other optionally modifiable parameters whose explanation can be found in the standard specification document. Figure 2 will be helpful to understand certain concepts.

Some parameters are set in each slave, while others are set in the master. The slaves are limited to set their internal parameters (see *5.4 Definition of dcpSlaveDescription Element, pp. 80-82*), of which, the essential parameters for configuring the interface are defined by the following elements:

- Time resolution (*5.9 Definition of TimeRes Element, pp. 87-88*). It defines one atomic step of the slave and it is represented by an element that contains a list of permissible single time resolutions or a list of resolution ranges. To set a single resolution, that is what we are going to do next, we have to set two sub-parameters: `numerator` and `denominator`. Where the `numerator` divided by the `denominator` represents the time resolution of the slave.

- Transport protocol (*5.11 Definition of TransportProtocols Element, p. 88*). The DCP supports multiple transport protocols and this element is used to store their specific settings. For instance, if UDP transport protocol is used, this element must indicate it and contain the `host` and `port` data of the slave.

- Variables (*5.13 Definition of Variables Element, pp. 92-98*). This element contains the information about the variables of the slave, they can be either an Input, an Output, a Parameter, or a Structural Parameter. Among its sub-parameters, the indispensable ones to configure our interface are: `valueReference`, `dataType`, and `declaredType`. `valueReference` is the identifier of each variable and its value must be unique, in Figure 2 it can be seen how each I/O of each slave has different `valueReference` (vr) values. With `dataType` we declare the data type of the variable,



(a) Original DCP Slave design representation. Internally linked DCP Slave and Model. One specific DCP Slave for each Model.

(b) Proposed interface. New DCP Slave design. Externally linked DCP Slave and Model. The same slave with easily configurable Input/Outputs.

Figure 1. Comparison between original DCP Slave design and our interface.

see *Table 174: Data type elements* to know the accepted data types by the DCP standard. Finally, with `declaredType` we indicate whether the variable is used as an input/output that connects to another slave or as an input/output that connects to an external model. For this purpose, we have two predefined options: *default*, for communication between slaves, and *interface* for communication with the models.

As the task of the interface is to communicate different co-simulation environments, inputs and outputs will always have to be declared in pairs. Each pair will be internally linked in an automatic way as long as their `valueReferences` are consecutive. In other words, the `valueReference` parameters of an input-output pair must be consecutive. These values shall consist of the pairs 1-2, 3-4, ..., regardless of which of the two is the value of the input and which of the output. Additionally, if an output of one interface communicates with an input of another interface, both must have the same `valueReference`. This is shown in Figure 2. In this way we determine the link between interfaces and certify a correct communication.

The master, on the other hand, configures how the inputs and outputs of the slaves should communicate with each other. That is to say, it indicates to each slave where the inputs corresponding to its outputs are and vice versa; in addition, it establishes the sending frequency of each output. To do this, the following parameters must be configured:

- Step Size. The master defines the step size of each output of all slaves. The step size is a multiple of the time resolution parameter mentioned in the slave configuration. That is, the step size of each output is defined by the time resolution of its slave multiplied by the `step` parameter.
- Data Identifier (`data_id`). When exchanging information between slaves, Outputs are communicated to Inputs via DAT_input_output PDUs. Thanks to the `data_id` parameter, the values of several outputs of a slave can be grouped in a single DAT_input_output PDU. Only outputs that have the same configuration, i.e. sender, receiver and step size, can be grouped together.

4 Methodology

In this section we explain the experiments that we performed to show how the architecture presented in (Segura, Poggi, and Barcena 2021) is applicable on several hardware platforms, and how it is applicable on a distributed real-time co-simulation application. To do so, we apply it on four different platforms, which are introduced in subsection 4.1. As proof-of-concept use case, we use a closed-loop control model explained in subsection 4.2. In subsection 4.3 we present the co-simulation scenarios we use to demonstrate the applicability of the interface. Finally, in subsection 4.4, we explain how to configure our generic co-simulation interface for this particular use case.

4.1 Hardware platforms

In order to test the applicability of our architecture, and thus of our interface, it has been decided to work with hardware platforms designed for different purposes, concretely we used:

- Hardware platforms with integrated FPGA, such as the Xilinx Zynq UltraScale+ and the Xilinx Zynq-7000 SoC ZC702.
- Hardware platforms with integrated GPU, such as the NVIDIA Jetson Nano.
- Generic hardware platforms such as the Raspberry Pi 3B, which is very accessible and widely used.

By working with platforms that integrate FPGAs or GPUs, we are able to introduce these technologies into co-simulations, expanding our design to new applications, such as simulation accelerators, artificial intelligence, or image processing.

However, for now our interface has two limitations. Firstly, it is only implemented to run on soft real-time (SRT) and hard real-time (HRT) operation modes, the non-real-time (NRT) operation mode has not yet been implemented. Therefore, as both implemented modes require a common clock reference shared by all computing elements, the interface must be implemented in an environment that can provide it. Secondly, for the moment, we have only implemented UDP communication, so the boards must have an Ethernet port.

4.2 Use case: control of a closed-loop system

As was done in (Segura, Poggi, and Barcena 2023), as a proof of concept we use a closed-loop control system. This system consists of two parts: a plant, which is modeled as a discrete time first-order system, represented by Equation 1; and a PI control algorithm, represented by Equation 2. The simulation analysis is done by observing the time evolution of the closed-loop system response to a step input, comparing both the transient and steady-state parts. It is worth pointing out that this work is focused on testing the capability of the interface to communicate distributed systems in real-time. Therefore, in order to facilitate the demonstration process, we have chosen this simple use case.

$$y(k) = a \cdot y(k-1) + b \cdot u(k) \tag{1}$$

Where:

- u is the control signal.
- y is the plant output or feedback signal.

- a is a constant parameter, and it was permanently set to $a = 0.99$.
- b is a constant parameter, and it was permanently set to $b = 0.01$.

$$u(k) = \left[K_p + K_i T_s \frac{1}{z-1}\right] e(k) \qquad (2)$$

Where:

- u is the control signal.
- K_p is the proportional gain coefficient.
- K_i is the integral gain coefficient.
- T_s is the sampling period.
- $e(k) = r(k) - y(k)$ is the error signal.
- $r(k)$ is the target reference signal.
- z is the unit delay operator.

4.3 Co-simulation scenarios

In (Segura, Poggi, and Barcena 2023) we presented different scenarios to explain the development process of the interface. We started with a scenario composed only of Simulink and ended up with a scenario where the control algorithm was running in an UltraScale+ and the plant in Simulink. However, in order to implement the control subsystem on the UltraScale+ board, we manually created a DCP slave using C++ code that was specifically adapted to work on this board and be compatible with the control algorithm. Now we want to progress in the development of the interface and test its applicability. To do so, following the MBD methodology, we have automatically generated the interface code from the Simulink model, using the Embedded Coder tool. In order to be able to generate code correctly, we adapted the Simulink model by adding blocks and creating functions compatible with this generation. After that, we were able to generate directly implementable code, without the need for any changes, in any of the platforms presented in subsection 4.1.

Figure 2 represents the co-simulation scenario. In it, we can see the two models that compose the use case, defined in the subsection 4.2, located in different simulation environments and communicated by two entities/slaves of our interface. On the left we can see the *Model 1*, where the control algorithm is located. On the right, we can see the *Model 2*, which is composed of the plant and a synchronisation mechanism. The latter has the function of ensuring a controlled start of closed-loop control applications. Guaranteeing identical starts in all executions help us analyze the interface behaviour. For a detailed description of its operation refer to (Segura, Poggi, and Barcena 2023). The communication medium used to link both systems is UDP.

To demonstrate the easy implementation capability of the interface on different hardware platforms and, at the same time, to analyse its scope for performing real-time XIL simulations, we have considered the following scenarios:

- Scenario 1.A: Control algorithm and interface in the ARM-based processor of the ZC702 and Plant in Simulink.
- Scenario 1.B: Control algorithm and interface in the ARM-based processor of the UltraScale+ and Plant in Simulink.

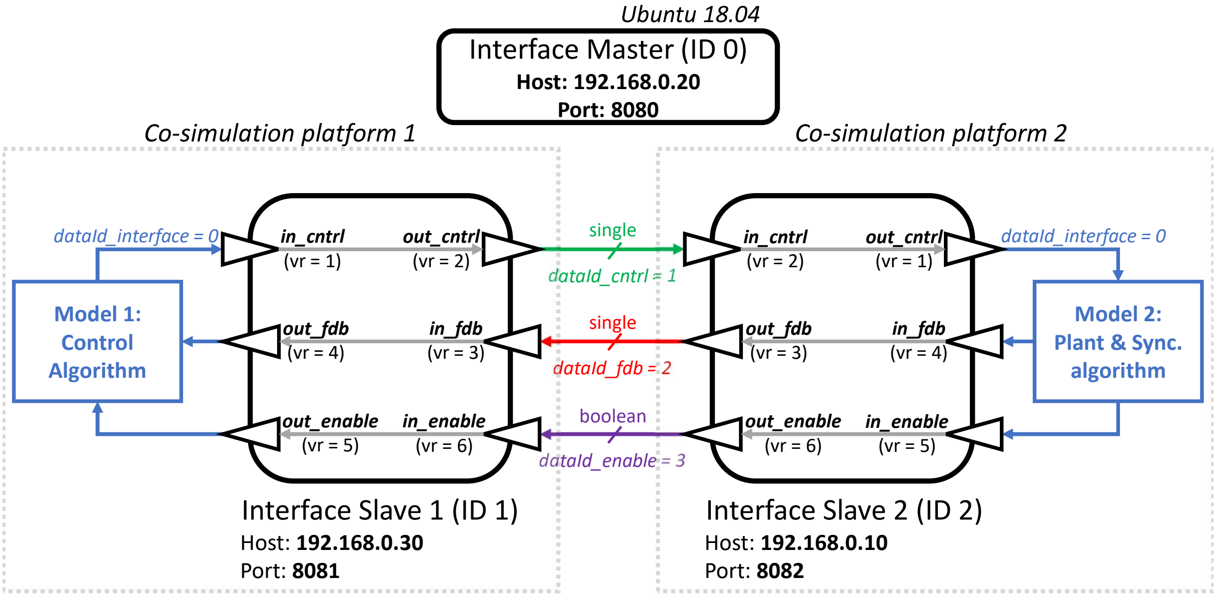


Figure 2. Interface configuration of the employed use case.

- Scenario 2: Control algorithm and interface in the ARM-based processor of the Jetson Nano and Plant in Simulink.
- Scenario 3: Control algorithm and interface in the CPU of the Raspberry-Pi and Plant in Simulink.
- Scenario 4.A: Control algorithm in the FPGA of the ZC702, interface in the ARM-based processor of the ZC702, and Plant in Simulink.
- Scenario 4.B: Control algorithm in the FPGA of the UltraScale+, interface in the ARM-based processor of the UltraScale+, and Plant in Simulink.

Additionally, to demonstrate its applicability in distributed real-time executions, we have proposed the following scenario:

- Scenario 5: Control algorithm in the FPGA of the ZC702, interface in the ARM-based processor of the ZC702, and Plant in the CPU of the Raspberry-Pi, see Figure 3.

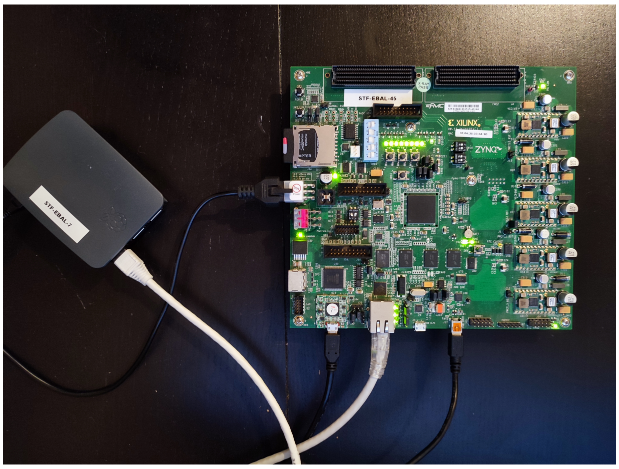


Figure 3. Scenario 5.

It is worth to mention that in all scenarios we have kept the same interface configuration (see subsection 4.4), thus facilitating the interoperability between simulation tools.

As we mentioned in subsection 4.1, our interface is limited to work in real-time operation mode, therefore all the scenarios have to be executed in real-time. The DCP standard defines that in its real-time mode, all components within the simulation must be synchronised with POSIX time. That is to say, they have to synchronise their clock by reference to 1 January 1970, 00:00:00 UTC. Consequently, we have to make all the components run in an environment that supports it and make them synchronised with each other. To do this possible on the platforms, we have installed an Ubuntu operating system on the CPUs of all of them, whose clock will be synchronised with the POSIX time. Therefore, the interface and the system (plant or controller) will run on it. Regarding Simulink, as explained above, by default it works in non-real-time mode. However, we will use its Simulink Desktop Real-Time tool in I/O mode, which allows us to synchronise the UDP ports with the wall-clock. This way, it will be also be synchronised to the POSIX time. It should be noted that we will run Simulink on a conventional PC that contains a 6 core Intel Core i7 CPU processor and using a Windows 10 operating system.

The use of the interface must not alter the behaviour of the closed-loop system in any of the scenarios. Therefore, we need a reference in order to be compared with the scenarios. To this end, using the Embedded Coder and HDL Coder tools provided by Simulink, we conducted Processor-in-the-Loop (PIL) and FPGA-in-the-Loop (FIL) simulations equivalent to the scenarios. In this way, we obtained a reference response for each of the scenarios. In other words, for scenarios 1.A, 1.B, 2, and 3 we performed PIL simulations, one with each hardware platform. While for scenarios 4.A and 4.B we performed FIL simulations. Each of these simulations are used as a reference for their respective scenario. Regarding scenario 5, we compare its responses to those of the system running entirely in Simulink.

4.4 Interface and simulations configuration

Two types of configurations were applied to conduct the experiment: the configuration of the models to be simulated and the configuration of the interface.

Regarding the configuration of the models, in (Segura, Poggi, and Barcena 2023) we saw that the behaviour of the interface varied depending on the execution time, therefore we performed the tests using six different configurations. In the current article, instead, we are going to focus on working only with the most limiting configuration that we encountered, which is shown at Table 1.

Configuration I	$T_s = 10$ ms	$K_p = 10$ $K_i = 10$

Table 1. Configuration for the simulation

Regarding the configuration of the co-simulation interface. In subsection 3.1 we explain the indispensable parameters to be configured. Now, we specify which values we chose for our particular use case:

- Time resolution. With this parameter we indicate under which step-size the state machine of the interface is executed. We set it to the lowest resolution that Simulink Desktop Real-Time allows when working in I/O mode. Therefore, we set it to $1ms$: $numerator = 1$ and $denominator = 1000$. It is also worth mentioning that the step-size of the interface must be lower than that of the model (Segura, Poggi, and Barcena 2023).
- Transport protocol. As mentioned, we use UDP. In

the Figure 2 we can see the chosen host and port values.

- Variables. We declared 3 inputs and 3 outputs to each interface. In the Figure 2 we can see the data types and the value reference of each one.
- Data identifier. In the Figure 2 we can see that each slave has been assigned four *data_id*. This means that each variable that is transmitted between slaves has grouped independently, while the outputs that go from the interfaces to each of the models have been grouped together.
- Step size. We assigned a step value of 3 to the three *data_id*s that are transmitted between slaves (i.e. dataId_cntrl, dataId_fdb, and dataId_enable) and a step value of 10 to dataId_interface. With this configuration we could have assigned the same data_id to the three signals that are transmitted between slaves, but this makes it easier in case of future modifications.

5 Results

In this section, we present the simulation results of the scenarios explained in subsection 4.3. They will be discussed in section 6. In order to prove that the system behaves identically every execution, as done in (Segura, Poggi, and Barcena 2023), we perform 25 executions of each scenario. Between each run, the system is reset in order to ensure identical starting conditions in each of them. Subsequently, we compare the time response of the $y(k)$ output of each scenario with the respective reference. From this comparison we obtain the error, which is calculated by means of Equation 3.

$$err = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\left[y(k) - y^{ref}(k)\right]^2} \qquad (3)$$

where N represent the steps executed in a simulation, $y(k)$ is the response of the closed-loop system at step k under a specific scenario, and $y^{ref}(k)$ is the response of the closed-loop system under the corresponding reference. Table 2 reports the maximum, the minimum, the mean and the standard deviation of the error over the 25 repetitions.

Figure 4 helps us understand the results of the table. It comprises two graphs, each comparing the response $y(k)$ of a scenario (red line) with its corresponding reference (green line). There are 25 red lines in each graph, corresponding to the 25 repetitions that were executed for each configuration. We have decided to show only these two scenarios because they are sufficient to explain the behavior of the rest.

6 Results Analysis

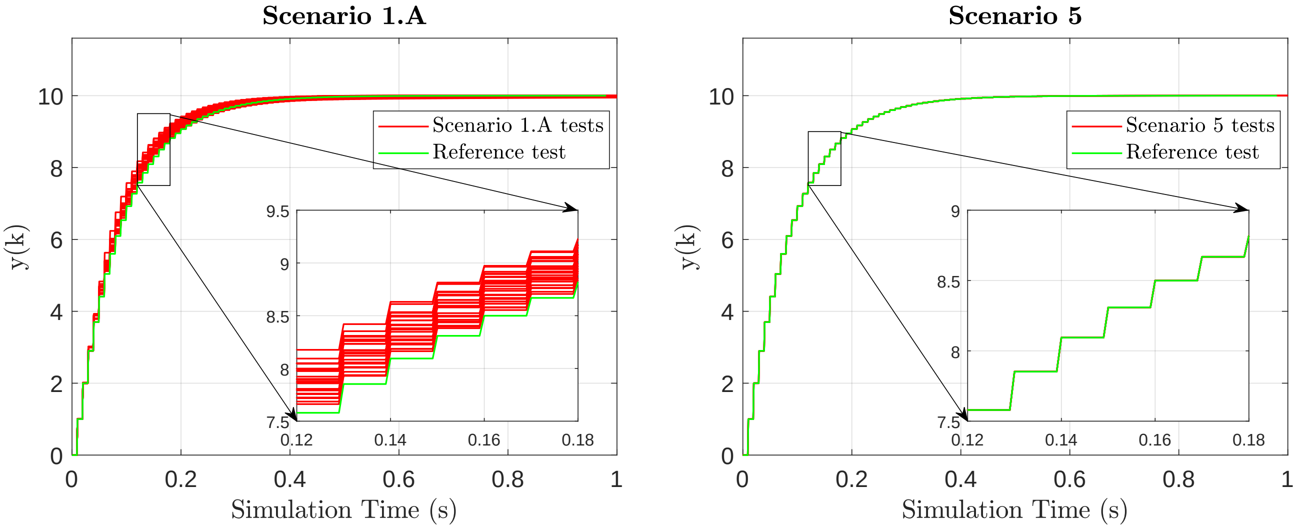
There are three topics we have discussed in this article, so this analysis will also be divided into three parts.

Scenario 1.A - PIL in ZC702	
max = 0.23479	min = 0.042779
mean = 0.12957	sd = 0.05131
Scenario 1.B - PIL in UltraScale+	
max = 0.21266	min = 0.024876
mean = 0.13098	sd = 0.048035
Scenario 2 - PIL in Jetson Nano	
max = 0.20944	min = 0.030067
mean = 0.11274	sd = 0.043965
Scenario 3 - PIL in RaspberryPi	
max = 0.20592	min = 0.050682
mean = 0.13517	sd = 0.04197
Scenario 4.A - FIL in ZC702	
max = 0.14277	min = 0
mean = 0.04963	sd = 0.0404
Scenario 4.B - FIL in UltraScale+	
max = 0.12031	min = 0
mean = 0.051518	sd = 0.036303
Scenario 5 - Distributed co-simulation	
max = 0	min = 0
mean = 0	sd = 0

Table 2. Comparison between MathWorks PIL/FIL solution and our Interface.

The first point focuses on evaluate the viability and ease of implementation of the interface in a variety of hardware platforms. Since we developed it in Simulink and adapted all its functions to be compatible with the generation of generic C/C++ code, we were able to implement it easily on all the platforms mentioned in subsection 4.1. This way we demonstrate that the interface can be migrated between platforms without any extra effort.

As for our second point, we evaluated the viability of the interface to perform real-time executions between Simulink and the platforms described in subsection 4.1. In Table 2, from Scenario 1A to Scenario 4.B, we find the results of the tests carried out for this point. Additionally, Figure 4a displays graphically the results obtained in Scenario 1A. As the graphs obtained in Scenarios 1A to 4B are very similar, we decided to omit the rest and display only this one to assist in the interpretation of Table 2. Analysing this table, we observe that there is an error in every scenario. Simplifying the results, we can say that the mean error of PIL simulations (from Scenarios 1A to 3) is in the order of 0.215 units (±0.02), whereas the mean error of FIL simulations (Scenarios 4A and 4B) is about 0.13 units (±0.01). This error occurs because the PIL and FIL simulations made with MathWorks tools exhibit identical behaviour in each run, while our simulations vary in each of the 25 runs. This can be seen in Figure 4a, where there is single green line (corresponding to the MathWorks re-



(a) Scenario 1A.

(b) Scenario 5. The responses overlap.

Figure 4. Comparison of the responses of Scenarios 1A and 5 with their respective References.

sponse) and multiple red lines (each corresponding to one of the 25 tests). Nevertheless, it can also be seen that this error is focused in the transient state, while in the steady state it is minimum. In fact, in the steady state, from second 0.4 onwards, the mean error is of the order of 0.019 units (±0.008), with a standard deviation of another 0.019 units (±0.007).

The appearance of this error means that our solution is not deterministic, which is an indispensable quality in the real-time executions for the verification and validation processes of the CPS. Therefore, we can say that our interface is not suitable for linking Simulink and hardware platforms in real-time. However, we did not test how our interface would behave with these scenarios working in non real-time mode, that is, following the controlled read-compute-write sequence explained previously. In fact, this is the way that MathWorks perform PIL and FIL simulations. Nevertheless, as explained before, this is an operation mode that we have yet to implement.

Analysing the cause of this non-deterministic response, we have not been able to link Simulink simulation time with POSIX time. In other words, POSIX time is constantly moving forward, and in a period of time, the simulation time get blocked and it does not advance. Figure 5 demonstrates this behaviour, where the graph instead of showing a perfect diagonal line shows "jumps", which are sometimes more pronounced. As we discussed previously, real-time systems must guarantee a response every predefined period and this breaks do not allow it. This can be caused, for instance, due to an interruption in the computing platform. This is the reason why we have not been able to benefit from all the power that the Simulink Desktop Real-Time tool provides.

The effect of this desynchronization in our scenarios is that the plant, which is executing in Simulink, stops running for an indefinite period; while the control algorithm on the hardware platform continues to run. During this period, the control will not receive updated input values from the plant, however they will be processed, provoking incorrect output control signals. When the plant resumes running, it reads the last of this unwanted values, resulting in an incorrect feedback value being sent to the controller. This way, a single desynchronization in the execution can significantly impact the system's behavior, leading to non-deterministic behavior. It is worth mentioning that the faster the system runs, the smaller its execution steps will be. As a result, a pause in the simulation time will involve more simulation steps, creating a more adverse effect on the system's response.

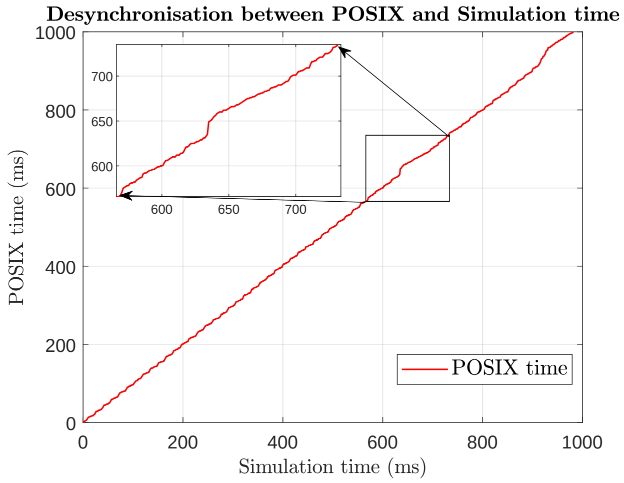


Figure 5. Desynchronisation between POSIX and Simulation time

Finally, it remains to analyse the response of our implementation in real-time distributed applications, i.e., the Figure 4b. Contrary to what happens in the previous tests, we can see how the 25 red lines are overlapped. On top of that, they have the same behaviour as the reference (green line). The fact that we obtained identical results in all executions means a deterministic simulation. Therefore, we can be assured that our interface is suitable for real-time distributed simulations. At the same time, these results demonstrate that the problem we had in linking Simulink with hardware platforms lies in the fact we were not able to link Simulink with the wall-clock correctly.

7 Conclusions and future work

In this paper we present an empirical demonstration of the applicability of the previously developed generic co-simulation interface. Specifically, we demonstrated i) its easy implementation on a variety of hardware platforms, and ii) how it can be used in real-time distributed simulations. Both capabilities are very useful in the process of verification and validation of cyber-physical systems, especially in those whose components are developed by different suppliers; in those where the system is split into smaller modules to spread the computational load across different processors; or in those where the integration of different simulators into a single system is required. Therefore our interface could save time, resources and money, as it facilitates coupling of elements, saves displacements for integration testing and helps to preserve Intellectual Property.

To test the interface we deployed it in different hardware boards and performed a closed-loop simulation between them, obtaining reliable responses. Consistent with the MBD methodology, as we have developed it in Simulink and take advantage of its code generation capabilities, our interface is easily implementable in a wide variety of simulation environments. Additionally, as our interface is based on the non-proprietary DCP standard, it is fully compatible with any other DCP slave.

Looking to extend our work to the future, in order to improve the linking capability to Simulink, we want to develop the non-real-time simulation mode. Additionally we have planned to test the applicability of this interface in a more complex use case involving hardware-in-the-loop simulations.

Acknowledgements

This work was supported by Basque Government through the ELKARTEK programme under the AUTOEV@L project (KK-2021/00123).

References

Alfalouji, Qamar et al. (2023). “Co-simulation for buildings and smart energy systems — A taxonomic review”. In: *Simulation Modelling Practice and Theory* 126, p. 102770. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2023.102770.

Attarzadeh-Niaki, Seyed Hosein and Ingo Sander (2020). “Heterogeneous co-simulation for embedded and cyber-physical systems design”. In: *Simulation: Transactions of the Society for Modeling and Simulation International* 96, pp. 753–765. DOI: 10.1177/0037549720921945.

Baumann, Peter et al. (2019). “Using the Distributed Co-Simulation Protocol for a Mixed Real-Virtual Prototype”. In: *Proceedings - 2019 IEEE International Conference on Mechatronics, ICM 2019*. IEEE, pp. 440–445. ISBN: 9781538669594. DOI: 10.1109/ICMECH.2019.8722844.

Böhm, Wolfgang et al. (2021). *Model-Based Engineering of Collaborative Embedded Systems*. 1st ed. Springer. Chap. 12 & 13. ISBN: 978-3-030-62135-3. DOI: https://doi.org/10.1007/978-3-030-62136-0.

Falcone, Alberto and Alfredo Garro (2019). “Distributed Co-Simulation of Complex Engineered Systems by Combining the High Level Architecture and Functional Mock-up Interface”. In: *Simulation Modelling Practice and Theory* 97. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2019.101967.

Hatledal, Lars Ivar et al. (2019). “A Language and Platform Independent Co-Simulation Framework Based on the Functional Mock-Up Interface”. In: *IEEE Access* 7, pp. 109328–109339. DOI: 10.1109/ACCESS.2019.2933275.

Ivanov, Valentin et al. (2019). “Connected and shared x-in-the-loop technologies for electric vehicle design”. In: *World Electric Vehicle Journal* 10, pp. 1–13. DOI: 10.3390/wevj10040083.

Köhler, Christian (2011). *Enhancing Embedded Systems Simulation*. 1st ed. Vieweg+Teubner. Chap. 2. ISBN: 978-3-8348-1475-3. DOI: https://doi-org.ehu.idm.oclc.org/10.1007/978-3-8348-9916-3.

Kopetz, Hermann (2011). *Real-Time Systems*. 2nd ed. Real-Time Systems Series. New York: Springer, pp. XVIII, 378. DOI: https://doi.org/10.1007/978-1-4419-8237-7.

Krammer, Martin et al. (2020). “A Protocol-Based Verification Approach for Standard-Compliant Distributed A Protocol-Based Verification Approach for Standard-Compliant Distributed Co-Simulation”. In: *Asian Modelica Conference 2020*. DOI: 10.3384/ecp20174133.

Marwedel, Peter (2021). *Embedded System Design - Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. 4th ed. Springer. Chap. 1. ISBN: 978-3-030-60909-2. DOI: https://doi.org/10.1007/978-3-030-60910-8.

MathWorks (2023). *Simulink Desktop Real-Time*. https://es.mathworks.com/help/sldrt/low-sample-rate-simulation.html.

Modelica Association (2023a). *DCP standard specification*. https://github.com/modelica/dcp-standard.

Modelica Association (2023b). *Distributed Co-Simulation Protocol (DCP) website*. https://dcp-standard.org/.

Segura, Mikel, Tomaso Poggi, and Rafael Barcena (2021). “Towards the implementation of a real-time co-simulation architecture based on distributed co-simulation protocol”. In: *35th Annual European Simulation and Modelling Conference 2021, ESM 2021*. EUROSIS-ETI, pp. 155–162. ISBN: 978-9-492-85918-1.

Segura, Mikel, Tomaso Poggi, and Rafael Barcena (2023). “A Generic Interface for x-in-the-Loop Simulations Based on Distributed Co-Simulation Protocol”. In: *IEEE Access* 11, pp. 5578–5595. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3237075.

Towards the separate compilation of Modelica: modularity and interfaces for the index reduction of incomplete DAE systems

Albert Benveniste[1] Benoît Caillaud[1] Mathias Malandain[1] Joan Thibault[1]

[1]Inria centre at Rennes University / IRISA, France,
{albert.benveniste,benoit.caillaud,mathias.malandain,joan.thibault}@inria.fr

Abstract

A key feature of the Modelica language is its object-oriented nature: components are instances of classes and they can aggregate other components, so that extremely large models can be efficiently designed as "trees of components". However, the structural analysis of Modelica models, a necessary step for generating simulation code, often relies on the flattening of this hierarchical structure, which undermines the scalability of the language and results in widely-used Modelica tools not being able to compile and simulate such large models.

In this paper, we propose a novel method for the modular structural analysis of Modelica models. An adaptation of Pryce's Sigma-method for non-square DAE systems, along with a carefully crafted notion of component interface, make it possible to fully exploit the object tree structure of a model. The structural analysis of a component class can be performed once and for all, only requiring the information provided by the interface of its child components. The resulting method alleviates the exponential computation costs that can be yielded by model flattening; hence, its scalability makes it ideally suited for the modeling and simulation of large cyber-physical systems.

Keywords: DAE, Modelica, object-oriented modeling, index reduction, structural analysis, linear programming, interface theory, difference bound matrices

1 Introduction

System modeling tools are key to the engineering of safe and efficient Cyber-Physical Systems (CPS). Although ODE-based languages and tools, such as Simulink (MathWorks, Inc. 1994–2023), are widely used in industry, there are two main reasons why DAE-based modeling is best suited to the modeling of such systems: it enables a modeling based on first principles of the physics; it is physics-agnostic, and consequently accomodates arbitrary combinations of physics (mechanics, electrokinetics, hydraulics, thermodynamics, chemical reactions, etc.).

The pioneering work by Hilding Elmqvist (Elmqvist 1978) led to the emergence of the Modelica community in the 1990s, and the DAE-based modeling language of the same name (Modelica Association 2023) has become a *de facto* standard, with its object-oriented nature enabling a component-based modeling style. Its combined use with the port-Hamiltonian paradigm (Rashad et al. 2020) results in a methodology that is instrumental to the scalable modeling of large systems, additionally ensuring that the model architecture preserves the system architecture, in stark contrast to ODE-based modeling (Benveniste, Caillaud, Elmqvist, et al. 2019; Benveniste, Caillaud, and Malandain 2022).

Consequently, DAE-based modeling requires that Modelica tools properly scale up to very large models. However, although Modelica enables the modeling of extremely large systems, its implementations (Dassault Systèmes 2002–2023; Fritzson et al. 2020) are often not capable of compiling and simulating such large models. Scaling has been and still is a subject for sustained effort by the Modelica community (Casella and Guironnet 2021), and although HPC issues belong to the landscape (Braun, Casella, and Bachmann 2017), a more specific issue is of uttermost importance for the Modelica language.

In the first steps of the compilation of a Modelica model, its hierarchical structure is flattened, thanks to a recursive syntactic inlining of the objects composing it.[1] The result of this flattening process is an unstructured DAE that can be exponentially larger than the source model. The *structural analyses* that are required for the generation of simulation code (namely, the *index reduction* of the DAE system, followed by a *block-triangular form* transformation of the reduced-index system) are then performed on this monolithic DAE model. As the compilation process does not fully take advantage of the hierarchical nature of the models it has to handle, the modeling capabilities offered by the Modelica language are undermined by performance issues on the structural analysis itself (Höger 2015; Höger 2019). Additionally, model flattening poses a challenge when attempting to extend DAE-based modeling to higher-order modeling or dynamically changing systems (Broman and Fritzson 2008; Broman 2010; Broman 2021).

In this paper, a new modular structural analysis algorithm is proposed that takes full advantage of the object tree structure of a DAE model. The bedrock of this method is a novel concept of *structural analysis-aware interface* for components. The essence of a component interface is to capture the necessary information about a Modelica class that needs to be exposed, in order to perform the structural

[1] See (Modelica Association 2023), Section 5.6 for a complete definition of the flattening process.

DOI
10.3384/ecp204255

analysis of a component comprizing instances of the former class, while hiding away useless information regarding the equations and all *protected* features it may contain.

In order to compute a component interface, one has to be able to perform the structural analysis of the possibly non-square DAE system that this component encapsulates, and to use the interfaces of the components it aggregates in this analysis. We base our algorithm on Pryce's Σ-method for index reduction (Pryce 2001), which essentially consists in the successive solving of two dual linear integer programs. The striking difference with Pryce's algorithm is that these problems are solved by parts, in a scalable manner.

Putting all of this together, it is then possible to perform a *modular structural analysis*, in which structural analysis is performed at the class level, and the results can then be instantiated for each component of the system model, knowing its context. Hence, structural information at the system level is derived from composing the result of component-level analysis. Modular structural analysis yields huge gains in terms of memory usage and computational costs, as the analysis of a single large-scale DAE is replaced with that of multiple smaller subsystems. Moreover, the analysis is performed at the class level, meaning that a single structural analysis is needed for all system components that are instances of the same class.

To the best of our knowledge, only (Höger 2015) addresses the specific issue of performing the structural analysis of a hierarchical model; Section 2.1 of this paper shows the approach proposed by Höger and explains why we regard it as an efficient but still partial solution. On the related subject of sorting equations, attention is paid in (Zimmermann, Fernández, and Kofman 2019) to methods that avoid unrolling loops and expanding arrays when sorting equations, an issue targeting the same overall objective as both (Höger 2015; Höger 2019) and the present work.

Section 2 introduces two simple examples of DAE systems, to be used for illustrative purposes, and explains the modular approach from (Höger 2015) on one of them. In Section 3, we provide background information on structural analysis, which includes the Σ-method used to perform index reduction.

In Section 4, we introduce Σ-systems as an abstraction of DAE systems suitable for structural analysis. Σ-systems include the linear programming problems associated with the Σ-method and the structural description of the BTF. They also provide a notion of composition that abstracts the composition of DAE systems. This formalization leads us to the main contribution of this paper, presented in Section 5, where we propose the notion of Σ-interface. A Σ-interface exposes the necessary information to assemble partial structural analyses into a system-level structural analysis for a DAE system. We discuss how Σ-interfaces can reduce the computational cost of structural analysis for large systems.

Finally, Section 6 introduces our prototype implementation of the resulting modular method, then presents numerical applications to both examples presented in Section 2.

2 Examples

In this section, we develop two illustrative examples: a slightly modified version of the chained circuit proposed by C. Höger in (Höger 2015), and a homemade chained mass-and-spring system. Both will be used to illustrate how our approach changes and improves the structural analysis process of models where several instances of a same class are connected, while also highlighting different features of our algorithm.

2.1 A chained circuit

2.1.1 The circuit model

This example is a minor modification (with a second inductance added) of the one developed in (Höger 2015). The following text is borrowed verbatim from this reference:

> The Modelica representation of the circuit *[...]* consists of a name (`Circuit`), declarations of parameters (`n`), unknowns (`u` and `i`) and sub-components (`c`). The physical behavior is defined directly by multiple equations, including the description of sub-circuit interconnection. Without knowledge about the internal structure of `SubCircuit` it is possible to use it inside the larger circuit, as long as it provides the corresponding interface (i.e. variables `u` and `i`). This composition of models is an easy and safe way to create more complex models out of simpler building blocks and is the very foundation of object-oriented modeling.

The `Circuit` and `SubCircuit` Modelica codes and schematics are given in Fig. 1 and 2 respectively.

2.1.2 The approach by C. Höger

While the flattening of the `Circuit` model is linear with `n`, the cost of the structural analysis of the resulting model is super-linear, thus preventing the classical approach from properly scaling up. In (Höger 2015), important contributions are proposed to cope with this problem. As far as we know, this paper is the first one pointing this issue very clearly. A faster method is proposed to perform the structural analysis of a hierarchical model, involving scoping and hiding.

This method is based on Pryce's Σ-method (Pryce 2001), which is explained in Section 3. The Σ-method involves solving a pair of dual Linear Programming problems (*primal* and *dual*), using a specific iterative algorithm for the dual. In Theorem 1 of (Höger 2015), the author provides a complexity argument to support the claim that the dual problem is not a bottleneck on its own.[2] Hence, the author focuses on the primal problem, for which a decomposition method is proposed.

[2] Our own numerical experiments confirm this observation.

```
model Circuit
  parameter Integer n = 10;
  SubCircuit[n] c;
  Real u,i;
equation
  c[1].i = i;
  u = sin(time);
  for j in 2:n loop
    c[j].i = c[j-1].i;
  end for;
  sum(c.u) = u;
end Circuit;
```

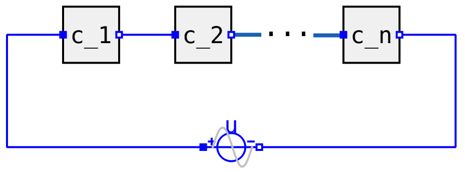


Figure 1. Modelica code and electrical schematics of the chained circuit.

```
class SubCircuit
  parameter Real R1, R2, L, C;
  Real i, u;
  protected
  Real u1,i1, u2,i2, uC,iC, uL1,iL1, uL2,iL2;
equation
  iC=C*der(uC);
  uL1=L1*der(iL1);
  uL2=L2*der(iL2);
  u1 = R1 * i1;
  u2 = R2 * i2;
  u2 = uL1;
  uC=u1+u2;
  i1=i2+iL1;
  u=u1+uL1+uL2;
  i=i1+iC;
  i=iL2;
end SubCircuit;
```

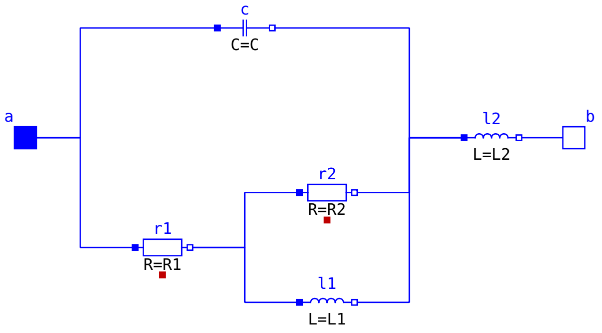


Figure 2. Modelica code of the **SubCircuit** class, and schematics of the corresponding electrical circuit.

2.2 A recursive mass-spring-damper system

Figure 3 shows the model, consisting of a chain of mass-spring-damper elements, defined with the 1D translational components of the Modelica Standard Library (MSL). Although recursive classes are not allowed in Modelica, we use a recursive definition of a chain of elements. This gives a binary-tree structure to the model, that is best suited for the modular structural analysis method presented in the sequel of the paper. Our prototype implementation of the method supports recursive classes, with conditional statements evaluated statically, at compile time.

2.3 Our contribution for these examples

Despite its important contribution, we believe that (Höger 2015) does not provide the ultimate answer. While a hierarchical algorithm for solving the primal problem is a great contribution in terms of computational costs, it still does not fully take advantage of object-oriented modeling. We would like instead to advance towards the separate compilation of components and systems, which consists in:

1. Proposing a notion of interface for model components, that is rich enough to subsequently perform system-level structural analysis; and
2. Proposing a modular structural analysis method, consisting of the needed algorithms performing hierarchical structural analysis based on interface information.

In doing so, both primal and dual problems, as well as the construction of a Block-Triangular Form (BTF) for the Jacobian, need to be addressed—this is also in contrast to the approach of (Höger 2015), where only the primal problem is considered.

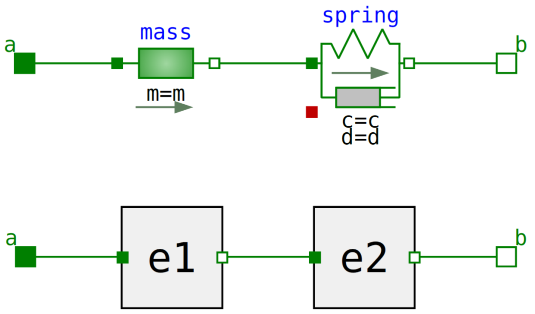
To simplify our presentation, we skip the discussion of the BTF; it will be developed in an extended version of this paper, that is currently in the works.

3 The Σ-method for DAE

The index reduction method proposed by J. Pryce (Pryce 2001), called the Σ-*method*, is an interesting alternative to the classical method originally proposed by C. Pantelides (Pantelides 1988). Its elegant and compact formulation as a pair of dual Linear Programming problems (LP) makes it particularly valuable for an extension to DAE components and architectures. The Σ-method can be summarized as follows:

The Σ-matrix

For $S = (F, X)$ a square DAE system involving equations $f{=}0$, where $f{\in}F$, and variables $x{\in}X$ and their derivatives, form the Σ-matrix $\Sigma = (\sigma_{f,x})_{(f,x)\in F\times X}$ of the DAE system, where $\sigma_{f,x}$ is the highest differentiation order of variable x in f, or $-\infty$ if x does not appear in f.



```
import ...;
model HarmonicString
   parameter Integer n = 1 "Number of elements";
   parameter Mass m = 1e-3 "Mass";
   parameter Distance l = 1e-2 "Length";
   parameter TranslationalSpringConstant c = 1;
   parameter TranslationalDampingConstant d = 1e-3;
   parameter Distance s0 = 0 "Initial position";
   Flange_a a;
   Flange_b b;
   static if n > 1 then
   protected
     parameter Integer n1 = n/2;
     parameter Integer n2 = n - n1;
     HarmonicString s1(n=n1, m=n1*m/n, ...);
     HarmonicString s2(n=n2, m=n2*m/n, ...);
   else
   protected
     Element e(m=m, l=l, c=c, d=d, s0=s0);
   end if;
equation
   ...
end HarmonicString;
```

Figure 3. A mass-spring-damper system; the element (top-left) is assembled from the 1-D translational mechanical components of the Modelica Standard Library. An assembly of two elements is shown at the bottom-left. A chain of mass-spring-damper elements of length n is defined by the recursive Modelica-like class shown on the right. Although Modelica does not allow for recursive classes, our software prototype allows recursion, provided conditional statements can be evaluated at compile-time.

Primal problem

The primal LP encodes the search for a maximum weight transverse of Σ, that will be described as $(f, x_f)_{f\in F}$ or, equivalently, $(f_x, x)_{x\in X}$ in what follows. The existence of a solution to the primal LP is a success check for the Σ-method.

Dual problem

The variables of the associated dual LP are variable offsets $(d_x)_{x\in X}$ and equation offsets $(c_f)_{f\in F}$, and we search for the minimal non-negative solution to this LP. (Pryce 2001) proves the uniqueness of this solution and proposes a relaxation method for finding it, given a solution to the primal problem.

The dual problem can be rewritten as the following constraint system, involving only the variable offsets $(d_x)_{x\in X}$:

$$\begin{array}{rrcl} \forall x \in X: & d_x & \geq & \sigma_{f_x,x} \\ \forall (f,x) \in E: & d_x - d_{x_f} & \geq & \sigma_{f,x} - \sigma_{f,x_f} \end{array} \tag{1}$$

and the equation offsets are then given by

$$c_f \quad = \quad d_{x_f} - \sigma_{f,x_f} \quad . \tag{2}$$

It is proved in (Pryce 2001) that the set of solutions of (1) does not depend on the particular choice of a solution to the primal problem.

Use of the offsets

Equation offset c_f indicates how many times equation f=0 needs to be differentiated to get the *index-reduced system*, whereas d_x indicates the maximum differentiation order of x in this index-reduced system. In addition, the solution of the primal problem is useful for computing a BTF for the Jacobian of this system.

C. Höger (Höger 2015) explains that the primal problem is the main bottleneck, hence it focuses on solving it efficiently, by decomposing it into smaller subproblems. In contrast, we extend and adapt the Σ-method for *open DAE systems*, which are DAE systems that can possess more variables than equations, and that can be composed with other open DAE systems by unifying their common variables.

4 Structural analysis of open DAE

The set of variables of an open DAE system can be decomposed as

$$X = X^s \uplus X^\ell$$

where X^s and X^ℓ are its sets of *shared* and *local* variables, respectively. For example, the **SubCircuit** of Fig. 2 possesses 2 shared variables and 10 local variables (declared as **protected** in Modelica).

4.1 Selectors

Since **SubCircuit** possesses 11 equations, we can regard it as a square system by assuming that one among the shared variables i, u is dependent and the other one is free (determined by the yet unspecified environment of this circuit):

$$X^s = X^{\mathsf{free}} \uplus Y$$

where $Y \subseteq X^s$ is a subset of the shared variables. The set of dependent variables of the system is then $X^{\mathsf{dep}} = Y \uplus X^\ell$.

If we compose open DAE system S with an environment S' (another open DAE system), then the two selectors Y

and Y' for S and S' must satisfy the condition

$$Y \cap Y' = \emptyset, \tag{3}$$

expressing that S' cannot claim determining variable x if the latter is already claimed by S, i.e., belongs to Y. We say that selectors Y and Y' are *compatible* if (3) holds.

For **`SubCircuit`**, two possible choices for selectors are

$$Y = \{\mathtt{i}\} \quad \text{or} \quad Y = \{\mathtt{u}\}. \tag{4}$$

4.2 Matching selectors in compositions

If $\{\mathtt{i}\}$ is selected, then **`u`** is free, i.e., it must be determined by the (yet unspecified) environment. Thus, we expect this environment to allow for a selector containing **`u`** but not **`i`**. This means that information (4) has to be exposed by S as part of its interface for structural analysis.

Then, by exposing (4), S sets structural constraints on the interface of any environment for it. We will show that

> information (4) is sufficient for characterizing the environments that are compatible with S. (5)

This information, however, is not sufficient to perform the structural analysis in a modular way. In the sequel, we will identify the information that is missing for this purpose.

5 Interfaces for the modular structural analysis of DAE systems

This section introduces the main contribution of this paper, which is the notion of component interface needed for the structural analysis of DAE systems. This notion is called Σ-interface as a reference to the Σ-method itself. Possible ways to perform the modular structural analysis of the **`Circuit`** example are also shown, as a way to illustrate the benefits of the modular approach.

5.1 The Σ-method for open DAE systems

We first show the result of the Σ-method, applied to the open DAE system **`SubCircuit`** (see Fig. 2), for the two possible choices of selectors given by (4).

5.1.1 Primal problem

The solutions of the primal problem for both selectors are given in Fig. 4. For each case, the free variable for the considered selector is written in blue. The chosen maximum weight transverse is indicated by highlighting in red the dependent variable associated to each equation, and we give in the third column the differentiation order of this variable in this equation. Two important observations will guide us in defining the primal Σ-interface:

1. The choice of the transverse, which fixes the assignment of variables to equations, *does not depend on the* (yet unspecified) *environment of* **`SubCircuit`**;
2. The total weight of the maximal transverse of **`SubCircuit`**, for a given selector, *is added* to the total weight of a maximal transverse of the environment (for a compatible selector), yielding the total weight of the overall transverse (that covers both the component and its environment).

5.1.2 Dual problem

The dual problem is the constraint system (1), whose dependent variables are the offsets $(d_x)_{x\in X}$. For $\mathtt{S} =_{\text{def}}$ **`SubCircuit`**, both selectors have to be considered:

- **`S`** has selector $\{\mathtt{i}\}$. A maximal weight transverse is shown in red on Fig. 4-left. With this transverse, the dual problem (1) is shown in Fig. 5-left.
- **`S`** has selector $\{\mathtt{u}\}$. A maximal weight transverse is shown in red on Fig. 4-right. With this transverse, the dual problem (1) is shown in Fig. 5-right.

Note that, while edges (f,x) are local (since all equations are local), variables can be shared between components; hence, dual problems only interact via the offsets of their shared variables (**`i`** and **`u`** for **`SubCircuit`**). This observation will guide us in defining the notion of dual Σ-interface.

5.2 Σ-interfaces

5.2.1 Primal Σ-interfaces

We say that a selector for a component is *consistent* if the primal problem has at least one solution for this selector.

Primal Σ-interface of a component Based on the discussion at the end of Section 5.1.1, it suffices to expose,

> for each consistent selector, the set of all pairs (selector, maximal transverse weight) (6)

at the interface of the considered component, for the primal problem. Thus, (6) defines the *primal Σ-interface* of an open DAE system. For the **`SubCircuit`** example, the primal Σ-interface is the set

$$\left\{ (Y = \{\mathtt{i}\}, \mathsf{J}_S^{Y=\{\mathtt{i}\}} = 3), (Y = \{\mathtt{u}\}, \mathsf{J}_S^{Y=\{\mathtt{u}\}} = 2) \right\} \tag{7}$$

Composing primal Σ-interfaces Let us consider two open DAE systems S_1 and S_2 whose composition $S = S_1 \cup S_2$ is another open DAE system. Then, solving the primal problem for S for a given selector Y is equivalent to:

1. Solving the primal problems for $S_i, i{=}1,2$, for each selector Y_i, thus producing optimal weights $\mathsf{J}_i^{Y_i}$, then
2. Selecting an *optimizing compatible pair* (Y_1, Y_2) of selectors, i.e., solve

$$\max\left\{ \mathsf{J}_1^{Y_1} + \mathsf{J}_2^{Y_2} \mid Y_1 \cap Y_2 = \emptyset \text{ and } Y_1 \cup Y_2 = Y \right\}. \tag{8}$$

```
class SubCircuit
  parameter Real R1, R2, L, C;
  Real i, u;
  protected
  Real u1,i1, u2,i2, uC,iC, uL1,iL1, uL2,iL2;
equation
  iC = C * der(uC);                           // 1
  uL1 = L1 * der(iL1);                        // 1
  uL2 = L2 * der(iL2);                        // 1
  u1 = R1 * i1;                               // 0
  u2 = R2 * i2;                               // 0
  u2 = uL1;                                   // 0
  uC = u1 + u2;                               // 0
  i1 = i2 + iL1;                              // 0
  u = u1 + uL1 + uL2;                         // 0
  iL2 = i1 + iC;                              // 0
  i = iL2;                                    // 0
end SubCircuit;
```

```
class SubCircuit
  parameter Real R1, R2, L, C;
  Real i, u;
  protected
  Real u1,i1, u2,i2, uC,iC, uL1,iL1, uL2,iL2;
equation
  iC = C * der(uC);                           // 1
  uL1 = L1 * der(iL1);                        // 1
  uL2 = L2 * der(iL2);                        // 0
  u1 = R1 * i1;                               // 0
  u2 = R2 * i2;                               // 0
  u2 = uL1;                                   // 0
  uC = u1 + u2;                               // 0
  i1 = i2 + iL1;                              // 0
  u = u1 + uL1 + uL2;                         // 0
  iL2 = i1 + iC;                              // 0
  i = iL2;                                    // 0
end SubCircuit;
```

Figure 4. The primal problem for **SubCircuit**, seen as an open DAE system, for two possible choices for the selector. Left: selector $Y = \{\mathtt{i}\}$ yields a maximal transverse (in red) of weight 3; right: selector $Y = \{\mathtt{u}\}$ yields a maximal transverse (in red) of weight 2. In each case, the contribution of each equation to the weight of the transverse is provided on the right of the equation, and the free variable (to be determined by the environment) is highlighted in blue.

$$
\begin{array}{rcr}
\forall \mathbf{x},\ d_{\mathbf{x}} & \geq & 0 \\
d_{\mathtt{uC}}, d_{\mathtt{iL1}}, d_{\mathtt{iL2}} & \geq & 1 \\
d_{\mathtt{iC}} - d_{\mathtt{uC}} & \geq & -1 \\
d_{\mathtt{uL1}} - d_{\mathtt{iL1}} & \geq & -1 \\
d_{\mathtt{uL2}} - d_{\mathtt{iL2}} & \geq & -1 \\
d_{\mathtt{u1}} - d_{\mathtt{i1}} & \geq & 0 \\
d_{\mathtt{i2}} - d_{\mathtt{u2}} & \geq & 0 \\
d_{\mathtt{u2}} - d_{\mathtt{uL1}} & \geq & 0 \\
d_{\mathtt{uC}} - d_{\mathtt{u1}} & \geq & 0 \\
d_{\mathtt{u2}} - d_{\mathtt{u1}} & \geq & 0 \\
d_{\mathtt{i1}} - d_{\mathtt{i2}} & \geq & 0 \\
d_{\mathtt{iL1}} - d_{\mathtt{i2}} & \geq & 0 \\
d_{\mathtt{u1}} - d_{\mathtt{uL2}} & \geq & 0 \\
d_{\mathtt{uL1}} - d_{\mathtt{uL2}} & \geq & 0 \\
d_{\mathtt{u}} - d_{\mathtt{uL2}} & \geq & 0 \\
d_{\mathtt{i1}} - d_{\mathtt{iC}} & \geq & 0 \\
d_{\mathtt{iL2}} - d_{\mathtt{iC}} & \geq & 0 \\
d_{\mathtt{iL2}} - d_{\mathtt{i}} & \geq & 0
\end{array}
\qquad
\begin{array}{rcr}
\forall \mathbf{x},\ d_{\mathbf{x}} & \geq & 0 \\
d_{\mathtt{uC}}, d_{\mathtt{iL1}} & \geq & 1 \\
d_{\mathtt{iC}} - d_{\mathtt{uC}} & \geq & -1 \\
d_{\mathtt{uL1}} - d_{\mathtt{iL1}} & \geq & -1 \\
d_{\mathtt{iL2}} - d_{\mathtt{uL2}} & \geq & +1 \\
d_{\mathtt{u1}} - d_{\mathtt{i1}} & \geq & 0 \\
d_{\mathtt{i2}} - d_{\mathtt{u2}} & \geq & 0 \\
d_{\mathtt{u2}} - d_{\mathtt{uL1}} & \geq & 0 \\
d_{\mathtt{uC}} - d_{\mathtt{u1}} & \geq & 0 \\
d_{\mathtt{u2}} - d_{\mathtt{u1}} & \geq & 0 \\
d_{\mathtt{i1}} - d_{\mathtt{i2}} & \geq & 0 \\
d_{\mathtt{iL1}} - d_{\mathtt{i2}} & \geq & 0 \\
d_{\mathtt{u1}} - d_{\mathtt{u}} & \geq & 0 \\
d_{\mathtt{uL1}} - d_{\mathtt{u}} & \geq & 0 \\
d_{\mathtt{uL2}} - d_{\mathtt{u}} & \geq & 0 \\
d_{\mathtt{i1}} - d_{\mathtt{iC}} & \geq & 0 \\
d_{\mathtt{iL2}} - d_{\mathtt{iC}} & \geq & 0 \\
d_{\mathtt{i}} - d_{\mathtt{iL2}} & \geq & 0
\end{array}
$$

Figure 5. The dual problem when **S** has selector $\{\mathtt{i}\}$ (left) and $\{\mathtt{u}\}$ (right).

The primal Σ-interface of S is then obtained by collecting the pairs (Y, J_S^Y) for every consistent selector of S. Given a consistent selector Y for S, we denote by

$$(\pi_1(Y), \pi_2(Y))$$

an optimizing selector pair. Remark that this pair may not be unique. However, the end result of the structural analysis (the offsets d_x and c_f) does not depend upon the choice of $(\pi_1(Y), \pi_2(Y))$.

5.2.2 Dual Σ-interfaces

Dual Σ-interface of a component Based on the discussion at the end of Section 5.1.2, we can now define the *dual* Σ-*interface* as collecting,

> for each consistent selector Y, the projection $\mathscr{D}^Y$ of system (1) on the subset of offsets $(d_x)_{x \in X^s}$. (9)

For the **SubCircuit** example, the dual Σ-interface is the following set (collecting two elements):

$$\left\{ \begin{array}{c} \left(Y = \{\mathtt{i}\},\ \mathscr{D}^{Y=\{\mathtt{i}\}} : \left\{ \begin{array}{l} d_{\mathtt{i}}, d_{\mathtt{u}} \geq 0 \\ d_{\mathtt{i}} \leq d_{\mathtt{u}}+1 \end{array} \right. \right) \\ , \\ \left(Y = \{\mathtt{u}\},\ \mathscr{D}^{Y=\{\mathtt{u}\}} : \left\{ \begin{array}{l} d_{\mathtt{i}}, d_{\mathtt{u}} \geq 0 \\ d_{\mathtt{u}} \leq d_{\mathtt{i}}+1 \end{array} \right. \right) \end{array} \right\} \qquad (10)$$

In (10), the two constraint systems are obtained by projecting, over the offsets $d_{\mathtt{i}}$ and $d_{\mathtt{u}}$, the constraint systems given in Fig. 5.

Composing dual Σ-interfaces Two open DAE systems $\mathbf{S}_1$ and $\mathbf{S}_2$ can only interact via their shared variables $X_1^s \cup X_2^s$. Hence, for their composition **S**:

> Given a consistent selector Y for **S**, the projection of the dual problem of **S** onto the offsets of $X^s \subseteq X_1^s \cup X_2^s$ is the composition of the projections of the dual problems $\mathscr{D}_i^{\pi_i(Y)}$ of $\mathbf{S}_i, i{=}1,2$ over the offsets of X^s. (11)

5.3 Using Σ-interfaces in DAE systems

In this section, we illustrate the use of Σ-interfaces for the modular structural analysis of DAE systems. To each open DAE system, we associate its Σ-*interface*, by fusing the primal and dual Σ-interfaces defined above: it is a set of triples whose elements are a consistent selector, the weight of a solution of the corresponding primal problem, and the projection of the dual problem on the offsets of the shared variables. We shall then use (8) and (11) to perform the structural analysis of a DAE system in a modular way.

5.3.1 The `Circuit`, seen as a chain

From (7) and (10), one gets the following Σ-interface for `SubCircuit`:

$$\left\{\begin{array}{l} \left(Y=\{\mathtt{i}\},\ \mathrm{J}_S^Y=3,\left[\begin{array}{l} d_\mathtt{i},d_\mathtt{u}\geq 0\\ d_\mathtt{i}\leq d_\mathtt{u}+1\end{array}\right]\right)\\ \left(Y=\{\mathtt{u}\},\ \mathrm{J}_S^Y=2,\left[\begin{array}{l} d_\mathtt{i},d_\mathtt{u}\geq 0\\ d_\mathtt{u}\leq d_\mathtt{i}+1\end{array}\right]\right)\end{array}\right\} \tag{12}$$

In what follows, successive instances of `SubCircuit` are chained, as shown in Fig. 1. The structural analysis will be performed by induction on the length n of the chain.

Call $\mathtt{S}_n$ the chain of length n; in particular, $\mathtt{S}_1 = \mathtt{SubCircuit}$, denoted by `SC` in what follows. Call `i`, $\mathtt{u}_n$ the shared variables of $\mathtt{S}_n$. Seen an a chain, $\mathtt{S}_n$ is obtained by composing $\mathtt{S}_{n-1}$ with `SC` and adding a Kirchhoff equation for voltages. We regard this last equation as a component with no local variables, denoted by `eq` below.

Note that we also have to rename `u` as `v` in `SC` in order to avoid name clashes, which we write `[u/v]`, and that variable hiding has to be used on the result of the composition, as variables $\mathtt{u}_{n-1}$ and `v` have to be made local in $\mathtt{S}_n$.

As a result, for all $n \geq 2$, $\mathtt{S}_n$ is defined as the following composition:

$$\mathtt{S}_n = \mathtt{hide}\ \mathtt{u}_{n-1},\mathtt{v}\ \mathtt{in}\left\{\begin{array}{l}\mathtt{S}_{n-1}\\ \mathtt{SC[u/v]}\\ \mathtt{u}_n = \mathtt{u}_{n-1}+\mathtt{v}\end{array}\right. \tag{13}$$

For every n, $\mathtt{S}_n$ has exactly one more variable than it has equations. Thus, there are two possible selectors for $\mathtt{S}_n$: $Y_{\mathtt{S}_n} = \{\mathtt{i}\}$ and $Y_{\mathtt{S}_n} = \{\mathtt{u}_n\}$.

Case $Y_{\mathtt{S}_n} = \{\mathtt{u}_n\}$ Then, `i` is free and there is only one triple of compatible selectors in the composition occurring in the right-hand side of (13):

$$\left(Y_{\mathtt{S}_{n-1}} = \{\mathtt{u}_{n-1}\}, Y_\mathtt{SC} = \{\mathtt{v}\}, Y_\mathtt{eq} = \{\mathtt{u}_n\}\right)\ .$$

An immediate induction argument shows that the optimal weight for $\mathtt{S}_n$ is $2(n-1)+0+2 = 2n$.

We prove by induction that the dual Σ-interface is

$$\left\{\begin{array}{l} d_\mathtt{i}, d_{\mathtt{u}_n}\geq 0\\ d_{\mathtt{u}_n}\leq d_\mathtt{i}+1\end{array}\right. \tag{14}$$

for every n. This holds for $n = 1$, as (14) then yields the dual Σ-interface of class `SubCircuit` for selector $\{\mathtt{u}\}$ (see (10)). Assuming that (14) holds for $n-1$, the dual Σ-interfaces compose as follows:

$$\mathtt{hide}\ \mathtt{u}_{n-1},\mathtt{v}\ \mathtt{in}\left\{\begin{array}{l} d_\mathtt{i}, d_{\mathtt{u}_{n-1}}, d_\mathtt{v}, d_{\mathtt{u}_n}\geq 0\\ d_{\mathtt{u}_{n-1}}\leq d_\mathtt{i}+1\\ d_\mathtt{v}\leq d_\mathtt{i}+1\\ d_{\mathtt{u}_n}\leq d_{\mathtt{u}_{n-1}}\\ d_{\mathtt{u}_n}\leq d_\mathtt{v}\end{array}\right.$$

which yields (14) when projected on $\mathtt{u}_n$, `i`.

Consequently, the variable offsets of a `SubCircuit` in a `Circuit` are independent of the number of chained instances, and the equation offset of an equation in the k-th component of the chain is equal to that of the same equation in the original class.

Case $Y_{\mathtt{S}_n} = \{\mathtt{i}\}$ Then, $\mathtt{u}_n$ is free and there are two triples of compatible selectors:

$$\left(Y_{\mathtt{S}_{n-1}} = \{\mathtt{u}_{n-1}\}, Y_\mathtt{SC} = \{\mathtt{i}\}, Y_\mathtt{eq} = \{\mathtt{v}\}\right) \tag{15}$$

$$\left(Y_{\mathtt{S}_{n-1}} = \{\mathtt{i}\}, Y_\mathtt{SC} = \{\mathtt{v}\}, Y_\mathtt{eq} = \{\mathtt{u}_{n-1}\}\right) \tag{16}$$

By adding the contributions of the components, in the order they are written above, in each case, we get:

- In case (15): $[2(n-1)]+0+3 = 2n+1$
- In case (16): $[2(n-1)+1]+0+2 = 2n+1$

Thus, each of the two triples (15) and (16) is optimizing, which brings an important question: *Does the dual Σ-interface depend on the choice of a tuple of selectors?*

If triple (15) is used: System (14) for $n-1$ provides us with the dual Σ-interface of $\mathtt{S}_{n-1}$ for selector $\mathtt{u}_{n-1}$. For $\mathtt{S}_n$, the dual Σ-interfaces then compose as follows:

$$\mathtt{hide}\ \mathtt{u}_{n-1},\mathtt{v}\ \mathtt{in}\left\{\begin{array}{l} d_\mathtt{i}, d_{\mathtt{u}_{n-1}}, d_\mathtt{v}, d_{\mathtt{u}_n}\geq 0\\ d_{\mathtt{u}_{n-1}}\leq d_\mathtt{i}+1\\ d_\mathtt{i}\leq d_\mathtt{v}+1\\ d_\mathtt{v}\leq d_{\mathtt{u}_{n-1}}\\ d_\mathtt{v}\leq d_{\mathtt{u}_n}\end{array}\right.$$

which yields

$$\left\{\begin{array}{l} d_\mathtt{i}, d_{\mathtt{u}_n}\geq 0\\ d_\mathtt{i}\leq d_{\mathtt{u}_n}+1\end{array}\right. \tag{17}$$

If triple (16) is used: We will prove by induction that (17) still holds. Assuming that (17) holds for $n-1$, the dual Σ-interfaces compose as follows:

$$\mathtt{hide}\ \mathtt{u}_{n-1},\mathtt{v}\ \mathtt{in}\left\{\begin{array}{l} d_\mathtt{i}, d_{\mathtt{u}_{n-1}}, d_\mathtt{v}, d_{\mathtt{u}_n}\geq 0\\ d_\mathtt{i}\leq d_{\mathtt{u}_{n-1}}+1\\ d_\mathtt{v}\leq d_\mathtt{i}+1\\ d_{\mathtt{u}_{n-1}}\leq d_\mathtt{v}\\ d_{\mathtt{u}_{n-1}}\leq d_{\mathtt{u}_n}\end{array}\right.$$

which yields again (17).

Recall that solutions to the Σ-method's dual problem are independent of the particular choice of a solution for the primal problem. This property generalizes to the composition of dual interfaces:

the dual Σ-interface of a composition does not depend on the choice of a particular optimizing tuple. (18)

5.3.2 The `Circuit`, seen as a tree

Instead of a chain, the architecture of the `Circuit` can be regarded as a binary tree:

$$\begin{tikzcd} & & \widehat{\mathtt{SC}} \arrow[dl] \arrow[dr] & \\ \mathtt{T}_m =_{\text{def}} & \mathtt{T}^-_{m-1} & & \mathtt{T}^+_{m-1} \end{tikzcd} \tag{19}$$

where:

- the length ℓ_m of $\mathtt{T}_m$ is defined by $\ell_m = 2\ell_{m-1}+1$, which is exponential in m;
- $\mathtt{T}_m$ has shared variables $\mathtt{i}_m$, $\mathtt{u}_m$;
- $\mathtt{T}^{\pm}_{m-1}$ are two copies of $\mathtt{T}_{m-1}$ with renamings $[\mathtt{i}^{\pm}_{m-1}/\mathtt{i}]$;
- $\hat{\mathtt{SC}}$ is made of an instance of $\mathtt{SC}$ and a component (similar to component $\mathtt{eq}$ above) made of the connection equation $\mathtt{u}_m = \mathtt{u}^-_{m-1}+\mathtt{u}+\mathtt{u}^+_{m-1}$.

We reuse the Σ-interface (12) of class **`SubCircuit`**, the optimal weights $2\ell_m$ and $2\ell_m{+}1$ for selectors $Y_{\mathtt{T}_m}{=}\{\mathtt{u}_m\}$ and $Y_{\mathtt{T}_m}{=}\{\mathtt{i}\}$ respectively, and we focus on the dual interface. It is computed by the recursion shown in Fig. 6.

$$Y_{\mathtt{T}_m}{=}\{\mathtt{u}_m\}: \quad \text{hide } \mathtt{u},\mathtt{u}^{\pm}_{m-1} \text{ in } \left\{\begin{array}{rcl} 0 & \leq & d_{\mathtt{i}}, d_{\mathtt{u}^{\pm}_{m-1}}, d_{\mathtt{u}_m} \\ d_{\mathtt{u}^-_{m-1}} & \leq & d_{\mathtt{i}}+1 \\ d_{\mathtt{u}^+_{m-1}} & \leq & d_{\mathtt{i}}+1 \\ d_{\mathtt{u}} & \leq & d_{\mathtt{i}}+1 \\ d_{\mathtt{u}_m} & \leq & d_{\mathtt{u}^{\pm}_{m-1}} \\ d_{\mathtt{u}_m} & \leq & d_{\mathtt{u}} \end{array}\right.$$

$$\text{which yields} \quad \left\{\begin{array}{rcl} 0 & \leq & d_{\mathtt{i}}, d_{\mathtt{u}_m} \\ d_{\mathtt{u}_m} & \leq & d_{\mathtt{i}}+1 \end{array}\right.$$

$$Y_{\mathtt{T}_m}{=}\{\mathtt{i}\}: \quad \text{hide } \mathtt{u},\mathtt{u}^{\pm}_{m-1} \text{ in } \left\{\begin{array}{rcl} 0 & \leq & d_{\mathtt{i}}, d_{\mathtt{u}^{\pm}_{m-1}}, d_{\mathtt{u}_m} \\ d_{\mathtt{u}^-_{m-1}} & \leq & d_{\mathtt{i}}+1 \\ d_{\mathtt{u}^+_{m-1}} & \leq & d_{\mathtt{i}}+1 \\ d_{\mathtt{i}} & \leq & d_{\mathtt{u}}+1 \\ d_{\mathtt{u}} & \leq & d_{\mathtt{u}^{\pm}_{m-1}} \\ d_{\mathtt{u}} & \leq & d_{\mathtt{u}_m} \end{array}\right.$$

$$\text{which yields} \quad \left\{\begin{array}{rcl} 0 & \leq & d_{\mathtt{i}}, d_{\mathtt{u}_m} \\ d_{\mathtt{i}} & \leq & d_{\mathtt{u}_m}+1 \end{array}\right.$$

Figure 6. The dual Σ-interface of tree shaped architecture (19).

5.3.3 Discussion

It is worth comparing the chain-based and tree-based approaches above. At first glance, since recursion arguments were used in both cases, the two approaches may seem equivalent in terms of computational costs. However, this impression shall not last once we detail how implementations should proceed.

- For the chain architecture of Section 5.3.1, each induction step consists in the computation of the Σ-interface of prefix $\mathtt{S}_k$ as a function of the Σ-interface of $\mathtt{S}_{k-1}$, for k increasing from 2 to n.
- For the tree-shaped architecture of Section 5.3.2, the induction step expresses the Σ-interface of the root of each subtree $\mathtt{T}_k$ as a function of the Σ-interface of each subtree $\mathtt{T}^{\pm}_{k-1}$. Remark that the Σ-interface of each subtree $\mathtt{T}_k$ is only computed once, as all leaf components are instances of the same class, and all subtrees of height k have, by construction, identical interfaces.

The number of steps is proportional to n in the first case, and $m \sim \log n$ in the second case. Since the computational complexity of each induction step is roughly the same, treating the **`Circuit`** model as a tree-shaped architecture can dramatically improve performance.

6 Implementation and experimental results

6.1 Implementation considerations

A significant difficulty in our modular structural analysis is the consideration of selector-dependent structural analyses. Both the primal and dual problems are functions of the selectors, which can be numerous for components with a large number of public variables. Although Modelica models are usually sparse, with components exposing only a few variables, a mere enumeration of selectors can result in an exponential growth of the handled data structures.

To deal with this issue, we advocate the approach proposed in (Benveniste, Caillaud, Malandain, and Thibault 2022; Caillaud, Malandain, and Thibault 2020) for multimode DAE systems. In these works, a *dual representation* is introduced, where equations are labeled with a predicate on mode variables, characterizing the modes under which they are active. Using this representation (instead of a direct one, that lists the active equations for each mode of the model) provides a compact representation of the structure of multimode systems. Reduced Ordered Binary Decision Diagrams, or ROBDD (Bryant 1986), provide not only an adapted data structure, but also efficient computation algorithms that can be used for performing the whole structural analysis in an “all-modes-at-once” fashion.

For modular structural analysis, a similar representation can be used, but with selectors instead of modes. The whole structural analysis chain can then be performed in an “all-selectors-at-once” fashion: (Benveniste, Caillaud, Malandain, and Thibault 2022) provides all building blocks for the computation of primal interfaces and their compositions, as presented in Section 5.2.1. The computation and composition of dual interfaces reduce to projecting parametrized constraint systems such as the one illustrated Fig. 5. Such systems are called *Difference Bound Matrices* (DBM) and come equipped with a rich calculus (Dill 1989; Miné 2001) with a polynomial computational complexity. Although there are excellent implementations of DBM, parametric DBM remain to be implemented, possibly using tuples of ROBDD for the representation of matrix elements.

6.2 Benchmarks and measured performance

Experimental results have been obtained on the modular structural analysis method, using two benchmarks: the

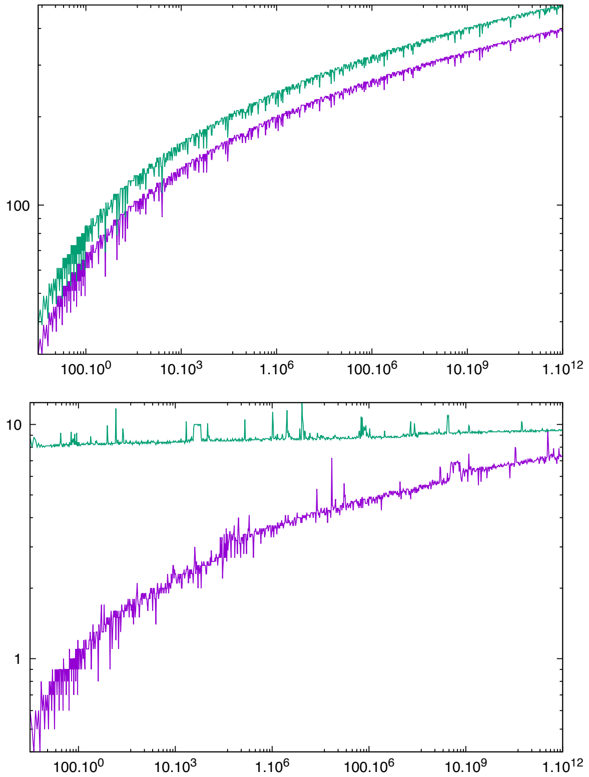


Figure 7. Number of interfaces computed (top plot) and computation time (bottom plot, in ms) for the mass-spring-damper (purple curves) and chain circuit (green curves) models, as functions of parameter n.

chain circuit (Section 2.1, Figure 2) and the mass-spring-damper model (Section 2.2, Figure 3). Models of increasing sizes have be analyzed, up to $n = 10^{12}$.

The experimental results presented below have been obtained with a prototype implementation of the method, based on an enumeration of the selectors. This has a limited impact on performance, for these particular models, since the selector combinatorics is limited to 2 cases for the chain circuit, and 6 cases for the mass-spring-damper model. The key feature of the prototype implementation is the use of a dynamic programming approach for the computation of the interfaces of model tree nodes. For this purpose, a memoization table (see (Cormen et al. 2022), pages 390–392) is used to store the computed interfaces. The software consists in about 10 kLOC of OCaml code and performance has been measured on a MacBook Pro with a 2.4GHz 8-core i-9 Intel processor with 16GB of RAM.

Figure 7 shows the computation times for both models as a function of parameter n. It clearly appears that the empirical time complexity of the method is a logarithmic function of n, like in (Höger 2015). Memory usage is very modest, with about 1MB to store the memoization table.

7 Conclusion and perspectives

In this paper, we present a modular index-reduction method for (possibly incomplete) DAE systems, based on John Pryce's Σ-method (Pryce 2001) and extending the seminal work of Christoph Höger on scalable algorithms for the compilation of Modelica (Höger 2015; Höger 2019). Our method is built upon three key contributions: (i) a concept of *selector* that allows to characterize, from a structural analysis point of view, the possible effects of unknown environments on an incomplete DAE system; (ii) a concept of *interface* for the primal and dual problems of the Σ-method, that encapsulates the minimal information regarding a subsystem that needs to be exposed to its environment in order to perform the structural analysis; and (iii) interface *composition* and *transformation* operators that allow to compute the interface of a system from the interfaces of its parts.

Our modular structural analysis method is well-suited to the Modelica language, since the interface of a class can be computed inductively from the interfaces of the objects contained in the class. Modelica models are often sparse, meaning that each component shares only a few variables with its environment. This guarantees that the interface of a component remains small, independently of the number of components, variables and equations it may contain.

We believe that this concept of interface, and the modular structural analysis method it yields, pave the way towards a genuine separate compilation of Modelica, that can scale up to extremely large models.

The benchmarks performed with our prototype implementation demonstrate that extremely large models, organized in a component tree with sufficient regularity, can be analyzed in a few milliseconds.

Future work shall focus on a more robust implementation of the method, based on our IsamDAE multimode DAE structural analysis software (Benveniste, Caillaud, Malandain, and Thibault 2022; Caillaud, Malandain, and Thibault 2020). We plan to use a functional, BDD-based, representation of interfaces, in order to curb the combinatorics of selectors that is expected when computing interfaces of Modelica classes with a large number of public variables. This functional representation is also a promising approach to the extension of the notion of interface to multimode models.

The structural analysis of Modelica models comprising *for loops* could be done by (i) computing a *tree decomposition* of the component graph obtained from the evaluation of the loops, followed by (ii) the modular structural analysis of the resulting component tree. Benchmarking this approach on the scalable test suite (Casella and Guironnet 2021) would be of great interest.

Several features of the Modelica language, such as *inner/outer* declarations, *expandable* connectors and *overconstrained* connections, may turn out to be difficult to deal with. In the Modelica Language Specification (Modelica Association 2023), the semantics of these features is expressed in terms of an elaboration phase, that is not modular. Devising a modular transformation of these features into the Modelica kernel language is a challenge that needs to be addressed before the modular structural analysis method can be applied to the full Modelica language.

There are also a few fundamental issues, related to the modular structural analysis, that ought to be investigated: (i) Is it possible to decide, by inductive reasoning, whether a parametric model is structurally nonsingular for every valuation of its parameters? (ii) Is it possible to perform the structural analysis of DAE systems that have a large treewidth, such as systems organized as a grid of dimension 2 or higher?

Answers to these questions are key to the design of scalable separate compilation methods for the Modelica language. One could envision the standardization of interfaces, possibly as an extension of the FMI standard for model exchange, so that precompiled Modelica libraries, *equipped with their* Σ*-interfaces,* could be built and reused.

References

Benveniste, Albert, Benoît Caillaud, Hilding Elmqvist, et al. (2019). "Multi-Mode DAE Models - Challenges, Theory and Implementation". In: *Computing and Software Science - State of the Art and Perspectives*. Vol. 10000. Lecture Notes in Computer Science. Springer, pp. 283–310. DOI: 10.1007/978-3-319-91908-9_16.

Benveniste, Albert, Benoît Caillaud, and Mathias Malandain (2022). "From Hybrid Automata to DAE-Based Modeling". In: *Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*. Vol. 13660. Lecture Notes in Computer Science. Springer, pp. 3–20. DOI: 10.1007/978-3-031-22337-2_1.

Benveniste, Albert, Benoît Caillaud, Mathias Malandain, and Joan Thibault (2022). "Algorithms for the Structural Analysis of Multimode Modelica Models". In: *Electronics* 11.17. ISSN: 2079-9292. DOI: 10.3390/electronics11172755. URL: https://www.mdpi.com/2079-9292/11/17/2755.

Braun, Willi, Francesco Casella, and Bernhard Bachmann (2017). "Solving large-scale Modelica models: new approaches and experimental results using OpenModelica". In: *Proceedings of the 12th International Modelica Conference*.

Broman, David (2010). "Meta-Languages and Semantics for Equation-Based Modeling and Simulation". PhD thesis. Linköping University, Sweden. URL: https://nbn-resolving.org/urn:nbn:se:liu:diva-58743.

Broman, David (2021). "Interactive Programmatic Modeling". In: *ACM Trans. Embed. Comput. Syst.* 20.4, 33:1–33:26. DOI: 10.1145/3431387.

Broman, David and Peter Fritzson (2008). "Higher-Order Acausal Models". In: *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools, EOOLT 2008, Paphos, Cyprus, July 8, 2008*. Vol. 29. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 59–69. DOI: 10.11128/sne.19.tn.09921.

Bryant, Randal E. (1986). "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* C-35.8, pp. 677–691. DOI: 10.1109/tc.1986.1676819.

Caillaud, Benoît, Mathias Malandain, and Joan Thibault (2020-04). "Implicit Structural Analysis of Multimode DAE Systems". In: *23rd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2020)*. Sydney, Australia. DOI: 10.1145/3365365.3382201.

Casella, Francesco and Adrien Guironnet (2021-09). "ScalableTestGrids - An Open-Source and Flexible Benchmark Suite to Assess Modelica Tool Performance on Large-Scale Power System Test Cases". In: *Proceedings of the 14th International Modelica Conference*. Linköping Electronic Conference Proceedings 181. Linköping, Sweden: Modelica Association and Linköping University Electronic Press, pp. 351–358. ISBN: 978-91-7929-027-6. DOI: 10.3384/ecp21181351.

Cormen, Thomas H et al. (2022). *Introduction to algorithms*. MIT press. ISBN: 9780262046305.

Dassault Systèmes (2002–2023). *Dymola official webpage*. Accessed: 2023-06-12. URL: https://www.3ds.com/products-services/catia/products/dymola/.

Dill, David L. (1989). "Timing Assumptions and Verification of Finite-State Concurrent Systems". In: *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989, Proceedings*. Vol. 407. Lecture Notes in Computer Science. Springer, pp. 197–212. DOI: 10.1007/3-540-52148-8_17.

Elmqvist, Hilding (1978). "A structured model language for large continuous systems". PhD thesis. Universiteit i Lund.

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Höger, Christoph (2015). "Faster Structural Analysis of Differential-Algebraic Equations by Graph Compression". In: *IFAC-PapersOnLine* 48.1. 8th Vienna International Conference on Mathematical Modelling, pp. 135–140. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2015.05.100. URL: https://www.sciencedirect.com/science/article/pii/S2405896315001019.

Höger, Christoph (2019). "Compiling Modelica: about the separate translation of models from Modelica to OCaml and its impact on variable-structure modeling". PhD thesis. TU Berlin. DOI: 0.14279/depositonce-8354.

MathWorks, Inc. (1994–2023). *Simulink official webpage*. Accessed: 2023-06-12. URL: https://www.mathworks.com/products/simulink.html.

Miné, Antoine (2001). "A New Numerical Abstract Domain Based on Difference-Bound Matrices". In: *Programs as Data Objects, Second Symposium, PADO 2001, Aarhus, Denmark, May 21-23, 2001, Proceedings*. Vol. 2053. Lecture Notes in Computer Science. Springer, pp. 155–172. DOI: 10.1007/3-540-44978-7_10.

Modelica Association (2023). *Modelica, A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.6*. Accessed: 2023-06-12. URL: https://modelica.org/documents/MLS.pdf.

Pantelides, Constantinos C. (1988). "The Consistent Initialization of Differential-Algebraic Systems". In: *SIAM Journal on Scientific and Statistical Computing* 9.2, pp. 213–231. DOI: 10.1137/0909014.

Pryce, John D. (2001). "A Simple Structural Analysis Method for DAEs". In: *BIT Numerical Mathematics* 41.2, pp. 364–394. DOI: 10.1023/a:1021998624799.

Rashad, Ramy et al. (2020). "Twenty years of distributed port-Hamiltonian systems: a literature review". In: *IMA J. Math. Control. Inf.* 37.4, pp. 1400–1422. DOI: 10.1093/imamci/dnaa018.

Zimmermann, Pablo, Joaquín Fernández, and Ernesto Kofman (2019). "Set-based graph methods for fast equation sorting in large DAE systems". In: *EOOLT '19: 9th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, Berlin, Germany, 5 November, 2019*. ACM, pp. 45–54. DOI: 10.1145/3365984.3365991.

Object-Oriented Formulation and Simulation of Models using Linear Implicit Equilibrium Dynamics

Dirk Zimmer

Institute of System Dynamics and Control,
German Aerospace Center (DLR),
dirk.zimmer@dlr.de

Abstract

New robust and yet powerful Modelica libraries have been developed such as the DLR ThermoFluid Stream library or the introduction of the Dialectic Mechanics library. These libraries apply a special modeling approach that uses linear implicit equilibrium dynamics. In this paper, we study the basic motivation of this approach, its benefits and drawbacks before we finally demonstrate how to get from models to applicable simulation code.

Keywords: Object-oriented modeling, Code Generation, Modeling principles

1 Introductory Example

Classic continuous laws of physics can be interpreted as communicating by means of waves. When you read these lines, your eye's photon receptors measure the electromagnetic waves communicating the corresponding visual information. When we speak, pneumatic waves communicate our audible voices. In a mechanics, pressure waves distribute the impulse in seemingly rigid bodies.

Even for things that we consider not to be alive, this analogy may be applied. A famous example is called: "communicating vessels" (in German: "kommunizierende Röhren") where various vessels filled with a homogenous liquid (let us use water) agree on a common surface level. This agreement is reached by hydraulic pressure waves going through the pipes, finally establish the hydrostatic equilibrium.

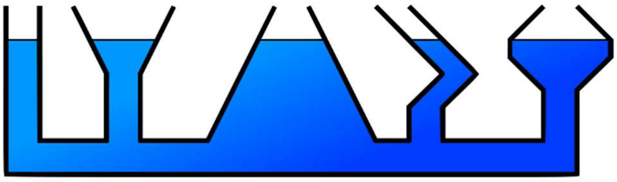


Figure 1: Depiction of communicating vessels displaying the hydro-static equilibrium. Public domain from Wikipedia.

Evidently, the macroscopic motion of a system can be interpreted as the emerging behavior of wave functions agreeing on an (quasi-) equilibrium state. One straightforward way to model and simulate classic physics is thus simply to implement the corresponding wave equations directly using a spatial discretization scheme.

1.1 Example: Communicating Vessels

We can implement this in Modelica for the example of the communicating vessels by using a staggered grid, where the inertia and compression of the fluid is alternately placed such as in the lower half of Figure 2.

When we model the wave equation in an object-oriented way, we need an interface to connect the distributed elements. Since a wave can be interpreted as the rotation within two dimensions as in Figure 3, it is a natural choice to choose two variables on the two corresponding orthogonal axes. Each of these variables thereby indicates a different form of energy storage.

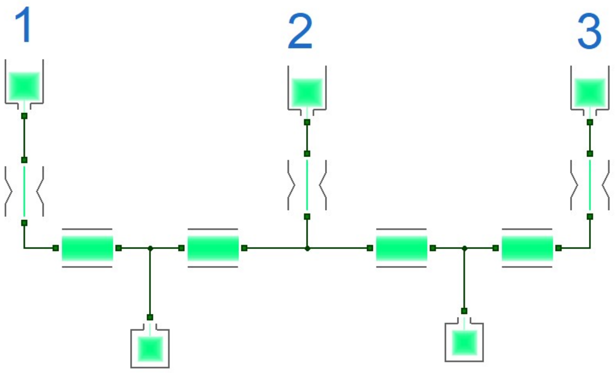


Figure 2: A model of 3 communicating vessels using a simple hydraulics library. Different from the depiction in Figure 1, the speed of flow is modulated by three narrow orifices at each tank. The one-dimensional hydraulic wave is modeled using a staggered grid for discretization. From top to bottom the layered icons represent the following elements: open-tank, non-linear pressure drop, fluid inertia, fluid compressibility.

In our hydraulic example, these two axes are: pressure p and volume flow $\dot{V}$. The pressure represents the potential energy of the compressed element whereas the volume flow rate represents the kinetic energy. We may call one of them a potential variable and the other one a flow variable. Since we work with an Eulerian framework, choosing the volume flow as flow is the natural choice.

Figure 4 shows the simulation result corresponding to our example. We can see that at the end of the simulation, we reach the desired equilibrium point. However, the computational efficiency is abysmal if this point is the only result we are interested in. The pressure waves have a very high frequency (artificially lowered here) and so the simulation had to take many, very small time-steps.

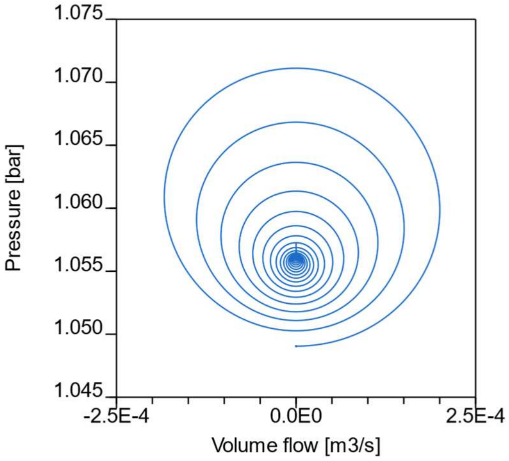


Figure 3: Trajectory of the pressure wave for the compressible volume in the two dimensions spanned by pressure and volume flow rate

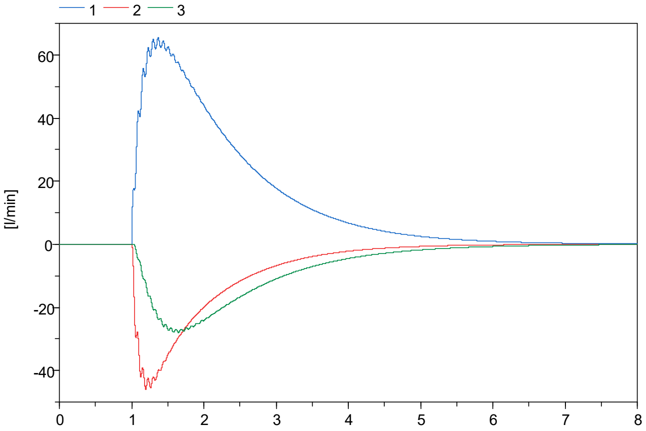


Figure 4: Step response of the communicating vessels after height of vessel 1 being increased at time = 1, showing the volume-flow through the valve openings. For the sake of illustration, the compressibility of water has been divided by 1000(!). Using the actual values, frequency would be much higher and ripples on volume flow barely visible.

Fortunately, we can avoid having to deal with high frequencies if we reduce the wave to its role as a conveyor of energy. The energy contained in our linear hydraulic wave is

$$E = \frac{1}{2}\rho_{lin}A^2\omega^2 c$$

where ρ_{lin} is the linear density, A the amplitude, ω the frequency and c the speed of sound. If the macroscopic phenomenon of interest is orders of magnitudes larger than the amplitude and slower than the frequency, we can presume the wave to be an instantaneous transmitter of energy that simply has to uphold the conservation of energy (given that also the speed of sound is quick enough over the required distance).

This transfer of power can be modelled by the same pair of variables that we have used to describe the wave equation. In our case the product of the pair represents a flow of energy:

$$\dot{E} = p\dot{V}$$

The power produced by a component with two such pairs is thus:

$$P = p_1\dot{V}_1 + p_2\dot{V}_2$$

If we are simply interested in the exchange of potential gravitational energy between the vessels over dissipative valve openings, we can choose to ignore the modeling of the hydraulic wave completely and simply connect the elements directly as in Figure 5.

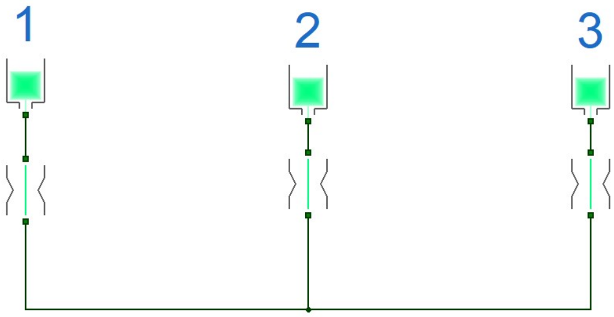


Figure 5: Modelling the communicating vessels by the sheer exchange of potential energy

By doing so, we have created an implicit non-linear algebraic equation system: the pressure level below the valves has to be found so that the corresponding volume flows resulting from the pressure drop are in balance. In our case, this system can be reliably solved, even for the case of the step response as depicted in Figure 6.

In general, we may have more than one solution or none at all. Also, the equation system is only available in implicit form. We thus replace the physical method to compute the transfer of energy with the solution for an algebraic system. Whether this works or not is simply down to luck in the general case. Here we were lucky.

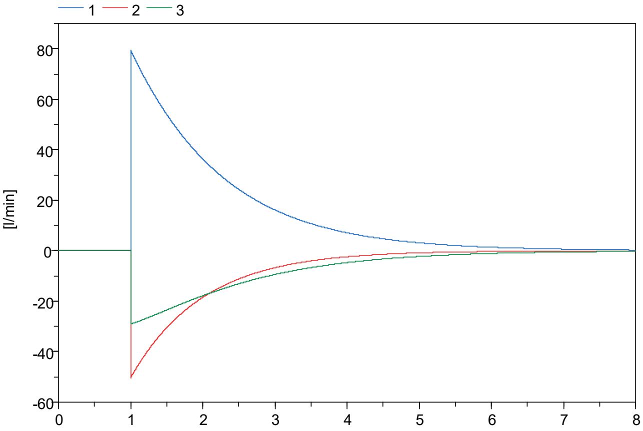


Figure 6: Corresponding step response of the direct model

1.2 Comparing the Modeling Approaches

We have created our first model using *explicit wave dynamics*.

We have created our second model using *non-linear implicit power dynamics*

Generating code for explicit wave dynamics is rather easy. All the equations are in explicit form and can be directly written as an ODE. Setting up the simulation code is thus principally rather trivial.

Simulating explicit wave dynamics is often computationally very expensive. Worse than the potentially high number of state variables is the that the frequency of the wave dynamics is often several orders of magnitude higher than the frequency of the macro-phenomenon of interest.

Simulating non-linear implicit power dynamics is much more efficient. Assuming an instantaneous transfer of power enables us to ignore the high frequency and phase shifts of the wave and we only have to deal with the low frequency of the macro-phenomenon. Also, we may use significantly fewer states.

Generating code for implicit power dynamics however, is far from trivial. Our system above had permutation index 1, because it requires the solution of a non-linear equation system. In mechanical systems, higher-index systems are common that require a reduction of the differential index for instance by applying Pantelides (Pantelides 1988). Because the simulation code is of high algorithmic complexity, we like to have a Modelica compiler creating it for us.

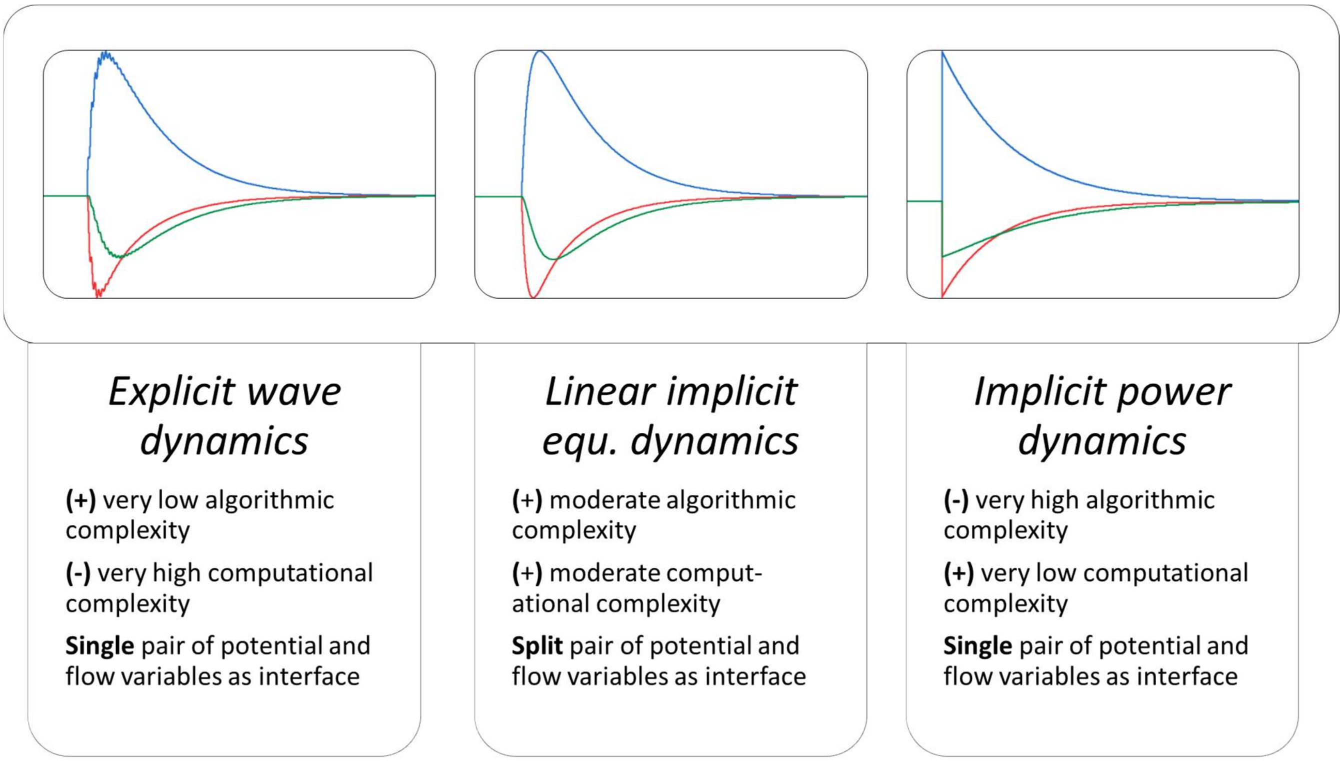
Choosing between these two can thus be seen as a trade-off between computational complexity (time needed for simulation) and algorithmic complexity (length of program for model generation) of the simulation code. This comparison is also highlighted in Figure 7 where wave dynamics is on the left and power dynamics is on the right.

For both forms of complexities, it is in practice nearly impossible to determine their theoretical limits. Since the computational complexity includes the ODE solver, we would need to determine the solver that reaches the desired precision within the shortest amount of time. The algorithmic complexity is to be interpreted in terms of algorithmic information theory (Chaitin 1987) and we would need to find the shortest possible program for code generation. In practice, it is however feasible to work with the numbers at hand: measuring the code-length of the compiler and measuring the time the simulation took. For our considerations, the general concepts suffice.

Explicit wave dynamics is computationally complex but can be low in algorithmic complexity. Non-linear implicit power dynamics is often of high algorithmic complexity but lower in computational complexity.

However, Figure 7 also shows that there seems to be an interesting middle ground in between these two classes of models, that might offer a very favorable trade-off. I denote this class: Linear Implicit Equilibrium Dynamics (LIED) (you can keep the pun). This class is typically not found in classic text-books and I presume that the primary reason for this is simply that we have to use an extended interface, which is not intuitive to come up with in the first place.

Figure 7: Illustration of different modeling approaches and their impact on algorithmic and computational complexity



2 The Idea behind Linear Implicit Equilibrium Dynamics

Figure 9 illustrates the desired result. To a step change we react neither with a high-frequency wave function nor with a discrete jump but by approaching the desired equilibrium with replacement dynamics. These dynamics shall reach the same steady-state behavior than the original wave dynamics and exhibit only a limited deviation for slow-mode behavior. Any deviation shall be of dissipative nature in case energy conservation cannot be upheld. There is one additional catch though: we shall limit our equations which are in implicit form to constitute a purely linear system.

The motivation for restricting ourselves to linearity for the implicit part is, to enable a robust solution of the system at all time, something that cannot be guaranteed for non-linear systems in general.

To put these statements in formal terms: if a system is described by differential algebraic equations (DAEs) in the following implicit form:

$$\mathbf{0} = F(\mathbf{x}_P, \dot{\mathbf{x}}_P, \mathbf{u}, t)$$

where $\boldsymbol{x}_P$ is the vector of potential states, $\dot{\boldsymbol{x}}_P$ represents all time derivatives, $\boldsymbol{u}$ the input vector and t time.

We aim to transform this system into the following form with an implicit linear part and an explicit non-linear part:

$$\mathbf{L}\dot{\mathbf{x}}_L = g(\mathbf{x}_L, \mathbf{x}_N, \mathbf{u}, t)$$

$$\dot{\mathbf{x}}_N = f(\mathbf{x}_L, \mathbf{x}_N, \mathbf{u}, t)$$

where $\mathbf{x}_L$ and $\mathbf{x}_N$ are both disjoint sub-vectors of $\mathbf{x}_P$. $\mathbf{L}$ is a linear matrix and f and g are non-linear functions. The original DAE system F is defined as a LIED system if and only if the functions f and g can be constructed just by ordering the corresponding equations of F.

Techniques for symbolical reduction of the differential index (Leimkuhler 1985) or the permutation index (Campbell 1995) may hence only be applied to derive the matrix $\mathbf{L}$. Hence, all non-linearities have to be brought into an explicit form and placed in either f or g. Equations in implicit form (including constraints between potential states) have to be linear and to be placed in $\mathbf{L}$.

How can we construct such DAEs for classic physical systems? And how to do this in an object-oriented form? The basic idea is simple: we find a part in the transient dynamics that is suitable for linear approximation and that completely vanishes at steady-state. A suitable candidate is often the dynamics of kinetic energy since it has a linear characteristic for a wide range of systems.

To enable this extraction, we have to split our interface, especially suited are variables that contain the flow of impulse (force, pressure, etc.) because here we can apply the superposition principle. Otherwise it may be very hard to separate the linear part in implicit from the non-linear part in explicit form.

All of the above is much easier said than done. I have spent several months figuring it out for thermo-fluid domain and later for the mechanical domain. Refinement took years for thermo-fluids and is still in the process for mechanics. The good news is: once we have identified a suitable interface, the remaining part of implementation is straightforward, often even easy.

2.1 LIED for Thermofluid Systems

Here is the full interface for thermo-fluid streams:

- r: inertial pressure (potential)
- $\dot{m}$: mass-flow rate (flow)
- Θ: Vector repr. state of medium (signal)
  - $\hat{p}$: steady-mass flow pressure
  - $\hat{h}$: steady-mass flow enthalpy
  - X: mass fractions

For the thermo-fluid streams, we have to split the potential variable into two parts: The steady-state pressure $\hat{p}$ and the inertial pressure r. The dynamics for the inertial pressure can be described by an implicit linear system using the law of inertance using the fluids inertia L:

$$r = L\frac{d\dot{m}}{dt}$$

For the steady-state with a constant mass flow rate, r will thus go to zero. To enable the approximation during transients, the impact of r on the thermodynamic state has to be neglected and hence Θ is composed using $\hat{p}$.

When the interface is used correctly, the whole thermofluid system will be a LIED system. $\mathbf{x}_L$ will form a vector that describes all mass-flow rates of the system in non-redundant manner. Typically, the dummy derivatives method (Mattsson 1993. Pantelides 1988) needs to be applied to construct the Matrix $\mathbf{L}$. Its coefficients are then formed by linear combinations of the inertances. $\mathbf{x}_N$ will contain all other states (such as specific enthalpy, etc.). Using these states the functions f and g can be computed in a downstream manner. More details on this interface and the implementation of a full library can be found in (Zimmer 2020, 2022). Models using this interface are especially suitable for the simulation of complex thermal architectures with bypasses and switches even under hard real-time constraints.

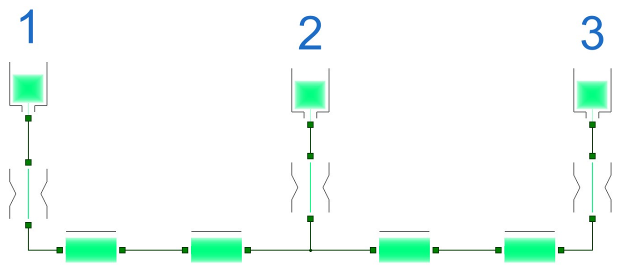


Figure 8: For this particular model of the communicating vessels, the LIED approach has an equivalent counterpart using conventional connectors. Modeling the inertia but leaving out the compressibility does the trick here.

In the particular case of our example with the communicating vessels, the LIED approach is equivalent to using only inertias for the fluid but disregarding the compressibility. Figure 8 shows the equivalent model diagram and Figure 9 depicts the corresponding simulation results.

This simple equivalence does however only work in this example because we treat the water as having constant density and also neglect any influence of temperature. Hence in this example we can mimic the LIED approach using the basic connectors. Using more realistic media models, the ThermoFluid Stream approach works more subtly and the split interface is needed.

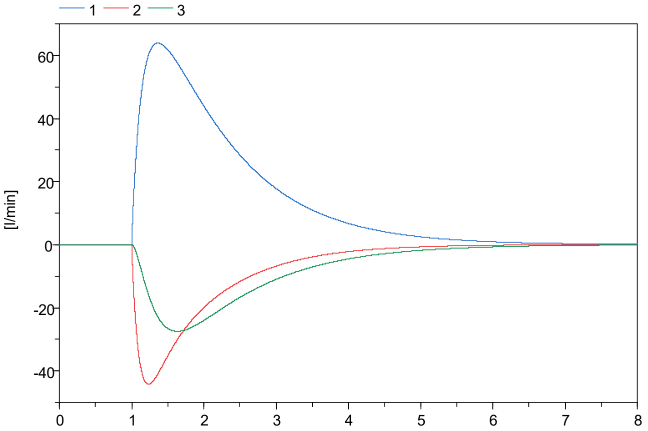


Figure 9: Modelling the communicating vessels by the sheer exchange of potential energy

2.2 LIED for Mechanical Systems

For mechanical systems, the interface is defined as follows:

- s: position (potential)
- f_{el}: elastic force (flow)
- v: velocity (potential)
- f_{ki}: kinetic force (flow)

We have thus two pairs of effort and flow not one. The derivative of the position s is thereby defined as v_{el}. The velocity v is also denoted as v_{ki}. The difference $\Delta v = v_{el} - v_{ki}$ should ideally be zero at all times. To enable a linear implicit approximation, we tolerate non-zero values for Δv at fast transients but establish a first order dynamics that ensures zero is approached for slow dynamics with the dialectic time constant T_D:

$$\frac{d\Delta v}{dt} T_D = -\Delta v$$

Because this interface separates the regimes of elastics and kinetics, I have denoted it as dialectic mechanics. First implementations and analysis are presented in (Zimmer 2023) and (Oldemeyer 2023). Models using this interface are especially suitable for the simulation of contacts and limited joints also under hard-real time constraints.

Dialectic mechanics are also LIED systems: the vector $\mathbf{x}_L$ will contain all the (generalized) positions in a non-redundant form so that all degrees of freedom are described. $\mathbf{x}_N$ then typically consists in the corresponding kinetic velocities. f and g can then be computed from the mechanical root of the system to the branches. Kinematic loops are explicitly closed using elastic elements with high stiff springs which is the preferred way in dialectic mechanics since high frequencies can be suppressed.

The details of the domain specific implementation shall not be the topic of this paper. But evidently this class of models is very useful and hence we shall further investigate its implications for the generation of simulation code.

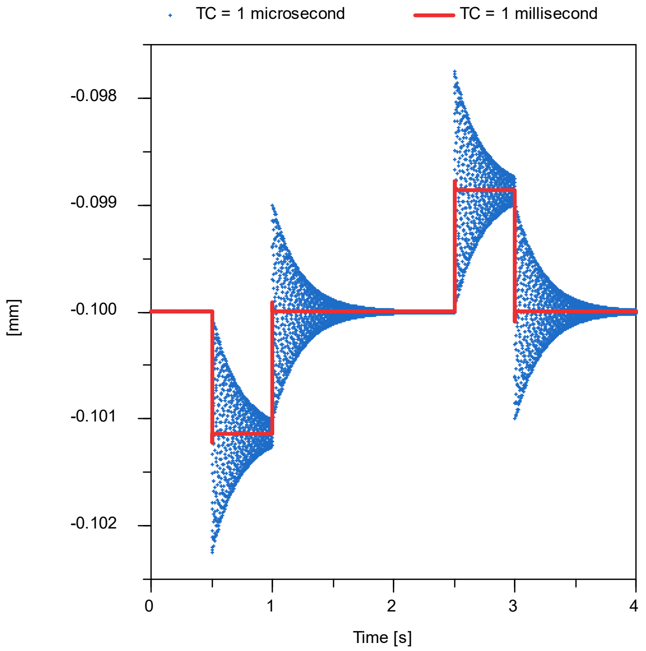


Figure 10: Penetration depth into the left claw represented by an elasto-gap, for the choice of two different time constants. Both agree on the time-averaged solution.

Just for the sake of quick illustration: Figure 10 from (Zimmer 2023) is repeated here again that shows the dynamics of a lightweight object moved in clamp modeled by a very stiff spring. The figure simply illustrates how the oscillatory dynamics is replaced with a replacement dynamics leading to the same (quasi) steady-state solution.

3 How to Create Simulation Code for LIED Systems?

The original intention of the LIED approach was simply to ensure that no non-linear system is created that spans across the components and hence a robust solution of the model evaluation could be taken for granted, given robust component models. When we started with it, we expected it to be the only notable change from other Modelica models and that all other features of a Modelica compiler (state selection, differential index-reduction, tearing of linear equation systems, etc.) would basically remain untouched.

However, over time, we realized that LIED systems are much simpler to transfer to simulation code than general DAEs resulting from non-linear implicit power dynamics. Let us go through the observed simplifications one-by one:

- Because we avoid the creation of non-linear equation systems, we do not need a non-linear equation system solver anymore.
- For the same reason, constraint equations among potential states cannot be non-linear and hence no dynamic state selection is needed (Mattsson 2000).
- Even stronger: we can select the states on component level. This is less obvious but ultimately the connection rules that enforce the linearity of the system also enforce this rule.
- Because we can select the states on component level, this means that the dummy-derivative method can be applied also on component level before system composition.
- Since the goal of the linear equation system is to have a synchronized replacement dynamic towards the equilibrium, we know suitable tearing variables for this system. These will be the linear state derivatives: $\dot{\mathbf{x}}_L$ or at least a subset of it.
- The residual for a tearing variable can be attributed to the same component as the tearing variable.

The items above represent observations resulting from modeling many components and system examples using the LIED approach. However, these observations have profound implications: For each component we know:

- the set of pairs of state-variables and their derivatives it adds to the system.
- the set of pairs of tearing and residual variables it adds to the system.

If this is the case, we can basically causalize everything already on the component level. In concrete terms, this means for each component:

- we stipulate the states
- we stipulate the tearing variables of the linear system and the corresponding residuals
- we perform the dummy derivative method on those equations where necessary.
- we define the causality of the interface variables
- we causalize all equations into assignments in a particular order
- we group the list of assignments depending on their dependence of the inputs.

Practical experience so far indicates that performing index reduction to construct the matrix **L** can be performed in a very methodical and deterministic manner. It is thus far easier to generate simulation code for the LIED modeling approach than it would be for general higher-index DAEs. Neither there is a need for global flattening anymore nor are elaborate heuristics needed for the selection of state or tearing variables. Indeed, the generation of simulation code is so easy that a direct implementation in C++ becomes feasible. The following code excerpts illustrate the implementation for a ThermoFluidStream Library (using idealized water) in C++.

First, we have to define the interface. This is naturally more tedious than in Modelica because there is no direct support in the C++ language. Yet, it is feasible and after all, interfaces only need to be defined once:

Listing 1. ThermoFluid Interface in C++

```
class ThermodynamicStateOut: public Signal{
  public:
    double p;
    double h;
    […]
};

class ThermodynamicStateIn:
  public ThermodynamicStateOut
{
  public:
    void connect(ThermodynamicStateOut* o);
    […]
};

class MassFlowOut : public Signal{
  public:
    double flow;
    double flow_der;
    […]
};

class MassFlowIn : public MassFlowOut{
  public:
     void connect(MassFlowOut* o);
    […]
};

class InertialPressureOut : public Signal{
  public:
    double r;
    […]
};

class InertialPressureIn :
  public InertialPressureOut
{
  public:
    void connect(InertialPressureOut* o);
    […]
};

class ThermalPlugOut : public Signal{
   public:
     ThermodynamicStateOut state{};
     MassFlowOut m{};
     InertialPressureIn inertial{};
     […]
};
```

```
class ThermalPlugIn : public Signal{
  public:
    ThermodynamicStateIn state{};
    MassFlowIn m{};
    InertialPressureOut inertial{};
    void connect(ThermalPlugOut* o);
    […]
};

class Connection {
  public:
    Connection(ThermalPlugOut* o,
               ThermalPlugIn* i) {
                i->connect(o);
     };
};
typedef std::vector<Connection>  Connections;
```

To best understand the interface, let us look at the classes `ThermalPlugOut` for a nominal outlet flow and at `ThermalPlugIn` for a nominal inlet flow first. These contain the same 3 components as the corresponding Modelica connector of the DLR ThermoFluid Stream library.

There are two notable differences however. In Modelica, inertial pressure and mass flow were not causalized signals as in the C++ implementation. Also the mass-flow signal in the C++ library consists of the mass-flow rate and its derivative. In Modelica, this is not necessary since symbolic differentiation can be applied by the Modelica compiler. Using this interface, we can now implement a component such as the pressure drop:

Listing 2. Implementation of a pressure drop component

```
class PressureDrop : public Component{
  public:
    ThermalPlugIn inlet;
    ThermalPlugOut outlet;
    PressureDrop(double v_ref,double dp_ref)
    void evalState();
    void evalFlow();
    void evalInertial();
    double v_ref;
    double dp_ref;

    virtual void metainfo(Meta& meta)
      override;
    […]
};
```

First, we declare our interface for outlet and inlet. Then we have to implement three blocks represented by methods. The first is `evalState` and computes the thermodynamic state downstream:

Listing 3. Calculation of the pressure drop by the corresponding method

```
void PressureDrop::evalState() {
   const double v =
     inlet.m.flow / density(inlet.state);
   const double v_norm = v/v_ref;
   const double dp = 0.5*dp_ref*
     (v_norm + v_norm*v_norm);
   outlet.state.h = inlet.state.h;
   outlet.state.p = inlet.state.p - dp;
};
```

The second method is `evalFlow` to ensure what flows in is what flows out. However, this constraint is restated for the derivative. This is because the dummy derivative method is applied on the component level.

Listing 4. Trivial implementation of `evalFlow`

```
void PressureDrop::evalFlow() {
  outlet.m = inlet.m;
}
```

The third one is `evalInertia` that implements the law for the inertance as in the ThermoFluid Stream Library.

Listing 5. Calculation of the inertial pressure

```
void PressureDrop::evalInertial() {
   inlet.inertial.r = outlet.inertial.r
                    + L*inlet.m.flow_der;
}
```

Meta-information can be collected by a dedicated virtual method to register state and tearing variables as well as to track the signal dependence of the computing blocks.

Listing 6. The meta information of the component is described in a virtual method.

```
void PressureDrop::metainfo(Meta& meta)
{
  meta.regComp (&inlet, “inlet”);
  meta.regComp (&inlet, “outlet”);
  meta.addBlock(this,
     LambdaFuncCalling(this->evalState()),
     Signals{&inlet.state,&inlet.m},
     Signals{&outlet.state});
  meta.addBlock(this,
    LambdaFuncCalling(this->evalFlow()),
    Signals{&inlet.m},
    Signals{&outlet.m});
  meta.addBlock(this,
    LambdaFuncCalling(this->evalInertial),
    Signals{&outlet.inertial,&inlet.m},
    Signals{&inlet.inertial});
}
```

For the pressure drop the signal dependencies of the methods have to be registered as vital structural information. Because of the horribly bad support of method function pointers in C++, the implementation requires the use of a lambda function which is done here in pseudo-code for the sake of readability.

In similar manner the other components of our introductory example can be implemented. Each of these components declares its interfaces, defines and implements methods representing the computational blocks and then registers these blocks as well as states, etc by overriding the virtual `metainfo` method. It is not as convenient as Modelica but also not overburdening.

Finally, we can compose the introductory example:

Listing 7. Total system composition

```
class ComVessels : public Component {
public:
  OutTank t1{};
  InTank t2{};
  InTank t3{};
  Splitter s{};
  PressureDrop p1{};
  PressureDrop p2{};
  PressureDrop p3{};

  Connections con {
    Connection{&t1.outlet, &p1.inlet},
    Connection{&p1.outlet, &s.inlet},
    Connection{&s.outlet1, &p2.inlet},
    Connection{&p2.outlet, &t2.inlet},
    Connection{&s.outlet2, &p3.inlet},
    Connection{&p3.inlet, &t3.inlet},
  };

  virtual void metainfo(Meta& meta) override{
    meta.regComp(&t1, "t1: first vessel");
    meta.regComp(&t2, "t2: second vessel");
    meta.regComp(&s,  "s: flow split");
    meta.regComp(&t3, "t3: third vessel");
    meta.regComp(&p1, "p1: first valve");
    meta.regComp(&p2, "p2: second valve");
    meta.regComp(&p3, "p3: third valve");

  };
};
```

Regarding that C++ is a statically compiled imperative general-purpose language, the end result is astonishingly close to what we are used to from Modelica.

When an instance of the class is coupled to a simulator, a crawler for meta information collects all blocks recursively as well as the structural information regarding the signals, the states and the tearing variables for the linear equation system. Then the blocks are put into right order for complete or partial model evaluation. The full model evaluation is thus simply a list of method calls that are called in their respective order.

This also means that all component code is statically compiled but the system composition is performed at run-time. Advanced tasks such as variable structure systems would thus be comparably easy to achieve.

This is also the purpose of this code demonstration. It is not suggesting that we should use C++ instead of Modelica but to highlight that for a certain class of models the object-oriented Modelica code could be translated to object-oriented imperative code that can be statically compiled even before system composition.

This would avoid the flattening of all equations before code generation and help to overcome many limitations of current Modelica compilers and you can of course choose a different target than C++.

4 Conclusions

That a more restrictive class of modeling enables a simpler compilation scheme is not surprising. The same can be said about the many conventional signal-based modeling schemes or simple modeling schemes as Forrester's System Dynamics (Junglas 2016). Typically, the disadvantage is that the easier generation of simulation code has to be paid by an inferior modeling approach and indeed modeling complex mechanics or thermo-fluid streams is painful when using signal-based approaches (nevertheless this pain has been taken in industrial practice all too often).

The remarkable thing about the LIED approach is that you have a simple scheme for code generation but you can conveniently model both mechanics and thermo-fluid streams in a very robust manner. The corresponding Modelica Libraries prove this (Zimmer 2022, Zimmer 2023). Both application domains are known to be rather difficult but LIED can even be applied to the challenging parts of these fields such as handling stiff contact mechanics or complex by-passes in thermal architectures. It is yet unclear for what other domains LIED is an attractive choice.

Figure 11 attempts to qualitatively depict the trade-off between computational complexity and algorithmic complexity. LIED forms a very exposed point on a hypothetical Pareto front. This means that for a large number of applications it is a very attractive choice.

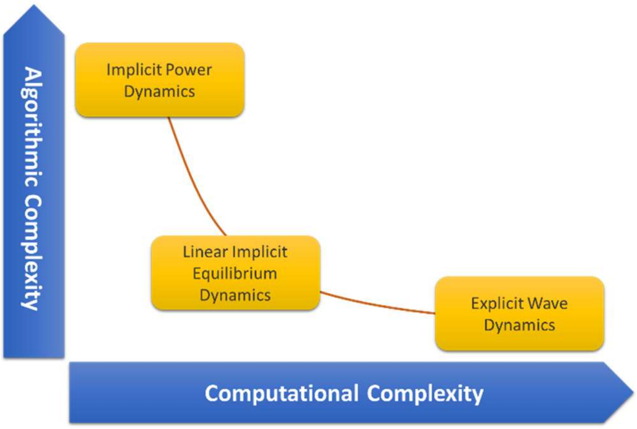


Figure 11: Hypothetical Pareto front weighing computational complexity against algorithmic complexity for code generation. LIED systems form an attractive compromise. The ultimate choice of the modeling approach depends however on the concrete application.

What does this mean for the Modelica community? We should recognize that fully within our standard, this particular class of LIED models has been hidden. Due to their non-conventional interfaces (that appear in no textbook), this class has been overlooked for more than 20 years. It is a robust class of models that scales for complex systems and also it is suitable for hard-real time simulation since everything non-linear is explicit and there are effective methods to manipulate fast eigendynamics.

I think it is justified to give this class extra support, by enabling the following features:

- The modeler shall be enabled to mark components that are compatible to LIED, and provide additional meta-information to this end. The Modelica compiler can then check whether this is true.
- The Modelica compiler can then enable the component-wise compilation of such components, at least for explicit ODE solvers and for implicit solvers with the numerical computation of the Jacobian.

The primary motivation of this paper is to raise awareness on this class of models and the possibilities it enables for the generation of simulation code. The provided code examples are not necessarily the best approach and can be improved on. Furthermore, this modeling approach is still new and open for further investigation.

Many statements in this paper require further validation also the topic is not very tangible. Hence, I want to encourage the reader to play with the open-source library ThermoFluid Stream and study dialectic mechanics. The practical way is the best way to develop an understanding for this modeling style.

As a final remark, I shall say that I am very grateful for Modelica and its tool-set. I am not sure whether I would have ever found this particular class without it. Quick experimentation within Modelica was certainly extremely helpful.

References

Campbell, S.L., C. William (1995), The Index of General Nonlinear DAEs. In: Numerische Mathematik, Vol. 72, pp. 173–196

Chaitin, G.J. (1987). Algorithmic Information Theory. Cambridge University Press. ISBN 9780521343060

Junglas, P. (2016), Causality of System Dynamics Diagrams, SNE Simulation Notes Europe 26/3, 147-154

Leimkuhler, B., C.W. Gear, G.K. Gupta (1985), Automatic integration of Euler-Lagrange equations with constraints. In: J. Comp. Appl. Math., Vol. 12&13, pp. 77–90.

Mattsson, S.E., Gustaf Söderlind (1993). “Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives” In: SIAM Journal on Scientific Computing 1993 14:3, 677-692

Sven Erik Mattsson, H. Olsson, H. Elmqvist (2000) “Dynamic Selection of States in Dymola”. Proceedings of Modelica Workshop 2000.

Oldemeyer, C., D. Zimmer (2023). “Dialectic Mechanics: Extension for Hard Real-time Simulation”. Proceedings of the 15th International Modelica Conference, Aachen.

Pantelides, C. (1988), The consistent initialization of differential-algebraic systems, SIAM J. Sci. Statist. Comput., 9 (1988), 213–231

Zimmer, D. (2020), Robust Object-Oriented Formulation of Directed Thermofluid Stream Networks . Mathematical and Computer Modelling of Dynamic Systems, Vol 26, Issue 3.

Zimmer, D. N. Weber, M. Meißner (2022) The DLR ThermoFluid Stream Library. MDPI Electronics - Special Issue.

Zimmer, D. C. Oldemeyer (2023). “Introducing Dialectic Mechanics”. Proceedings of the 15th International Modelica Conference, Aachen.

Accelerating the simulation of equation-based models by replacing non-linear algebraic loops with error-controlled machine learning surrogates

Andreas Heuermann [1] Philip Hannebohm [1] Matthias Schäfer [2] Bernhard Bachmann [1]

[1] Institute for Data Science Solutions, Bielefeld University of Applied Sciences and Arts, Germany,
{first.last}@hsbi.de
[2] LTX Simulation GmbH, Germany, Matthias.Schaefer@ltx.de

Abstract

When simulating a Modelica model, non-linear algebraic loops may be present, which involves solving multiple equations simultaneously. The classical Newton-Raphson method is commonly employed for solving a non-linear equation system (NLS). However, the computational burden of using this method during simulation can be significant. To tackle this issue, utilizing artificial neural networks (ANNs) to approximate the solution of algebraic loops is a promising approach. While ANN surrogates offer fast performance, ensuring the correctness of the computed solution or quantifying reliability can be challenging.

This publication presents a prototype, based on the OpenModelica compiler (OMC) (Fritzson et al. 2020), that automates the extraction of time-consuming algebraic loops. It generates training data, trains ANNs using machine learning (ML) methods, and replaces the algebraic loops with ANN surrogates in the simulation code. A hybrid approach, combining the trained surrogate with the nonlinear Newton solver, is then used to compute the solution with a desired level of accuracy.

Keywords: Machine Learning, Dynamic Systems, Surrogate Model, Non-Linear System, Error Control

1 Introduction

Modelling and simulation play a major role in many fields of science, technology, engineering and mathematics. Modelica (Mattsson and Elmqvist 1997) is an established object-oriented language for multi-domain modeling. It is easy to develop model-based components using simple textbook equations and combine them into detailed and complex cyber-physical systems. With increasing complexity even on modern Modelica compilers simulation performance can slow down.

One way to computational accelerate Modelica components is using ML surrogates. Such a surrogate approximates the equation-based Modelica model with a data-driven approach. When sufficiently trained, a surrogate can replace the corresponding Modelica equations and the resulting speedup can be utilized e.g., in parameter optimization.

Different data-driven ML methods are used in the context of modelling and simulation. ANNs as methods of artificial intelligence (AI) are often used, in particular physics informed neural networks (PINNs) (Lawal et al. 2022), long short-term memory (LSTM) networks (Hochreiter and Schmidhuber 1997), continuous-time echo state networks (CTESNs) (Anantharaman et al. 2020) could demonstrate impressive speedups of simulation time for complex models. While these methods are fast and precise no guarantees for correctness can be made that the surrogate solutions stays within the desired error tolerances.

So called hybrid physical-AI based models are a compromise between classical and ML models. A hybrid model can consist of equations derived from first principle physics as well as data-driven ML models. They offer better simulation performance with acceptable accuracy. While (Hübel et al. 2022) could show improved performance with a reduced order model the resulting hybrid model cannot be used outside of the trained area or ensure a given error tolerance.

In this publication the authors present a partially automated method to replace non-linear algebraic loops of Modelica models with error-controlled ML surrogates to generate hybrid physical-AI based models. The relation between inputs and outputs of the loop are learned from synthesized data and reference simulations. It could be shown, that with the use of ANN the simulation time could be sped up by a factor of 1.5 while keeping the surrogate prediction within a given error tolerance.

This enables users to select the tradeoff between accuracy and speedup.

Paper Organization

Subsection 3.1 describes the use of profiling to identify NLSs that are worthwhile to replace. Different methods to generate artificial training data from the original model are discussed in Section 3.2. The exemplary training of feedforward neural networks (FNNs) is illustrated in Section 3.3 while Section 3.4 shows an approach to reduce the demand for generated training data. Section 3.5 presents

DOI
10.3384/ecp204275

the integration of the trained surrogate into the simulation while the error control is discussed in Section 3.6. Results are shown in Section 4.2.2 with models from the ScalableTranslationStatistics Modelica library. Finally section 5 discusses results and encountered problems and section 6 concludes the paper.

2 Problem Statement

Developing a detailed high fidelity Modelica model named M_{hf} is an elaborated task and simulating such a model can take a significant amount of time. For applications like parameter fitting simulation speed outweighs fidelity, so another Modelica model named M_{sur} is needed to complete the given task in an acceptable amount of time. The classic approach is to manually replace expensive equations of the original M_{hf} model and reduce the complexity of the modeled physics. Another approach is to generate large sets of artificial data from the expensive to solve M_{hf} model to then train a ML surrogate.

The ordinary differential equation (ODE) of M_{hf} can have subsystems of equations that need to be evaluated simultaneously. These subsystems, also known as algebraic loops, can consist of linear or non-linear equations. This paper only discusses the treatment of non-linear loops, since in general they are harder to solve than linear loops.

An algebraic loop can be described in its residual form

$$\begin{gathered} f_{res} : I_t \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_{in}} \times \mathbb{R}^{n_{out}} \to \mathbb{R}^{n_{out}}, \\ f_{res}(t, p, z_{in}, z_{out}) \overset{!}{=} 0 \end{gathered} \tag{1}$$

with simulation time $I_t := [t_{start}, t_{stop}] \subset \mathbb{R}$, parameters $p \in \mathbb{R}^{n_p}$, used variables $z_{in} \in \mathbb{R}^{n_{in}}$ computed in preceding model equations and unknowns $z_{out} \in \mathbb{R}^{n_{out}}$. Define a function

$$f_{NLS} : I_t \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_{in}} \to \mathbb{R}^{n_{out}}, \;\; f_{NLS}(t, p, z_{in}) = z_{out} \tag{2}$$

that solves Equation 1 explicitly. For simplicity non-unique solutions to Equation 2 are ignored for now. In practice this function is approximated by iterative root finding methods solving Equation 1, for example the Newton-Raphson method.

Instead of replacing all equations of M_{hf} with ML surrogates the authors propose to replace only the slowest non-linear equation systems f_{NLS} with faster ML surrogates f_S to reduce the time each evaluation of the right-hand side of the ODE takes.

Because the residual f_{res} and its Jacobian J are available, it is possible to compute error approximations for prediction $\tilde{z}_{out} = f_S(t, p, z_{in})$. Therefore it is possible to ensure, that the prediction $\tilde{z}_{out}$ stays within a given tolerance and the default non-linear solver can improve the solution if necessary.

By using an error controlled surrogate overall simulation time could be improved by a factor of 1.5 as shown in Section 4.2.2.

3 Method

Dependency information of the equations as well as a callable C function are necessary to replace f_{NLS} with a fast surrogate f_S. Because of this the authors choose the open-source OMC. It offers ways to interfere with the compilation process in order to retrieve the aforementioned requirements.

The method to generate a FMU containing the fast surrogates for slow non-linear equation systems consists of four main steps:

1. Find NLSs worth replacing (Section 3.1).
2. Generate training data (Section 3.2).
3. Train surrogates (Section 3.3).
4. Integrate trained surrogates into original model (Section 3.5).

Steps 1, 2 and 4 can be handled by the prototype implementation while the training process in step 3 still needs human intervention.

3.1 Profiling Model

The first step consists of analyzing how much time is spent for each NLS in relation to the total simulation time. Profiling for each model equation is performed to find all NLSs that need more of the total simulation time than a defined threshold. Since NLSs from the initial systems are solved only once at simulation time $t = t_{start}$ they are not considered for replacement.

Solving an NLS is time consuming, independent of the chosen Modelica tool. The example `ScaledNLEquations.NLEquations_5` from Section 4.2 was profiled in two popular tools: Dymola and OpenModelica. Most of the simulation time is spent solving the 8 non-linear systems, as can be seen in Table 1. Dymola spends around 53.05% and OpenModelica spends around 51.50% of the total simulation time solving these non-linear systems.

To find suitable equations for surrogate replacement the OpenModelica profiler (Sjölund 2015) is used to identify the Modelica equations corresponding to the slowest NLSs. The generated simulation results are used in Section 3.2 to specify the relevant input space for generating training data.

3.2 Data Generation

After identifying equations for replacement data driven surrogates need training, validation, and test data.

For the NLS all reached values for z_{in} are recorded. To only generate data in the area of interest the reference solution from the profiling step can be used to define a hypercube around the reference variables:

$$H_{in} := I_t \times I_{in} \tag{3}$$

$$I_{in} := \{ z \in \mathbb{R}^{n_{in}} \mid a_j \leq z_j \leq b_j \forall j \} \tag{4}$$

where

$$a_j := \min_{t \in I_t} z_{in}(t)_j, \; b_j := \max_{t \in I_t} z_{in}(t)_j. \tag{5}$$

Table 1. Profiling non-linear equation systems of ScalableTranslationStatistics.Examples.ScaledNLEquations.NLEquations_5

OpenModelica			*Dymola*		
Index	*Total* [s]	*Fraction* [%]	*Block*	*Total* [s]	*Fraction* [%]
	5.892	100.00		4.830	100.00
907	0.434	7.54	1100	0.334	6.91
928	0.433	7.51	1117	0.334	6.91
834	0.423	7.35	1187	0.333	6.91
813	0.396	6.88	1136	0.327	6.76
882	0.392	6.81	1204	0.323	6.68
951	0.392	6.80	1153	0.322	6.65
857	0.386	6.70	1170	0.319	6.60
987	0.111	1.92	1083	0.272	5.63

If the user has knowledge of the input variables it is viable to refine I_{in} to limit the training area further. For example physical constraints or boundary conditions can enforce that the surrogate only has to be valid in a specific region or that some combinations of different inputs are not reachable.

For the training process the input space I_{in} must be sampled sufficiently dense and corresponding solutions saved. One straightforward approach is to initialize the model at time t_{start} to set all constants and parameters. Subsequently pairs (z_{in}, z_{out}) are computed, where $z_{in} \in I_{in}$ random and z_{out} is computed by Equation 2.

Improvements of the data generation regarding the ANN training effort are discussed in Section 3.2.4.

3.2.1 `evaluateEquation` C Interface

All equations f_{NLS} can be evaluated with `evaluateEquation` without evaluating any other equations. The input variables z_{in} must be set using the appropriate `set` function before evaluating the equation. Afterwards the solution z_{out} can be inquired with the corresponding `get` functions.

```
status evaluateEquation(model c,
                        const size_t eqNumber);
```

- Argument `c` is the pointer to the model specific data structure of OMC.
- Argument `eqNumber` specifies the unique equation index of the equation to be evaluated.
- If the equation was successfully evaluated `success` is returned, otherwise `discard` is returned.

3.2.2 Implementation Details

The methods described in this paper are implemented in the prototypical Julia package NonLinearSystemNeuralNetworkFMU.jl[1]. For the function `evaluateEquation` an OpenModelica source-code ModelExchange 2.0.4 Functional Mock-up Unit (FMU) is built from the original Modelica model `M` using the following compiler flags[2]:

```
    --fmiFilter=internal
    --fmuCMakeBuild=true
    --fmuRuntimeDepends=modelica
```

Then the C source files for `evaluateEquation` are included into the FMU. The binaries and `M.fmu` are recompiled using the provided `CMakeLists.txt` file. The resulting extended FMU is called `M.interface.fmu`.

To compute input-output pairs (z_{in}, z_{out}) Julia package FMI.jl[3] (Thummerer, Mikelsons, and Kircher 2021) is used to instantiate the FMU, set z_{in} and call `evaluateEquation` to evaluate $f_{NLS}(t, p, z_{in})$ with the Newton-Raphson method. The resulting input-output pairs (z_{in}, z_{out}) are saved to a CSV file for each NLS.

The Functional Mock-up Interface (FMI) standard is used because it provides a standardized way to instantiate, initialize, and solve an ODE system. At the time of writing FMI.jl allows the illegal call of `fmi2SetXXX`, allowing this workflow for a provisional but functioning prototype. This might change in a future version. In a matured implementation these steps must be performed directly in the Modelica compiler or its simulation runtime.

3.2.3 Preprocessing Data

It is possible for a NLS to have multiple solutions $z_{out_1} \neq z_{out_2}$, such that for the same input z_{in}

$$f_{res}(t, p, z_{in}, z_{out_1}) = f_{res}(t, p, z_{in}, z_{out_2}) = 0. \qquad (6)$$

In this case Equation 2 is multi-valued and no unique functional relation exists. In case the training data has two sets

[1] NonLinearSystemNeuralNetworkFMU.jl v0.5.1: github.com/AnHeuermann/NonLinearSystemNeuralNetworkFMU.jl

[2] OpenModelica User's Guide on compiler flags: openmodelica.org/doc/OpenModelicaUsersGuide/1.20/omchelptext.html

[3] ThummeTo/FMI.jl v0.10.2: github.com/ThummeTo/FMI.jl

of input data that are close together but the corresponding outputs are far apart, the surrogate will not be able to approximate it. The training data has to be processed in a way that the relation from input to output is sufficiently continuous, i.e. not jumping between different solution branches.

Modelica tools usually follow one solution continuously if the step size of the ODE solver is small enough and the previous solution of the NLS is a good enough start value for the root finding method. Algorithm 1 from Section 3.2.4 tries to mimic this behavior, so on one trajectory the solution should not jump between different branches. However this only guarantees local uniqueness as two different trajectories to an input z_{in} can still lead to two different outputs $z_{out_1} \neq z_{out_2}$.

3.2.4 Improving Data Generation

Iterative non-linear solver methods need a decent start value to converge to a solution. Whether the method converges and if so at which rate highly depend on the chosen start values. An intuitive method to generate training data using small random perturbations is described in Algorithm 1.

Algorithm 1 Random Walk

```
procedure RANDOMWALK($\delta, \Delta_t$)
    $z_{out} \leftarrow 0$
    $z_{in} \leftarrow$ random value from $I_{in}$
    for $t = t_{start}, t_{start} + \Delta_t, \dots, t_{end}$ do
        $z_{out} \leftarrow f_{NLS}(t, p, z_{in})$, using previous $z_{out}$ as start value
        Save $(z_{in}, z_{out})$
        $z_{in} \leftarrow z_{in} + \delta\omega$, where $\omega \in [-1,1]^{n_{in}}$ random
        Ensure $z_{in} \in I_{in}$
```

With small enough $\delta > 0$ and $\Delta_t > 0$ the number of iteration steps needed to solve the NLS for a given tolerance should be low (close or equal to 1), since the previously computed solution z_{out} is a good start value for the next small random perturbation of input vector $z_{in} \leftarrow z_{in} + \delta\omega$. The data generation process consists of creating several of these `randomWalk` trajectories to cover I_{in} densely.

3.3 Supervised Learning

Using the generated training data from Section 3.2 it is possible to train a ML surrogate for each NLS. This paper restricts itself to simple FNN models:

$$model_1(t, z_{in}) \approx f_{NLS}(t, p, z_{in}) = z_{out} \tag{7}$$

Due to implementation limitations parameters p are not changeable now and constant during the training process. That means each parameter configuration requires its own surrogate. It is planned to address this in future work.

To solve the issue of ambiguous solutions a different FNN

$$model_2(t, z_{in}, \hat{z}_{out}) \approx f_{NLS}(t, p, z_{in}) = z_{out} \tag{8}$$

is defined, where the solution from the previous time step $\hat{z}_{out}$ is given as an additional input. With the information from the previous ODE integrator step the surrogate should learn to predict a solution that is close to the previous solution and not jump to a different solution branch.

3.4 Active Learning

Data for the NLS can be generated at arbitrary inputs inside an appropriate region as discussed in Section 3.2. However, a call to the root finding algorithm is expensive in general. Therefore, a variation of active learning (AL) (Settles 2009; Wu, Lin, and Huang 2018) can be used for training the surrogate f_S.

The general idea of AL is to let the surrogate decide which samples to label, i.e. which inputs to generate the corresponding outputs for. Between training steps the performance of f_S is tested on new inputs z_{in}. Samples from unfit inputs i.e., inputs for which f_S performs poorly, are added to the set T for the next training step. This is described more precisely in Algorithm 2, where m is the number of active learning steps, n is the number of total samples to generate, and $1 - p$ is the fraction of samples that are generated randomly for the first training step, so $p = 0$ is equivalent to not using AL.

Algorithm 2 Active Learning

```
procedure ACTIVELEARN($m, n, p$)
    $T \leftarrow$ initial data set with $|T| = (1-p)n$
    for $i = 1, \dots, m$ do
        train $f_S$ on $T$
        $T' \leftarrow$ FINDUNFITSAMPLES($\frac{pn}{m}$)
        $T \leftarrow T \cup T'$
    return $f_S$
procedure FINDUNFITSAMPLES($n$)
    $T \leftarrow \emptyset$
    for $i = 1, \dots, n$ do
        choose $z_{in} \in I_{in}$ with large expected error
        $z_{out} \leftarrow f_{NLS}(t, p, z_{in})$
        $T \leftarrow T \cup \{(z_{in}, z_{out})\}$
    return $T$
```

If Equation 6 does not apply, the residual norm

$$\tau_{abs}(z_{in}, f_S) := \|f_{res}(t, p, z_{in}, f_S(z_{in}))\|_2 \tag{9}$$

gives a comparatively cheap measure for identifying unfit inputs, without the need for root finding. However, residual equations may still contain expensive computations, so in either case evaluations need to be done economically. Since there is some freedom in generating new data, finding unfit inputs can be seen as a multimodal optimization problem inside the classical ML optimization problem

$$\min_{f_S} \left\{ \max_{z_{in}} \tau_{abs}(z_{in}, f_S) \right\} \tag{10}$$

which can be solved by any cheap optimization heuristic and points generated along the way can be added to the data set T. This corresponds to line 11 in Algorithm 2. A simple variant of the bees algorithm (Pham et al. 2006) was chosen which combines global and local search.

An analysis of the effectiveness of AL is given in Section 4.3.

3.5 Integrate Surrogate into Simulation

The trained Flux model from Section 3.3 is exported in the Open Neural Network Exchange (ONNX) format (Bai, Lu, Zhang, et al. 2017) using ONNXNaiveNASflux.jl[4]. For each f_{NLS} that is replaced by a surrogate the corresponding ONNX file is copied into the `M.interface.fmu` resources directory. The ONNX Runtime (ORT) (ONNX Runtime developers 2021) is used to interact with the ONNX object. During `fmi2Instantiate` all ORT data is initialized and the ONNX files are loaded. In the C code responsible for evaluating the NLS it is possible to switch between the iterative solver method and the evaluation of the surrogate FNN. The FMU is then compiled and packed into `M.onnx.fmu`.

3.6 Surrogate Error Control

Using Equation 1 it is possible to define an error control algorithm. Computing the residual error from Equation 9 is cheap, but for many examples it is important to use the scaled residual norm instead:

$$\tau_s(J) := \|s(J) \circ f_{res}(t,p,z_{in},\tilde{z}_{out})\|_2 \tag{11}$$

Here $\circ$ is element-wise multiplication, s a scaling vector

$$s_i(J) := \frac{1}{\|J_{i,*}\|_\infty}, \quad i = 1,\dots,n_{out}, \tag{12}$$

with $J_{i,*}$ being the i-th row of the Jacobian J of f_{NLS} and $\|\cdot\|_\infty$ the maximum norm. To utilize the scaled residual norm for the error control it is necessary to evaluate the Jacobian J at each time step. Especially for numeric Jacobians this can be costly to evaluate, but it is still cheaper than a Newton-Raphson step, where the Jacobian needs to be inverted in addition.

Hoping that the Jacobian does not change too much during simulation, Algorithm 3 reuses J from the initialization and updates it whenever the default iterative method needs to evaluate the Jacobian anyway.

If f_S is not performing well on z_{in} or $z_{in} \notin I_{in}$, Equation 2 can be solved by the iterative solver method with a start value from the surrogate or extrapolated from previous solutions $\hat{z}_{out}$.

3.7 (Re-)Initialization and Events

The Modelica language is able to express models that can have discontinuities in the right-hand side of their ODE system. For $model_1$ from Equation 7 events and reinitialization are no issue, if the event is not changing the system structure of the NLS.

Algorithm 3 Error Control

```
procedure ERRORCONTROL($\tau$, $J$)
    if $z_{in} \in I_{in}$ then
        $z_{out} \leftarrow f_S(t,p,z_{in})$
        if $\tau_s(J) > \tau$ then
            $z_{out}, J \leftarrow f_{NLS}(t,p,z_{in})$ with start value $z_{out}$
    else
        $z_{out} \leftarrow extrapolate(\hat{z}_{out})$
                    ▷ extrapolate $z_{out}$ from previous time step(s)
        $z_{out}, J \leftarrow f_{NLS}(t,p,z_{in})$ with start value $z_{out}$
```

In contrast $model_2$ from Equation 8 uses the solution $\hat{z}_{out}$ from the previous time step. If the NLS is solved for the first time or if the previous solution is invalid because an event occurred the previous solution is not available and the original equation f_{NLS} is evaluated first.

The `delay` and `spatialDistribution` operators of the Modelica language specification are not considered but they don't seem to be an issue as long as they are not used inside f_{res}.

4 Experiments

The method described in section 3 is tested on a mechanical mass-spring system with scalable non-linear equation systems. The library is presented in Section 4.2 and the generation of surrogates is described in Section 4.2.1. In Section 4.2.2 the simulation results are compared to the Newton-Raphson method used by OpenModelica. Section 4.3 demonstrates reduced data consumption on a Modelica toy model.

4.1 Test Setup

All examples were run on a test server with an Intel Xeon Gold 6248R CPU @ 3.00GHz, 192 GB DDR4 RAM @ 2933 MT/s, NVIDIA Quadro RTX 6000 GPU with 24 GB SDRAM on Ubuntu 22.04.2 LTS. Julia v1.9.0 with packages FMI.jl v0.12.2, Flux.jl v0.13.16, ONNXNaiveNASflux.jl v0.2.7, OMJulia.jl v0.2.1 together with OpenModelica v1.22.0-dev-156 was used.

4.2 ScalableTranslationStatistics Library

In this section the Modelica library ScalableTranslationStatistics[5] is presented, which will be used as an example for the algebraic loop replacement in Section 4.2.1. The ScalableTranslationStatistics library offers many possibilities to create models of specified numerical complexity. It can be used to create generic examples for specific numeric problem classes and thus avoids the need of sharing confidential models.

Model properties like number of algebraic loops, continuous time states or use of numeric Jacobians can be specified a priori via structural parameters in the model. These are the structural properties the model will have after the translation. Due to different algorithms used during

[4] GitHub Repository: DrChainsaw/ONNXNaiveNASflux.jl

[5] ltx.de/download/ModelicaLibraries/ScalableTranslationStatistics

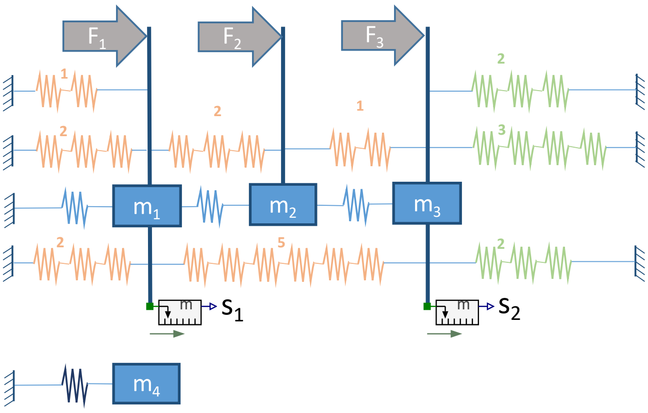


Figure 1. Scalable mass-spring system with parametrization `num_masses=4`, `Lin_equations = {2,3,2}` and `NL_equations = {2,1,5,1,2,2}`.

the translation in different Modelica tools, the final structural properties might slightly differ from the given parameters, but the principal functionality is not tool-dependent.

The principal idea of this library - to scale models - is based on the ScalableTestSuite[6]. But since this library doesn't offer the possibility to scale structural properties like the size of algebraic loops, the ScalableTranslationStatistics library was developed. Figure 1 shows the physical representation for an exemplary parametrization of the model.

A specified number of continuous time states is reached by introducing masses, each having two state variables (position and velocity). In the most simple case the masses are only connected with simple linear springs (blue springs in Figure 1), to avoid in any case singular systems due to free, unconnected masses. To obtain linear or nonlinear equation systems, springs are directly connected with each other. Thus, an algebraic loop of size $M-1$ is created, where M is the number of springs. Depending on the spring characteristics the algebraic loop has a linear or nonlinear behavior.

A simple way to enforce numeric block Jacobian is to introduce an assert-statement in the spring-characteristic to limit the force to a given maximum value, or reading the characteristics from a file. In both cases the characteristic cannot be differentiated analytically but needs to be solved numerically. In the latter case an arbitrary characteristic can be defined.

Furthermore, following additional features are implemented:

- External forces (F_i) can act as inputs to the system, position measurement sensors (S_i) as outputs. By an appropriate definition of the input-forces (e.g. via a TimeTable) a desired number of time- or state-events can be reached.
- Additional parameters, time varying variables and alias variables can be added to the model. They have no influence on the physical behavior, but increase the size of the model.
- The model contains an independent mass-spring system with a different stiffness of the spring (m_4 in Figure 1). Thus, the stiffness can be adjusted to increase the effort for an integrator to solve the model.
- Two-dimensional springs can be added to the model. The directional stiffness of these springs depends on the deflection in both directions. In this way partial derivatives are introduced.

Besides the principal model, the library offers numerous examples of different scaling and for the above-mentioned features.

4.2.1 Generating Surrogates

To test the method presented in section 3 the model `ScalableTranslationStatistics.Examples.ScaledNLEquations.NLEquations_N` is used with $N \in \{5, 10, 20, 40\}$ where N scales the number of iteration variables. The model has eight NLSs with $2N$ unknowns. Because of tearing (Täuber et al. 2014) there are N iteration variables and N inner variables. The first seven systems cannot be differentiated symbolically, therefore numeric Jacobians are used. The fastest system is differentiated symbolically. ODE integrator DASSL (Petzold 1982) is used with tolerance 10^{-6} to simulate in $I_t = [0, 10]$. For small systems ($N \in \{5, 10, 20\}$) the dense Newton-Raphson method is used. For the larger systems ($N \geq 40$) the sparsity of J is 7 %, so the sparse solver KINSOL (Alan C Hindmarsh et al. 2005; Alan C. Hindmarsh et al. 2023) is used instead of the dense one. After profiling Figure 2 shows a significant amount of the simulation time is spent to solve seven of the NLSs, which is an upper bound for the amount of time that could be saved by a faster surrogate.

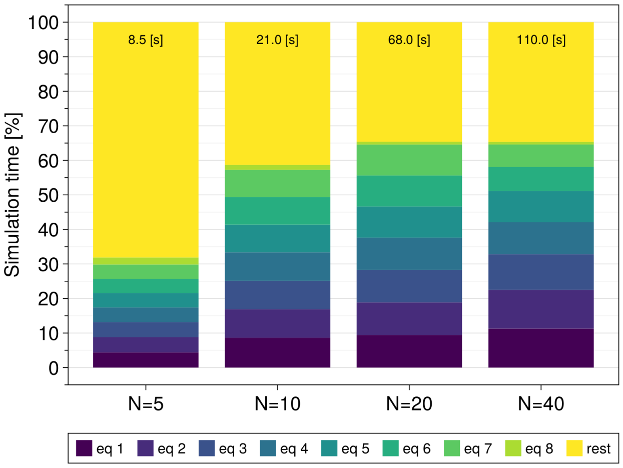


Figure 2. Profiling relative simulation times for `ScalableTranslationStatistics.Examples.ScaledNLEquations.NLEquations_N` with $N \in \{5, 10, 20, 40\}$ and the eight NLS *eq* 1, ..., *eq* 8 as well as all remaining equations *rest*.

[6] github.com/casella/ScalableTestSuite

During data generation Algorithm 1 is used to generate 5,000 data points in 100 batches of 50 calls to `randomWalk` with $\delta = 0.01$ for each N.

80 % of the available data points are used for the training set, the remaining 20 % for the validation set. The model is simulated over $I_t = [0, 10]$ to test the ANN.

For FNN Equation 7 is fitted to the normalized training data $\tilde{I}_{in}$ using Julia package Flux.jl (Michael Innes et al. 2018; Mike Innes 2018). A model with one input, one hidden and one output layer is created:

```
model1 = Flux.Chain(
  Flux.Dense(nIn,      nIn*10,  σ),
  Flux.Dense(nIn*10,  nOut*10, tanh),
  Flux.Dense(nOut*10, nOut)
)
```

with `nIn` $= 1 + n_{in}$ and `nOut` $= n_{out}$, activation functions sigmoid σ and hyperbolic tangent tanh. The mean square error is used as loss function and Adaptive Moment Estimation (Adam) optimization is used to train the model.

model$_1$ is trained over 1000 epochs or until the loss of the training set is below 10^{-6}.

4.2.2 Simulation Results

With the generated FMU containing surrogates for all eight NLSs the simulation times are measured and compared to the Newton-Raphson method as reference. The models are simulated with an explicit Euler method with fixed step size of 0.001 in time interval $[0, 10]$. In Figure 3 the simulation times and speedup factors are plotted for different values of N. While the evaluation of *model*$_1$ is slower for small NLS, with growing N the surrogates are significantly faster to evaluate (up to 9 times). With more sophisticated ANN structures even better performance is expected. Unfortunately, the overall savings for the total simulations are not as large as expected. For $N = 40$ the surrogate is only 1.55 times faster than the original high fidelity model.

In Figure 4 the simulation results of iteration variables `scalableModelicaModel.springChain[1].spring[m].s_rel` for $m \in \{1, 3, 4, 5\}$ from NLS equation 808 are displayed. It can be observed, that while the results of the iteration variables of the surrogate compared to the reference solution have a low absolute error

$$|z_{out_i} - \tilde{z}_{out_i}|, \ i \in \{1, \dots, 4\} \tag{13}$$

the error $\tau_s(J)$, displayed in Figure 5, of the residual is relatively large. If $\tau_s(J) > 1$ the original Newton-Raphson method was used to solve the NLS.

The relevant output variables are the positions of eight masses `output[m]` for $m \in \{1, \dots, 8\}$. The results of the surrogate simulation are compared to the reference solution in Figure 6.

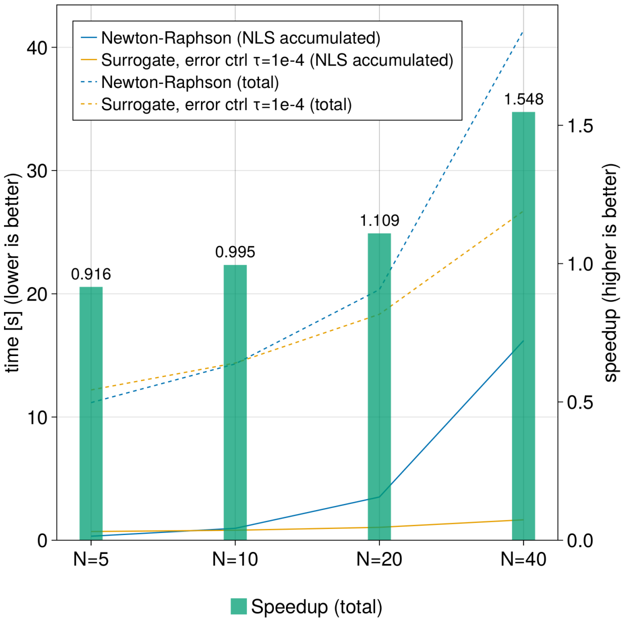


Figure 3. The simulation time of the surrogates are plotted against the reference Newton-Raphson method as well as the respective speedup in total simulation time.

4.3 Controlled Data Generation

The training improvements of AL from Section 3.4 are studied using the simple Modelica model `SimpleLoop` which has a single NLS of size 2. It describes the intersection of a circle with a line, both changing over time:

```
model SimpleLoop
  Real r = 1+time;
  Real s = sqrt((2-time)*0.9);
  Real x(start=1.0), y(start=-0.1);
equation
  r^2 = x^2 + y^2;
  r*s = x + y;
  annotation(experiment(StopTime=2));
end SimpleLoop;
```

This model has two distinct solutions since the equations are symmetric in `x` and `y`. The NLS is torn to a single iteration variable and has two inputs, `r` and `s`.

For this model a surrogate with a total of 441 parameters was trained. Figure 7 shows the peaks in τ_{abs} from Equation 9 for different training scenarios. For $p = 0$ surrogates improve slightly with increasing data size n. Using AL on $p = \frac{1}{4}$ of the generated data improves the surrogate significantly, even for small n. Generating $p = \frac{1}{2}$ to $\frac{3}{4}$ with AL seems to be optimal with slight improvements over $p = \frac{1}{4}$. However, surrogates with no pre-training whatsoever perform no better or even worse than with $p = \frac{1}{2}$ pre-training. In particular, for the scenario $n = 800, p = 1$, two out of ten simulations showed bad results and so did one simulation for $n = 1000, p = 0.5$. Still, $p = 0$ produces worse results regardless of the value of n.

5 Results

When increasing the size of the non-linear systems the advantage of surrogates becomes more and more evident.

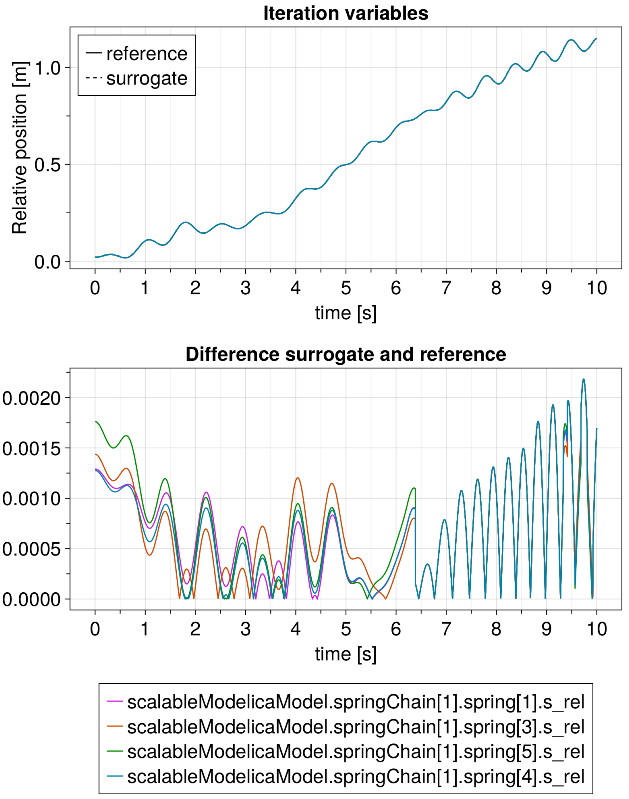


Figure 4. For $N = 5$ the iteration variables of the surrogate and the reference solution are compared in the upper graph. The solid lines are the reference and the dotted lines the surrogate solution. They are so close, that the dotted lines are not visible. The variables are describing a relative position of springs in a chain of springs. The lower graph shows the absolute difference between the reference and surrogate solution.

However, the examined Modelica model is simple. When more complex and application-oriented examples where investigated several problems were encountered, that are not yet solved and discussed in the following subsection.

The AL approach was tested on a small toy example and proved to outperform the method of random data generation both in precision and in data efficiency. However further experiments need to be done to see to what extent AL can impact the surrogate training process for more complex Modelica models.

5.1 Encountered Problems

For larger and highly non-linear NLS it is more complicated to train accurate enough FNN. Simple NLS that have only a few iteration variables can use a lot of previously computed variables. The surrogate tries to approximate a function from the used variables to the solution of the iteration variables. In this case an easy to solve NLS becomes difficult to train for an ANN.

An especially difficult problem are iteration variables that influence the ODE states. There are two different issues with this. When trying to solve the simulation executable with the surrogates the error control cannot manage the imprecise solution of the surrogate. While the approximation would be good enough for the use case the error control of the ODE method will reduce the step size until the lower limit is reached. And when the error control of ODE integrator is deactivated coupled states are a problem.

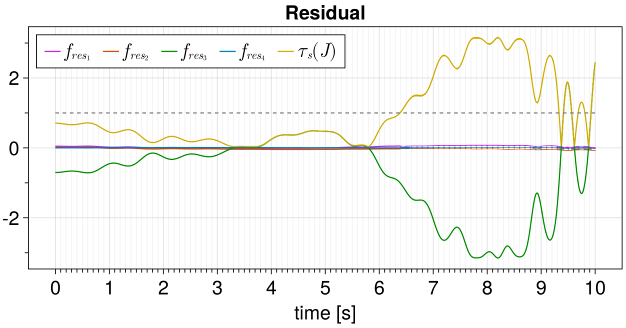


Figure 5. For $N = 5$ equation 808 the residual vector and its norm over the simulation time are displayed. If the value of $\tau_s(J)$ is larger than 1 Algorithm 3 switches to the original nonlinear solver method to refine the prediction from the surrogate.

For analytic stable ODE a well-suited integration method transports this stability to the numeric approximation of the solution. This means numeric errors will vanish over time and the numeric solution converges to the analytic solution. In contrast it seems that small errors from the surrogate will escalate over time and the numeric solution gets worse over time. This needs to be investigated more. Iteration variables that have a high sensitivity towards states can be a significant issue.

6 Summary and Outlook

The presented prototype NonLinearSystemNeuralNetworkFMU.jl aims to lower the bar for Modelica users to utilize hybrid modeling approaches in their models. Even though the scripts are in an early development stage they could pave the way for tool supported integration of ML into Modelica models.

The example from section 4 shows that there is some potential in replacing sufficient large NLS with machine learning surrogates. The focus of this paper is to automate the workflow for data generation and integration of trained surrogates into the simulation executable. So further refinements of the ANN could result in faster and more precise evaluations of the surrogates. The ability to use hybrid ML methods for problems with discrete events and inputs distinguishes the presented method from methods replacing all of the right-hand side of the ODE and ensures correctness of the surrogates.

The workflow can be extended to profile linear equation systems as well as external Modelica functions and automate data generation for these systems or functions. Especially functions computing media properties, e.g. for thermal fluid systems, can be expensive and library developers and users are searching for ways to decrease the

DOI
10.3384/ecp204275

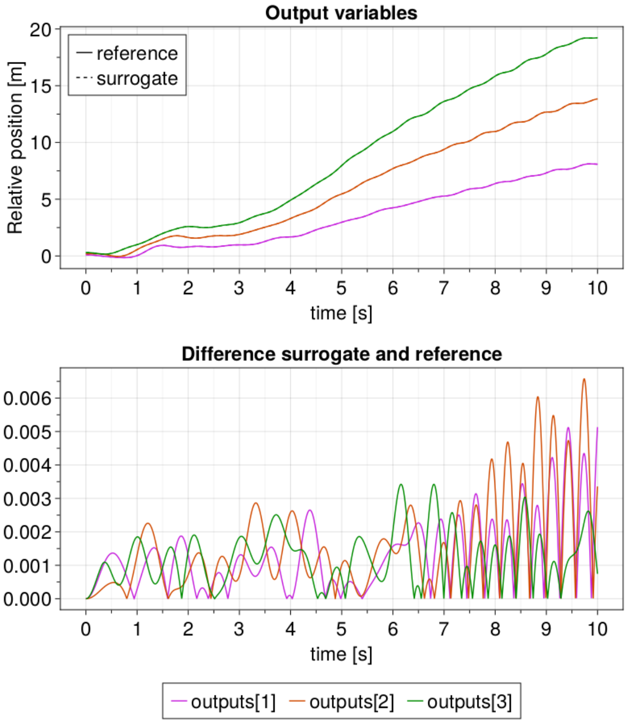


Figure 6. Simulation results in the upper graph and absolute errors in the bottom graph for $N = 5$ equation 808. In the upper graph the solid lines are the reference and the dotted lines the surrogate solution. They are so close, that the dotted lines are not visible.

time spent evaluating these functions.

Instead of using ANN we plan to use symbolic regression to obtain equations that represent the underlying physics while being less of a black box.

Acknowledgements

This work has been supported by the German Federal Ministry for Economic Affairs and Climate Action in the framework of PHyMoS - Proper Hybrid Models for Smarter Vehicles (grant number 19I20022G and 19I20022E).

References

Anantharaman, Ranjan et al. (2020). *Accelerating Simulation of Stiff Nonlinear Systems using Continuous-Time Echo State Networks*. DOI: 10.48550/ARXIV.2010.04004. URL: https://arxiv.org/abs/2010.04004.

Bai, Junjie, Fang Lu, Ke Zhang, et al. (2017). *ONNX: Open Neural Network Exchange*. https://github.com/onnx/onnx. Version: v1.12.0.

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Hindmarsh, Alan C et al. (2005). "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software (TOMS)* 31.3, pp. 363–396. DOI: 10.1145/1089014.1089020.

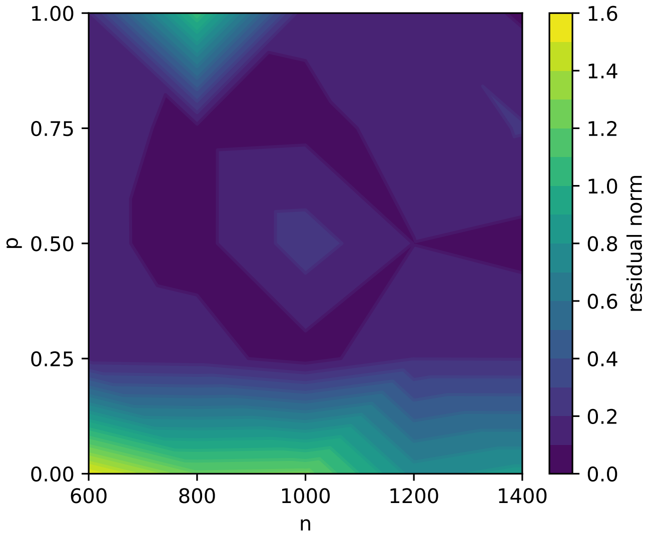


Figure 7. Peaks of τ_{abs} after simulating surrogates trained with $m = 10$ and different values of $n \in \{600, 800, 1000, 1200, 1400\}$ and $p \in \{0, 0.25, 0.5, 0.75, 1\}$. Each data point is the average over 10 surrogates.

Hindmarsh, Alan C. et al. (2023). *User Documentation for KINSOL*. v6.5.1.

Hochreiter, Sepp and Jürgen Schmidhuber (1997-11). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

Hübel, Moritz et al. (2022-11). "Hybrid physical-AI based system modeling and simulation approach demonstrated on an automotive fuel cell". In: *Modelica Conferences*. Ed. by Tielong Shen, Rui Gao, and Yutaka Hirano. Linköping University Electronic Press, pp. 157–163. DOI: https://doi.org/10.3384/ecp193157.

Innes, Michael et al. (2018). "Fashionable Modelling with Flux". In: *CoRR* abs/1811.01457. arXiv: 1811.01457. URL: https://arxiv.org/abs/1811.01457.

Innes, Mike (2018). "Flux: Elegant Machine Learning with Julia". In: *Journal of Open Source Software*. DOI: 10.21105/joss.00602.

Lawal, Zaharaddeen Karami et al. (2022). "Physics-Informed Neural Network (PINN) Evolution and Beyond: A Systematic Literature Review and Bibliometric Analysis". In: *Big Data and Cognitive Computing* 6.4. ISSN: 2504-2289. DOI: 10.3390/bdcc6040140. URL: https://www.mdpi.com/2504-2289/6/4/140.

Mattsson, Sven Erik and Hilding Elmqvist (1997). "Modelica-An international effort to design the next generation modeling language". In: *IFAC Proceedings Volumes* 30.4, pp. 151–155.

ONNX Runtime developers (2021). *ONNX Runtime*. https://onnxruntime.ai/. Version: 1.12.1.

Petzold, Linda R (1982). *Description of DASSL: a differential/algebraic system solver*. Tech. rep. Sandia National Labs., Livermore, CA (USA).

Pham, D.T. et al. (2006). "The Bees Algorithm — A Novel Tool for Complex Optimisation Problems". In: *Intelligent Production Machines and Systems*. Ed. by D.T. Pham, E.E. Eldukhri, and A.J. Soroka. Oxford: Elsevier Science Ltd, pp. 454–459. ISBN: 978-0-08-045157-2. DOI: https://doi.org/10.1016/B978-008045157-2/50081-X. URL: https://www.sciencedirect.com/science/article/pii/B978008045157250081X.

Settles, Burr (2009). *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison.

Sjölund, Martin (2015). *Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models*. Vol. 1664. Linköping University Electronic Press.

Täuber, Patrick et al. (2014). "Practical Realization and Adaptation of Cellier's Tearing Method". In: *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. EOOLT '14. Berlin, Germany: Association for Computing Machinery, pp. 11–19. ISBN: 9781450329538. DOI: 10.1145/2666202.2666204. URL: https://doi.org/10.1145/2666202.2666204.

Thummerer, Tobias, Lars Mikelsons, and Josef Kircher (2021-09). "NeuralFMU: Towards Structural Integration of FMUs into NeuralNetworks". In: *Proceedings of 14th Modelica Conference 2021*. Ed. by Adrian Pop Martin Sjölund Lena Buffoni and Lennart Ochel. Linköping University Electronic Press, pp. 297–306. DOI: 10.3384/ecp21181297.

Wu, Dongrui, Chin-Teng Lin, and Jian Huang (2018). *Active Learning for Regression Using Greedy Sampling*. arXiv: 1808.04245 [cs.LG].

Exploiting Modelica and the OpenIPSL for University Campus Microgrid Model Development

Fernando Fachini[1] Srijita Bhattacharjee[1] Miguel Aguilera[2] Luigi Vanfretti[1] Giuseppe Laera[1]
Tetiana Bogodorova[1] Ardeshir Moftakhari[3] Michael Huylo[4] Atila Novoselac[4]

[1]Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, USA,
{fachif, bhatts10, vanfrl, laerag, bogodt2}@rpi.edu
[2]OPAL-RT Technologies, Canada, Miguel.Aguilera@opal-rt.com
[3]Architectural Engineering, Pennsylvania State University, USA, abm6934@psu.edu
[4]Department of Civil, Architectural, and Environmental Engineering, University of Texas at Austin, USA,
{mhuylo,atila}@utexas.edu

Abstract

The need for modeling different aspects of microgrid design and operation has seen the development of various tools over time for different analysis purposes. In this study, Modelica has been adopted as the language of choice to construct a University Campus Microgrid model, utilizing the Modelica Standard Library and the OpenIPSL library. This paper explores the advantages of utilizing Modelica for campus microgrid modeling, emphasizing its benefits and unique features. Modelica features, such as the use of record structures and replaceable templates prove to be particularly advantageous for the modeling task, enabling flexibility and efficiency in the modeling process. Furthermore, comprehensive validation tests are conducted to ensure the accuracy and reliability of sub-systems (e.g. specific power generator systems), before assembling the microgrid network model as a whole. The results demonstrate the efficacy of Modelica in accurately modeling and simulating microgrids, highlighting its potential for advancing microgrid research and development.

Keywords: Microgrid modeling, Modelica, OpenIPSL, record structures, replaceable templates

1 Introduction

1.1 Background and Related Works

The growing deployment of microgrids over the past few decades can be attributed to a multitude of factors that have collectively shaped their development. These include significant technological advancements, the growing trend towards decentralization in power generation and distribution, the increasing utilization of renewable energy sources, a heightened focus on grid resilience, and comprehensive studies on energy security. Microgrid modeling and simulation provide designers and operators the flexibility to explore diverse design options, and moreover, operational power dispatch scenarios and control strategies, without risking any disruptions in the actual operation of the physical system. This enables the identification of potential issues and facilitates the optimization of control strategies, ensuring the provision of a reliable and efficient power supply.

Over the years, numerous tools have been developed and employed specifically for the purpose of modeling microgrids. These tools serve as essential resources for a microgrid's stakeholders (e.g., operators, owners, etc.) seeking to accurately represent and analyze different aspects involved from the design, to deployment and operation of microgrid systems (Feng et al. 2018). The Distributed Energy Resource Customer Adoption Model (DER-CAM), pioneered by the Berkeley lab, integrates thermal and electrical storage sizing while optimizing equipment selection and operation to effectively reduce energy costs (Marnay et al. 2008).
The National Renewable Energy Laboratory (NREL) has developed the Hybrid Optimization Model for Multiple Energy Resources (HOMER), an optimization model for microgrids. This tool enables evaluating of various equipment options, accommodating various constraints and sensitivities (Mendes, Ioakimidis, and Ferrão 2011).
Diverging from the aforementioned tools, the Smart Grid Computational Tool (SGCT), created by the Electric Power Research Institute, takes a distinct approach to performing techno-economic analysis, providing valuable insights into the economic feasibility and advantages of smart grid implementations by considering a range of factors beyond energy balance (Xu et al. 2017).
The agent-based smart grid simulation software GridLAB-D developed by the Pacific Northwest National Laboratory offers comprehensive capabilities for simulating both the power flow of transmission networks and the performance of individual components within microgrids (Chassin, Schneider, and Gerkensmeyer 2008).

While most of these software tools focus on important microgrid design and analysis aspects, they do not provide adequate means to represent the system's dynamics, where the Modelica language excels. The primary motivation behind selecting Modelica as the modeling language for this work is its adoption by multiple software tools that

DOI
10.3384/ecp204285

support the language. This permits the use of these models across several software platforms, enhancing flexibility and enabling wider accessibility for analysis features beyond power flow and time-series simulations. In particular, Modelica tools provide user-friendly environments for effortlessly linearizing models, which is challenging with traditional power system analysis tools. Meanwhile, dynamic simulation analysis, including transient and steady-state simulations, can be performed efficiently, providing valuable insights into the system's behavior under different operating conditions. Additionally, these features of the Modelica language allow to perform stability analysis, enabling the assessment of stability margins and the identification of potential stability issues within the microgrid's power system models.

In addition, Modelica-based libraries enable the utilization of diverse models that encapsulate the behavior and interactions of systems across different domains such as electrical, mechanical, thermal, and more. As highlighted in (Winkler 2017), Modelica exhibits substantial potential as a robust power system modeling tool when utilized in conjunction with the Open-Instance Power System Library (OpenIPSL) library (Baudette et al. 2018; Castro et al. 2023). The distinct features of the Modelica language, such as the ability to create and manage record structures, prove particularly advantageous for power system models that necessitate initialization with power flow data in various power dispatch scenarios. Furthermore, the inherent replaceable model structure and object orientation of Modelica greatly facilitate the modeling of a microgrid's sub-systems and components.

This paper presents the modeling of a University Campus Microgrid utilizing the Modelica language (Fritzson and Engelson 1998; Fritzson 2014) and the OpenIPSL (Baudette et al. 2018; Castro et al. 2023) and explores the benefits of utilizing specific unique features of Modelica for microgrid modeling. Through this work, valuable insights are gained into the potential of Modelica as a versatile modeling language for microgrid modeling, that can ultimately contribute to advancements in microgrid research, design, operation, and optimization.

1.2 Motivation

On the topic of microgrid modeling utilizing the OpenIPSL library, the author's previous work (Fachini et al. 2023) aimed to demonstrate the use of Modelica as a viable modeling language for microgrid, as well as the engineering rationale on how to translate document information into power system models using OpenIPSL. This work, on the other hand, is focused on sharing the exploitation of Modelica features and modeling details of a real-world microgrid system, that albeit similar in implementation as in paper (Fachini et al. 2023), represents a different facility in Texas. The topics explored in this paper address the utilization of records for both component parameter and initial condition instantiation, implementation of reusable templates (using `replaceable`) with a structure for generation unit implementation and characterization of variants, generation unit validation through simulation comparison, and description and simulation results of the microgrid model.

1.3 Contributions

The paper's main contribution is to illustrate how Modelica-specific features can be used in the implementation of a real-world microgrid. Thus, the contributions are:

- To describe the record structures used for initialization of the dynamic microgrid model implemented utilizing the OpenIPSL Modelica library, and component parameter instantiation.
- To describe the implementation of a synchronous generator-based replaceable model that allows the user to create variants by easily configuring a generation unit model and the use of inheritance.
- Description of the validation of the generation units utilized in the microgrid model using **csv compare** (Modelica-Tools n.d.) to contrast with results from the domain-specific tool.
- Description of the implementation of the microgrid model and closed-loop system stability through root locus analysis.

1.4 Paper Structure

This paper is structured as follows: Section 2 describes the university campus microgrid model and the advantages of using Modelica for microgrid modeling including the records structure proposed to handle the power-flow variables. In Section 3, we illustrate how this data container can be linked to OpenIPSL, validate the generation units developed, and benchmark the power flow values against the results obtained with commercial tools. Section 4 presents the simulation result. Finally, Section 5 concludes the work.

2 University Campus Microgrid Modeling utilizing the OpenIPSL Modelica Library

When modeling power systems dynamics, there are two distinct mathematical representations that one can choose: Electromagnetic Transient (EMT) based models, or Phasor domain (RMS) based models. EMT-based power system models provide a three-phase waveform representation of instantaneous values for currents and voltages, used for the analysis of high-frequency events in the grid, such as power electronic switching, lightning phenomena, etc (Mahseredjian, Dinavahi, and Martinez 2009). RMS-based power system models provide a positive sequence phasor representation of currents and voltages, used to model electromechanical oscillations in the system. This modeling representation considers only the fundamental frequency of the AC voltages and currents of the real-life electrical system. The OpenIPSL library components are validated against Siemens PTI PSSE (Siemens PTI 2017), therefore

DOI 10.3384/ecp204285

the majority of the models within the library are RMS-type models.

Figure 1 displays the campus microgrid model, described in this paper, and implemented utilizing OpenIPSL components. This university campus microgrid is located in Texas and is connected to the local utility through four 69kV feeder lines as shown in Figure 1. The utility bus voltage level is reduced to 12kV through four step-down transformers, namely *T1 - T4*. The microgrid also contains a 4.16kV portion in bus *B15*, stepped down from 12kV to 4.16kV through transformers *T5*, and *T6*.

The university campus microgrid is powered by two combustion turbo generators (CTs) and four steam turbo generators (STs). The two oldest steam turbine generation units rarely operate, typically online for a few days a year in extreme load conditions. For that reason, the model in Figure 1 contains two CTs and two STs, producing power at 12kV each. The maximum amount of power that the combined generation units can produce is approximately 127MW. The CTs and STs are composed of three essential components: the synchronous machine, which converts mechanical energy into electrical energy, the prime mover, which drives the synchronous machine, and the control system, which regulates and monitors the overall operation. Together, these components work in tandem to ensure the optimal performance of the CTs and STs within the power plant. Both the *CT1* and *ST1* utilize the *GENROU* synchronous machine, *IEEEVC* as the voltage compensator, a power system stabilizer (*PSS*), and the *ESST4B* and *ESST4A* as the excitation system respectively. The *CT2* and *ST2* models share a similar internal structure, with the distinction that they lack a voltage compensator. Notably, the *CT2* and *ST2* models employ the *AC7B* excitation system. The transformers are denoted as *T1*, *T2*, etc. The power transmission lines are denoted as *X1*, *X2*, and so on. Correspondingly, the buses within the network are labeled as *B01*, *B02*, etc, and the loads are symbolized by *L01*, *L02*, and so forth. The blocks labeled *Y* in the network represent the shunt component with conductance and susceptance as its parameters. The microgrid model also displays an electrical contingency block, namely *pwFault*, used for three-phase to ground fault testing. Lastly, the *UTI* component represents the utility bulk power system, modeled as an infinite source model.

This campus microgrid has a peculiarity to its operation: it is only allowed to purchase a limited amount of power from the utility grid in emergency situations, i.e., it operates to meet the campus demand on a continuous basis. Currently, the university satisfies the campus power demand from internal generation sources, however, with the increasing variability in the price of natural gas, the university management is studying the possibility of both purchasing more power and also selling excess power when lucrative. The development of this microgrid's model in OpenIPSL, utilizing the Modelica modeling language, will open the possibility of expanding the current model for multiple purposes, including the optimization of operation and revenue of both electrical and thermal domains of the combined heat and power system. This paper will focus on the electrical domain, while future work will expand it to represent the thermal domain, i.e., heat generation and distribution.

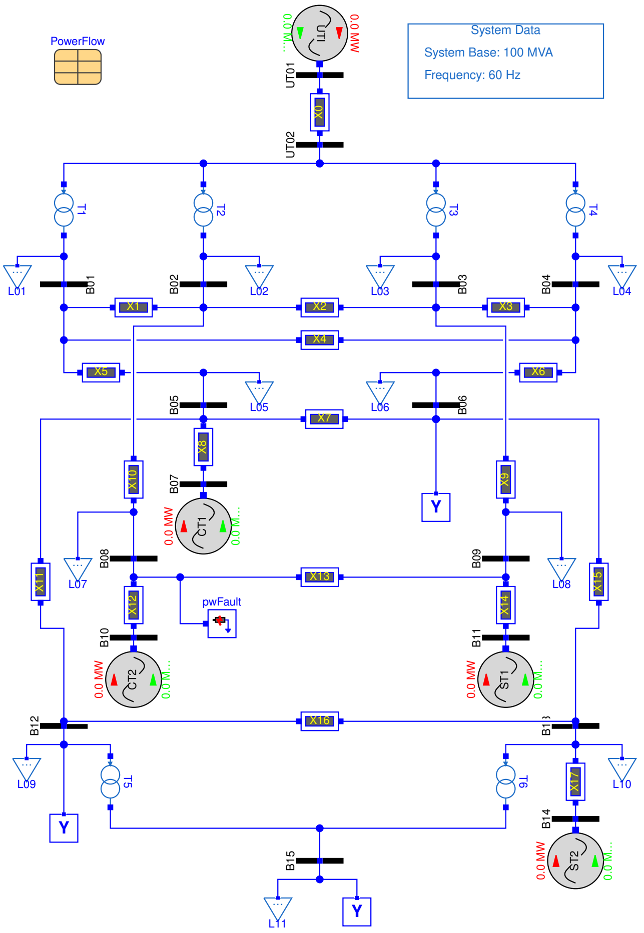


Figure 1. University Campus Microgrid "Virtual" Testbed Model.

2.1 Advantages of using Modelica for Microgrid Modeling

The Modelica modeling language was developed to provide a systematic approach to developing models of cyber-physical systems through mathematical equations (Fritzson and Engelson 1998). With that in mind, the OpenIPSL Library was implemented as a means to study power system dynamics with a modern and modular approach to power system modeling through the usage of the Modelica modeling language. The modeling paradigm used for the implementation of the OpenIPSL Library is the phasor-domain representation of power system components, meaning that the models are represented in terms of a set of differential-algebraic equations that represent the electromechanical transients in the power system (Kundur and Malik 2022). The general form of the differential-algebraic set of equations in power system studies is:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}(\mathbf{t}),\mathbf{y}(\mathbf{t}),\mathbf{t}), \\ 0 &= \mathbf{g}(\mathbf{x}(\mathbf{t}),\mathbf{y}(\mathbf{t}),\mathbf{t}), \end{aligned} \qquad (1)$$

with initial conditions, $(\mathbf{x}_0,\mathbf{y}_0,\mathbf{t})$, i.e. $\mathbf{0}=\mathbf{f}(\mathbf{x}_0,\mathbf{y}_0,\mathbf{t})$ (Kundur and Malik 2022).

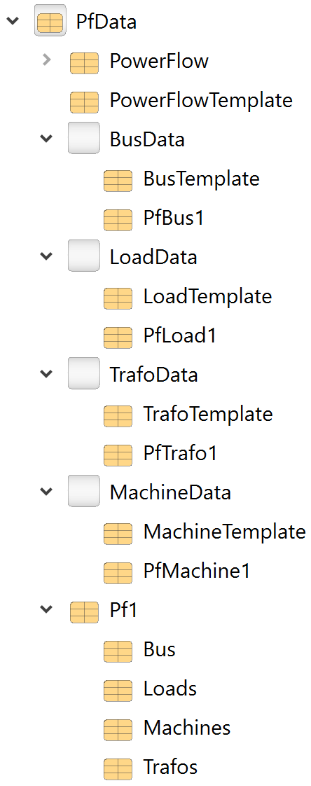
When implementing phasor-domain dynamic models, the assumption that is taken is that the balanced system is at its nominal system frequency, being it 50 or 60 Hz, depending on the system in study. This particular simplification allows representing passive system components, such as transmission lines, with algebraic equations that rely on lumped impedance values instead of a set of differential equations. Therefore, the differential equations in the phasor-domain models describe the states of generation units and their controls. The initial condition values utilized to generate different simulation starting points are derived from steady-state simulations of the electrical grid, known as power flow analysis (Powell 2004).

Unlike traditional power system tools for phasor-domain dynamic simulation, OpenIPSL was built utilizing the Modelica modeling language. With its object-oriented constructs, models can be reused and modeled conveniently, especially intricate systems containing multiple components (Fritzson and Engelson 1998; Fritzson 2014). Among several benefits of utilizing the Modelica language for power system simulation, two are of major interest and are implemented to be used in the microgrid models: (1) record structures, herein used for both system initialization and system characterization and (2) replaceable model templates, herein used for creating variants of generation units.

2.2 Initial Guess Record Structure

On the modeling front, the Modelica language utilizes its object-oriented paradigm to enable users to create models in a hierarchical manner. Similarly, one can also manage model parameter values and initial guess data through a hierarchical structure based on records. Figure 2 displays the *PfData* record structure that is used to initialize the dynamic power system model. The initial guess data necessary for the start of the dynamic simulation utilizing OpenIPSL components, which is obtained through power flow simulations, are voltage magnitude and angle of all buses, active and reactive power injection/consumption from the generation units, and the respective load profile of the microgrid.

The *PowerFlowTemplate* is a record template for the entire record structure which constrains the power flow records. The record files *BusTemplate*, *LoadTemplate*, and *MachineTemplate* define partial models that define the parameters that are extended in the data-filled record files. The record files *PfBus1*, *PfLoad1*, *PfTrafo1* and *PfMachine1* are extension records from the record templates, where the initial guess parameters from the templates are the attributed values. The *Pf1* record holds the template for all the record components. The numbering has been assigned to signify the initial record, creating room for incorporating multiple additional values for each component within their respective records across diverse power dispatch scenarios. This means that the defined record templates can have a multitude of different parameter values for multiple different initial guess values in the same system. The implementation of such a record structure can be done manually, but it can also be done automatically through an automation tool, namely the **pf2rec** Python script, that converts power flow simulation results into `*.mo` record file (Dorado-Rojas et al. 2021).



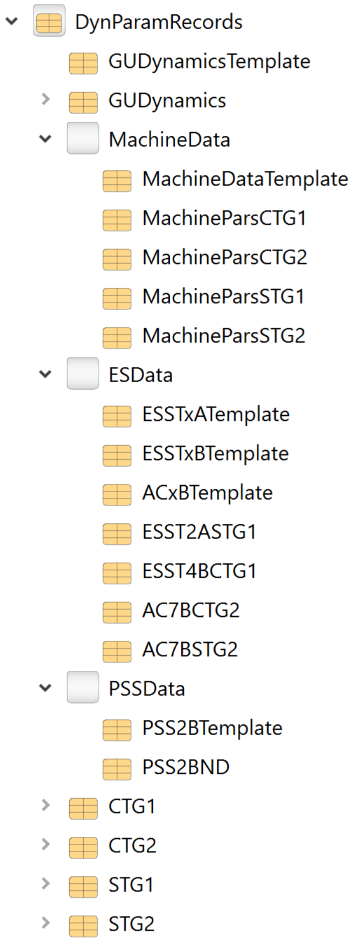
- PfData
  - PowerFlow
  - PowerFlowTemplate
  - BusData
    - BusTemplate
    - PfBus1
  - LoadData
    - LoadTemplate
    - PfLoad1
  - TrafoData
    - TrafoTemplate
    - PfTrafo1
  - MachineData
    - MachineTemplate
    - PfMachine1
  - Pf1
    - Bus
    - Loads
    - Machines
    - Trafos

Figure 2. Record Structure for Dynamic Model Initial Guess

2.3 System Characterization Record Structure

Records can also be used for model parameter setup, which is useful when the user has the intention of modifying multiple system parameters to effectively study different grid or component models, without actually re-implementing the model from the ground up. This feature empowers users to easily customize and adapt the model based on specific requirements or scenarios by manipulating the record structure. This capability not only saves time and effort but also enhances model reusability and promotes iterative model refinement.

Figure 3 displays the *DynParamRecords* record structure, which serves as a repository for parameter values related to the model equipment, including the synchronous generator (Machine), exciter (ES), and the power system stabilizer (PSS). This record structure efficiently stores and organizes the data associated with each individual component, and visually demonstrates the arrangement of the record structure, showcasing how the data for each machine or exciter model is maintained within this organized framework. *MachineDataTemplate*, *ESSTxATemplate, ESSTxBTemplate* and *ACxBTemplate* record files are also partial record models that define the parameters of the generation unit equipment. The set of record models, excluding the aforementioned ones, in the sub-packages *MachineData*, *ESData*, and *PSSData* extends their template files to attribute values to the defined component parameters. *CT1*, *CT2*, *ST1*, and *ST2* as denoted in the record structure as *CTG1*, *CTG2*, *STG1*, and *STG2* respectively, extend the *GUDynamicTemplate*, which is used as a `replaceable` in *GUDynamics*. This last record file is utilized in the models in order to (re)parametrize the components in the model.



DynParamRecords
- GUDynamicsTemplate
- GUDynamics
- MachineData
  - MachineDataTemplate
  - MachineParsCTG1
  - MachineParsCTG2
  - MachineParsSTG1
  - MachineParsSTG2
- ESData
  - ESSTxATemplate
  - ESSTxBTemplate
  - ACxBTemplate
  - ESST2ASTG1
  - ESST4BCTG1
  - AC7BCTG2
  - AC7BSTG2
- PSSData
  - PSS2BTemplate
  - PSS2BND
- CTG1
- CTG2
- STG1
- STG2

Figure 3. Data Record Structure for Generation Units

2.4 Replaceable Model Template for the Generation Units (CT and ST)

Because of the way the models are implemented in OpenIPSL, each type of generation unit component has multiple extensions to a base class. This means that it is possible to implement an “ALL-IN-ONE” generation unit model architecture that can be used to create variants that represent different generator units based on the different types of synchronous generator models, exciter models, and power system stabilizer models. For instance, Figure 4 displays an example of the *CT1* gas-turbine generation unit.

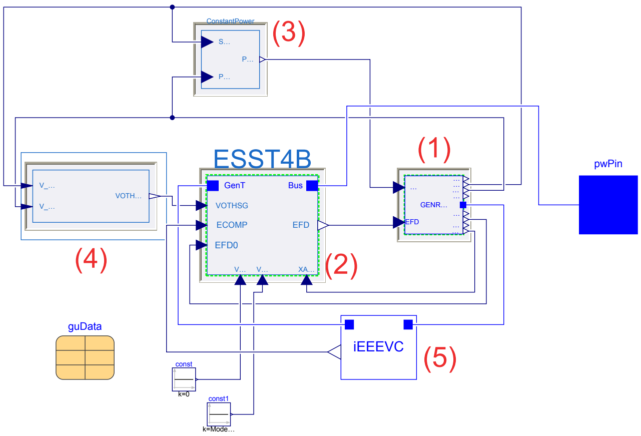
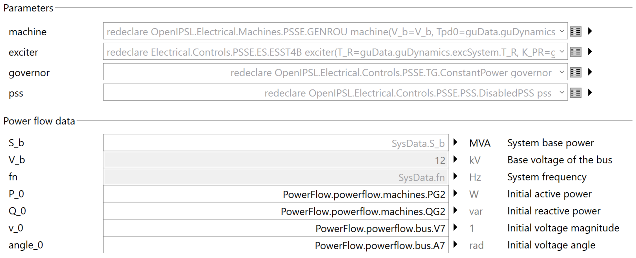


Figure 4. Replaceable Generation Unit Model Example

The sunken gray components are replaceable models for the synchronous generator (1), exciter (2), PSS (4), and turbine-governor model (3). Component (5) from Figure4 is a voltage compensation block. For the purpose of this work, the turbine-governor model is defined as a constant mechanical power to the generator, because of the lack of information on the turbine during the model implementation phase, however, this will be expanded in future work. Inspecting the generation unit component in the microgrid model, shown in Figure 5, one can observe the result of enabling the use of the records for parameter definition and model initialization (i.e. the use of the power flow data as an initial guess for the initialization problem).



Parameters

machine	redeclare OpenIPSL.Electrical.Machines.PSSE.GENROU machine(V_b=V_b, Tpd0=guData.guDynamics
exciter	redeclare Electrical.Controls.PSSE.ES.ESST4B exciter(T_R=guData.guDynamics.excSystem.T_R, K_PR=g
governor	redeclare OpenIPSL.Electrical.Controls.PSSE.TG.ConstantPower governor
pss	redeclare OpenIPSL.Electrical.Controls.PSSE.PSS.DisabledPSS pss

Power flow data

S_b	SysData.S_b	MVA	System base power
V_b	12	kV	Base voltage of the bus
fn	SysData.fn	Hz	System frequency
P_0	PowerFlow.powerflow.machines.PG2	W	Initial active power
Q_0	PowerFlow.powerflow.machines.QG2	var	Initial reactive power
v_0	PowerFlow.powerflow.bus.V7	1	Initial voltage magnitude
angle_0	PowerFlow.powerflow.bus.A7	rad	Initial voltage angle

Figure 5. Parameters and Power Flow Data from the CT1 Model

The image displays the possibility of the user selecting different component models for the replaceable *machine*, *exciter*, *governor*, and *pss* components. On the Power flow data section, the user is able to select initialization values for *P_0, Q_0, v_0,* and *angle_0* based on values from the *PowerFlow* record file shown in Figure 2.

3 Validation System Models for the Generation Units

Before assembling the entire microgrid model, this work proposes the development of validation models that allows checking if the generator unit models have been characterized correctly by reutilizing a partial model from the OpenIPSL library that is used for unit testing of the library. This helps in minimizing the complexity of debugging when assembling the microgrid system model, as the most crucial dynamic sub-systems have been verified in this step. To this end, the modeling and verification process is discussed next.

3.1 Generators

The campus microgrid power plant utilizes two gas-fired combustion turbo-generators (*CT1* and *CT2*) that produce steam at high pressure. This steam is then directed towards the two steam turbo-generators (*ST1* and *ST2*) in order to generate electricity. This section provides an in-depth explanation of the *CT* model, shedding light on the internal structure and components of the model.

Figure 4 provides a visual representation of the *CT1* model, offering a diagram view that highlights its key components. The central element is the *GENROU* synchronous machine, which is driven by a constant power component. The *IEEEVC* component acts as a voltage compensator, supplying the necessary voltage to the exciter, represented by the *ESST4B* component. The output of the exciter model i.e. the excitation voltage *EFD* is fed to the synchronous machine. Meanwhile, the power system stabilizer (*PSS*) continuously monitors the generator's shaft speed and electrical power output. To interface with the wider grid, the *pwPin* serves as the connection point between the generation unit and the rest of the system.

The *CT2* model, depicted in Figure 6, follows a similar approach, with the components following a similar numbering convention as the one in Figure 4. However, there are distinct differences, specifically in the exciter model, which is *AC7B* in this case. Additionally, the absence of the voltage compensator sets it apart from the previous model. The *guData* component within the model is used to parameterize the model components, including the machine, exciter, and power system stabilizer. By selecting the data record associated with *CT2*, the component parameters are automatically populated from the record structure, as depicted in Figure 3. This approach streamlines the process of configuring the model components with the appropriate values, enhancing efficiency and ease of use.

3.2 Single Machine Infinite Bus Test System

To ensure dynamic modeling accuracy, the generation units consisting of the two combustion turbo-generators (CTs) and the two steam turbo-generators (STs) undergo a validation process through the creation of individual test models. The single-machine infinite bus test system model is implemented utilizing the Modelica modeling language with the OpenIPSL Library components, simulated using Dymola (Brück et al. 2002), and in the Siemens PTI PSSE tool. The simulation results from both tools are then compared via **csv compare** tool. This comparison algorithm fits a tolerance tube around the data set, where the tube is linked to a tolerance value set by the user, and the simulation result values are then checked to see if they lie within the tube. In order to validate the Modelica-based model against the reference Siemens PTI PSSE model, a fault event is generated in order to evaluate the dynamic response of the generation unit and its controls.

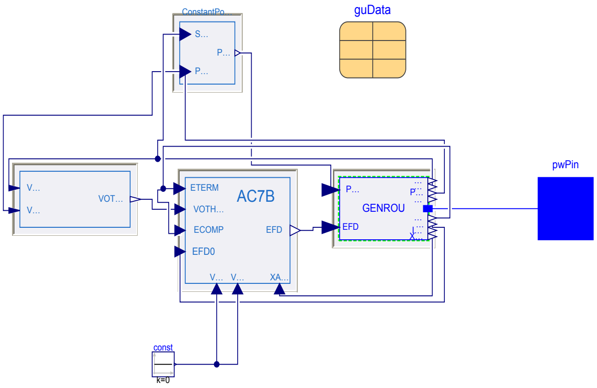


Figure 6. Combustion Turbo-Generator (CT2) Model

Figure 7 depicts the test model for the *CT2*, in which the *pwFAULT* component from the OpenIPSL library is utilized to simulate a three-phase electrical fault contingency. This fault is applied to the designated *FAULT* bus, specifically from 2 to 2.15 seconds. The introduction of this fault condition results in a reduction in the voltage magnitude at the specified bus, thereby reflecting the ensuing impact on the system dynamics. Figure 8(a) and Figure 8(b) depict the active, and reactive power injection from the generation unit being validated (*CT2*).

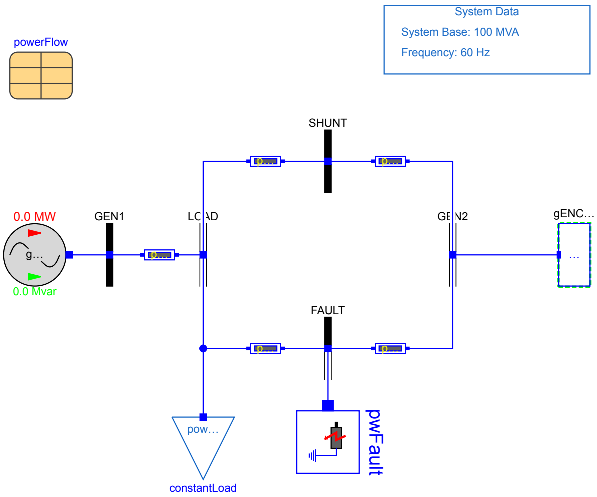
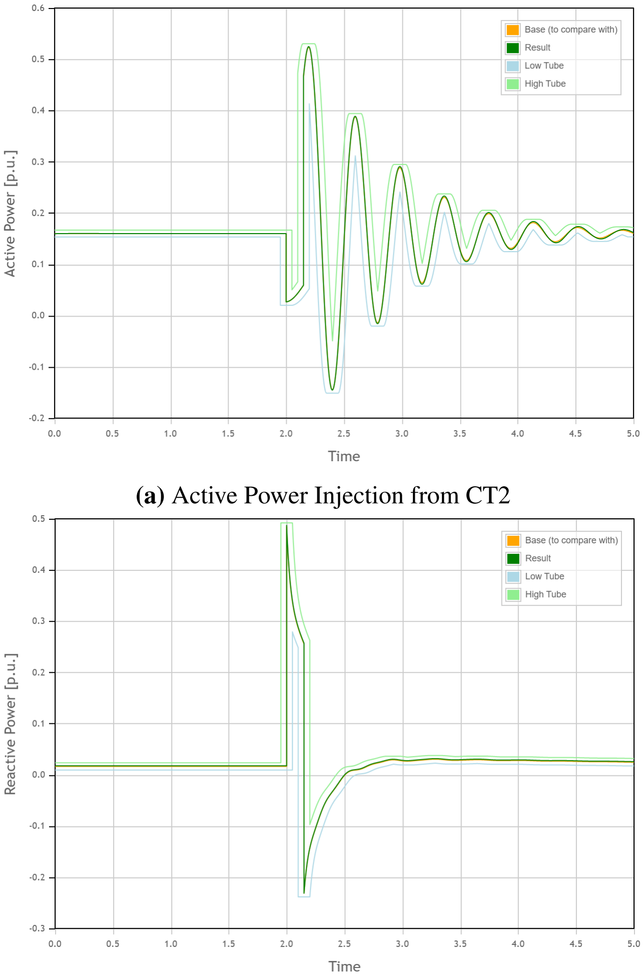


Figure 7. Single Machine Infinite Bus Validation Test Model with CT2

The orange curve is the simulation plot from Siemens PTI PSSE (Base), while the dark green curve is the simulation plot from Dymola (Result). The blue and light green curves are the lower and upper values of the tube mentioned earlier in the subsection, with a tolerance value chosen to be 0.01. From the comparison result, one can observe that the simulation in Dymola is practically identical to the simulation results from Siemens PTI PSSE and that both curves are inside the tolerance tube. This means that the simulation passed the validation process and it is adequate to be used in the microgrid model. The same validation process is conducted in all the generation unit models implemented in this work, namely the two gas turbines and the two steam turbine units.



(a) Active Power Injection from CT2

(b) Reactive Power Injection from CT2

Figure 8. Active power injection comparison simulation results utilizing **csv compare** algorithm Figure 8(a), and reactive power injection comparison simulation results utilizing **csv compare** Figure 8(b).

4 Simulation Results

Paper (Fachini et al. 2023) discussed two simulation results for a similar microgrid model of another university campus in Colorado: an electrical contingency study in the microgrid, and an eigenvalue analysis of the microgrid. To expand on available Modelica capabilities in this work, using the newly developed model for a real-world microgrid in Texas, the authors explore the stability of the microgrid system when varying the proportional gain K_p from the exciter proportional-integrator (PI) controller in *CT1*. It is worth mentioning that the addressed PI controller takes in as input the error signal between a voltage reference and the terminal voltage.

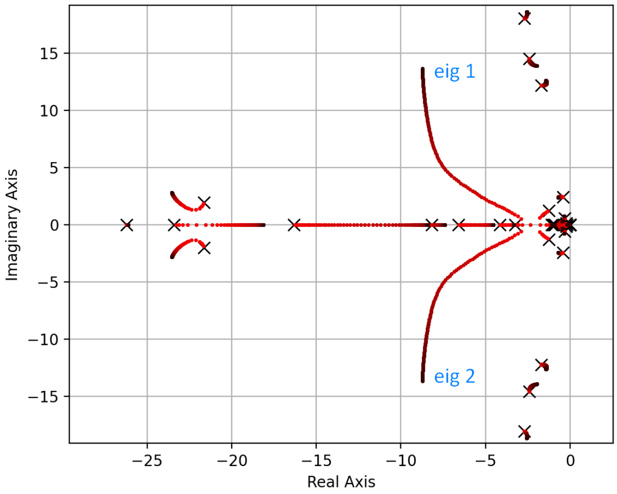


Figure 9. Root Locus Result for Increasing Values of K_p in the CT1 Generation Unit

The equilibrium solution to the system when increasing the proportional gain in increments can be used to perform a root-locus analysis, as an exploration of the system's eigenvalues with changing proportional gain values. Figure 9 displays the root-locus analysis done in the microgrid model from Figure 1, where K_p of the exciter from *CT1* generation unit is systematically for the range $[1,500]$.

The X markers in Figure 9 define the eigenvalues from the first root locus simulation for $K_p = 1$. The subsequent iterations of the root locus plot are displayed in a gradient color spectrum, with vibrant red being values near the minimum of the K_p range and dark maroon being values near the maximum of the K_p range. The root locus plot displays a complex eigenvalues pair, labeled *eig 1*, and *eig 2*, which are associated with the exciter's voltage. As K_p increases, the complex eigenvalue pair present a reduction in the real components, which implies an increase of dampening.

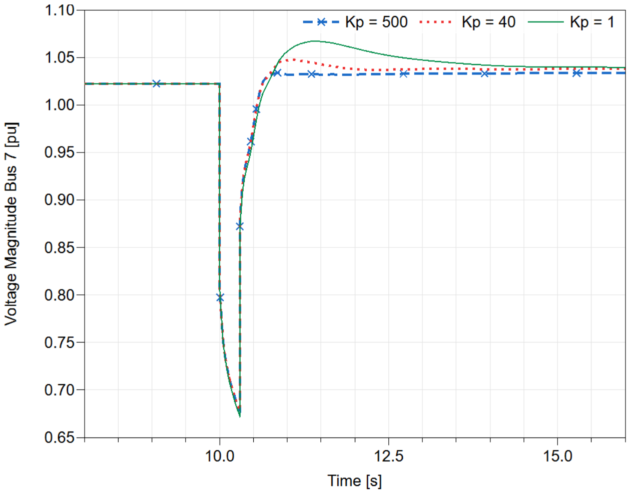


Figure 10. Time domain-simulation of Voltage in Bus 7 for different K_p values

This is verified in the plot shown in Figure 10, where a fault and a consequent line trip are applied to the system in Figure 1 to display the oscillatory behavior of the voltage magnitude at Bus 7. The voltage magnitude at Bus 7 for three different values for the proportional gain in the exciter: $K_p = 1$, $K_p = 40$, and $K_p = 500$, are shown. As expected, increasing the value of the gain results in a more aggressive dampening of the voltage magnitude oscillations, all due to a reduction in the real component the complex eigenvalue pair *eig 1*, and *eig 2*.

5 Conclusions

In this work, Modelica and the OpenIPSL library have been utilized to model a real-world university campus microgrid. The implementation of Modelica allowed leveraging its unique features, such as the record structures and replaceable templates, to effectively design and parameterize the generation units of the microgrid. To ensure reliable performance, each generation unit underwent a separate validation test before being integrated into the main grid. By adopting this approach, the advantages of utilizing Modelica in the development and validation of microgrid models have been demonstrated, bringing added value with multiple potential uses, such as linear-model-based analysis, which is challenging with domain-specific tools.

The study has been based on the documentation from the plant engineers, but unfortunately, there were not enough documents on the turbine governor (*TG*). Future work includes the TG and thermo-fluidic system used for the optimization of the heat-and-power model.

Acknowledgements

This paper is in part, based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office, Award Number DE-EE0009139, and in part by the National Science Foundation Award No. 2231677.

References

Baudette, Maxime et al. (2018). "OpenIPSL: Open-instance power system library—update 1.5 to "iTesla power systems library (iPSL): A modelica library for phasor time-domain simulations"". In: *SoftwareX* 7, pp. 34–36.

Brück, Dag et al. (2002). "Dymola for multi-engineering modeling and simulation". In: *Proceedings of modelica*. Vol. 2002. Citeseer.

Castro, Marcelo de et al. (2023). "Version [OpenIPSL 2.0. 0]-[iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations]". In: *SoftwareX* 21, p. 101277.

Chassin, David P, Kevin Schneider, and Clint Gerkensmeyer (2008). "GridLAB-D: An open-source power systems modeling and simulation environment". In: *2008 IEEE/PES Transmission and Distribution Conference and Exposition*. IEEE, pp. 1–5.

Dorado-Rojas, Sergio A et al. (2021). "Power flow record structures to initialize openipsl phasor time-domain simulations with python". In: *Modelica Conferences*, pp. 147–154.

Fachini, Fernando et al. (2023). "Developing a Campus Microgrid Model utilizing Modelica and the OpenIPSL Library". In: *2023 11th Workshop on Modelling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE, pp. 1–6.

Feng, Wei et al. (2018). "A review of microgrid development in the United States–A decade of progress on policies, demonstrations, controls, and software tools". In: *Applied energy* 228, pp. 1656–1668.

Fritzson, Peter (2014). *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons.

Fritzson, Peter and Vadim Engelson (1998). "Modelica-a unified object-oriented language for system modeling and simulation". In: *ECOOP*. Vol. 98. Citeseer, pp. 67–90.

Kundur, Prabha S and Om P Malik (2022). *Power system stability and control*. McGraw-Hill Education.

Mahseredjian, Jean, Venkata Dinavahi, and Juan A Martinez (2009). "Simulation tools for electromagnetic transients in power systems: Overview and challenges". In: *IEEE Transactions on Power Delivery* 24.3, pp. 1657–1669.

Marnay, Chris et al. (2008). "Optimal technology selection and operation of commercial-building microgrids". In: *IEEE Transactions on Power Systems* 23.3, pp. 975–982.

Mendes, Gonçalo, Christos Ioakimidis, and Paulo Ferrão (2011). "On the planning and analysis of Integrated Community Energy Systems: A review and survey of available tools". In: *Renewable and Sustainable Energy Reviews* 15.9, pp. 4836–4854.

Modelica-Tools (n.d.). *Modelica-Tools/CSV-compare: Tool to compare curves from one CSV files with curves from other CSV files using an adjustable tolerance*. URL: https://github.com/modelica-tools/csv-compare.

Powell, Lynn (2004). *Power system load flow analysis*. McGraw Hill professional.

Siemens PTI (2017). "PSS®E 34.2.0 model library". In: *Siemens Power Technologies International, Schenectady, NY*.

Winkler, Dietmar (2017). "Electrical power system modelling in modelica–comparing open-source library options". In: *Proceedings of the 58th Conference on Simulation and Modelling (SIMS 58) Reykjavik, Iceland, September 25th–27th, 2017*. 138. Linköping University Electronic Press, pp. 263–270.

Xu, Liu et al. (2017). "A review of the ARRA smart grid projects and their implications for China". In: *January. LBNL-1007122*.

Application of the OpenModelica-MATLAB Interface to Integrated Simulation and Successive Linearization Based Model Predictive Control

Mohammad Hadi Alizadeh[1] Ali M. Sahlodin[1] Arunkumar Palanisamy[2] Francesco Casella[3]
Peter Fritzson[2]

[1]Process Systems Engineering Laboratory, Department of Chemical Engineering, Amirkabir University of Technology (Tehran Polytechnic), Iran, {m.hadi,sahlodin}@aut.ac.ir
[2]Department of Computer and Information Science (IDA), Linköping University, Sweden,
{arunkumar.palanisamy,peter.fritzson}@liu.se
[3]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy,
francesco.casella@polimi.it

Abstract

This paper presents the implementation of successive linearization based model predictive control (SLMPC) efforts through the interfacing of OpenModelica and MATLAB using the OMMatlab tool. The dynamic system (here a chemical process) and the model predictive control (MPC) algorithm are implemented in OpenModelica and MATLAB, respectively. The model linearization procedure is carried out through OMMatlab, which is highly optimized in terms of run-time by using a single executable file and adapting it at each sample time. Also, necessary theories for a continuous model discretization are discussed for both nonlinear Modelica and linearized continuous models. A procedure for constructing an Extended Kalman Filter (EKF) from a continuous Modelica model is also presented. The usability of the OpenModelica-MATLAB interface for SLMPC is demonstrated by control of liquid levels in a tanks-in-series problem.

Keywords: Model predictive control, OpenModelica, OMMatlab, Extended Kalman filter.

1 Introduction

In the recent years, the Modedica language has been widely used for modeling of systems described by differential algebraic equations (DAEs). The object-oriented nature of Modedica facilitates modular model construction and the use of powerful solvers as provided by commercial and open source Modedica-based simulators (e.g., Dymola, OpenModelica (Fritzson et al. 2020), Jmodelica). Some integrated features such as Optimica (Åkesson 2008) and CasAdi are also supplied for solving dynamic optimization problems. The reader is referred to (Ruge et al. 2014; Magnusson and Åkesson 2015) for a thorough discussion of these topics. Moreover, model predictive control (MPC) problems have been implemented in OpenModelica; see e.g., (Bachmann et al. 2012) for MPC implementation of a batch reactor.

Conventional nonlinear model predictive control (NMPC) works by predicting the future behavior of a system and solving a dynamic optimization problem for minimizing a performance index, e.g., tracking error or operational cost; see, for example, (Ellis, J. Liu, and Christofides 2017; Heidarinejad, J. Liu, and Christofides 2013). Although using a rigorous nonlinear model increases the prediction accuracy, it leads to high computational expense (Zhakatayev et al. 2017) and potential convergence issues in the optimization routine. Successive linearization MPC (SLMPC) is an efficient alternative to NMPC (Seki, Ooyama, and Ogawa 2002; Cannon, Ng, and Kouvaritakis 2009; Cortinovis et al. 2014; C. Liu et al. 2015), especially in large-scale applications. In this method, the nonlinear model of the system is linearized successively at each sample time, and the prediction model is updated over time to preserve the prediction accuracy (Vrlić, Ritzberger, and Jakubek 2020). As a result of using a linear model, the dynamic optimization can be performed faster, favoring SLMPC over NMPC for large-scale systems in terms of computational expense, although the prediction accuracy may be undermined to some extent.

Many dynamic optimization or MPC efforts using Modelica models are reported in the literature. Franke (2002) employed Modelica to study optimal startup of a power plant, in which Dymola was used for generating S-Functions that would be imported into Simulink. Gräber et al. (2012) proposed a framework based on functional mockup interface generated from Modelica models for implementing NMPC on a vapor compression cycle. Also, L. Imsland, P. Kittilsen, and T. Schei (2010) integrated Dymola models into commercial software designed for NMPC and did a case study with an offshore oil and gas processing plant. Pandey et al. (2021a) employed OMJulia (Lie et al. 2019) for stochastic MPC of solar and hydropower plants described by a set of small-scale DAEs. Pandey et al. (2021b) implemented MPC for a power grid

DOI
10.3384/ecp204293

system, where a small-scale model was implemented in OpenModelica as the actual plant, and a separate model was developed in Julia for use in the control algorithm that included an unscented Kalman filter for the state estimation. OMJulia was used for interfacing OpenModelica with Julia. Also, when it comes to optimal control problems that require a linearized model, Jmodelica can offer a tool for linearizing the nonlinear model of the plant. Jmodelica supplies the interfacing capability with Python, enabling it to integrate Jmodelica simulation, optimization, and linearization features and design MPC problems that need linearization. It is possible to interact with the Modelica models and functional mockup units through IPython, a command shell for the programming environment (Andersson et al. 2018). The interested readers are referred to, e.g., ((Romero, Goldar, and Garone 2019; Pippia et al. 2021; Lars Imsland, Pål Kittilsen, and T. S. Schei 2009; Jorissen, Boydens, and Helsen 2019)) for other dynamic optimization studies involving Modelica models.

The use of engineering software such as MATLAB for Modelica models offers easy implementation for algorithm prototyping. However, a challenge in the application of MPC for Modelica models is the computational cost of interfacing the Modelica software with engineering software such as MATLAB. This is particularly true in case of SLMPC, where repeated model linearization through Jacobian calculations is required. The computational cost can grow quickly for large-scale models, making it prohibitively expensive to apply SLMPC or similar algorithms through interfacing of Modelica models with MATLAB or other engineering software. Therefore, the procedure needs to be carried out decently ensuring that the Jacobian calculations are undertaken in a time-efficient manner.

In this paper, efficient implementation of SLMPC for Modelica models in OpenModelica is addressed. In particular, the dynamic model of the system (serving as the virtual plant) is implemented in OpenModelica. Also, OMMatlab (*OpenModelica* 2021) is used to take control of the model simulation and make an interconnection between MATLAB and the OpenModelica model, which is transformed into an executable file. Moreover, the OMMatlab linearization method is enhanced to operate considerably faster than its older version, making the new version more favorable for large-scale SLMPC.

These enhancements can also be implemented for other OpenModelica interfaces such as OMOctave, OMPython, and OMJulia so they can support efficient prototyping of advanced control algorithms requiring successive linearization.

The rest of the paper is structured as follows. In section 2, the general structure of the SLMPC problem is briefly introduced. In section 3, the method of assembling the discrete system model from its continuous form is presented. Section 4 discusses the details of the linear prediction model, the pertinent theories, and the enhancements made to OMMatlab to boost the linearization process. The state estimator design algorithm is discussed in section 5. In section 6, the dynamic optimization problem solved at each sample time is formulated. A case study of the SLMPC implementation is demonstrated in section 7. Finally, the paper is concluded in section 8.

2 Overview of Problem Blocks

As depicted in Figure 1, the problem under study includes an often nonlinear model that represents the actual physical system (here a chemical plant). Some of the plant variables are measured at every sample time. Since measuring all the variables is impractical, a state estimator is used to estimate the unmeasured state variables. These values are used to initialize the prediction model, which is a linearized form of the nonlinear model updated at each sample time. It is also possible to increase the frequency of prediction model updates by regenerating the linear model at each step over the prediction horizon instead of updating it only at the beginning of each sample time. The optimal control inputs are calculated over the prediction horizon by minimizing the objective function (e.g., the cumulative tracking error). Then, the first element of the optimal input trajectory is passed to the plant, and the procedure is repeated in the next sample time. The building blocks of the SLMPC are detailed in the following subsections.

3 System Model

The plant model is represented in a continuous, nonlinear, time-invariant state-space form as

$$\dot{x} = f(x,u) \tag{1}$$

$$y = g(x,u) \tag{2}$$

where $x \in \mathbb{R}^{n_x}$ is the set of n_x system states, $u \in \mathbb{R}^{n_u}$ is the vector of n_u control inputs, $y \in \mathbb{R}^{n_y}$ is the vector of n_y system outputs, $f = [f_1, f_2, f_3, ..., f_{n_x}]$ and $g = [g_1, g_2, g_3, ..., g_{n_y}]$ are the sets of equations describing the state evolution and outputs of the system, respectively. OpenModelica automatically generates these functions from a high-level, object-oriented, equation-based model description, and can simulate the nonlinear model and generate an executable file that will be used for the state estimator and prediction model. The existing features of OMMatlab allow the user to take control of the executable file and overwrite the simulation setup and model parameters from MATLAB.

Equation 1 is converted to a discrete state-space form with additive white Gaussian noise for process states and measurements. This is done by the methodology presented in (Brembeck 2019; Brembeck, Otter, and Zimmer 2011), the implementation of which is adapted to the MATLAB-OpenModelica environment. The discrete form reads

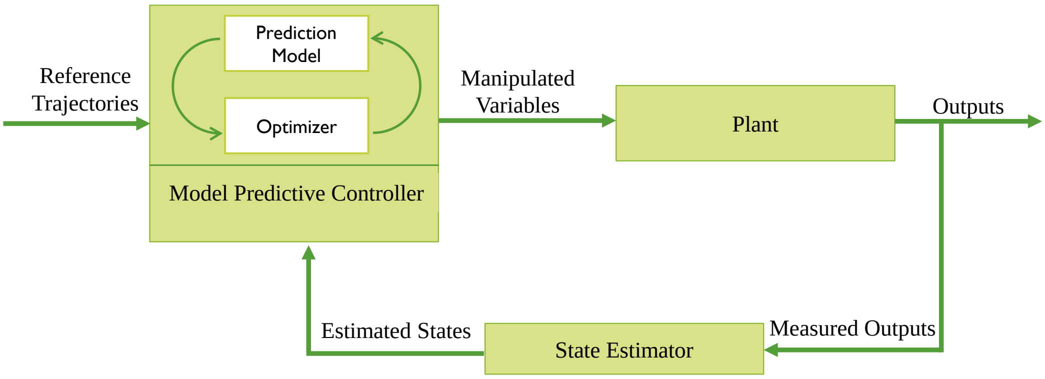


Figure 1. A general schematic of the MPC blocks.

$$x_k = f_{k|k-1} + w_{k-1} \tag{3}$$

$$f_{k|k-1} = x_{k-1} + \int_{t_{k-1}}^{t_k} f(x,u)\,dt \tag{4}$$

$$y_k = g(x_k, u_k) + v_k \tag{5}$$

$$E[w_k] = 0 \tag{6}$$

$$E[v_k] = 0 \tag{7}$$

$$E[w_k w_k^T] = Q_k \tag{8}$$

$$E[v_k v_k^T] = R_k, \tag{9}$$

where w_k and v_k are additive process and measurement noises, respectively; and Q_k and R_k denote the process and measurement noise covariance matrices, respectively.

The LHS of Equation 4 is computed by integration of the system over one sample time. In the continuous model implemented in OpenModelica, initial state values (defined in the `initial equation` section) are set parametrically so that the initial values can be overwritten and Equation 4 can be evaluated from MATLAB at each sample time. This is illustrated in Listing 1 for an arbitrary continuous Modelica model.

Listing 1. Continuous system model in Modelica

```
model List1
...
parameter Real x1_In;
parameter Real x2_In;
...
input Real u1;
input Real u2;
...
output Real y1;
output Real y2;
...
Real x1;
Real x2;
...
initial equation
x1=x1_In;
x2=x2_In;
...
equation
...
end List1;
```

The initial conditions x_{k-1} could be set from MATLAB by the `setParameters()` method of OMMatlab. Similarly, the system inputs u_{k-1} are specified by the `setInputs()` method. Finally, the integration in Equation 4 is performed over one sample time using built-in OpenModelica solvers such as DASSL and IDA.

4 Prediction Model

The system linearization required for the prediction model and its implementation are described in this section.

4.1 System linearization

Equation 1 and Equation 2 can be linearized around an arbitrary operating point (x_{op}, u_{op}) by using the first-order Taylor expansion as follows.

$$\dot{x} = f(x_{op}, u_{op}) + A_c(x - x_{op}) + B_c(u - u_{op}) \tag{10}$$

$$y = g(x_{op}, u_{op}) + C_c(x - x_{op}) + D_c(u - u_{op}), \tag{11}$$

where

$$A_c = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_{n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{n_x}(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_{n_x}(\mathbf{x})}{\partial x_{n_x}} \end{bmatrix} \tag{12}$$

Similarly, $B_c = \frac{\partial f}{\partial u} =$, $C_c = \frac{\partial g}{\partial x}$, and $D_c = \frac{\partial g}{\partial u}$. Considering Equation 1 and Equation 2, the bias values are

$$f(x_{op}, u_{op}) = \left.\frac{dx}{dt}\right|_{(x,u)=(x_{op},u_{op})} \tag{13}$$

$$g(x_{op}, u_{op}) = y|_{(x,u)=(x_{op},u_{op})} \tag{14}$$

Therefore, it is possible to construct continuous linearized models by having A_c, B_c, C_c, D_c, and the bias values. OMMatlab can provide all these matrices along with ($\dot{x}$, y) at any specific simulation time by the `linearize()`

and `getSolutions()` methods. Equation 10 and Equation 11 can then be expressed as

$$\dot{x} = A_c x + B_c u + \mathbb{K}_{xc} \tag{15}$$
$$y = C_c x + D_c u + \mathbb{K}_{yc}, \tag{16}$$

where $\mathbb{K}_{xc} = (f(x_{op}, u_{op}) - A_c x_{op} - B_c u_{op})$, and $\mathbb{K}_{yc} = (g(x_{op}, u_{op}) - C_c x_{op} - D_c u_{op})$. The continuous linearized model can be discretized for a given sample time T_s as (see (Zhakatayev et al. 2017; Vrlić, Ritzberger, and Jakubek 2020) for a detailed proof).

$$x_{k+1} = A_d x_k + B_d u_k + \mathbb{K}_{xd} \tag{17}$$
$$y_k = C_d x_k + D_d u_k + \mathbb{K}_{yd}, \tag{18}$$

in which

$$A_d = e^{A_c T_s} \tag{19}$$
$$B_d = A_c^{-1}(e^{A_c T_s} - I) B_c \tag{20}$$
$$\mathbb{K}_{xd} = A_c^{-1}(e^{A_c T_s} - I)\mathbb{K}_{xc} \tag{21}$$

and $C_d = C_c$, $D_d = D_c$, and $\mathbb{K}_{yd} = \mathbb{K}_{yc}$.

4.2 Improving OMMatlab for successive linearization

Repeated linearization of a nonlinear dynamic model was computationally inefficient in the previous OMMatlab distributions. For example, a single invocation of the `linearize()` method for a system of DAEs with about 1100 equations took around 420 seconds on an Intel Core i5 7200U CPU. This would make implementation of SLMPC impractical as the linearization task should be performed at every sample time. The leading cause for this inefficiency was that the OpenModelica model had to be recompiled each time the `linearize()` command was called. The resulting linearized model had to be rebuilt to create an XML file so that the values of matrices could be extracted into MATLAB.

To solve this problem, OMMatlab is edited by the authors so that the initial executable file can be adapted and used in all invocations without being recompiled. Moreover, instead of using the time-consuming perturbation and finite differences for Jacobian approximation, the built-in automatic differentiation algorithm (invoked by the `-generateSymbolicLinearization` flag) is employed for exact linearization and construction of the model matrices (Braun, Ochel, and Bachmann 2011). The generated linearized model is populated in a `.m` file, from which the matrix values are read. With these improvements, a single invocation of the `linearize()` command for the same DAE system now takes only a fraction of a second, which is a remarkable computational enhancement. This improvement can benefit any application requiring nonlinear Modelica model linearization through MATLAB, including SLMPC.

5 State Estimator

The extended Kalman filter (EKF) is used for state estimation. An EKF for Modelica models can be implemented in MATLAB based on the approach presented in (Brembeck 2019). The EKF uses linearization for state and measurement covariance propagation. The prediction and correction steps of the EKF for Equation 3 to Equation 5 proceed as follows.

- Prediction:

$$\hat{x}_k^- = f_{k|k-1}(x_{k-1}^+, u_{k-1}) \tag{22}$$
$$F_{k-1} = exp(\frac{\partial f}{\partial x}|_{x_{k-1}^+}.T_s) \tag{23}$$
$$P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + Q \tag{24}$$

- Correction:

$$G_k = \frac{\partial g}{\partial x}|_{x_k^-} \tag{25}$$
$$K_k = P_k^- G_k^T.(G_k P_k^- G_k^T + R)^{-1} \tag{26}$$
$$\hat{x}_k^+ = x_k^- + K_k(y_k^m - g(x_k^-)) \tag{27}$$
$$P_k^+ = (I - K_k.G_k).P_k^-, \tag{28}$$

where the $-$ and $+$ superscripts respectively denote the predicted and corrected values. The EKF is initialized by setting $\hat{x}_0^+$ and P_0^+ to arbitrary or approximate values. It should be noted that the required Jacobian values F_{k-1} and G_k can be calculated easily by a call to the `linearize()` method through OMMatlab.

6 Optimization Problem

MPC solves the following general optimization problem at the k-th sample time.

$$\min_{r \in U} \quad J_k(X, r) \tag{29}$$
$$\text{s.t.} \quad X_{k+i} = h(X_{k+i-1}, r_{k+i-1}), \quad i = 1, \ldots, N_{H_p} \tag{30}$$
$$M(X, r) \leq 0 \tag{31}$$
$$S(X, r) = 0 \tag{32}$$
$$X_k = \hat{x}_k^+, \tag{33}$$

where J_k is the objective function optimized over the prediction horizon H_p. Note that $N_{H_p} = \frac{H_p}{T_s}$. The decision variable set r is the trajectory of control inputs. Also, in order to reduce the computational cost, the control horizon is set as $H_c < H_p$, and $r_{k+N_{H_c}} = r_{k+N_{H_c}+1} = \cdots = r_{k+N_{H_p}-1}$, with $N_{H_c} = \frac{H_c}{T_s}$. The function h describes the discrete state evolution that is used for prediction. Also, M and S are the inequality and equality constraints, respectively, and X_k is the estimated current state vector used to initialize the prediction model.

The MPC objective can be a tracking index, an economic index, or a combination of the two; see e.g., (Heidarinejad, J. Liu, and Christofides 2013; Ellis, J. Liu,

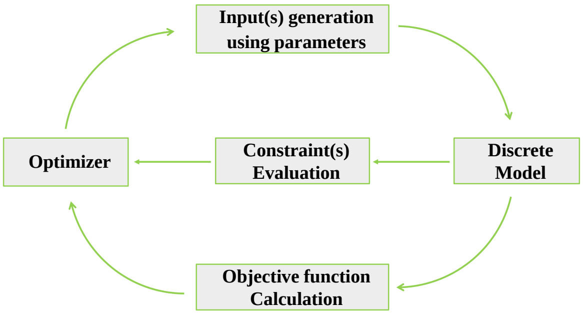

Input(s) generation using parameters
Optimizer
Constraint(s) Evaluation
Discrete Model
Objective function Calculation

Figure 2. Sequential dynamic optimization method.

and Christofides 2017). In this work, the dynamic optimization problem is solved using the sequential method (Chachuat 2009), where the control input trajectories are parameterized (usually in a piece-wise constant manner), and the dynamic model and the nonlinear optimization problem are solved in sequence as shown in Figure 2. This work uses the active-set optimization algorithm to solve the nonlinear optimization problem.

7 Case Study

A tracking SLMPC is implemented for a system of three interconnected cylindrical tanks as depicted in Figure 3. The flow rates of the three inlet streams are considered the control inputs. The outlet flow rates of the tanks are the measured outputs, and the tanks' liquid levels are regarded as the system states being estimated by the EKF. The tanks are empty at the initial time, and the control scenario is to move the tank levels to a specific set point by adjusting the inlet flow rates. The continuous dynamic model describing the system is as follows.

$$\dot{h_1} = \frac{q_{in1} - q_{o1}}{A_1} \tag{34}$$

$$\dot{h_2} = \frac{q_{o1} + q_{in2} - q_{o2}}{A_2} \tag{35}$$

$$\dot{h_3} = \frac{q_{o2} + q_{in3} - q_{o3}}{A_3} \tag{36}$$

$$q_{o1} = C_v\sqrt{h_1 - h_2} \tag{37}$$

$$q_{o2} = C_v\sqrt{h_2 - h_3} \tag{38}$$

$$q_{o3} = C_v\sqrt{h_3} \tag{39}$$

where A_1, A_2, and A_3 are the vessel cross-section areas, C_v is the valve coefficient; q_{in1}, q_{in2}, and q_{in3} are the volumetric flow rates of the inlet streams, and q_{o1}, q_{o2}, and q_{o3} are the volumetric flow rates of the outlet streams. In case of reverse flow through the valves, the square root term in the valve equations becomes negative, leading to a complex number and solver failure. To avoid this situation and preserve local Lipschitz continuity, the valve equations are regularized as (Barton, Banga, and Galán 2000; Sahlodin 2022; Casella 1998)

$$q_{o1} = C_v \frac{h_1 - h_2}{\sqrt{|h_1 - h_2| + \varepsilon}} \tag{40}$$

$$q_{o2} = C_v \frac{h_2 - h_3}{\sqrt{|h_2 - h_3| + \varepsilon}} \tag{41}$$

$$q_{o3} = C_v \frac{h_3}{\sqrt{|h_3| + \varepsilon}}, \tag{42}$$

where $\varepsilon > 0$. Let $A_1 = A_2 = A_3 = 1m^2$ and $C_v = 0.5$. The model nonlinearity comes from the outlet stream equations. Also, the desired set points for the tank levels are $h^{sp} = [0.56, 0.52, 0.36]^T$. For the dynamic optimization given in Equation 29, the following quadratic tracking objective function with control move penalization is defined.

$$J_k = \sum_{i=1}^{N_{H_p}} (h_{k+i} - h^{sp}_{k+i})^T.W_1.(h_{k+i} - h^{sp}_{k+i}) + \Delta r^T_{k+i-1}.W_2.\Delta r_{k+i-1} \tag{43}$$

The weighting factors W_1 and W_2 are positive-definite matrices set as

$$W_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 18 & 0 & 0 \\ 0 & 18 & 0 \\ 0 & 0 & 18 \end{bmatrix}$$

Also, the following path constraints are added to avoid reverse flow between the tanks.

$$\sum_{i=1}^{N_{H_p}} max(0, -q_{o1}(k+i)) \leq \delta \tag{44}$$

$$\sum_{i=1}^{N_{H_p}} max(0, -q_{o2}(k+i)) \leq \delta, \tag{45}$$

where $\delta > 0$ is a small regularization parameter (Chachuat 2009). The system is simulated for a time span of 50 minutes with sampling time of 6 seconds. The prediction and the control horizons are set to $H_p = 1.5$ and $H_c = 1$ min, respectively. The control inputs are bounded as $0 \leq q_{in1}, q_{in2}, q_{in3} \leq 0.3$. Note that a discretized linear model is applied as the prediction model is updated at each sample time.

The results of the SLMPC are presented in the sequel. Figure 4 shows trajectories of the measured outlet flow rates that are used to estimate the tank levels.

Figure 5 depicts the actual and estimated trajectories of the tank levels. The initial guess for the state estimator is $\hat{h}^+_0 = [0.1,\ 0.1,\ 0.1]^T$, which is different than the actual values $h_0 = [0,\ 0,\ 0]^T$. Despite this difference, the EKF is able to track the actual state trajectories successfully. It is also seen that the actual tank levels approach the set points shortly after the SLMPC is executed. The optimal

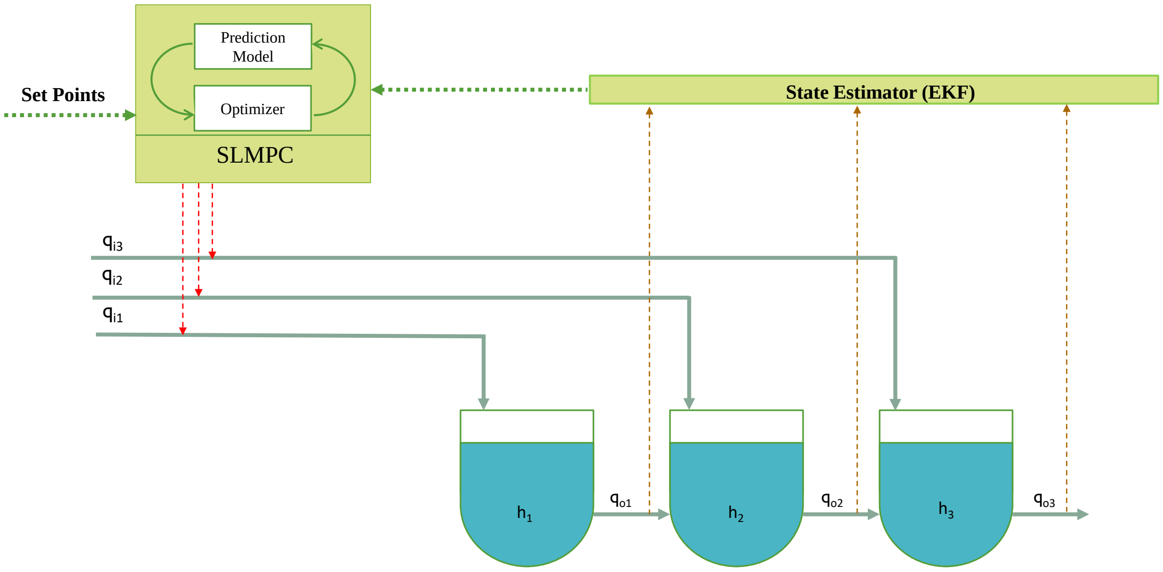


Figure 3. Schematic of the system with the SLMPC and estimator blocks.

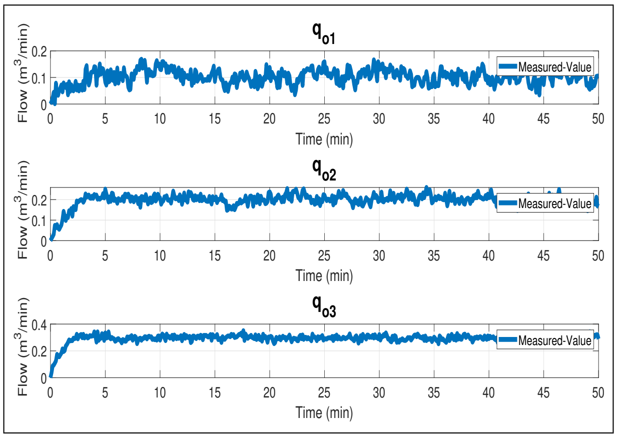


Figure 4. Measured system outputs.

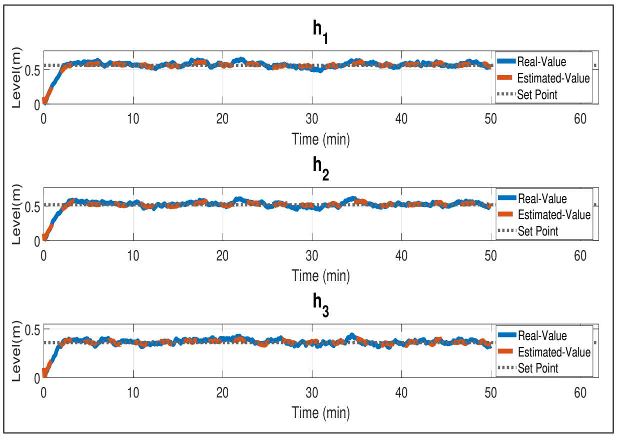


Figure 5. Actual and estimated system states.

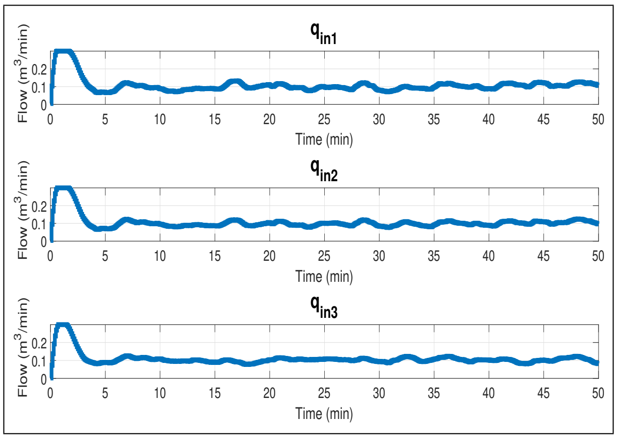


Figure 6. Optimal control inputs.

control inputs are plotted in Figure 6. It is observed that the control inputs are slightly oscillatory as a result of the process noise.

It is worth noting that the total simulation takes only 25 minutes by the enhanced OMMatlab, while it would take around 7 hours when employing the previous OMMatlab versions. This proves that the linearization part is the main bottleneck in the SLMPC procedure.

The code for the case-study presented in this manuscript can be found on GitHub at `https://github.com/pseAUT/SLMPC_OMMatlab`.

8 Conclusion

The implementation of the SLMPC algorithm using OpenModelica and MATLAB has been demonstrated in this paper. The OMMatlab API is upgraded to avoid repeated compilation of the OpenModelica model into an executable file at each sample time. In this way, the system can be linearized efficiently and the model matrices can be obtained directly in a `.m` file, thanks to the existing OpenModelica flags that make it possible to generate the linearized model in different formats. Therefore, the successive linearization runtime is optimized considerably. These enhancements can be implemented on other OpenModelica interfaces such as OMOctave, OMPython, and OMJulia to facilitate fast prototyping of control algorithms that require successive linearization.

References

Åkesson, Johan (2008). “Optimica—an extension of modelica supporting dynamic optimization”. In: *In 6th International Modelica Conference*. Citeseer, pp. 57–66.

Andersson, Christian et al. (2018). *JModelica.org User Guide*. English. Version Version 2.2. Modelon AB.

Bachmann, Bernhard et al. (2012). “Parallel multiple-shooting and collocation optimization with openmodelica”. In: *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*. 076. Linköping University Electronic Press, pp. 659–668.

Barton, P.I., J.R. Banga, and S. Galán (2000). “Optimization of hybrid discrete/continuous dynamic systems”. In: *Comput. Chem. Eng* 24.9, pp. 2171–2182. ISSN: 0098-1354. DOI: https://doi.org/10.1016/S0098-1354(00)00586-X. URL: http://www.sciencedirect.com/science/article/pii/S009813540000586X.

Braun, Willi, Lennart Ochel, and Bernhard Bachmann (2011). “Symbolically derived Jacobians using automatic differentiation-enhancement of the OpenModelica compiler”. In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 063. Linköping University Electronic Press, pp. 495–501.

Brembeck, Jonathan (2019). “A Physical Model-Based Observer Framework for Nonlinear Constrained State Estimation Applied to Battery State Estimation”. In: *Sensors* 19.20. ISSN: 1424-8220. DOI: 10.3390/s19204402. URL: https://www.mdpi.com/1424-8220/19/20/4402.

Brembeck, Jonathan, Martin Otter, and Dirk Zimmer (2011). “Nonlinear observers based on the functional mockup interface with applications to electric vehicles”. In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 63. Linköping University Electronic Press, pp. 474–483.

Cannon, Mark, Desmond Ng, and Basil Kouvaritakis (2009). “Successive linearization NMPC for a class of stochastic nonlinear systems”. In: *Nonlinear model predictive control: towards new challenging applications*, pp. 249–262.

Casella, Francesco (1998). “Modeling, simulation and control of a geothermal power plant”. PhD thesis. Politecnico di Milano, Italy, p. 72.

Chachuat, B.C. (2009). *Nonlinear and Dynamic Optimization: From Theory to Practice - IC-32: Spring Term 2009*. Polycopiés de l’EPFL. EPFL. URL: http://books.google.com/books?id=%5C_JOHYgEACAAJ.

Cortinovis, Andrea et al. (2014). “Safe and efficient operation of centrifugal compressors using linearized MPC”. In: *53rd IEEE Conference on Decision and Control*. IEEE, pp. 3982–3987.

Ellis, Matthew, Jinfeng Liu, and Panagiotis D. Christofides (2017). “Two-Layer EMPC Systems”. In: *Economic Model Predictive Control: Theory, Formulations and Chemical Process Applications*. Cham: Springer International Publishing, pp. 171–232. ISBN: 978-3-319-41108-8. DOI: 10.1007/978-3-319-41108-8_6. URL: https://doi.org/10.1007/978-3-319-41108-8_6.

Franke, Rüdiger (2002). “Formulation of dynamic optimization problems using Modelica and their efficient solution”. In: *Proceedings 2nd International Modelica Conference*, pp. 315–323.

Fritzson, Peter et al. (2020). “The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development”. In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Gräber, Manuel et al. (2012). “Using functional mock-up units for nonlinear model predictive control”. In: *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*. 076. Linköping University Electronic Press, pp. 781–790.

Heidarinejad, Mohsen, Jinfeng Liu, and Panagiotis D. Christofides (2013). “Algorithms for improved fixed-time performance of Lyapunov-based economic model predictive control of nonlinear systems”. In: *Journal of Process Control* 23.3, pp. 404–414. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2012.11.003. URL: https://www.sciencedirect.com/science/article/pii/S0959152412002545.

Imsland, L., P. Kittilsen, and T.S. Schei (2010). “odel-Based Optimizing Control and Estimation Using Modelica Model”. In: *Modeling, Identification and Control: A Norwegian Research Bulletin* 31.3, pp. 107–121. DOI: 10.4173/mic.2010.3.3. URL: https://doi.org/10.4173/mic.2010.3.3.

Imsland, Lars, Pål Kittilsen, and Tor Steinar Schei (2009). “Using modelica models in real time dynamic optimization–gradient computation”. In: *Proc. of Modelica*.

Jorissen, F., W. Boydens, and L. Helsen (2019). “TACO, an automated toolchain for model predictive control of building systems: implementation and verification”. In: *Journal of Building Performance Simulation* 12.2, pp. 180–192. eprint: https://doi.org/10.1080/19401493.2018.1498537. URL: https://doi.org/10.1080/19401493.2018.1498537.

Lie, Bernt et al. (2019). “Omjulia: An openmodelica api for julia-modelica interaction”. In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. 157. Linköping University Electronic Press.

Liu, Changchun et al. (2015). “Stochastic predictive control for lane keeping assistance systems using a linear time-varying model”. In: *2015 American Control Conference (ACC)*. IEEE, pp. 3355–3360.

Magnusson, Fredrik and Johan Åkesson (2015). “Dynamic Optimization in JModelica.org”. In: *Processes* 3.2, pp. 471–496. ISSN: 2227-9717. DOI: 10.3390/pr3020471. URL: https://www.mdpi.com/2227-9717/3/2/471.

OpenModelica (2021). https://openmodelica.org/doc/OpenModelicaUsersGuide/1.16. Accessed: 2023-07-23.

Pandey, Madhusudhan et al. (2021a). “Formulation of Stochastic MPC to Balance Intermittent Solar Power with Hydro Power in Microgrid”. In: *The First SIMS EUROSIM Conference on Modelling and Simulation, SIMS EUROSIM 2021*.

Pandey, Madhusudhan et al. (2021b). “Using MPC to Balance Intermittent Wind and Solar Power with Hydro Power in Microgrids”. In: *Energies* 14.4. ISSN: 1996-1073. DOI: 10.3390/en14040874. URL: https://www.mdpi.com/1996-1073/14/4/874.

Pippia, Tomas et al. (2021). “Scenario-based nonlinear model predictive control for building heating systems”. In: *Energy and Buildings* 247, p. 111108. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2021.111108. URL: https://www.sciencedirect.com/science/article/pii/S0378778821003923.

Romero, Alberto, Alejandro Goldar, and Emanuele Garone (2019). “A Model Predictive Control Application for a Constrained Fast Charge of Lithium-ion Batteries”. In: *International Modelica Conference*.

Ruge, Vitalij et al. (2014). “Efficient implementation of collocation methods for optimization using openmodelica and ADOL-C”. In: *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 096. Linköping University Electronic Press, pp. 1017–1025.

Sahlodin, Ali M (2022). “Optimally safe tank changeover operation using a smooth optimization formulation”. In: *ACS omega* 7.39, pp. 34974–34989.

Seki, Hiroya, Satoshi Ooyama, and Morimasa Ogawa (2002). “Nonlinear Model Predictive Control Using Successive Linearization Application to Chemical Reactors”. In: *Transactions of the society of instrument and control engineers* 38.1, pp. 61–66.

Vrlić, Martin, Daniel Ritzberger, and Stefan Jakubek (2020). “Safe and Efficient Polymer Electrolyte Membrane Fuel Cell Control Using Successive Linearization Based Model Predictive Control Validated on Real Vehicle Data”. In: *Energies* 13.20. ISSN: 1996-1073. DOI: 10.3390/en13205353. URL: https://www.mdpi.com/1996-1073/13/20/5353.

Zhakatayev, Altay et al. (2017). “Successive linearization based model predictive control of variable stiffness actuated robots”. In: *2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1774–1779. DOI: 10.1109/AIM.2017.8014275.

Parameter Estimation of Modelica Building Models Using CasADi

Carlos Durán Cañas[1] Javier Arroyo[1] Joris Gillis[1,2] Lieve Helsen[1,3]

[1]Department of Mechanical Engineering, KU Leuven, Heverlee, Belgium
carlosandres.durancanas@student.kuleuven.be,
{javier.arroyo,joris.gillis,lieve.helsen}@kuleuven.be
[2]Flanders Make@KU Leuven
[3]EnergyVille, Thor Park, Waterschei, Belgium

Abstract

Predictive control can substantially improve the energy performance of buildings during operation, but it requires a model of the building to be implemented. Gray-box model identification starts from a physics-based model (white-box element) and complements it with measurements from the operation of the building (black-box element). The level of detail of the original model is limited by the optimization problem that needs to be solved when estimating its parameters. Consequently, it is common to heavily simplify building models hindering the intelligibility of their parameters and limiting their application potential. This paper investigates the accuracy and scalability of different transcription methods for parameter estimation of building models. The methodology starts from a Modelica model as an initial guess which is transferred to CasADi using the Functional Mockup Interface to solve the parameter estimation problem. The study demonstrates the high effectiveness of multiple shooting. Single shooting and direct collocation could be more suitable for setups with faster integration times or with increased granularity in the training data, respectively.

Keywords: Gray-Box modeling, CasADi, Shooting Methods, Direct Collocation, OpenModelica

1 Introduction

Commercial and domestic buildings worldwide account for 30-45% of the global energy use (Mariano-Hernández et al. 2021). Therefore, efficient building energy use is a topic of growing interest. A popular strategy for building energy management systems (BEMS) is the use of model predictive control (MPC) (Drgoňa et al. 2020). MPC uses a building model and an optimizer to minimize a cost function generally comprised of two competing elements like occupant thermal discomfort and operational cost.

Gray-box model identification is a powerful tool for obtaining building models for predictive control through two input sources: *prior system knowledge* and *operational data* which are referred to as the white and black elements of a gray model, respectively (Bohlin 2006). This approach benefits from the advantages of both physics- and data-based modeling by calibrating the physical model parameters with operational data extracted from the actual building. Moreover, gray-box modeling can automatically and systematically tune the parameters for a model while having the reliability offered by physics (Bohlin 2006).

Prior knowledge can be introduced with well-known equations describing physical phenomena in buildings like heat transfer and thermal inertia. Other inputs like weather variables and internal gains are subject to uncertainty and need to be estimated with forecasting models. It is also crucial to obtain accurate values for parameters such as heat conductivities of materials, solar transmittance of the glazing in the windows, etc. However, obtaining detailed information on the thermal systems for the building is a difficult and time-consuming process in practice (Yu et al. 2019). Optimization is required to estimate the parameter values that minimize a predefined error function, which usually leads to a non-linear and non-convex problem that needs to be solved. Traditionally, this problem is solved by lumping the building parameters into basic models usually represented by a lower-order RC (resistance-capacitance) network. Some examples include (Beneventi et al. 2012), (Drgoňa et al. 2020) and (Saurav and Chandan 2017). These simplifications can decrease the usability and intelligibility of the building model and hamper the ability of the model to be used for fault detection and diagnostics mechanisms in the context of predictive maintenance.

Another approach to estimate the model's parameters is through black-box optimizers like brute-force or genetic algorithms. However, these methods can have scaling issues when increasing the number of parameters to be estimated. Since building models can have several parameters to calibrate, traditional gradient-based optimization methods are preferred for the envisaged application. Moreover, unlike off-line parameter estimations, efficient non-linear problem solvers are key for the application of on-line parameter estimations such as the ones involved in adaptive control. For these reasons, the use of efficient, gradient-based, parameter estimation methods is crucial in order to develop more advanced building models.

This work investigates the performance of different transcription methods for parameter estimation of building models by coupling two open-source toolboxes: OpenModelica (Fritzson et al. 2005) and CasADi (Andersson et al. 2019). The former is a tool for modeling physical systems using the Modelica language and the latter is

DOI
10.3384/ecp204301

a numerical optimization and algorithmic differentiation framework designed to solve highly complex optimization problems. Similar workflows were adopted by (Shitahun et al. 2013) and by (Decker 2021), but they were using deprecated versions of the software and their focus was not on parameter estimation of building models. Criteria such as accuracy, scalability, convergence time, and computational cost are investigated in this work. These criteria are then used to compare all investigated methods as well as to determine their optimal application range.

The outline of the paper is as follows: Section 2 gives theoretical background related to this work; Section 3 elaborates on the methodology used in this work to implement and compare different transcription methods for parameter estimation; Section 4 presents the results obtained from the implementation of each parameter estimation method. Finally, Section 5 draws the main conclusions and Section 6 suggests lines of further research.

2 Theoretical Background

2.1 Parameter Estimation Problem

The process of parameter estimation is a crucial step when configuring a building model. The process of calibrating the parameters of a physical model can be expressed mathematically as shown in Eqs. 1[1] The objective function to be minimized $f(\boldsymbol{p})$ is expressed as a (non-linear) least squares problem by means of the Eucledian (ℓ-2) norm for the error between the outputs from the physical model $\boldsymbol{y}(\boldsymbol{p},t)$ and the historical measurements $\hat{\boldsymbol{y}}(t)$ within a time horizon $t \in [t_0, t_T]$. Here, the error function is squared to achieve the sum of the squared differences. This in turn is divided by two as a convention in order to remove constants during the calculation of its derivative. The optimization is subject to equality and inequality constraints ($\boldsymbol{h}(\boldsymbol{p})$ and $\boldsymbol{g}(\boldsymbol{p})$, respectively) which are derived from the physics and limits of the real system, represented by the physical model. Here, the objective function as well as the equality and inequality constraints are dependent on the model parameter values $\boldsymbol{p} \in \mathbb{R}^m$, which must be tuned to minimize the residual between the model and measurements.

$$\min_{\boldsymbol{p}} f(\boldsymbol{p}) = \int_{t_0}^{t_T} \frac{1}{2} \left(\|\boldsymbol{y}(\boldsymbol{p},t) - \hat{\boldsymbol{y}}(t)\|_2\right)^2 dt \tag{1a}$$

$$\text{subject to: } \boldsymbol{h}(\boldsymbol{p}) = 0 \tag{1b}$$

$$\boldsymbol{g}(\boldsymbol{p}) \leq 0, \tag{1c}$$

Parameter estimation problems for building models are usually non-convex because of the multiplication of thermal resistances and capacitances that are commonly optimization variables. This non-convexity dictates the importance of the initial parameter guesses $\boldsymbol{p}_0$ with minimum and maximum values $\boldsymbol{p}_{min}$ and $\boldsymbol{p}_{max}$, respectively. For the case of a convex problem, initial parameter guesses become trivial since, by definition, all local minima in a convex problem are the global minimum (Wright, Nocedal, et al. 1999). The solution of a non-convex problem is most likely a local minimum which is accepted in practice due to the high complexity involved in the calculation and (in some cases) the non-physicality of a global minimum. For this reason, accurate initial guesses for the parameter values are paramount to obtaining a physical local minimum. Figure 1 depicts the results of a bad initial guess for an estimation problem with a single parameter. The importance of an accurate initial guess becomes clear even in this case with a single parameter for which all initial guesses lead to different local minima, none of which are the global minimum.

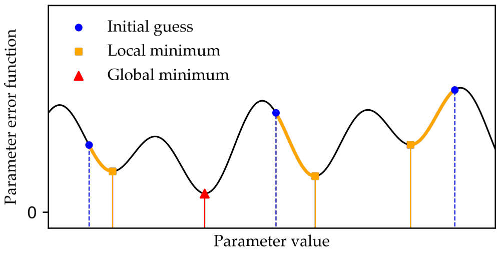


Figure 1. Non-convex minimization problem dependence on initial parameter guess.

2.2 Transcription Methods

Transcription methods are responsible for discretizing the originally continuous parameter estimation problem into a discrete non-linear program (NLP) in the form of Eqs. 2.

$$\min_{\boldsymbol{p}} \sum_{i=1}^{N_T} \frac{1}{2} \left(\|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\|_2\right)^2 \tag{2a}$$

$$\text{subject to: } h_i = 0 \tag{2b}$$

$$g_i \leq 0, \tag{2c}$$

for $i = 1, \ldots, N_T$ with N_T being the instance at the end of the time horizon. Once this transcription takes place, NLP solvers are used to minimize the objective function and output the optimal parameter values (M. P. Kelly 2015). The transcription method used to discretize the problem is critical to the complexity and the outcome of a parameter estimation problem since it dictates the number of decision variables and the sparsity of the eventual NLP to be solved. Multiple algorithms exist for this process, yet they can all be classified into two main groups: *shooting methods* and *collocation methods*.

2.3 Shooting Methods

Shooting methods take states as decision variables and integrate over a set of intervals, approximating the function

[1] Vectors and matrices are expressed in bold while scalars are expressed in regular text ($\boldsymbol{y}$ vs y).

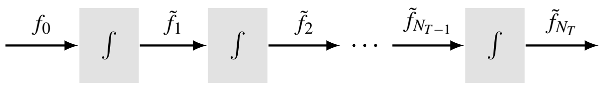


Figure 2. Block diagram for a single shooting transcription algorithm.

dynamics along the integration path based on given constraints. The term shooting methods refers to their resemblance to the operation of a projectile deployment device such as a cannon. In this example, the decision variables are only known in the first instance (e.g. firing angle and power), while the trajectory is subject to the projectile dynamics and physical constraints.

The simplest variation of the shooting algorithms is the single shooting method. Single shooting works by taking only the very first state as a decision variable which is used to determine a prediction for the following state. Subsequently, this prediction is used for the integration of the following state, starting a chain reaction of predictions until the integration horizon is reached (Schittkowski 2002). Figure 2 shows a block diagram demonstrating the integration process in a single shooting algorithm with f_i being decision variables and $\tilde{f}_i$ being predictions. Here, it can be seen that for every integration step, the previous prediction is taken as an input, generating the following prediction until the instance N_T is reached.

The main advantages of a single shooting algorithm stem from its simplicity and the compact representation of the eventual NLP. They are suitable for simple differential-algebraic equations (DAE) where extremely good initial guesses can be provided. Nevertheless, they may pose convergence problems for complex systems because of the need to integrate over the entire time horizon for every iteration of the optimization (Michalik, Hannemann, and Marquardt 2009).

Multiple shooting algorithms operate in a very similar way to single shooting algorithms but break down the time horizon into multiple intervals. Instead of taking a single decision variable, multiple shooting methods take a decision variable for every time step within the integration horizon, making a single integration over that time step (M. P. Kelly 2015). As the segments become shorter, the integration paths tend to become linear. Additionally, since each step is not dependent on the previous step, each integration can be computed in parallel, leading to shorter integration times. Since each prediction step does not perfectly match the decision variable for the following segment, the difference (known as the *defect*) must be stated in the constraint equations leading to larger and sparser programs. The increase in the number of constraints can increase the total computational time (M. Kelly 2017). A block diagram for this process is visualized in Figure 3. The use of single and multiple shooting methods is highly dependent on the application case and its level of complexity. Figure 4 shows an example of both shooting methods being applied to a parameter estimation problem.

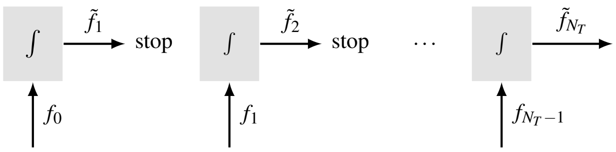


Figure 3. Block diagram for a multiple shooting transcription algorithm.

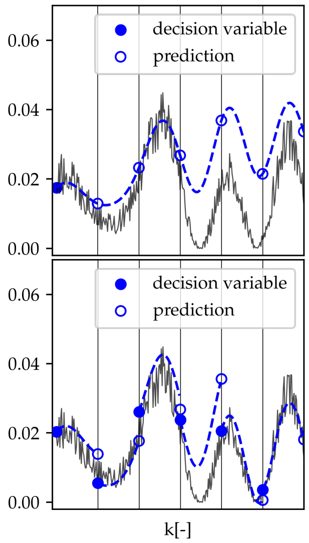


Figure 4. Example for a single (top) and multiple (bottom) shooting algorithm applied to the transcription of an error function.

2.4 Collocation Methods

The basis of collocation methods is the use of spline functions made up of polynomial sequences. The motivation is the effortless derivation and integration of these spline functions as well as their capability to be easily expressed in terms of coefficients (M. Kelly 2017). The integration path followed for each segment in a direct collocation algorithm is determined by two factors: the desired order for the polynomial to be fitted and the slope of the function at the collocation points (Bellomo et al. 2007). Depending on the desired order for the polynomial fit n, $n-1$ collocation points must be placed within the segment. Once these collocation points are determined, the algorithm adjusts the values for them such that a spline function going through the initial decision variable matches the slope of the function at each collocation point. If, like in most applications for collocation methods, this algorithm is applied within a multiple shooting framework, this procedure is completed for multiple segments covering the entire integration horizon. Similarly to multiple shooting, this process can be realized in parallel, yielding a series of predictions and their corresponding defects which must be accounted for in the constraint equations of the NLP.

Two approaches are widely used with the goal of reducing the defects between the predictions and decision variables. These are referred to as mesh refinement pro-

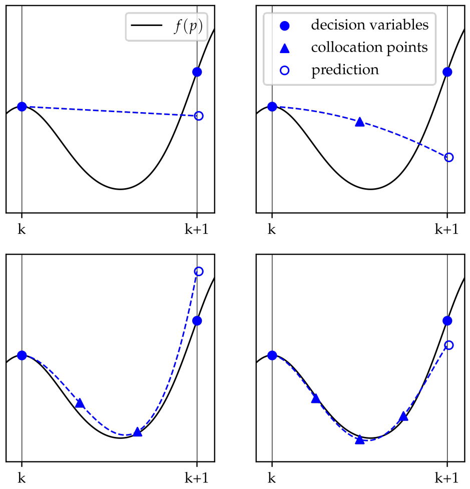


Figure 5. Example for a p-refinement procedure on direct collocation for polynomial orders ranging from first (top left) to fourth (bottom right) order.

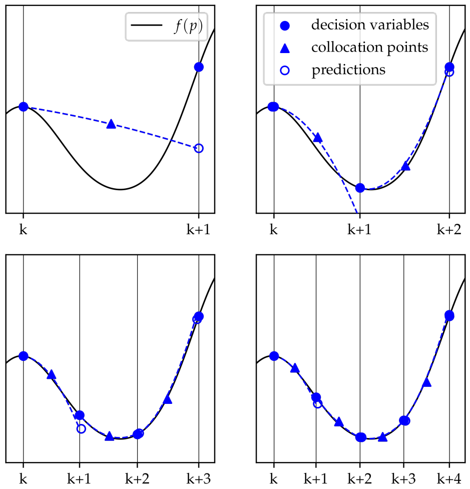


Figure 6. Example for an h-refinement procedure on direct collocation using second degree polynomials.

cedures, namely p- and h-refinement. A *p-refinement* procedure increases the order of the polynomial for each collocation segment such that the integration path can better follow the trajectory dynamics, ultimately resulting in a better prediction. An *h-refinement* procedure reduces the segment length such that the functions become more linear.

Figure 5 shows an example of a p-refinement procedure with varying polynomial orders. Here, a focused view on a single segment (k to $k+1$) is shown for polynomial orders ranging from first to fourth order for a direct collocation method on a function $f(p)$. It is clear that the defect decreases and the dynamics are better represented as the polynomial order increases. Similar to Figure 5, Figure 6 shows an h-refinement procedure for varying segment lengths for a function $f(p)$ using a second-order direct collocation method. A noticeable decrease in the prediction defect can be seen as the segment lengths become smaller.

3 Methods

The workflow followed in this work is hosted in the following open-source repository under a BSD license:

```
https://gitlab.kuleuven.be/positive-energy-districts/mocaspy
```

The necessary steps to generate the original physics-based model are illustrated with white blocks in Figure 7. These white blocks of Figure 7 represent the steps handling the model with true system dynamics. The original model is reconfigured into a so-called wrapped model that redeclares the parameters as inputs to tune its values during the optimization. Multiple open-source Modelica libraries exist for building modeling that can be used to configure the original building model such as IDEAS (Jorissen et al. 2018) and Buildings (M. Wetter et al. 2014). OpenModelica is used to compile a Functional Mockup Unit (FMU) to later transfer the model into CasADi.

The data collection process is illustrated by the black blocks shown in Figure 7. In practice, the data needed for calibration would be directly gathered from the actual building. However, in this work, OpenModelica is also used to emulate operational data. That is, we use the same model to generate the data as the model that is later used for parameter estimation. In this way, the true parameter values are known, and the accuracy of the parameter estimation process can be measured for a given deviation that is artificially introduced. This provides a hermetic environment for the investigation which would be unachievable in a real setting. Measurements of interest comprise weather and building variables such as the ambient temperature $\hat{T}_{amb}$ and the total heat input into the thermal zone $\hat{Q}_{hea,coo}$. Finally, the recorded operative zone temperature $\hat{T}_{zon}$ is used as the target variable in Eqs. 1. The training period t_T is set to one week (604800 seconds) with a sampling time t_s of 30 seconds, leading to 20160 samples in total.

Once the model FMU and the operational data are ready, the parameter estimation problem is formulated with the CasADi framework. This process is shown by the gray blocks of Figure 7. CasADi's `DaeBuilder` class was recently extended to import Model Exchange FMUs of version 2.0 (Andersson 2023 submitted). From Eqs. 1, the parameter estimation problem is relaxed to Eqs. 3 where a slack variable S is introduced to use soft constraints. The objective function minimizes the Euclidean norm of the error between the (virtually) measured $\hat{T}_{zon}$ and the modelled T_{zon} thermal zone temperatures.

$$\min_{\boldsymbol{p}} f(\boldsymbol{p}) = \int_{t_0}^{t_T} \left[\left(\left\| T_{zon}(\boldsymbol{p},t) - \hat{T}_{zon}(t) \right\|_2 \right)^2 + S \right] dt \tag{3a}$$

$$\text{subject to:} \quad \boldsymbol{p} \leq \boldsymbol{p}_{max} + S \tag{3b}$$

$$\boldsymbol{p} \geq \boldsymbol{p}_{min} - S \tag{3c}$$

$$\boldsymbol{S} \geq 0 \tag{3d}$$

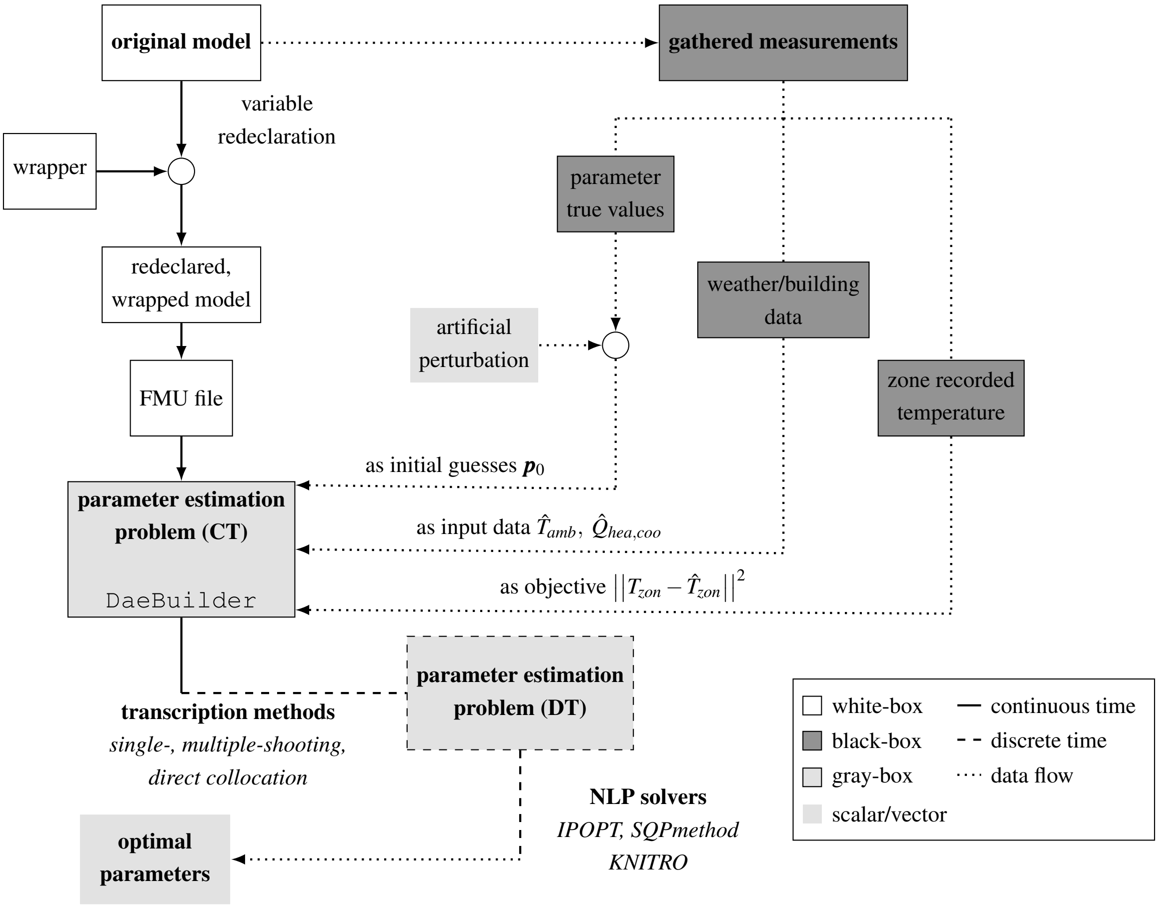


Figure 7. Parameter estimation workflow. The CasADi's `DaeBuilder` class is used to load the building model and to transcribe the continuous-time (CT) parameter estimation problem into a discrete-time (DT) problem that an NLP solver can address.

In Eqs. 3, $\boldsymbol{p}_{max}$ and $\boldsymbol{p}_{min}$ represent the upper and lower boundaries of the parameter vector $\boldsymbol{p}$, respectively.

The parameter estimation problem must be discretized by means of transcription methods. The Rockit framework (Gillis et al. 2020) is used here since it is built on top of CasADi and offers readily available formulations of different transcription methods. In this study, single shooting, multiple shooting, and collocation are investigated. Multiple variations of these algorithms are implemented to study the effect of h- and p-refinement. Upon completion of the problem transcription, the parameter estimation problem is solved by means of an NLP solver. IPOPT is chosen here due to its widely recognized reputation and capabilities for large-scale optimization. It is used in a setting with a limited-memory Hessian approximation. The results are compared based on three factors: accuracy for the parameter estimation, number of iterations, and the computational time required for convergence. These factors determine the application range for each transcription method based on the desired level of accuracy and available computational power.

The model `SimpleRoomOneElement` from the IDEAS library is chosen as a good trade-off between the level of detail and simplicity. This model represents a single thermal zone equipped with multiple, double-paned windows at different wall orientations (declared as `corGDouPan`), and is shown in Figure 8. Additionally, the model is equipped with a heater operating under a simple on/off control at a given time of the day (represented by the data table `intGai`). The weather data is simulated by the module `weaDat` and inputted into the building thermal zone, `thermalZoneOneElement`.

The parameters of the original model are redeclared as inputs in a so-called *wrapped* model to enable 1) their variability as decision variables in the CasADi `DaeBuilder` object, and 2) derivative information of model outputs. Moreover, some elements of the model had to be bypassed with inputs obtained from a previous simulation as the `DaeBuilder` class does not yet support time events.

Three studies are considered: the estimation of a single parameter, three parameters, and five parameters. Table 1 shows the parameters estimated during all studies along their units, true values $\boldsymbol{p}_{real}$, lower $\boldsymbol{p}_{min}$ and upper $\boldsymbol{p}_{max}$ limits, and their initial guesses after being artificially perturbed $\boldsymbol{p}_0$. Parameters that are commonly unknown, desired and/or hard to obtain in practice are selected to be estimated like U_{win} and $h_{con,win,out}$ which represent the transmission and convective coefficients associated with the installed windows, respectively. $h_{con,wall,out}$ represents

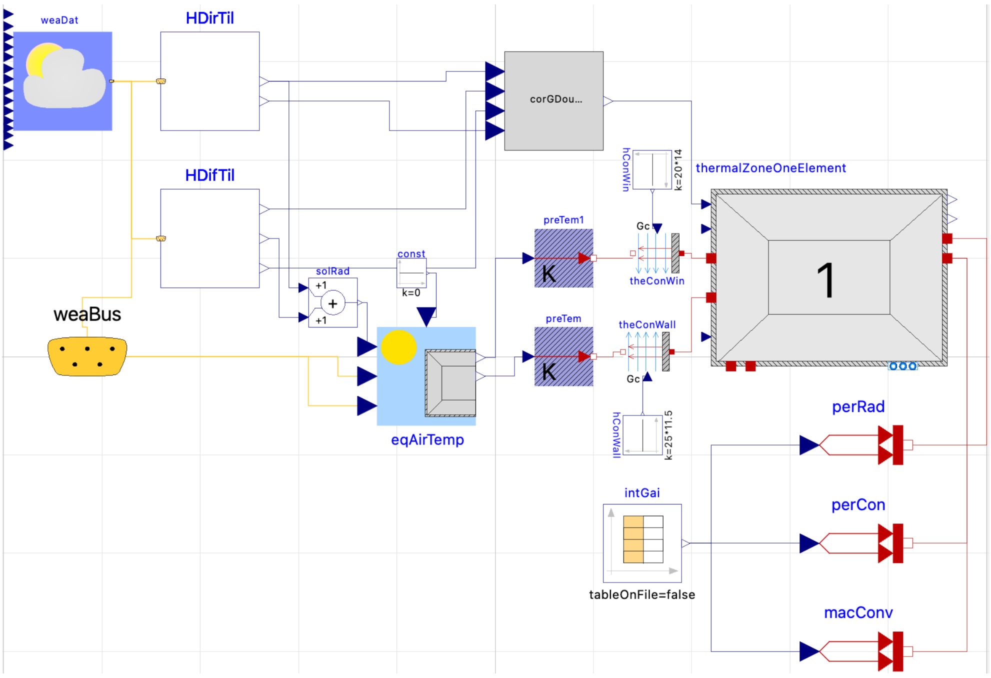


Figure 8. Graphical representation of `SimpleRoomOneElement.mo` model from IDEAS (Jorissen et al. 2018).

Table 1. Parameter values, boundaries and initial guesses for all studies conducted.

# Params.	p	*Unit*	p_{real}	p_{min}	p_{max}	p_0
1	U_{win}	W/m^2K	2.1	1.9	2.4	6.3
3	$h_{con,win,out}$	W/m^2K	20	18	22	60
	$h_{con,wall,out}$	W/m^2K	20	18	22	60
5	h_{rad}	W/m^2K	5	4	6	15
	a_{ext}	-	0.7	0.5	0.9	1

the convective coefficient for the exterior walls. Finally, h_{rad} is the coefficient of radiative heat transfer for the zone walls, and a_{ext} represents the thermal absorption coefficient for the exterior walls. From Table 1 it can be seen that the first study estimates the window transmission coefficient, the second study includes the window and exterior wall convective coefficients and the final study incorporates h_{rad} and a_{ext}. For all instances, the initial guesses are shown. These are decided to be three times larger than the actual value for the parameter, except for a_{ext} since it is a percentage and has a maximum value of 1.

The number of steps per discretization interval M is decided through a sensitivity analysis for a simple model integration with respect to the results of a reference integrated with an arbitrarily high value of $M = 3000$ steps. A value of $M = 10$ is taken as a compromise between integration error and computational demand. Table 2 contains a detailed list of all variations conducted to compare the transcription methods for parameter estimation. Notably, each variation is implemented for all cases outlined in Table 1 with one, three, and five parameters, which results in a total of 60 optimizations being carried out.

Table 2. Conducted investigations for all considered test cases. First, the refinement scheme is studied in detail for multiple shooting and collocation schemes (Sections 4.1,4.2,4.3). Then, all three transcription methods are compared for specific refinement schemes (Section 4.4).

Method	*Refinement Scheme*		*Transcription methods*	
	h (N)	p (deg)	h (N)	p (deg)
Single Shooting	-	-	100 200	-
Multiple Shooting	100 200 300 400	-	100 200	-
Direct Collocation	100 200 300 400	2, 3, 4, 5, 6, 7, 8, 9	100 200	3

4 Results and Discussion

All optimizations were run on an Apple M1 MacBook Air (2020 version) with 8 GB of RAM and MacOS Ventura 13.0. A comparison between the thermal zone temperatures using the perturbed initial guess values and the estimated parameter values can be seen in Figure 9. All valid estimations, i.e. those optimization runs that converged, led to proper fitting and their solutions resulted in original parameter values within a tolerance of 10%, in many cases landing on the local minima at the parameter boundaries

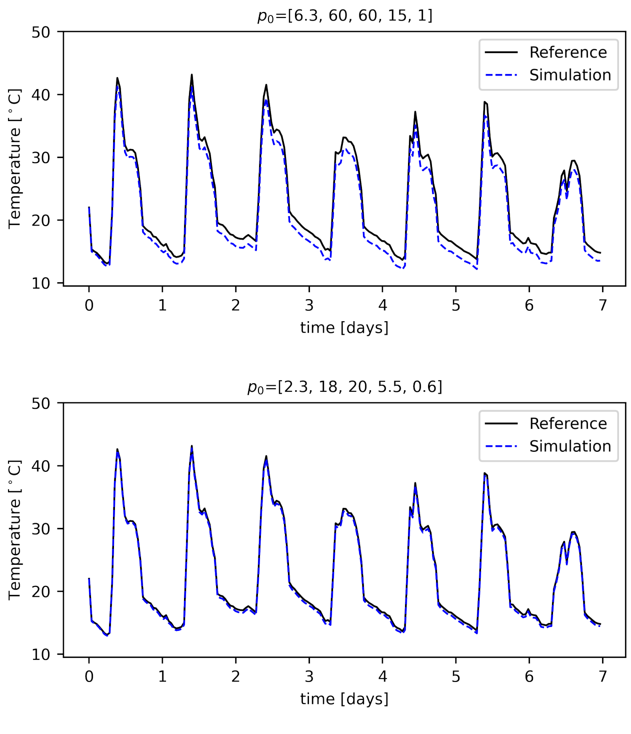


Figure 9. Temperature profiles of the model using the parameter initial guesses (top) and the estimated values (bottom) compared to the actual model temperature profile (Reference).

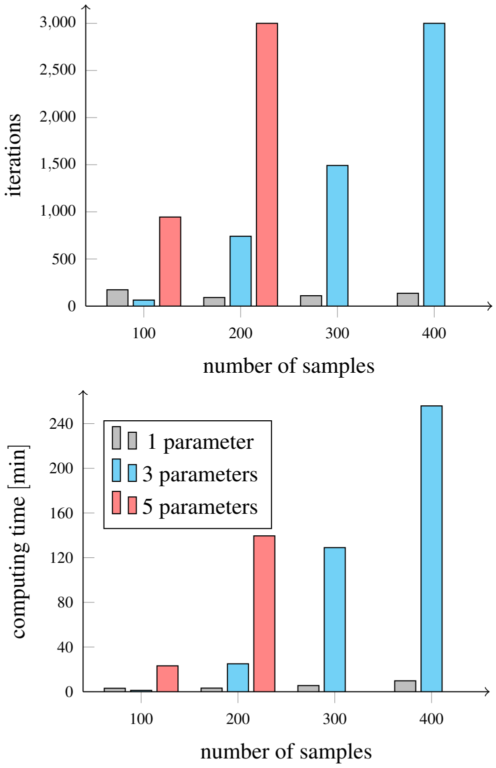


Figure 10. h-refinement investigation for a multiple shooting algorithm.

(see Table 1). Therefore, all achieved solutions are considered successful in terms of accuracy and the focus of this section is on the computational demand of obtaining these values.

4.1 Multiple Shooting h-refinement

The number of iterations and computing times for all variations to investigate h-refinement in multiple shooting are shown in Figure 10. The first aspect to highlight is that the multiple shooting algorithm succeeds to estimate the parameters for all cases with one, three, and five parameters. It is evident that the multiple shooting algorithm demonstrates remarkable suitability in the scenario of a single estimated parameter. It is able to estimate the chosen parameter without encountering any significant challenges despite the increasing estimation problem granularity. Its convergence times remain significantly low compared to the other cases with more parameters. The computational limits when increasing the problem granularity for the multiple shooting are reached in the cases involving three and five estimated parameters. Upon closer examination, it becomes apparent that the number of iterations and computational time for the three-parameter case with 100 samples closely resemble those of the single-parameter case. However, a steep increase is observed starting at the estimation with 200 samples, increasing quasi-exponentially until the iteration limit for IPOPT of 3000 iterations is reached for the case using 400 samples. Similar behavior is observed for the five-parameter case, although at a much earlier point. The computational time necessary for this scenario is approximately eight times larger than that of the single- and two-parameter cases when using a sample size of 100. Moreover, it rapidly reaches the iteration limit with a sample size of 200.

Considering that all estimations led to an accurate representation of the model, it is clear that an h-refinement for the multiple shooting algorithm is not necessary for the envisaged application. Increasing the granularity merely increases the computational resources required for the estimation without providing any additional value to the solution. However, the granularity choice should be made carefully since lower values can result in a loss of detail which can lead to an infeasible problem statement. Infeasible problem statements were encountered for sample sizes of 50, 85 and 90 for the single, three- and five-parameter cases, respectively.

4.2 Direct Collocation p-refinement

After exploring the h-refinement for multiple shooting, the impact of increasing the order for direct collocation is investigated. The sample size is set to 100, which represents the minimum value for a viable problem. Figure 11 presents the resulting number of iterations and computing times for all case studies. A slight increase in computational demand is observed for the three-parameter case around the collocation order of 9 and a sudden increase for the collocation order of 6, which can be qualified as an outlier. However, there is no clear correlation between the

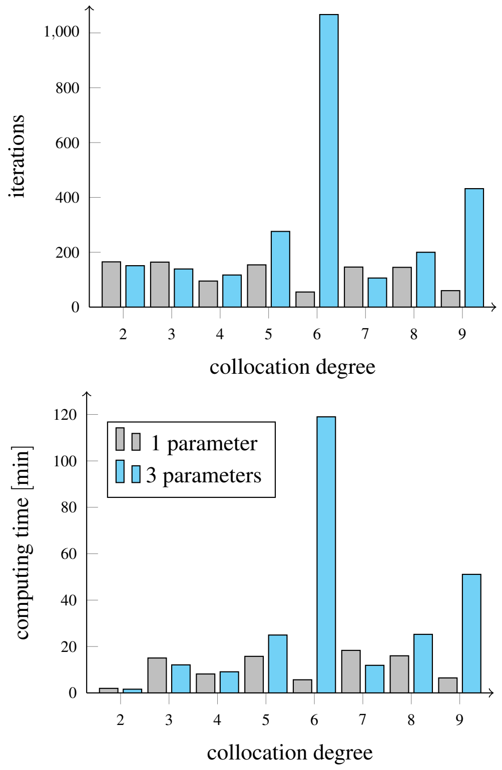


Figure 11. p-refinement investigation for a direct collocation algorithm with 100 samples.

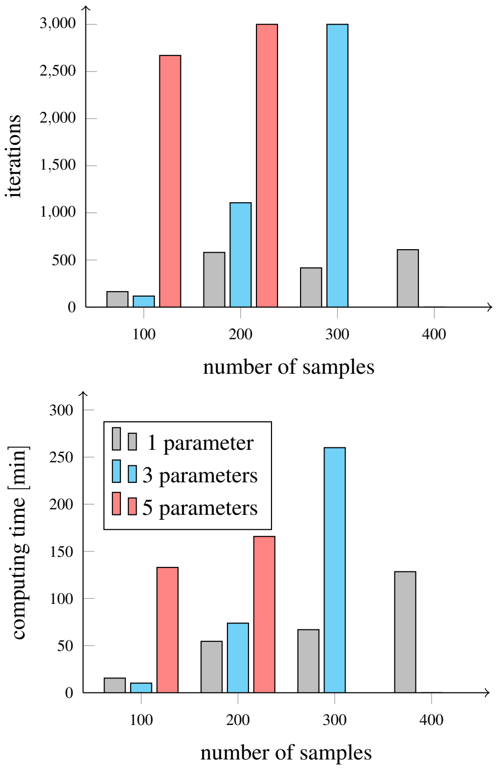


Figure 12. h-refinement investigation for a direct collocation algorithm with collocation degree of 3.

computational resources and the order of the collocation. As discussed in subsection 2.4, a higher collocation order results in a more accurate fit for the objective function, particularly for cases with lower granularity. The results from this study suggest that the collocation order is not affecting the computational strain thanks to a large enough granularity in the problem statement. Therefore, an increase in computational demand would be expected for a case with a higher number of samples.

4.3 Direct Collocation h-refinement

Similar to the multiple shooting case, an h-refinement investigation is conducted for a direct collocation algorithm. Here, a collocation degree of three is used based on the results of the p-refinement investigation. The results are shown in Figure 12. Similar patterns to those observed in the multiple shooting case emerge with a notable difference for the single parameter case. In comparison to the multiple shooting case, the estimation process using 400 samples takes approximately 13 times longer to converge, indicating a significantly higher computational strain for the direct collocation algorithm. Similar observations can be made when comparing the three- and five-parameter cases. While the overall trends are similar to the multiple shooting case, it is clear that the direct collocation algorithm experiences a higher computational burden. This is particularly evident in the three-parameter case, where the iteration limit is reached at 300 samples, compared to 400 samples in the multiple shooting case. In the five-parameter case, both studies reach the iteration limit at 200 samples. However, the computational strain is higher for direct collocation, as evidenced by the significantly longer computational time and more than twice the number of iterations required in the case with 100 samples.

Again, since all successful estimations yielded identical results, it is recommended to select a sample size that ensures proper convergence while minimizing the computational load. Excessively large sample sizes can lead to unnecessary waiting times or, in complex scenarios such as the five-parameter case, even convergence failure in the estimation process.

4.4 Transcription Method Comparison

Finally, a comparison of all transcription methods applied is carried out. Two sample sizes are investigated: one of 100 (shown in Figure 13) and another of 200 (shown in Figure 14). Single shooting, multiple shooting, and direct collocation with a collocation degree of three are compared for each sample size when estimating one, three, and five parameters. From examining the results in Figure 13 and Figure 14, several observations emerge.

Single Shooting This algorithm fails to converge to a solution for the five-parameter case. In the single- and three-parameter cases, it shows the fewest iterations required for convergence with the highest computing times for all scenarios. This could be seen as an advantage for a setup with more powerful computational resources. It is worth noting that the FMI simulations triggered by the

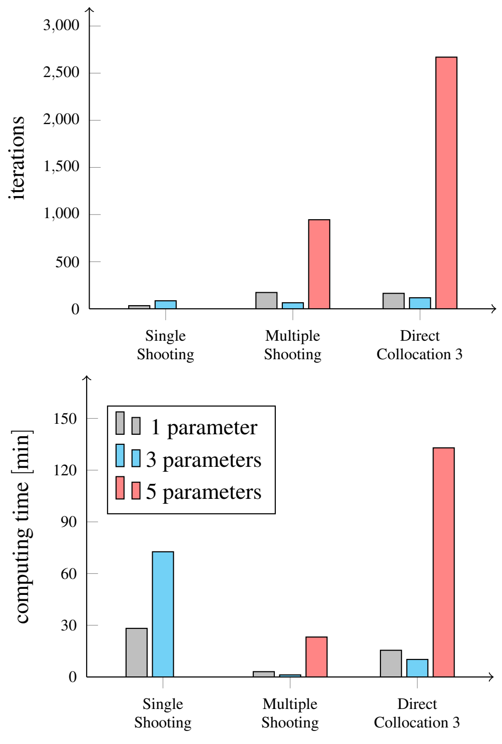


Figure 13. Transcription method performance comparison using 100 samples.

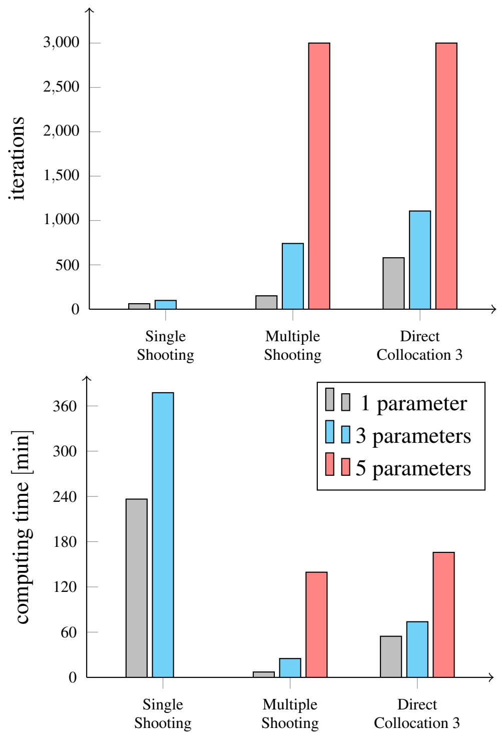


Figure 14. Transcription method performance comparison using 200 samples.

`DaeBuilder` object took more than 20 times longer than the calculations needed in each iteration for the optimization. Hence, single shooting could become competitive if the FMU integration times can be sped up.

Multiple Shooting Overall, multiple shooting shows the best performance for all investigated transcription methods. As shown for both sample sizes, it offers the lowest computing times for all cases, demonstrating its scalability potential. Moreover, when compared to its usual counterpart, direct collocation, multiple shooting requires significantly fewer iterations to achieve these fast convergence times, particularly evident in the investigation with 100 samples. Furthermore, if the efficiency of the FMI is improved, multiple shooting has the potential to achieve even better performance. The advantage of multiple shooting over direct collocation is further supported by the h-refinement investigation where multiple shooting achieved the same level of model granularity at a much lower computational cost. In general, multiple shooting is able to converge to accurate solutions for all investigated cases while requiring less tuning compared to single shooting and direct collocation.

Direct Collocation Similar to multiple shooting, direct collocation is able to converge to an optimal solution for all investigated cases though it has a higher computational burden. However, by comparing the results from both sample sizes (Figs 13 and 14) a decrease in the performance gap between both algorithms is observed. While multiple shooting still achieves convergence with a lower computational burden, it experiences a significant increase in both iterations and computing time when the granularity of the problem increases. This suggests that for higher sample sizes direct collocation could outperform multiple shooting. Additionally, direct collocation offers the highest level of tunability, which can be optimized for a specific application, perhaps leading to better performance. However, this tuning process is time-consuming and requires deep system knowledge. Overall, direct collocation algorithms exhibit great potential for rapid convergence, although meticulous fine-tuning is necessary to fully exploit their capabilities.

5 Conclusion

Predictive control can substantially enhance energy efficiency in buildings. To calibrate building models with operational data, efficient discretization methods are needed for the associated parameter estimation problem. The implemented methodology formulates the original building model in Modelica and transfers the model to CasADi through its `DaeBuilder` class, which relies on the Functional Mockup Interface. Multiple variations of transcription methods and the effect of different refinement schemes are investigated. The study demonstrates the high effectiveness of the multiple shooting algorithm for the envisaged application. Multiple shooting successfully converges for all cases investigated and shows the smallest convergence time. Single shooting has the highest computing times but requires fewer iterations to converge, so it may also work for simpler models or setups with more computational resources. Finally, a trend is observed toward better performance in cases with increased granularity for direct collocation.

6 Future Work

Although the use of the CasADi's `DaeBuilder` to calibrate Modelica models shows huge potential, there are still some challenges when following this methodology. A large setback for the proposed workflow relates to the fixed variability for the parameters of an FMU compiled with OpenModelica. This requires a manual redeclaration of all parameters to be estimated as inputs to enable their variability in CasADi as decision variables. Furthermore, FMU simulations are slow when compared to the time needed in the NLP solver per iteration (approximately 20 times longer for the model used in this work) , which leads to excessive computing times. Finally, the lack of support for time events in CasADi's `DaeBuilder` required major changes in the model to accommodate their introduction as inputs, similar to the parameter variability issue. The introduced methodology shows promise in coupling Modelica and CasADi for optimization. However, as shown in this work, it is still in its early stages and there is ample room for further improvements. Joint efforts are needed to come up with a workable solution, which will then be illustrated on multiple applications.

Acknowledgements

The authors acknowledge the financial support by KU Leuven through the TECHPED - C2 project (C24M/21/021). The TECHPED project investigates TECHnically feasible and effective solutions for Positive Energy Districts. This research was partially supported by Flanders Make, the strategic research centre for the manufacturing industry, by SBO projects WORKDRIVE and DIRAC.

References

Andersson, Joel A E (2023 submitted). “Import and Export of Functional Mockup Units in CasADi”. In: *Proceedings of the 15th International Modelica Conference*. Aachen, Germany.

Andersson, Joel A E et al. (2019). “CasADi – A software framework for nonlinear optimization and optimal control”. In: *Mathematical Programming Computation* 11.1, pp. 1–36. DOI: 10.1007/s12532-018-0139-4.

Bellomo, Nicola et al. (2007). *Generalized collocation methods: solutions to nonlinear problems*. Springer Science & Business Media.

Beneventi, Francesco et al. (2012). “An effective gray-box identification procedure for multicore thermal modeling”. In: *IEEE Transactions on Computers* 63.5, pp. 1097–1110.

Bohlin, Torsten P (2006). *Practical grey-box process identification: theory and applications*. Springer Science & Business Media.

Decker, Sebe De (2021). “Generic Optimization Toolbox for Physical Systems Coupling OpenModelica, CasADi and Python”. Master thesis. Leuven, Belgium: KU Leuven.

Drgoňa, Ján et al. (2020). “All you need to know about model predictive control for buildings”. In: *Annual Reviews in Control* 50, pp. 190–232.

Fritzson, Peter et al. (2005). “The OpenModelica modeling, simulation, and development environment”. In: *46th Conference on Simulation and Modelling of the Scandinavian Simulation Society (SIMS2005), Trondheim, Norway, October 13-14, 2005*.

Gillis, Joris et al. (2020-03). “Effortless modeling of optimal control problems with rockit”. In: *Proceedings of the 39th Benelux Meeting on Systems and Control*. Elspeet, The Netherlands.

Jorissen, Filip et al. (2018). “Implementation and Verification of the IDEAS Building Energy Simulation Library”. In: *Journal of Building Performance Simulation* 11 (6), pp. 669–688. DOI: 10.1080/19401493.2018.1428361.

Kelly, Matthew (2017). “An introduction to trajectory optimization: How to do your own direct collocation”. In: *SIAM Review* 59.4, pp. 849–904.

Kelly, Matthew P (2015). “Transcription methods for trajectory optimization”. In: *Tutorial, Cornell University, Feb.*

Mariano-Hernández, D et al. (2021). “A review of strategies for building energy management system: Model predictive control, demand side management, optimization, and fault detect & diagnosis”. In: *Journal of Building Engineering* 33, p. 101692.

Michalik, Claas, Ralf Hannemann, and Wolfgang Marquardt (2009). “Incremental single shooting—a robust method for the estimation of parameters in dynamical systems”. In: *Computers & Chemical Engineering* 33.7, pp. 1298–1305.

Saurav, Kumar and Vikas Chandan (2017). “Gray-box approach for thermal modelling of buildings for applications in district heating and cooling networks”. In: *Proceedings of the Eighth International Conference on Future Energy Systems*, pp. 347–352.

Schittkowski, Klaus (2002). *Numerical data fitting in dynamical systems: a practical introduction with applications and software*. Vol. 77. Springer Science & Business Media.

Shitahun, Alachew et al. (2013). “Model-Based Dynamic Optimization with OpenModelica and CasADi”. In: *IFAC Proceedings Volumes* 46.21. 7th IFAC Symposium on Advances in Automotive Control, pp. 446–451. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20130904-4-JP-2042.00166. URL: https://www.sciencedirect.com/science/article/pii/S1474667016384117.

Wetter, M. et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270.

Wright, Stephen, Jorge Nocedal, et al. (1999). “Numerical optimization”. In: *Springer Science* 35.67-68, p. 7.

Yu, Xingji et al. (2019). “Investigation of the model structure for low-order grey-box modelling of residential buildings”. In: *Proc. Build. Simul.*

Presentation, Validation and Application of the *EnergyProcess* Library

François Nepveu[1], Sylvain Mathonnière[2], Gaël Enée[1], Nicolas Lamaison[2]

[1] CEA, CEA Pays de la Loire, 44343 Bouguenais cedex, France,
{francois.nepveu,gael.enee}@cea.fr

[2] Univ. Grenoble Alpes, CEA, Liten, Campus Ines, 73375 Le Bourget du Lac, France,
{sylvain.mathonniere,nicolas.lamaison}@cea.fr

Abstract

Green production of hydrogen and its derivatives is becoming a cornerstone of industry decarbonation. Apart from the technological development point of view, optimizing the overall production chains dynamically is essential for the competitiveness of these systems.

In this paper, we describe how we built, validated and used a Modelica-based library dedicated to the simulation and optimization of energy process for the production of green molecules. Especially, models of complex media, salt cavity hydrogen storage and electrolysis module are presented.

An example application shows that the models of the library are particularly handy for the modeling of a 5MW electrolysis module, which is used for the calibration of an optimization model.

Keywords: green molecules, dynamic simulation, production optimization

1 Introduction

Emissions reduction of industries such as petroleum refining or fertilizer production is challenging. Green hydrogen and its derivatives such as ammonia are expected to play a significant role in their decarbonation since alternative solutions are unavailable or difficult to implement. Currently, low-carbon hydrogen accounts for less than 1% of hydrogen production, but in the Net-Zero Scenario of IEA by 2050, it is expected to be massively used (IEA, 2022).

However, green hydrogen production is about 6 times more expensive than traditional hydrogen production. Thus, in order to reduce the specific costs, very large plants are planned to be built (IEA, 2022). For these plants, optimizing the design and more specifically the operation with the use of software becomes then decisive for cost competitiveness issues.

Our research group is currently involved in several research programs devoted to the definition of optimal operation of production chains for green molecules. More specifically, these research programs aim at building Model Predictive Controllers (MPC) to be used in the Energy Management System (EMS) of such plants. In this context, we have been working on the development of accurate and efficient dynamic multi-physics models for which the objectives are two-fold:

- The calibration of MPC models for the development of the EMS;
- The evaluation of the EMS on a physical simulator.

The purpose of this paper is thus to present the *EnergyProcess* library and show its usage for optimization models development and validation. After the presentation of the context and objectives in the present section, the rest of the paper is organized as follows:

- Section 2 presents a review of existing Modelica libraries for the modelling of energy processes dedicated to green molecules production;
- Section 3 introduces the core of the library with special focuses on the media, electrolyzer and storage packages;
- Section 4 highlights an example of model validation and usage for optimization purposes with the description of a 5 MW electrolysis module;
- Section 5 summarizes the main messages of the present article and gives perspectives for the development roadmap regarding the library.

2 Review of existing libraries

Various Modelica-based libraries exist for the modelling of chemical energy processes with some library specifically dedicated to the decisive impact of the modelling of media. The present section displays a review of these existing libraries for media modelling on the one hand (see Table 1) and chemical processes system modelling on the other hand (see Table 2). Criteria such as compatibility with the Modelica Standard Library (Modelica Association and contributors, 2020), open source accessibility, H2 systems modelling capabilities, and consistency with the objectives of the present library are discussed.

DOI
10.3384/ecp204311

Table 1. Review of Media libraries

Library Name	***Description***	***Stronger points***	***Weaker points***
MSL Media	Media package of Modelica Standard Library	Free	Limited available media models
External Media (Casella, 2006)	Allows Modelica.Media compatible interfacing with external codes	Compatible with MSL Free with access to Coolprop (Bell et al., 2014)	Computationally heavy Limited for mixture Difficulties for FMU translation
TILMedia Suite (TLK-Thermo GmbH, 2020)	Properties for incompressible liquids, ideal gases and real fluids containing a vapor liquid equilibrium	Exhaustive list of pure substances and mixtures	Only partly free Limited for mixture Incompatible with MSL
MultiPhaseMixture Media (Windahl et al., 2015)	Modelica.Media compatible framework for thermodynamic properties including an external C/C++ interface	Allows usage of C++ state-of-the-art non-linear systems resolution Free	Not maintained since 2016

Table 2. Review of Chemical Energy Processes libraries

Library Name	***Description***	***Stronger points***	***Weaker points***
Transient (Andresen et al., 2015)	Dedicated to coupled energy networks with recent versatile electrolyzer models (Webster and Bode, 2019)	Large scope of models Models with different labeled levels of accuracy	Incompatible with MSL Based on TILMedia thus only partly free
Hybrid (IdahoLab, 2023)	Dedicated to various integrated energy systems including, among others, nuclear, electrolysis and desalination	Very large scope of models Compatible with MSL Free	H2 related processes not user friendly Still on-going structural changes
ThermoSyspro (El Hefni and Bouskela, 2019)	Dedicated to power plants and energy systems	Free Large scope and detailed models regarding thermal processes	Incompatible with MSL No media Not related to H2 modelling
Hydrogen (Dassault Systèmes, 2023)	Dedicated to PEM fuel cell stacks and systems	Compatible with MSL Efficient real gas model compatible with MSL (Kormann and Krüger, 2019)	Not free Limited models scope
Modelon FCL (Modelon, 2023)	Dedicated to PEM and Solid Oxide Fuel cell systems	Detailed models Own exhaustive media library	Not free Incompatible with MSL Limited models scope
ThermoFluidStream (Zimmer et al., 2021)	Dedicated to complex thermofluid architectures	Free Very robust modeling due to a new computational scheme	Incompatible with MSL Not related to H2 modelling

Among all the libraries, the most promising one for Media modelling is *ExternalMedia* (Casella, 2006). However, prohibitive calculation time, difficulties to generate Functional Mock Up (FMU) and limited mixture modelling capabilities with Coolprop (Bell et al., 2014) led to the decision to develop our own media package. Moreover, many different reasons led us to develop our own library of H2 components. First of all, the necessity to control the development, architecture as well as the components layout and complexity of the physics implemented combined with time and intellectual property constraints required by clients projects were hardly compatible with basing our library on an existing one. Second, another library on District Heating (Giraud et al., 2015) has been developed in our laboratory based on similar requirements and was used as a reference in the development of this library to ensure compatibility for complex system description. Lastly, open-source development requires development standards (naming, formulations, etc.) and community work not easily compatible with projects time constraints. Nevertheless, efforts are being carried to format the library (packages structure, modularity, coding standards, and internal documentation) in order to eventually release it as open-source.

3 Presentation of the Library

3.1 Overall presentation

The CEA Energy Process library (CEPL) has been historically developed for hydrogen systems modelling following requests for the modelling of industrial systems. Nowadays the focus of the library is still largely on H2 models but NH3 models (synthesis reactor, complex media) are being actively developed to study power-to-green ammonia system based on an electrolyzer and a Haber-Bosh reactor. The CEPL, built with the software Dymola®, is compatible with the MSL library thus providing direct compatibility with another CEA library of models for district heating modelling named District Heating. (Giraud et al., 2015) The library is being actively developed on our internal forge using Git, therefore a methodology of continuous integration based on non-regression tests using the Testing library from Dassault (Hammond-Scott, 2022) is being followed.

3.2 Structure

The library is structured on packages, similarly to the MSL as depicted in Figure 1. The core packages are the *Electrolyzers (FuelCell)*, *Storage* and *Media* packages. They will be further detailed in the present paper. The development of the package *Reactor* is still on-going with the objective to extend the library towards other green molecules such as NH3. The packages *Machines,* *HeatExchangers*, *Condensers* and *GeneralCircuitry* contain more traditional models oriented for use in a hydrogen system. Those models are similar to the MSL models but have been slightly modified either to increase computational speed or improve overall robustness.

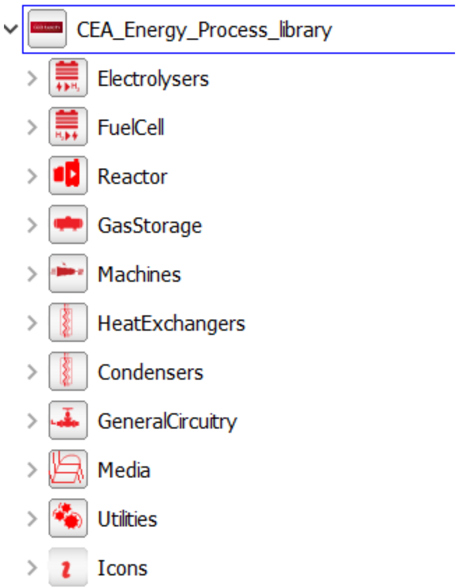


Figure 1. Structure of the *EnergyProcess* Library

3.3 Focus on Media package

The *Media* package is at the core of the present library. Media models must be numerically robust to avoid generating errors during dynamic simulation as well as fast, accurate, and capable of calculating the thermodynamic properties of the fluid over a wide range of pressures and temperatures. Models with best accuracy, such as models based on Helmholtz's free energy, suffer from slowness because they are computationally intensive. On the other end of the media model spectrum, there are fast models with few calculations but which are imprecise (ideal gas models). This observation is driving the development of new media models for CEPL.

The models of the *Media* package are divided into two categories: Monophasic and Diphasic media. Monophasic models are based on SRK (Soave Redlich Kwong) Equation of state (Soave, 1972), SBLT (Spline Based Look-up Table)(Ungethüm and Hülsebusch, 2009) or Helmholtz models using pressure and temperature as explicit variables. Diphasic models for pure substance are based on SRK, Helmholtz and SBLT as well using pressure and specific enthalpy as explicit variables. Mixture models are the most complex of all and involve several models. They are either based on SRK/PSRK models or in-house models (mixture NH3 / H2 / N2 or Moist gas) based on Henry and Raoult's laws coupled with correlations for the Vapour Liquid Equilibrium (VLE) calculation (Alesandrini et al., 1972; Reamer and Sage, 1959a, 1959b; Sawant et al., 2006). The media composing the package are represented in Figure 2.

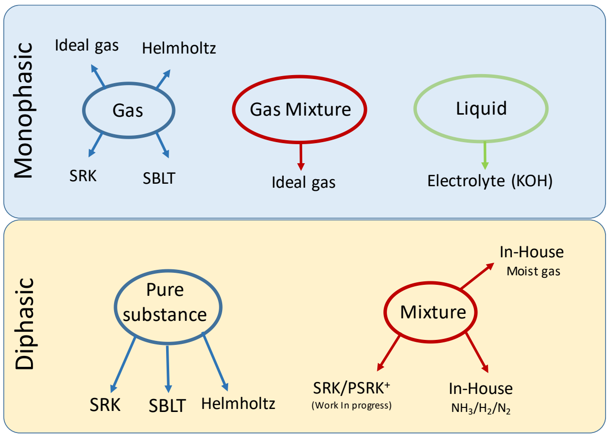


Figure 2. *Media* package of the library including monophasic and diphasic models for pure substance and mixture.

3.4 Focus on Electrolyzers and FuelCell packages

The *Electrolyzers* package is composed of three subpackages, each addressing a specific electrolyzer technology: PEM (Proton Exchange Membrane), SOEC (Solid Oxide Electrolyze cell) and Alkalin. The PEM subpackage has a dedicated electrolyzer model, which itself includes several replaceable physics package such as voltage, temperature, converter and mass flow models. The latter model structure is largely inspired by the model from Webster and Bode (2019).

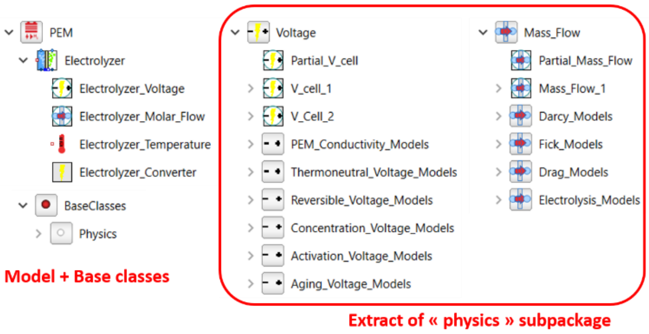


Figure 3. On the left, the electrolyzer model in the PEM package including the "physics" package. On the right, an extract of the "physics" package highlighting the voltage and mass flow replaceable subpackages.

Figure 3 represents an extract of the PEM highlighting its “physics” package. Several voltage models are represented in the “physics” package ranging from activation loss, concentration loss and reversible potential up to aging model (Espinosa-López et al., 2018; Falcão and Pinto, 2020; Olivier et al., 2017). The aging model is an empirical model that takes into account the history of fluctuation of power output of the electrolyzer to determine voltage losses due to stack/cells degradation. The strength of this approach lies in its modularity allowing the user to easily change physical modules, insert its own or to fit parameters to match experimental data (as shown later in Section 4.2). The other subpackages SOEC and Alkalin share this modular approach with different physics.

The remaining of the main package is composed of auxiliary models such as deoxodryer, scrubber, lye-gas separation or water gas separation models to assist in the modelling of the balance of plant for each technology.

The *FuelCell* package is very similar in structure to the *Electrolyzers* one. It incorporates PEM (Proton Exchange Membrane) and SOFC (Solid Oxide Fuel Cell) models and uses the same modular structure than the *Electrolyzers* package.

3.5 Focus on Storage package

The *Storage* package includes a compressed gas storage to model type I to type IV composite tank (thermo-hydraulic modeling). This package includes as well an underground, salt-cavity based, storage model (Gabrielli et al., 2020). The particularity of this model is to represent the diffusion of gas through porous rock using Fick's law in a 1D model. It takes into account the viscosity of the gas as well as the tortuosity of the rock to model diffusion. Figure 4 represents a schematic view of the physical model. The first node is the underground storage cavern from/to which the gas can be extracted/injected. From node 1, the gas is allowed to diffuse to node 2 up to node n through Fick's law. Equations of diffusion are fully reversible and help representing the latency of back diffusion during extraction. The effect is particularly pronounced for long and thin cavern that are created by lixiviation in situ.

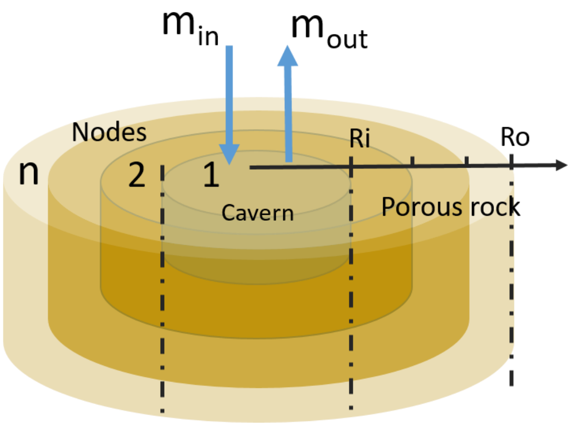


Figure 4. 1-D model of the underground gas storage.

3.6 Other packages

The remaining packages are composed of models used in the balance of plant of hydrogen system. Most of those models are inspired by the MSL but modified for computational performances or stability considerations. Among those models, the modular, quasi static or dynamic heat exchangers model using the ε-NTU model as well as the compressor model stands out as the most complex.

4 Validation and Application

4.1 Focus on Media package

As discussed in section 3.3, the media package is at the core of the *EnergyProcess* Library. A wrong choice by the user on the selection of the media model can lead to problematic physical results and/or extensive CPU time calculations.

Performances of the pure substance media models available in the *EnergyProcess* Library are studied on the classical use case of a fast filling of hydrogen tanks (Figure 5). The filling process consists of:

- A hydrogen source at constant mass flow of 0.05 kg/s.
- A buffer (1 m^3) and a discharge valve opening at 150 Bar.
- A rack of hydrogen tanks with a total volume of 5 m^3.
- The hydrogen source is piloted with a hysteresis loop to fill up the tanks at a maximal pressure of 750 Bar.

Simulations of the process are performed using the pure substance models composing the media package:

- Monophasic and diphasic SRK models using a cubic equation of state.
- Monophasic and diphasic Helmholtz models.
- Ideal gas model from Modelica Standard Library.

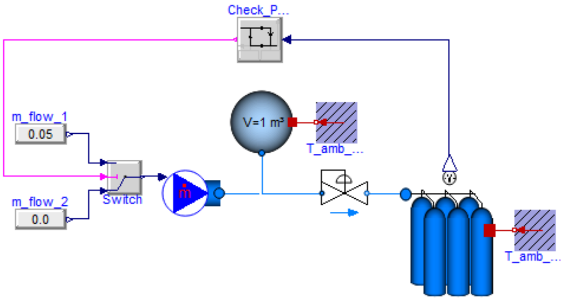


Figure 5. Process of a fast filling of hydrogen tanks

The obtained results are compared to a reference model from the ExternalMedia library (Table 1). Figure 6 shows the changes of gas pressure and gas temperature inside the tanks during the complete fast filling process with the ideal gas, monophasic SRK, monophasic Helmholtz and ExternalMedia hydrogen models. The CPU time with a Radau IIa solver required for the integration on the time simulation are also indicated. Classically, at high pressure and low temperature, ideal gas model is very imprecise in comparison to the three other models but it is very fast. ExternalMedia and Helmholtz models give similar physical results and CPU time calculations. This observation is logical because the ExternalMedia model is itself based on a Helmholtz formulation from C++ Coolprop library. Finally, SRK model is a very good trade-off between CPU time calculations and physical results.

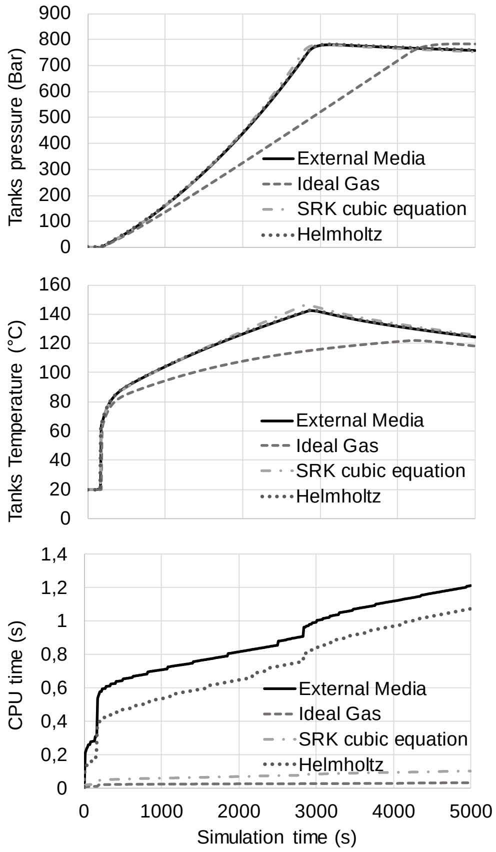


Figure 6 : Hydrogen pressure (up) and Temperature (middle) inside the tanks during the fast filling process obtained with 4 media models. Associated CPU time calculations (down).

Table 3 gives the results for the six hydrogen models focusing on CPU time and the mass of hydrogen stored in the tanks at the end of filling process. ExternalMedia is used as the reference model to compare results.

Table 3 : Comparison between six Hydrogen models during a fast filling process

Media model	*CPU time (s)*	*Stored H2 mass (kg)*
External Media	1.215	101
Ideal gas	0.034 (-97 %)	145 (+44 %)
Monophasic SRK	0.103 (-91 %)	99 (-2 %)
Diphasic SRK	0.157 (-87 %)	99 (-2 %)
Monophasic Helmholtz	1.074 (-11 %)	101 (+ 0 %)
Diphasic Helmholtz	2.418 (+99 %)	101 (+0 %)

4.2 5 MW PEM electrolysis module modeling

In this section, the *EnergyProcess* Library is used to develop a non-linear dynamic model of the Ex-2125D electrolyzer system manufactured by Plug Power (2023). This 5 MW electrolyzer module is composed of:

- Five Allabash PEM (Proton Exchange Membrane) stacks originally developed by GingerELX. Each stack is considered to be composed of 154 cells of 1250 cm² in series operating on a nominal current of 3750 A (current density of 3 A/cm²), a pressure of 41 bar and a temperature of 70°C (343.15 K). NREL performed polarization (I-V curve) scans from 350 to 3750 A, while maintaining cathode pressure and stack temperature at its nominal values (Harrison, 2021).
- A balance of plant including the following main components: two gas/water separator for the stack output conditioning, a feed water/cooling circuit composed of a heat exchanger, a water pump and valves for the stack temperature regulation.
- Additionally, a deoxo-dryer system is needed to purify hydrogen, an AC/DC converter to supply the DC current of the stacks from AC source and a dry cooler to evacuate heat from cooling circuit.

At first, to validate the stack model, polarization curve using the cell voltage calculation by Webster and Bode (2019) and Afshari et al. (2021) are compared to the NREL data (Harrison, 2021). The resulting I-V curves are given in Figure 7. Results shows a good match between both models and NREL measurements. Moreover, The Webster approach is simply correlated to measurements (RMSE = $4.6.10^{-3}$ V, 0.25 %) using a constant additional ohmic resistance (due to electrode, a lower membrane humidity or a contact resistance) for the ohmic overvoltage calculation.

Then, as shown in Figure 8, the Ex-2125D electrolyzer system is assembled from components model of the *EnergyProcess* Library. Each component (heat exchanger, pump, valves, dry-cooler) is sized at the nominal values of 3750 A, 70°C corresponding to an electric power consumption of 5.9 MWe.

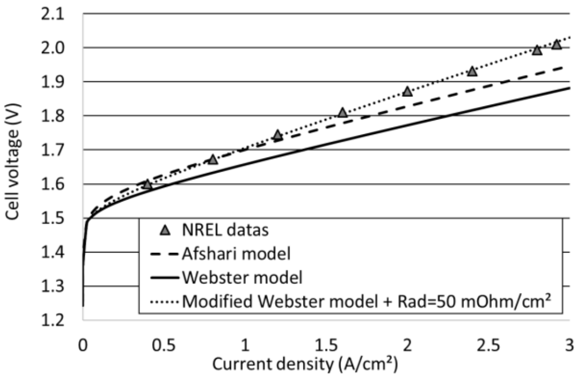


Figure 7. Polarization curves of Allabash cell from models and NREL data at 70°C and 41 bar

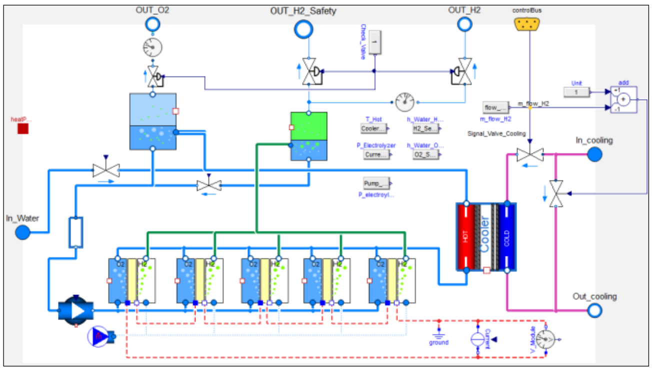


Figure 8. Ex-2125D electrolyzer system: From Physical to virtual system with *EnergyProcess* Library

The stack model includes a temperature submodel which implements a lumped thermal capacitance model and thermal losses (radiation and convection). The thermal capacity of the Allabash PEM stack is determined from geometry defined by Tiktak (2019) to be 219 kJ/K value.

The control-command system consists of three valves to pilot:

- The feed-in water directly as a function of the water consumption by electrolysis reaction in the five stacks.
- The level of water in both separators. The valve is used to discharge the overwater level from H2/water separator to O2/water separator due to pressure difference between the anodic and cathodic circuits and water crossover trough the cell membranes.
- The temperature of the stack using a bypass of a fraction of the flowrate of the cooling fluid in the cooler.

Component models are associated with a media package. In-house Moist O2 (using ideal gas model for gas phase and constant properties for liquid water) and Moist H2 (using SRK model for gas phase and constant properties for liquid water) mixtures (see Figure 2) are selected for anodic and cathodic circuits.

Simulations are performed to study dynamic or quasi-static response of the Ex-2125D electrolyzer and study system performances. For example, Figure 9 gives the evolution of the stack temperature in response to a step of the electric load (trapezoid source from MSL with 5 minutes of rising duration):

- Using a constant opening of the bypassing valve of the cooler.
- Using a PID regulator to control the opening of the bypassing valve as a function of the outlet temperature of the stacks.

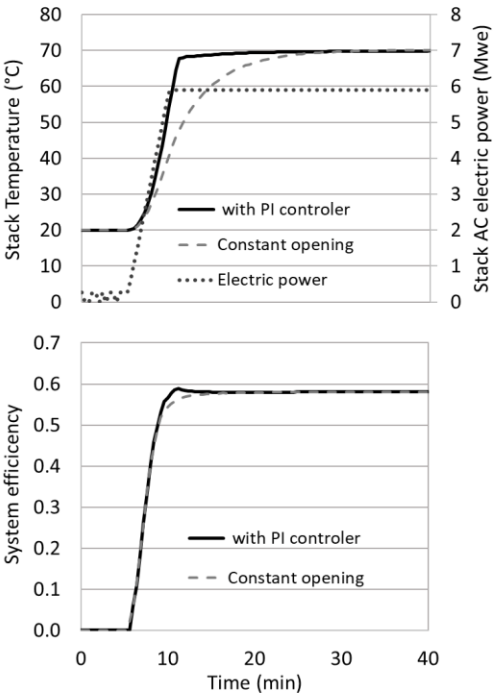


Figure 9. Stack temperature (Up) and module efficiency (Down) evolution in response to a step of the electric load with or without PI controller

4.3 Use for calibration of optimization model

With the physical model of the module, we were able to characterize its behavior for the entire range of current density and at a temperature of operation of 70°C. Figure 10 presents the results obtained in terms of efficiency with details regarding the different contributors to the efficiency loss. With that result, a model following the Mixed-Integer Linear Programing formalism could be built by piecewise linearization as shown in Figure 11. This model can then be used within the high-level controller model of an Energy Management System (EMS), as shown in next Section.

DOI
10.3384/ecp204311

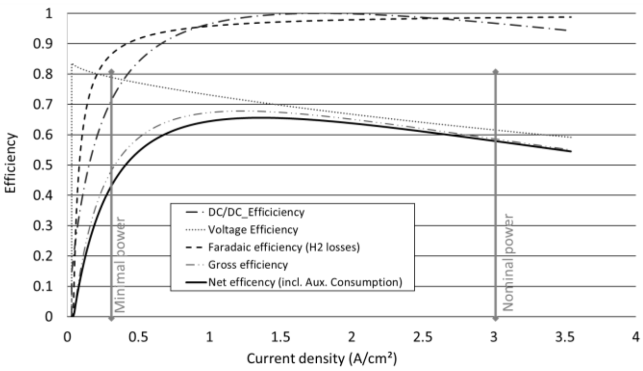


Figure 10. Different simulated contributions to the loss of efficiency and final net efficiency of the module

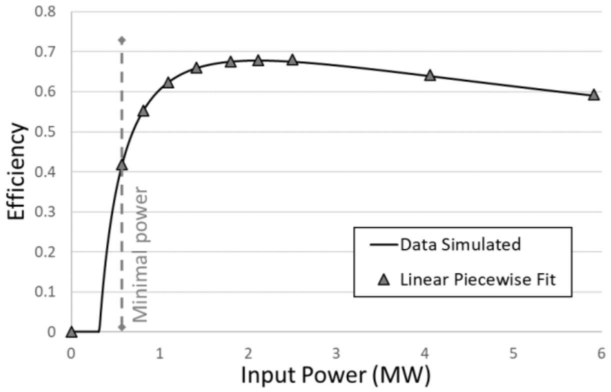


Figure 11. Piecewise fit of the part-load efficiency curve calculated using the *EnergyProcess* model

4.4 Use as a simulator for EMS evaluation

A complete green hydrogen production chain simulator was built with the use of the *EnergyProcess* library combining different modules of Section 4.1 in parallel, a salt cavern gas storage, a transportation pipeline and different compressors. This physical non-linear simulator was then used to evaluate the performance of a MILP-based predictive EMS prior to field deployment. Figure 12 explains schematically the interaction between the physical simulator and the EMS. The latter receives i) predictions of cost and H2 demand for the next 24 hours and ii) state-returns from the physical simulator (e.g. state-of-charge of the storage) and will calculate set points (e.g. input power to the different modules) for the next 24 hours at a time step of e.g. 15 minutes. The set points calculation takes place every hour. Figure 12 also highlights the other use of the *EnergyProcess* library, which is the off-line calibration of MILP models (as explained in Section 4.2).

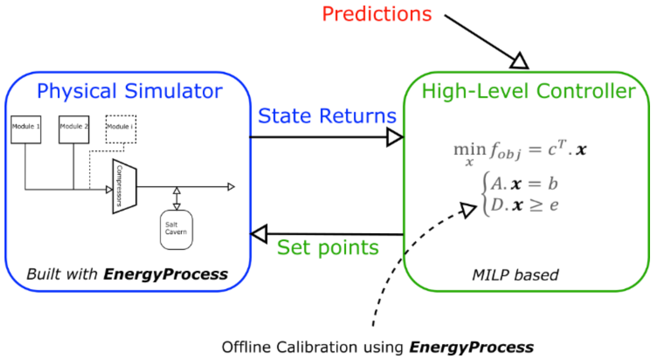

Predictions
Physical Simulator
Module 1
Module 2
Module i
Compressors
Salt Cavern
Built with EnergyProcess
State Returns
Set points
High-Level Controller
$\min_{x} f_{obj} = c^T.\boldsymbol{x}$
$\begin{cases} A.\boldsymbol{x} = b \\ D.\boldsymbol{x} \geq e \end{cases}$
MILP based
Offline Calibration using EnergyProcess

Figure 12. Schematic of the *EnergyProcess* library uses as i) physical simulator and ii) offline calibration tool for MILP model

5 Conclusion

In this paper, we describe how we built and used the *EnergyProcess* library, a Modelica-based library dedicated to the simulation and optimization of energy process for the production of green molecules. After a review of existing libraries, the development of this new library finds its roots in our need for i) media models computationally light and able to represent various multiphasic mixtures, ii) FMU and MSL compatible models, iii) exhaustive component models for various green molecules production chains, and iv) intellectual property and continuous integration control.

The structure of the library was then presented with a specific focus on the core packages, i.e. *Media*, *Electrolyzers* and *Storage*. The specific application of a 5 MW PEM electrolysis module was finally addressed with i) the polarization curve validation for a single stack, ii) the dynamic performance simulation of the entire module, iii) the optimization model calibration of this module and iv) the use of a complete simulator including this module to characterize an Energy Management System (EMS). The authors do not see any limitation in extending the electrolyzer model to smaller scale. As for larger scale, a simplified version (for the sake of computational time) of the 5 MW model was used as a subsystem to simulate a 500 MW electrolyzer in one project.

The development of the library is still on-going with for example the current modeling of ammonia synthesis catalytic reactor. Apart from the initial ambitions of the library which were to either be used for control models calibration or as a physical simulator to evaluate EMS, further uses are currently envisioned. The latter specifically comprises data reconciliation and surrogate model to be used inside the EMS.

Although the *EnergyProcess* library is limited to our internal use for now, a wider diffusion of some components under an open-source license is envisioned in the frame of a European project.

Acknowledgements

We wish to thank the CEA internal TRILOGY project for it financial support in the development of the library. This project aims at developing a simulation and optimization platform for energy networks at local, industrial, district and national scales. An illustration of the project accomplishment can be seen at Rava et al. (2022).

References

Afshari, E., Khodabakhsh, S., Jahantigh, N., Toghyani, S., 2021. Performance assessment of gas crossover phenomenon and water transport mechanism in high pressure PEM electrolyzer. International Journal of Hydrogen Energy, Fuel cell and hydrogen energy technology 46, 11029–11040. https://doi.org/10.1016/j.ijhydene.2020.10.180

Alesandrini, C.G., Lynn, S., Prausnitz, J.M., 1972. Calculation of Vapor-Liquid Equilibria for the System NH3-N2-H2-Ar-CH4. Ind. Eng. Chem. Proc. Des. Dev. 11, 253–259. https://doi.org/10.1021/i260042a017

Andresen, L., Dubucq, P., Peniche Garcia, R., Ackermann, G., Kather, A., Schmitz, G., 2015. Status of the TransiEnt Library: Transient Simulation of Coupled Energy Networks with High Share of Renewable Energy. Presented at the The 11th International Modelica Conference, pp. 695–705. https://doi.org/10.3384/ecp15118695

Bell, I.H., Wronski, J., Quoilin, S., Lemort, V., 2014. Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp. Ind. Eng. Chem. Res. 53, 2498–2508. https://doi.org/10.1021/ie4033999

Casella, F., 2006. ExternalMedia. Modelica 3rd-party libraries.

Dassault Systèmes, 2023. Systems Hydrogen Library (HYZ) [WWW Document]. URL https://cloud.academy.3ds.com/explorer/r2019x/role_hyz.html (accessed 5.17.23).

El Hefni, B., Bouskela, D., 2019. Modeling and Simulation of Thermal Power Plants with ThermoSysPro: A Theoretical Introduction and a Practical Guide. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-05105-1

Espinosa-López, M., Darras, C., Poggi, P., Glises, R., Baucour, P., Rakotondrainibe, A., Besse, S., Serre-Combe, P., 2018. Modelling and experimental validation of a 46 kW PEM high pressure water electrolyzer. Renewable Energy 119, 160–173. https://doi.org/10.1016/j.renene.2017.11.081

Falcão, D.S., Pinto, A.M.F.R., 2020. A review on PEM electrolyzer modelling: Guidelines for beginners. Journal of Cleaner Production 261, 121184. https://doi.org/10.1016/j.jclepro.2020.121184

Gabrielli, P., Poluzzi, A., Kramer, G.J., Spiers, C., Mazzotti, M., Gazzani, M., 2020. Seasonal energy storage for zero-emissions multi-energy systems via underground hydrogen storage. Renewable and Sustainable Energy Reviews 121, 109629. https://doi.org/10.1016/j.rser.2019.109629

Giraud, L., Bavière, R., Vallée, M., Paulus, C., 2015. Presentation, Validation and Application of the DistrictHeating Modelica Library. Versailles, France, p. 10. https://doi.org/10.3384/ecp1511879

Hammond-Scott, H., 2022. Testing Library: A Look Inside. Claytex. URL https://www.claytex.com/tech-blog/testing-library-a-look-inside/ (accessed 6.12.23).

Harrison, K., 2021. MW-Scale PEM-Based Electrolyzers for RES Applications: Cooperative Research and Development Final Report, CRADA Number CRD-18-00742. Renewable Energy.

IdahoLab, 2023. HYBRID. Idaho National Laboratory.

IEA, 2022. Hydrogen – Analysis [WWW Document]. IEA. URL https://www.iea.org/reports/hydrogen (accessed 5.17.23).

Kormann, M., Krüger, I.L., 2019. Application of a Real Gas Model by Van-der-Waals for a Hydrogen Tank Filling Process. Presented at the The 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019, pp. 665–670. https://doi.org/10.3384/ecp19157665

Modelica Association and contributors, 2020. Modelica Standard Library. https://github.com/modelica/ModelicaStandardLibrary.

Modelon, 2023. Fuel Cell Library [WWW Document]. Modelon. URL https://modelon.com/library/fuel-cell-library/ (accessed 5.17.23).

Olivier, P., Bourasseau, C., Bouamama, Pr.B., 2017. Low-temperature electrolysis system modelling: A review. Renewable and Sustainable Energy Reviews 78, 280–300. https://doi.org/10.1016/j.rser.2017.03.099

Plug Power [WWW Document], 2023. URL https://www.plugpower.com/, https://www.plugpower.com/ (accessed 6.12.23).

Rava, L., Wantier, A., Vallee, M., Ruby, A., Lamaison, N., 2022. Assessment of varying coupling levels between electric and thermal networks at district level using Co-simulation and model-predictive Control. Presented at the ECOS 2022 - Efficiency, Cost, Optimization and Simulation of energy conversion systems and processes, Copenhague, Denmark.

Reamer, H.H., Sage, B.H., 1959a. Phase Behavior in the Hydrogen-Ammonia System. J. Chem. Eng. Data 4, 152–154. https://doi.org/10.1021/je60002a012

Reamer, H.H., Sage, B.H., 1959b. Phase Behavior in the Nitrogen-Ammonia System. J. Chem. Eng. Data 4, 303–305. https://doi.org/10.1021/je60004a005

Sawant, M.R., Patwardhan, A.W., Gaikar, V.G., Bhaskaran, M., 2006. Phase equilibria analysis of the binary N2-NH3 and H2-NH3 systems and prediction of ternary phase equilibria. Fluid Phase Equilibria 239, 52–62. https://doi.org/10.1016/j.fluid.2005.10.014

Soave, G., 1972. Equilibrium constants from a modified Redlich-Kwong equation of state. Chemical Engineering Science 27, 1197–1203. https://doi.org/10.1016/0009-2509(72)80096-4

Tiktak, W.J., 2019. Heat Management of PEM Electrolysis.

TLK-Thermo GmbH, S., 2020. TIL Media Suite [WWW Document]. URL https://www.tlk-thermo.com (accessed 5.17.23).

Ungethüm, J., Hülsebusch, D., 2009. Implementation of a Modelica Library for Smooth Spline Approximation. Presented at the The 7 International Modelica Conference, Como, Italy, pp. 669–675. https://doi.org/10.3384/ecp09430013

Webster, J., Bode, C., 2019. Implementation of a Non-Discretized Multiphysics PEM Electrolyzer Model in Modelica. Presented at the The 13th International Modelica

Conference, Regensburg, Germany, March 4–6, 2019, pp. 833–840. https://doi.org/10.3384/ecp19157833

Windahl, J., Pr&ouml, Lss, K., Bosmans, M., Tummescheit, H., van Es, E., Sewgobind, A., 2015. MultiComponentMultiPhase - A Framework for Thermodynamic Properties in Modelica.

Zimmer, D., Meißner, M., Weber, N., 2021. The DLR Thermofluid Stream Library.

Import and Export of Functional Mock-up Units in CasADi

Joel A. E. Andersson[1]

[1]Freelance software developer and consultant, Madison, Wisconsin (USA) joel@jaeandersson.com

Abstract

This paper presents the recently added support for import and export of functional mock-up units (FMUs) in CasADi, an open-source software framework for numerical optimization. Of particular interest is the efficient calculation of derivatives, especially in the context of sensitivity analysis and dynamic optimization. We show how the import interface allows for both first and second derivatives can be efficiently and accurately calculated and – importantly – validated for correctness. We also outline the FMU export interface, which leverages CasADi mature and efficient support for forward and adjoint derivative calculation and C code generation. Finally, potential future developments of the support are discussed.

Keywords: CasADi, FMI, FMU, Modelica, optimal control

1 Introduction

The work presented here is intended to be general-purpose, but will typically include some sort of optimization formulation with a high-fidelity simulation model adhering to the FMI standard, cf. Section 1.2 below. Examples of such applications include:

- Parameter estimation applications, which may use parametric sensitivty approaches to obtain estimates of confidence intervals for estimated parameters
- A wide range of different optimal control formulations, i.e. problems with free control trajectories to be determined by the optimization algorithm
- Optimization-based control techniques such as (nonlinear) model predictive control (MPC), including their deployment on embedded systems
- Steady-state optimization formulations, i.e. problems that lack time derivatives or have the time derivatives fixed to some value (typically zero)

For more details on the implementation of such algorithms and on possible applications, we refer to the various textbooks on the topic, including (Biegler 2010) and (Rawlings, Mayne, and Diehl 2020).

1.1 CasADi

CasADi (J. A. E. Andersson et al. 2019) is an open-source software package for C++, Python, MATLAB and Octave. CasADi offers a flexible approach to solve numerical optimization problems in general and numerical optimal control in particular. At the lowest level, it offers all the building blocks needed to efficiency address all the problem formulations listed above. At the core of the package is a symbolic expression framework implementing *algorithmic differentiation* (AD) in forward and reverse (adjoint) modes. CasADi's symbolic expressions can contain embedded *function objects*, which offer a standard interface to generic, differentiable functions. These function objects can be defined in a number of ways, including by other symbolic expressions, by systems of nonlinear equations, by initial-value problems in differential-algebraic equations or by external function calls.

1.2 FMI

The *functional mock-up interface* (FMI) (Modelica Association 2020; Modelica Association 2023) is an open standard for exchanging information about dynamic system models between tools. The format specifies the structure of self-contained zip archives called *functional mock-up units* (FMUs), The FMUs contain, in particular, an XML file with static meta information and a C library for evaluating model equations and their derivatives. The C library is designed for either static linking or dynamic linking and can be distributed either in source form or as a compiled dynamically linked library (DLL), available in the FMU.

The FMI standard describes connections either at the dynamic equation level, referred to as *model exchange*, or on an input-output level at discrete communication time points, referred to as *co-simulation*. In this work, we are mainly concerned with the connections at the level of dynamic equations and references to "FMI" implicitly implies FMI for model exchange.

Derivative information in FMUs

FMI 2.0 (Modelica Association 2020) specifies two types of derivative information:

- Firstly, the FMI XML API contains information about which output variables depend on what input variables (`dependencies` attribute) and whether this dependency is linear (`dependenciesKind` attribute).
- Secondly, the FMI C API includes the routine for calculating (forward) directional derivatives i.e. Jacobian-times-vector products, for selected subsets of the FMU inputs and outputs

(fmi2GetDirectionalDerivative function).

Together, these two pieces of information can be used to obtain sparse Jacobians, as outlined in Section 3.2.

In FMI 3.0 (Modelica Association 2023), where forward directional derivatives are calculated using the routine fmi3GetDirectionalDerivative, the C API is extended with the ability to calculate adjoint directional derivatives, i.e. Jacobian-transposed times vector products, via the routine fmi3GetAdjointDerivative. Furthermore, the variable dependency information in the XML API can now be refined at runtime, with the addition of fmi3GetVariableDependencies in the C API.

2 Other integration efforts of CasADi and Modelica

This work does not represent the first time Modelica models has been integrated into CasADi. Indeed, it was a Modelica application that motivated the start of the CasADi project in 2009 (Ahlbrink et al. 2009). The first integration effort, between JModelica.org and CasADi, was described in (J. Andersson et al. 2011). That approach was based on the FMI version 1.0, where the XML model description had been extended with a symbolic representation of the model equations. Later, support for export of the same format was also added to OpenModelica (Shitahun et al. 2013).

Eventually, JModelica.org replaced the XML-based format with a tighter integration based on a Java interface generated using SWIG (Beazley 2003), an approach which is also used in OCT, JModelica.org's closed-source successor code from AB Modelon. As of this writing, the CasADi-backend of OCT represents the most mature *symbolic* coupling between generic Modelica models and CasADi.

Two additional interfaces between Modelica and CasADi are available via the open-source Pymoca[1] and Cymoca[2] packages on Github. Both these packages include native CasADi-based backends, using CasADi's Python and C++ APIs, respectively.

3 Importing FMUs into CasADi

CasADi 3.6 introduces the ability to import standard FMUs, adhering to FMI version 2.0, which can be generated from Modelica or non-Modelica models. Unlike the other efforts described in Section 2, this approach uses the standard C API, as defined by the FMI standard, for evaluating model equations and any derivative information.

3.1 Postponing the creation of CasADi function objects

A fundamental difficulty with all the integration efforts described in Section 2 is that not all Modelica expressions can be efficiently represented as CasADi expressions. In particular, CasADi does not support arithmetics involving string-valued expressions and expressions where the dimensions or sparsity patterns are unknown. There are also fundamental differences in how flow control can be implemented, e.g. if-then-else statements and for/while loops.

In the new FMI support of CasADi, we are able to overcome these limitations by *postponing the creation of CasADi functions objects*. Rather than creating a CasADi function directly from the FMU – which is how CasADi's other foreign function interfaces work – the FMU is imported in the form of a *mutable* representation of the physical model. This mutable representation, in the form of instances of the DaeBuilder class in CasADi, allows the user to change properties, set values and perform certain manipulations before an immutable (stateless) CasADi function object is finally created. The corresponding function objects are instances of the newly added FmuFunction class, and upon creation saves a snapshot of the current DaeBuilder state. Each FMU function object can have multiple vector-valued inputs and multiple vector-valued outputs, where the user defines the composition of each input or output. We typically only include the real-valued, differentable model variables that are manipulated by the various simulation and optimization algorithms available in CasADi. Inputs that are non-differentiable – including string-valued and integer-valued variables – or known to be fixed are assumed to be set prior to the creation of FmuFunction instance. Calculated quantitites of interest that are non-differentiable, and hence cannot be used in simulation or optimization algorithms in CasADi, can be obtained via the statistics-functionality of the FMU function objects.

3.2 Derivative calculation

The typical use cases for FMI and/or Modelica models within CasADi involve calculation of derivatives. This includes optimial control formulations solved with gradient-based optimization algorithms as well as dynamic simulation with sensitivity analysis.

To acommodate such use cases, the interface has been designed to:

- Be as efficient as possible, by leveraging parallelization and all available analytic derivative and sparsity information
- Support for both first and second order derivatives (even through the FMI standard only includes first order derivatives)
- Ensure that any calculated derivative quantities can be validated for correctness
- Ensure that the derivative calculation is *predictable* and customizable from a user standpoint

In the following sections, we briefly summarize the kinds of supported derivative information and how they

[1] https://github.com/pymoca/pymoca

[2] https://github.com/jgoppert/cymoca

are calculated by the interface. We will not discuss user syntax for calculation of derivative information as it is the same as for any other function object in CasADi. For illustration, we will consider a set of FMU model equations that can be represented as a differentiable function $y = f(x)$, where $x \in \mathbb{R}^{n_x}$ is a set of inputs or state vector components and $y \in \mathbb{R}^{n_y}$ is a set of calculated outputs or state derivatives.

Forward derivatives

Forward derivatives, i.e. Jacobian-times-vector products $\frac{\partial f(x)}{\partial x} v$ for some x and v, can be calculated using `fmi2GetDirectionalDerivative` in the FMI C API (Modelica Association 2020). Alternatively, we can estimate the same information using one of three finite difference schemes:

- Forward differences, i.e. $(f(x+hv) - f(x))/h$ for some small h, with approximation error $O(h)$
- Central differences, i.e. $(f(x+hv) - f(x-hv))/(2h)$ for some small h, with approximation error $O(h^2)$
- A generalized, smoothness seeking, scheme using 5-point stencils, $f(x-2hv), f(x-hv), f(x), f(x+hhv), f(x+2hv)$, with approximation error $O(h^4)$

The above schemes represent different tradeoffs between accuracy and computational overhead, requiring 1, 2 and 4 additional function evaluations, respectively. For all of the schemes, we will select a fixed, and predictable, perturbation size h, by default 10^{-6}. This necessitates that the directional derivative seed v is properly scaled.

The intended purpose of the finite difference implementation is not to serve as an alternative to analytic derivatives, but to validate that the provided analytic derivatives are correct. This is achieved by, optionally, allowing a selected finite difference implementation to run in a “shadow mode”, ensuring that the two derivative estimates agree up to some absolute and relative tolerance. This validation also helps ensuring that the finite difference perturbation is correctly chosen, which is important for the calculation of second order derivatives.

Jacobians

We use CasADi’s *greedy*, *distance-2*, *unidirectional* algorithm (Gebremedhin, Manne, and Pothen 2005) to calculate large-and-sparse Jacobians, i.e. $\frac{\partial f(x)}{\partial x}$ for the above example. This approach exploits a priori knowledge of the Jacobian sparsity pattern, which can be derived from the variable dependency information provided in the FMI standard. In most use cases, this technique reduces the problem of calculating the sparse Jacobian to one of calculating a reasonably small set of forward directional derivatives.

We allow this directional derivative calculation to be performed in parallel, using either `std::thread` in the C++ standard or OpenMP. We also scale derivative seeds with nominal values of the FMU and adjust the sign of the perturbation to respect variable bounds, when necessary. By using a fixed step size scaled by the nominal value for the derivative seeds, we ensure that the calculation becomes predictable and customizable as the user can adjust the individual nominal values.

Adjoint derivatives

Support for adjoint derivatives, i.e. Jacobian-transpose-times-vector products $\left[\frac{\partial f(x)}{\partial x}\right]^T w$ for some x and w, was added in FMI 3. As the import code, as of this writing, was limited to FMI 2, we instead use the above Jacobian calculation to calculate adjoint derivatives, i.e. we multiply the transpose of the Jacobian, which may not be formed explicitly, with the vector w from the right.

Note that such an approach may be significantly less efficient than using `fmi3GetAdjointDerivative`, assuming a reverse mode algorithmic differentiation is used for the calculation. As the CasADi FMI import transitions to FMI 3, the intention is for the existing implementation to be used as an optional *validation* of the adjoint directional derivatives, provided by the FMI C API. We predict that such validation will prove important when using the interface for complex physical systems.

Forward-over-adjoint derivatives

The FMI standard, whether FMI 2 or FMI 3, does not include an API for second order derivatives, i.e Hessian-times-vector products $\frac{\partial^2 [w^T f(x)]}{\partial x^2} v$, for some v and w. Nevertheless, we can calculate this information with acceptable efficiency and accuracy using finite difference perturbations of the (analytical) adjoint derivatives. For example, can we approximate the second derivative using central differences:

$$\frac{1}{2h}\left(\left[\frac{\partial f}{\partial x}(x+hv)\right]^T w - \left[\frac{\partial f}{\partial x}(x-hv)\right]^T w\right), \quad (1)$$

where we calculate the adjoint derivatives as described above.

Hessians

Our main intrest for calculating forward-over-adjoint derivatives is to obtain a large-and-sparse Hessian, i.e. $\frac{\partial^2 [w^T f(x)]}{\partial x^2}$ for the above example. In large-scale gradient-based optimization, knowledge of the exact Hessian can be used to get faster and more robust convergence.

We use CasADi’s *greedy*, *distance-2*, *star-coloring* algorithm (Gebremedhin, Manne, and Pothen 2005) to calculate sparse Hessians. For this calculation, we use the (incomplete) knowledge of the Hessian sparsity pattern that can be extracted from the FMI XML API. In particular, we know that variables that enter linearly will be absent from the Hessian sparsity pattern, which occurs whenever the `dependenciesKind` field is set to something other than *dependent*.

As for the Jacobian calculation, we allow the different directional derivatives to be calculated in parallel, using `std::thread` in the C++ standard or OpenMP.

Since the Hessian calculation relies on approximations, it is especially important to validate the results. We do this validation by comparing each Hessian entry with a reference value:

- For diagonal entries of the Hessian, we compare the result against the corresponding second order finite difference formula, i.e. $(f(x+hv)-2f(x)+f(x-hv))/h^2$.
- For off-diagonal entries of the Hessian, we compare the result against the mirror element, which will be calculated by a finite difference perturbation of a different variable.

In both cases, the validation can be done with little additional overhead and can thus be used as an on-the-fly diagnostics check. The main additional overhead comes from having to disable the star-coloring algorithm to get every Hessian element validated – avoiding to calculate mirror elements in Hessians is a fundamental property of star-coloring, cf. (Gebremedhin, Manne, and Pothen 2005). Whenever the calculated value deviates significantly from the reference value, a warning is issued, helping the user to either resolve non-smoothness issues in the model, detect toolchain bugs or adjust the nominal values.

4 Generalized support for ODE/DAE integration and sensitivity analysis in CasADi

The FMU import described in Section 3 has multiple potential use cases, including simply being used for validated Jacobians and Hessians of the model equations. Another use case is for dynamic simulation with forward and/or adjoint sensitivity analysis. Such an analysis is particularly important if we use *shooting method* to reformulate a dynamic optimization problem into a nonlinear program (NLP). How this type of reformulations can be implemented using CasADi was detailed in (J. A. E. Andersson et al. 2019).

A key ability of CasADi is to embed solvers of initial-value problems (IVPs) in ordinary differential equations (ODE) or differential-algebraic equations (DAE) – which we will refer to as *integrators* - into symbolic expressions and have the framework calculate forward and adjoint sensitivity analysis, including higher order, automatically and efficiently. This support is relatively mature and has been used in numerous applications. However, in order to use this feature with models defined by FMUs, a number of challenges had to be overcome:

- There was previously no support for *controls* in CasADi, i.e. external inputs that change at certain time points. While such problems could still be solved by constructing multiple calls to integrator instances with parametric inputs, this solution is particularly inefficient for FMUs as it would cause the FMUs to be reinitialized at every control point.
- While there was already support for outputting a solution at multiple time points (as opposed to just the end time), this feature was never made to work together with the automatic sensitivity analysis. So as in the case for controls, the solver would need to be called repeatedly, for each segment, again causing excessive reinitilizations.
- The implementation of the automatic forward and adjoint sensitivity analysis only worked well for models given as symbolic expressions. When the model equation was a function object as is the case here, a more limited range of derivative information is efficiently available.
- The ODE/DAE integrators in CasADi did not scale very well to large dimension. In particular, the structure of the forward and adjoint sensitivity equations were insufficiently exploited.

All the above points were addressed in the major refactoring of the ODE/DAE integrator in CasADi 3.6. In particular, the integrators now explicitly exploits forward sensitivity equation structure, adjoint sensitivity equation structure and forward-over-over adjoint sensitivity equation structure. While there may certainly be bottlenecks left in the code, there is – to the best knowledge of the author – no longer any fundamental limitation in CasADi for large-scale ODE/DAE sensitivity analysis, including for FMU models.

5 Exporting FMUs from CasADi

Another addition to CasADi 3.6 is support for exporting FMUs from CasADi. The FMU export is done from instances of `DaeBuilder`, a class which originates from the original (symbolic) coupling between CasADi and JModelica.org as described in Section 2. The FMU export thus reuses the data structures used for the FMU import descibed in Section 3.

As of this writing, a proof-of-concept implementation of FMUs adhering to FMI 3.0 exists in the framework. The implementation is based on the comprehensive support export of self-contained C code from symbolic expressions CasADi that exists in in CasADi. The generated FMUs contain support for both forward and adjoint derivative calculation.

6 Examples and Tutorial

While parts of the FMI support in CasADi are still rudimentary, the framework has been used successfully for real-world applications, including for parameter estimation with Modelica building models (Cañas et al. 2023).

In the CasADi Github repository, a step-by-step Jupyter Notebook tutorial (`fmu_demo.ipynb`) can be found that demonstrates the main capabilities of the FMU import described in Section 3, including:

- Compilation of FMUs from Modelica
- Loading FMUs into CasADi and creating function objects
- Calculation of Jacobians and Hessians
- Integration and forward/adjoint sensitivity analysis
- Dynamic optimization using a direct collocation approach

For up-to-date information about this and other examples, we refer to the CasADi user guide and website.

7 Conclusions and Outlook

The intention of the FMI support in CasADi is to provide numerically efficient and mature interfaces to FMUs, both for import and for export. In particular, such interfaces can enable the implementation of efficient simulation and optimization formulations, for existing physical models available as FMUs. These formulations can include forward, adjoint and forward-over-adjoint sensitivity analysis for the simulation problems and as well as sensitivity calculation for the optimization formulations. To enable such applications, special care has been taken to provide validation and diagnostics of provided derivative information as well as the efficient calculation of second derivatives.

The FMU interfaces are intended to be general-purpose and can be used for both static (steady-state) and dynamic problem formulations. In the dynamic case, both open-loop and closed-loop formulations are of interest.

As of this writing, the interface was still in active development and future additions to the support will be driven mainly by industrial and academic interest.

In the following, we list some of the main future developments the framework.

7.1 Support for FMI 3 import

FMI 3.0 is a natural fit with CasADi as it adds features that are important to many use cases of CasADi. These features especially include the added support for adjoint derivatives and better Jacobian sparsity information, as discussed in Section 1.2. The addded the support for vector-valued variables is also important as all expressions in CasADi are all matrix-valued.

At the time of this writing, only FMU 2 was supported for the FMI import.

7.2 C code generation for imported FMUs

A common use case of CasADi – especially for industrial applications – is to use the code generation support to generate self-contained C code, which can then be run on an embedded system. This code can represent just the evaluation of a function and its derivatives or a higher-level operation, including the solution of an optimal control problem.

A useful extension of the FMI import would be to allow for C code export of imported FMUs. It would for example allow CasADi symbolic expressions to be exported to an embedded system with static or dynamic linking to the FMU shared libary.

It would be possible to use the C code generation of imported FMU together with the FMU export described in Section 5. For example, we could use the CasADi framework to import multiple FMUs, connect them together into an aggregated system model and then export the aggregated model as a new FMU.

7.3 Support for hybrid systems

FMI – and the Modelica modeling language – provides a flexible modeling and execution paradigm for hybrid systems, i.e. systems with event dynamics. To allow such models to be used within CasADi, an extension of the framework would be needed. In particular, we may want extend the simulation and sensitivity analysis support described in Section 4 to also handle hybrid systems. While handling hybrid systems in the context of dynamic optimization – and in CasADi – is often not possible with the same generality as in the context of system simulation, several interesting problems could be addressed this way.

No explicit support for hybrid systems exists in CasADi as of this writing, although many hybrid systems can be reformulated and solved as multi-stage formulations. In addition, integer valued decision variables can be handled by using one of the interfaced solvers for mixed-integer quadratic programs (MIQP) or mixed-integer nonlinear programs (MINLP).

References

Ahlbrink, N. et al. (2009). “vICERP – The virtual Institute of Central Receiver Power Plants: Modeling and Simulation of an Open Volumetric Air Receiver Power Plant”. In: *Proceedings MATHMOD09 Vienna*. Vol. 263.

Andersson, J. et al. (2011). “Integration of CasADi and JModelica.org”. In: *8th International Modelica Conference*.

Andersson, Joel A E et al. (2019). “CasADi – A software framework for nonlinear optimization and optimal control”. In: *Mathematical Programming Computation* 11.1, pp. 1–36. DOI: 10.1007/s12532-018-0139-4.

Beazley, D. M. (2003). “Automated scientific software scripting with SWIG”. In: *Future Gener. Comput. Syst.* 19, pp. 599–609.

Biegler, Lorenz T. (2010). *Nonlinear Programming*. Society for Industrial and Applied Mathematics. DOI: 10.1137/1.9780898719383. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898719383. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898719383.

Cañas, Carlos Durán et al. (2023). “Parameter Estimation of Modelica Building Models Using CasADi”. In: *15th International Modelica Conference*.

Gebremedhin, Assefaw Hadish, Fredrik Manne, and Alex Pothen (2005). “What color is your Jacobian? Graph coloring for computing derivatives”. In: *SIAM Review* 47, pp. 629–705.

Modelica Association (2020-12). *Functional Mock-up Interface for Model Exchange and Co-Simulation Version 2.0.2*. Tech. rep. Linköping: Modelica Association. URL: https://fmi-standard.org.

Modelica Association (2023-12). *Functional Mock-up Interface for Model Exchange and Co-Simulation Version 3.0*. Tech. rep. Linköping: Modelica Association. URL: https://fmi-standard.org.

Rawlings, J.B., D.Q. Mayne, and M. Diehl (2020). *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing. ISBN: 9780975937754. URL: https://books.google.com/books?id=0IJezgEACAAJ.

Shitahun, Alachew et al. (2013). “Model-Based Dynamic Optimization with OpenModelica and CasADi”. In: *IFAC Proceedings Volumes* 46.21. 7th IFAC Symposium on Advances in Automotive Control, pp. 446–451.

On the Characteristics of One-Dimensional Compressible Flow

Hongtao Qiao[1,*] Takashi Kobayashi[2] Christopher R. Laughman[1] Scott A. Bortoff[1]

[1]Mitsubishi Electric Research Laboratories, Cambridge, MA, USA,{qiao,laughman,bortoff}@merl.com
[2]Mitsubishi Electric Corporation, Hyogo 661-8661, Japan,
kobayashi.takashi@ds.mitsubishielectric.co.jp

Abstract

Eigen decomposition of the governing equations that describe one-dimensional compressible flow has been presented. Analytical solution of the characteristics of the flow was derived. Simulation studies were conducted to support the theoretical analyses and wave propagation results were discussed in detail. It was found that acoustic effect introduced by the dynamic momentum led to a significant slowdown in the simulation and could be neglected in models without significant loss in accuracy for applications where energy transfer is of greater interest.

Keywords: model reduction, characteristic speed, compressible flow, hyperbolic PDE, acoustic effect

1 Introduction

The equations that govern the motion of one-dimensional compressible flow are the system of hyperbolic partial differential equations (PDEs). These consist of conservation laws for mass, momentum, and energy. Acoustic waves appear in a compressible flow, and pressure perturbation involves perturbation of density, velocity, and other parameters. To capture the acoustic effect often requires very small integration steps when solving the governing equations, which significantly increases the computational cost. In contrast, energy transfer often involves a much slower time constant than other transport phenomena, and acoustic effect resulting from pressure perturbation is of minor importance. Consequently, many efforts have explored model reduction techniques to improve computational efficiency without significantly compromising model integrity.

One area of focus has been the simplification of the momentum equation since it usually does not affect the thermal system behavior over the time scales of interest in the applications where acoustic effect is not a top concern. Qiao and Laughman (2018) compared different model reduction techniques and showed that neglecting the phenomena on small time scales can improve numerical efficiency with a minimal loss in prediction accuracy based on simulation studies. However, the paper failed to present a theoretical proof to support their conclusions. Brasz and Koenig (1983) discussed the consequence of eliminating some terms in the original set of governing equations and found that the magnitude of the maximum characteristic speed was a key factor to limit the integration time step. Unfortunately, the authors did not provide detailed derivations to help readers fully understand the underlying reasoning. To fill in the gap, this paper will present a theoretical study for the propagation of waves in a fluid flowing through a channel, with an emphasis on applying the method of characteristics to solve the governing equations and elucidate the effect of dynamic momentum and acceleration pressure loss on the acoustic effect.

The remainder of the paper is organized as follows. In Section 2, we present detailed derivations for the eigen decomposition of governing equations for one-dimensional flow. In Section 3, we perform case studies to discuss the effect of dynamic momentum and acceleration pressure loss on the behavior of two-phase flow. Conclusions from this work are then summarized in Section 4.

2 Eigen Decomposition of Governing Equations

Without taking into account the gravitational force and axial heat conduction, the governing equations of mass, momentum and energy for one-dimensional flow with constant cross-sectional area can be written as

$$\frac{\partial \rho}{\partial t}+\frac{\partial G}{\partial x}=0 \tag{1}$$

$$\frac{\partial G}{\partial t}+\frac{\partial}{\partial x}\left(\frac{G^2}{\rho}\right)=-\frac{\partial p}{\partial x}-\frac{4\tau_w}{D_H} \tag{2}$$

$$\frac{\partial}{\partial t}(\rho e)+\frac{\partial}{\partial x}(Gh)=\frac{G}{\rho}\left(\frac{\partial p}{\partial x}+\frac{4\tau_w}{D_H}\right)+\frac{4q''}{D_H} \tag{3}$$

where ρ, G, p, e, h, τ_w, D_H and q'' are fluid density, mass flux, pressure, specific internal energy, specific enthalpy, wall shear stress, hydraulic diameter and heat flux, respectively.

DOI
10.3384/ecp204327

δ_1 and δ_2 is introduced here to evaluate the impact of the dynamic momentum term $\frac{\partial G}{\partial t}$ and the acceleration pressure loss term $\frac{\partial}{\partial x}\left(\frac{G^2}{\rho}\right)$ on the transient characteristics of fluid flow. Multiplying G/ρ on both sides of Eq. (2) and substituting it into Eq. (3) yields

$$\delta_1 \frac{\partial G}{\partial t} + \delta_2 \frac{\partial}{\partial x}\left(\frac{G^2}{\rho}\right) = -\frac{\partial p}{\partial x} - \frac{4\tau_w}{D_H} \tag{4}$$

$$\frac{\partial}{\partial t}(\rho e) + \frac{\partial}{\partial x}(Gh) + \frac{G}{\rho}\left[\delta_1 \frac{\partial G}{\partial t} + \delta_2 \frac{\partial}{\partial x}\left(\frac{G^2}{\rho}\right)\right] = \frac{4q''}{D_H} \tag{5}$$

In general, pressure p, specific enthalpy h and mass flux G are selected as dynamic states in the simulation of vapor compressor cycles. Using the chain rule to expand $d\rho = \left(\frac{\partial \rho}{\partial p}\right)_h dp + \left(\frac{\partial \rho}{\partial h}\right)_p dh$, Eq. (1), (4) and (5) can be written in the quasilinear form after some manipulations

$$B(U)\frac{\partial U}{\partial t} + A(U)\frac{\partial U}{\partial x} = R \tag{6}$$

where

$$U = \begin{bmatrix} p \\ h \\ G \end{bmatrix},\ R = \frac{4}{D_H}\begin{bmatrix} 0 \\ -\tau_w \\ q'' \end{bmatrix},$$

$$B = \begin{bmatrix} \frac{\partial \rho}{\partial p} & \frac{\partial \rho}{\partial h} & 0 \\ 0 & 0 & \delta_1 \\ -1 & \rho & \delta_1 v \end{bmatrix},\ v = \frac{G}{\rho},$$

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 1-\delta_2 v^2 \frac{\partial \rho}{\partial p} & -\delta_2 v^2 \frac{\partial \rho}{\partial h} & 2\delta_2 v \\ -\delta_2 v^3 \frac{\partial \rho}{\partial p} & G-\delta_2 v^3 \frac{\partial \rho}{\partial h} & 2\delta_2 v^2 \end{bmatrix}$$

It is evident that the conservation laws described by Eq. (6) are hyperbolic inhomogeneous PDEs. The characteristic speed of information propagation dx/dt can be found by solving the eigenvalue of $B^{-1}A$ via $|\lambda I - B^{-1}A| = 0$, where $|\ldots|$ indicates the determinant and λ is the eigenvalue.

However, B is singular and does not have an inverse if the dynamic term in Eq. (4) is neglected, i.e., $\delta_1 = 0$. Note that A is always invertible regardless of the values of δ_1 and δ_2. Hence, Eq. (6) can be rewritten as

$$\frac{\partial U}{\partial x} + A^{-1}B\frac{\partial U}{\partial t} = A^{-1}R \tag{7}$$

Eq. (7) is also hyperbolic PDEs with characteristic speed dx/dt that is the reciprocal of the eigenvalue of $A^{-1}B$, i.e., $\left|\frac{dt}{dx}I - A^{-1}B\right| = 0$.

One can augment Eq. (6) with the total derivative of U

$$\begin{bmatrix} B & A \\ dtI & dxI \end{bmatrix}\begin{bmatrix} \frac{\partial U}{\partial t} \\ \frac{\partial U}{\partial x} \end{bmatrix} = \begin{bmatrix} R \\ dU \end{bmatrix} \tag{8}$$

where I is a 3×3 identity matrix.

It is shown below that the characteristic speeds of the system described by Eq. (6), dx/dt, are actually the roots of the determinant of the coefficient matrix of Eq. (8), regardless of whether B is singular or not.

$$0 = \begin{cases} \left|\frac{dx}{dt}I - B^{-1}A\right| & \text{for } |B| \neq 0 \\ \left|\frac{dt}{dx}I - A^{-1}B\right| & \text{for } |B| = 0 \end{cases}$$

$$\Rightarrow 0 = \begin{cases} (dt)^3\left|\frac{dx}{dt}I - B^{-1}A\right||B| & \text{for } |B| \neq 0 \\ -(dx)^3\left|\frac{dt}{dx}I - A^{-1}B\right||A| & \text{for } |B| = 0 \end{cases} \tag{9}$$

$$\Rightarrow \Delta = \begin{vmatrix} B & A \\ dtI & dxI \end{vmatrix} = \left|\frac{dx}{dt}B - A\right|(dt)^3 = 0$$

With both dynamic momentum term and acceleration pressure loss term included in Eq. (6), i.e., $\delta_1 = \delta_2 = 1$, the determinant of the coefficient matrix is given by

$$\Delta = \frac{\rho}{c^2}\left(\frac{dx}{dt} - v\right)\left[c^2 - \left(\frac{dx}{dt} - v\right)^2\right](dt)^3 = 0 \tag{10}$$

where $c = \sqrt{1/\left(\frac{\partial\rho}{\partial p} + \frac{1}{\rho}\frac{\partial\rho}{\partial h}\right)}$. It is evident that there are three distinct characteristics for the system

$$\frac{dx}{dt} = v - c, v, v + c \tag{11}$$

Next, we show that c is the speed of sound. Since the change in density can be determined as $d\rho = \left(\frac{\partial\rho}{\partial p}\right)_h dp + \left(\frac{\partial\rho}{\partial h}\right)_p dh$, one can have

$$\left(\frac{\partial\rho}{\partial p}\right)_s = \left(\frac{\partial\rho}{\partial p}\right)_h + \left(\frac{\partial\rho}{\partial h}\right)_p \left(\frac{\partial h}{\partial p}\right)_s \tag{12}$$

Using the relation $dh = Tds + \frac{1}{\rho}dp$, one can obtain

$$\left(\frac{\partial h}{\partial p}\right)_s = \frac{1}{\rho} \tag{13}$$

Substituting Eq. (13) into Eq. (12) yields

$$c = \sqrt{\frac{1}{\left(\frac{\partial\rho}{\partial p}\right)_h + \frac{1}{\rho}\left(\frac{\partial\rho}{\partial h}\right)_p}} = \sqrt{\left(\frac{\partial p}{\partial\rho}\right)_s} \tag{14}$$

The right-hand side of Eq. (14) is the definition of the speed of sound.

We use the symbols {−, ◦, +} to denote these three eigenvalues of $B^{-1}A$, i.e., v - c, v, v + c. The corresponding eigenvectors are

$$r^{(-)} = -\frac{2c}{\rho}\begin{bmatrix} c^2 \\ \frac{\partial h}{\partial\rho} + c^2\frac{\partial h}{\partial p} \\ v - c \end{bmatrix}, r^{(\circ)} = \frac{1}{T\frac{\partial\rho}{\partial h}}\begin{bmatrix} 0 \\ \frac{\partial h}{\partial\rho} \\ v \end{bmatrix},$$
$$r^{(+)} = \frac{2c}{\rho}\begin{bmatrix} c^2 \\ \frac{\partial h}{\partial\rho} + c^2\frac{\partial h}{\partial p} \\ v + c \end{bmatrix} \tag{15}$$

where the coefficients are carefully chosen so that the characteristic variables introduced below can be in a simple form.

Letting matrix Q be the matrix of eigenvectors, i.e., $Q = \left(r^{(-)}\middle|r^{(\circ)}\middle|r^{(+)}\right)$, $B^{-1}A$ can be diagonalized as

$$Q^{-1}B^{-1}AQ = \begin{bmatrix} v - c & 0 & 0 \\ 0 & v & 0 \\ 0 & 0 & v + c \end{bmatrix} = \Lambda \tag{16}$$

Defining the characteristic variables W as $dW = Q^{-1}dU$, we can write Eq. (7) as

$$\frac{\partial W}{\partial t} + \Lambda\frac{\partial W}{\partial x} = Q^{-1}B^{-1}R \tag{17}$$

Eq. (17) is equivalent to Eq. (18) - (20).

$$dW^{(-)} = dv - \frac{dp}{\rho c} = -\frac{4}{\rho D_H}\left(\frac{q''}{c}\frac{1}{\rho\left.\frac{\partial h}{\partial p}\right|_\rho - 1} + \tau_w\right)dt$$
$$\text{for } \frac{dx}{dt} = \lambda^{(-)} = v - c \tag{18}$$

$$dW^{(\circ)} = ds = -\frac{4q''}{c^2 D_H T\frac{\partial\rho}{\partial h}}\frac{1}{\rho\left.\frac{\partial h}{\partial p}\right|_\rho - 1}dt$$
$$\text{for } \frac{dx}{dt} = \lambda^{(\circ)} = v \tag{19}$$

$$dW^{(+)} = dv + \frac{dp}{\rho c} = \frac{4}{\rho D_H}\left(\frac{q''}{c}\frac{1}{\rho\left.\frac{\partial h}{\partial p}\right|_\rho - 1} - \tau_w\right)dt$$
$$\text{for } \frac{dx}{dt} = \lambda^{(+)} = v + c \tag{20}$$

Eq. (19) denotes that entropy wave $W^{(\circ)} = s$ travels with local flow speed of v, whereas Eq. (18) and (20) denote that backward acoustic wave $W^{(-)} = v - \int\frac{dp}{\rho c}$ and forward acoustic wave $W^{(+)} = v + \int\frac{dp}{\rho c}$ travel with sonic speed of c with respect of local flow speed v, as shown in Fig. 1. When the flow is adiabatic and inviscid (R = 0), Eq. (6) becomes Euler equations and three waves will remain constant along their respective characteristic curve (Doyle, 2006). However, this is nonphysical and cannot occur in the real world. For fluid flow with heat transfer, the variations in the entropy are only determined by the heat flux along $dx = vdt$. Positive heat flux results in an increase in entropy and negative heat flux results in a decrease in entropy. For viscous flow, frictional losses always lead to the attenuation of both acoustic waves, as shown in Eq. (18) and (20). Meanwhile, Eq. (18) and (20) indicate that heat flux imposes opposite impact on two acoustic waves, because positive heat flux in the forward direction is equivalent to negative heat flux in the backward direction.

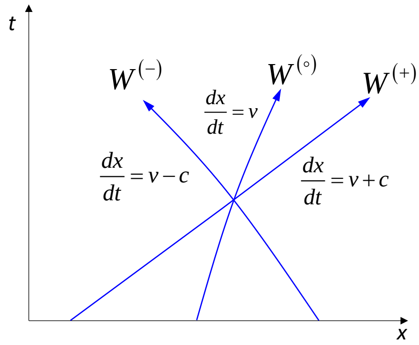


Figure 1. Characteristics of the original set of governing equations

Following the above analyses, one can compute the characteristic speeds of the system when acceleration pressure loss is neglected, i.e., δ₁ = 1, δ₂ = 0.

$$\Delta=\frac{\rho}{c^2}\left(\frac{dx}{dt}-v\right)\left[c^2-\left(\frac{dx}{dt}\right)^2\right](dt)^3=0 \tag{21}$$

It is evident that there are also three distinct characteristic speeds, i.e., $\frac{dx}{dt}=-c,v,c$. These three characteristics correspond to three waves, i.e., backward acoustic wave $W^{(-)}=-\int\frac{dp}{\rho c}$, entropy wave $W^{(\circ)}=s$ and forward acoustic wave $W^{(+)}=\int\frac{dp}{\rho c}$. Again, two acoustic waves travel with sonic speed c.

When dynamic momentum is neglected, i.e., δ_1 = 0, δ_2 = 1, the determinant of the coefficient matrix becomes

$$\Delta=\frac{\rho}{c^2}\left(\frac{dx}{dt}-v\right)\left(v^2-c^2-2v\frac{dx}{dt}\right)(dt)^3=0 \tag{22}$$

Only two distinct characteristic speeds exist in this case, $\frac{dx}{dt}=\frac{v^2-c^2}{2v},v$, corresponding to a supersonic backward acoustic wave and a forward entropy wave.

If neither dynamic momentum term nor acceleration pressure loss term is considered, i.e., $\delta_1=\delta_2=0$, the determinant of the coefficient matrix is

$$\Delta=\rho\left(\frac{dx}{dt}-v\right)(dt)^3=0 \tag{23}$$

In this case, the acoustic effect is eliminated and there is only one characteristic speed $\frac{dx}{dt}=v$, corresponding to the entropy wave travelling along with flow.

CFL stability condition must hold if the FTUS (Forward in Time, Upwind in Space) scheme is employed to solve Eq. (6).

$$\frac{|\lambda|_{\max}\Delta t}{\Delta x}\le C_{\max}=1 \tag{24}$$

where Δt is the time step, Δx is the length interval, and λ is the eigenvalue of the flux Jacobian ($B^{-1}A$) with the largest absolute value (or the reciprocal of the eigenvalue of $A^{-1}B$ with the smallest absolute value). It is evident from Eq. (24) that the time step is restricted by the fastest characteristic speed. When either dynamic momentum term or acceleration pressure loss is taken into consideration, time steps need to be extremely small to obtain a stable solution since the fastest characteristic speed contains the speed of sound. In comparison, when neither term is included, much larger time steps can be taken since the CFL condition becomes $\frac{v\Delta t}{\Delta x}\le 1$. In this case, v is the only characteristic speed, and it is often much smaller than the speed of sound c.

3 Results and Discussions

The finite volume method is often used to discretize the governing equations that describe the dynamics of fluid flow because it has been highly successful in approximating the solution of a wide range of thermal-fluid systems and maintaining quantity conservation. In many of these types of models, a staggered grid scheme is utilized to decouple the mass and energy balance equations from the momentum balance equation. As a result, the mass and energy balances are calculated within the volume cells while the momentum balance is calculated within the flow cells, as depicted in Fig. 2.

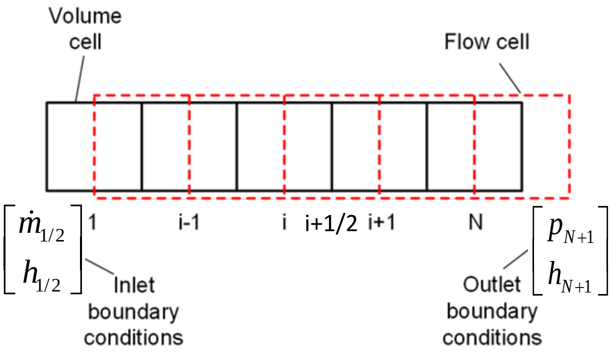

Volume cell
Flow cell
$\begin{bmatrix}\dot m_{1/2}\\ h_{1/2}\end{bmatrix}$
1
i-1
i
i+1/2
i+1
N
$\begin{bmatrix}p_{N+1}\\ h_{N+1}\end{bmatrix}$
Inlet boundary conditions
Outlet boundary conditions

Figure 2. Staggered grid scheme

To simplify the analysis, homogeneous equilibrium model (HEM), which assumes liquid and vapor phases are in thermodynamic equilibrium and have the same phasic velocities, was used to calculate thermodynamic properties of two-phase. As a result, Eq. (6) can be discretized as (Qiao et al., 2015)

$$\frac{\partial \rho_i}{\partial p_i}\frac{dp_i}{dt}+\frac{\partial \rho_i}{\partial p_i}\frac{dh_i}{dt}=\frac{G_{i-1/2}-G_{i+1/2}}{\Delta x_i} \tag{25}$$

$$\delta_1\frac{dG_{i+1/2}}{dt}+\delta_2\frac{\dot{I}_{i+1}-\dot{I}_i}{A\Delta x_i}=-\frac{p_{i+1}-p_i}{\Delta x_i}-\frac{4\tau_{w,i+1/2}}{D_H} \tag{26}$$

$$\begin{aligned}&\rho_i\frac{dh_i}{dt}-\frac{dp_i}{dt}+v_{i+1/2}\left(\delta_1\frac{dG_{i+1/2}}{dt}+\delta_2\frac{\dot{I}_{i+1}-\dot{I}_i}{A\Delta x_i}\right)\\&=\frac{G_{i-1/2}\left(h_{i-1/2}-h_i\right)-G_{i+1/2}\left(h_{i+1/2}-h_i\right)}{\Delta x_i}+\frac{4q''_i}{D_H}\end{aligned} \tag{27}$$

Based on the above discretized form, a dynamic model for one-dimensional channel flow was created in Modelica with inlet boundary conditions defined as mass flow rate and specific enthalpy and outlet boundary conditions as pressure and specific enthalpy, as depicted in Fig. 3. Please note that specific enthalpy at the outlet was only useful when reverse flow occurred.

Four case studies were conducted, and each case corresponded to a combination of δ₁ and δ₂ with different values, i.e., Case 1 (δ_1 = 1, δ_2 = 1), Case 2 (δ_1 = 1, δ_2 = 0), Case 3 (δ_1 = 0, δ_2 = 1) and Case 4 (δ_1 = 0, δ_2 = 0). The flow was assumed to be adiabatic so that the effect of heat transfer on wave propagation can be precluded. For Case 1 and 2, the flow was assumed to be inviscid ($\tau_W = 0$) since pressure could be decoupled from mass flux by the dynamic momentum. For Case 3 and 4, however, frictional pressure loss between adjacent segments needed to be considered since pressure and mass flux were coupled through algebraic relations. Frictional pressure loss was approximated as follow (Laughman and Qiao, 2018)

$$\Delta p=\Delta p_0\left(\dot{m}/\dot{m}_0\right)^2 \tag{28}$$

To perform a fair comparison between these cases, frictional pressure loss in the Case 3 and 4 needed to be very small. Δp_0 and $\dot{m}_0$ were chosen to be 5 Pa and 0.005 kg/s, respectively.

The flow channel was 6m long with hydraulic diameter of 0.01m and was divided into 50 segments with equal size. R-32 was the working fluid, and its properties were computed based on the patch-based B-spline approach (Laughman and Qiao, 2021). The mass flow rate and specific enthalpy at the source was 0.005 kg/s and 380 kJ/kg, respectively. The pressure and specific enthalpy of the sink was fixed at 820 kPa and 382 kJ/kg, respectively. For all cases, models were initialized with steady-state condition and subject to a step change in the specific enthalpy of the source at t = 0, from 382 kJ/kg to 458 kJ/kg. Simulations were carried out in the Dymola 2023x environment (Dassault Systemes, 2023). With the Euler solver, smaller time steps (Δt < 4e-7s) were required to yield stable solutions for Case 1 and 2. For Case 3 and 4, much larger time steps could be used (Δt < 2e-5s). The results shown below were obtained with the DASSL solver with a tolerance of 1e-6.

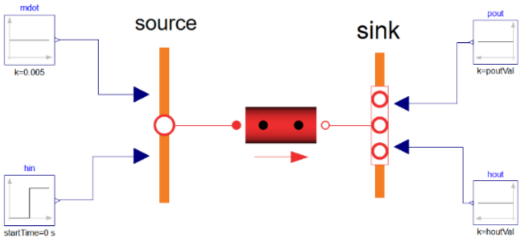
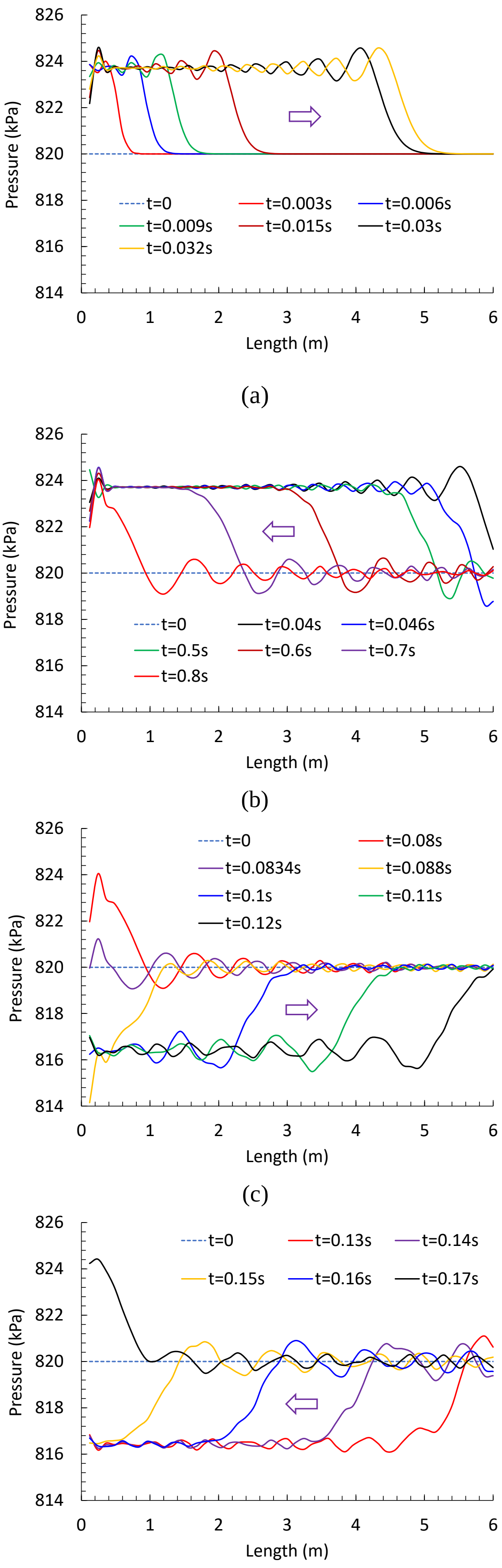


Figure 3. Modelica model of channel flow with a step change in inlet enthalpy

Fig. 4 illustrated the profiles of pressures along the flow channel at various time instants for Case 1. At t = 0, pressures were uniform everywhere. With a sudden increase in specific enthalpy at the inlet, two-phase flow with higher vapor quality entered the channel, resulting in an increase in pressure due to more vapor accumulating in the first segment. This high-pressure wave (it is related to the term of $\int \frac{dp}{\rho c}$ in $W^{(-)}$ and $W^{(+)}$) would travel forward with a speed of $v + c$ along the flow channel, as show in Fig. 4a. At 0.032 sec, the front of the pressure wave reached the exit of the channel and the average wave speed was around 187 m/s, which was very close to the value of $v + c$. The pressure in last segment could not vary freely since the pressure boundary at the exit was fixed. At 0.04 sec, the pressure in the last segment increased to its maximum. Due to the enlarged pressure difference between the last segment and the channel exit, more flow left the channel and pressure in the last segment started to decline. Accordingly, the upstream pressures would be affected, and pressure wave would bounce back. The reflection behavior was similar to the reflection of any waves with fixed end. It was interesting to notice that the wave front kept its shape after reflection. As shown in Fig. 4b, at 0.046 sec the pressure in the last segment reached to its minimum and corresponded to a peak of outflow. At around 0.08 sec, the backward wave reached the inlet of the channel and its average speed was about -176 m/s, which was very close to the value of $v - c$. Since the outflow of the first segment exceeded its inflow at this point, the pressure would decrease accordingly. Based on the behavior of wave reflection with free end, the pressure of the first segment could decrease to a much lower value than the equilibrium pressure, resulting in an inverted low-pressure wave. At 0.088 sec, the pressure of the first segment reached its minimum and then pressure wave traveled forward again (Fig. 4c). At 0.12 sec, the low-pressure wave reached the channel exit again and reflects with a similar shape. At 0.16 sec, the backward pressure wave reached the inlet of the channel once more (Fig. 4d). Then the pressure in the first segment would elevate quickly, and a high-pressure wave with the same phase as the initial forward wave built up and thus finished up a complete cycle.

Fig. 5 showed the profiles of mass flow rates along the flow channel for Case 1. At t = 0, mass flow rate was equal to 5 g/s everywhere. As the pressures near the inlet of the channel started to build up, resulting in negative pressure gradients. From Eq. (26), it was easy to deduce that negative pressure gradients tend to accelerate the flow. Therefore, a mass flow wave (it is related to the term of v in $W^{(-)}$ and $W^{(+)}$) that corresponded to the pressure wave formed and propagated forward. At 0.034 sec, mass flow wave reached the exit of the channel (Fig. 5a). At the same time, the reflecting backward pressure wave retained its negative slope at the wave front, resulting in further acceleration of the flow. Because the mass flow rate was not fixed at the exit, it was amplified by twice of the magnitude of the mass flow wave at 0.0445 sec (Fig. 5b). Similarly, the mass flow wave was inverted at the free end and traveled back towards the inlet of the channel. At 0.08 sec, the high mass flow wave reached the inlet. Since the inverted forward pressure wave resulted in positive pressure gradients at the wave front, the fluid flow started to decelerate. The reflected mass flow wave kept its shape since it was fixed at the inlet and reached the exit once again at 0.12 sec (Fig. 5c). At the exit, the corresponding pressure wave reflected with unchanged positive slopes at wave front, leading to further deceleration of the flow. Consequently, the mass flow rate at the exit declined by more than twice of the magnitude of the wave at 0.13 sec. As the inverted low mass flow wave traveled backwards, mass flow rates behind the wave front oscillated around the equilibrium (Fig. 5d). At 0.16 sec, the wave reached the inlet of the channel. As the corresponding high-pressure wave started to form at this point, the mass flow wave finished a complete cycle.

Fig. 6 depicted the profiles of entropies along the flow channel for Case 1. Initially, the entropy of the flow was 1147 J/(kg·K) everywhere. As a result of a sudden increase in specific enthalpy at the inlet, the entropy in the first segment increased rapidly. It can be observed that entropy wave traveled along with the flow and the wave front flattened out with time. At around 4 sec, entropy profile reached a new equilibrium. Different from pressure wave and mass flow wave, entropy wave traveled along with the flow at a much slower speed and did not bounce back at the boundaries, which corroborated the derivations in the previous section.



(a)

(b)

(c)

(d)

Figure 4. Pressure profiles along flow channel (Case 1)

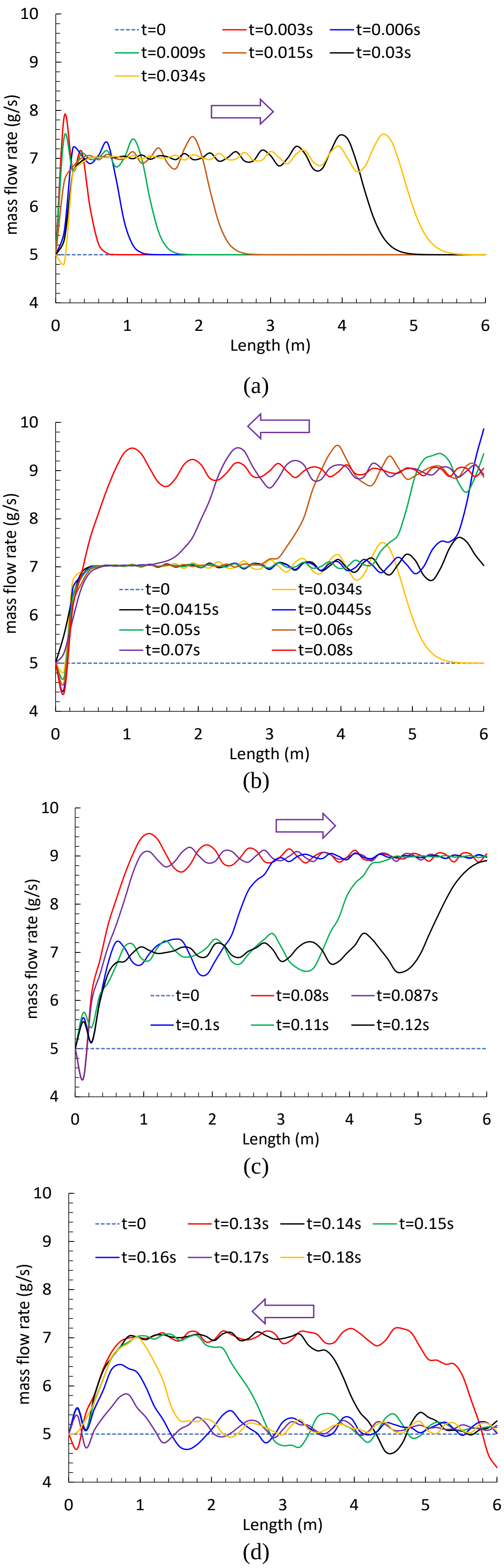


Figure 5. Flow profiles along flow channel (Case 1)

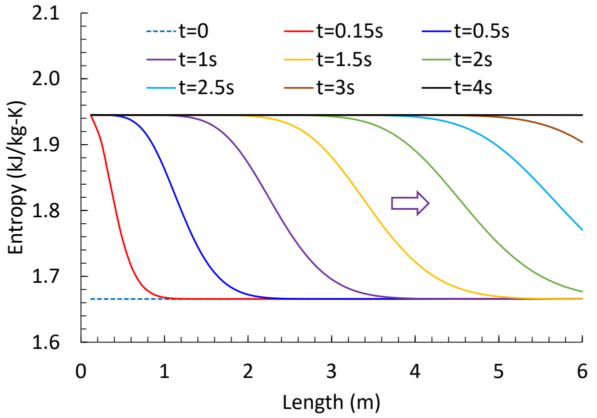
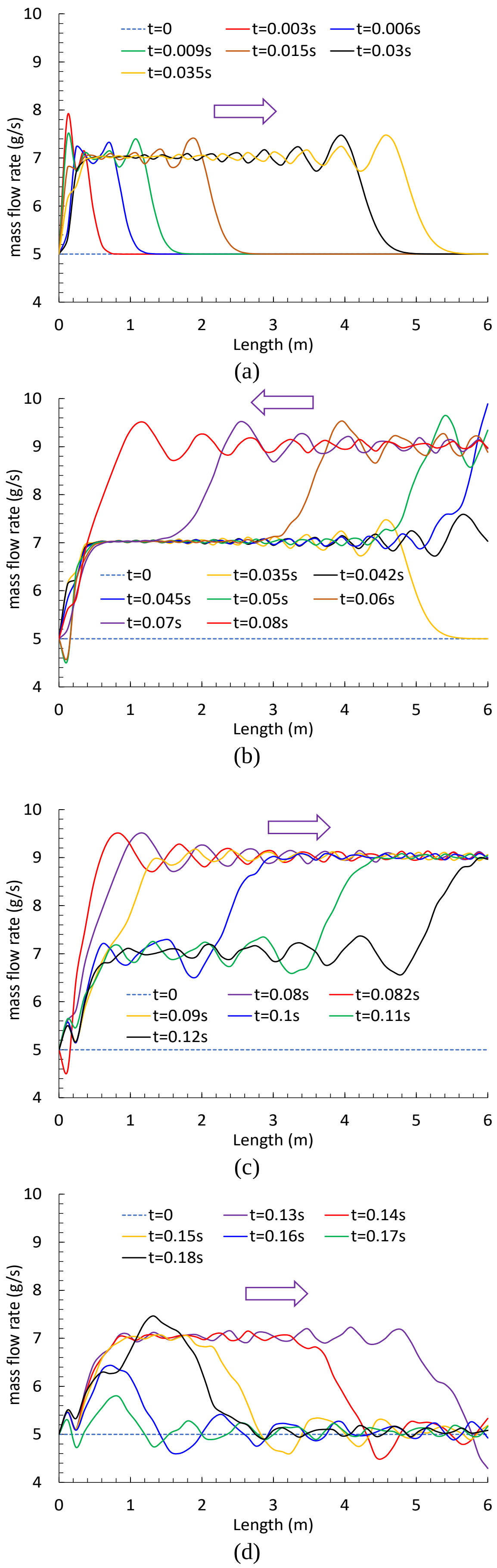


Figure 6. Entropy profiles along flow channel (Case 1)

The profiles of pressures, mass flow rates and entropies along the flow channel for Case 2 were plotted in Fig. 7 to 9. As discussed in the previous section, merely neglecting the acceleration pressure loss did not lead to major changes in the characteristics of the flow except that the speeds of acoustic waves changed slightly. Hence, the interpretation of the results in Case 2 will be the same as in Case 1.

Fig. 10 to 12 displayed the profiles of pressures, mass flow rates and entropies along the flow for Case 3. Although two analytical characteristics existed, i.e., $\frac{v^2-c^2}{2v}, v$, the backward supersonic wave was not physical and did not show up in the results. Like the entropy wave, the mass flow wave did not exhibit any acoustic effect. At 3.5 sec, the wave died out and mass flow rates were back to their equilibrium condition. Due to the frictional pressure loss, the equilibrium pressure profile was not flat. Since the dynamic term was neglected, the speed of pressure propagation should be infinite. Overall, pressure variations were well-behaved, and the rippling effect at the beginning dissipated very quickly.



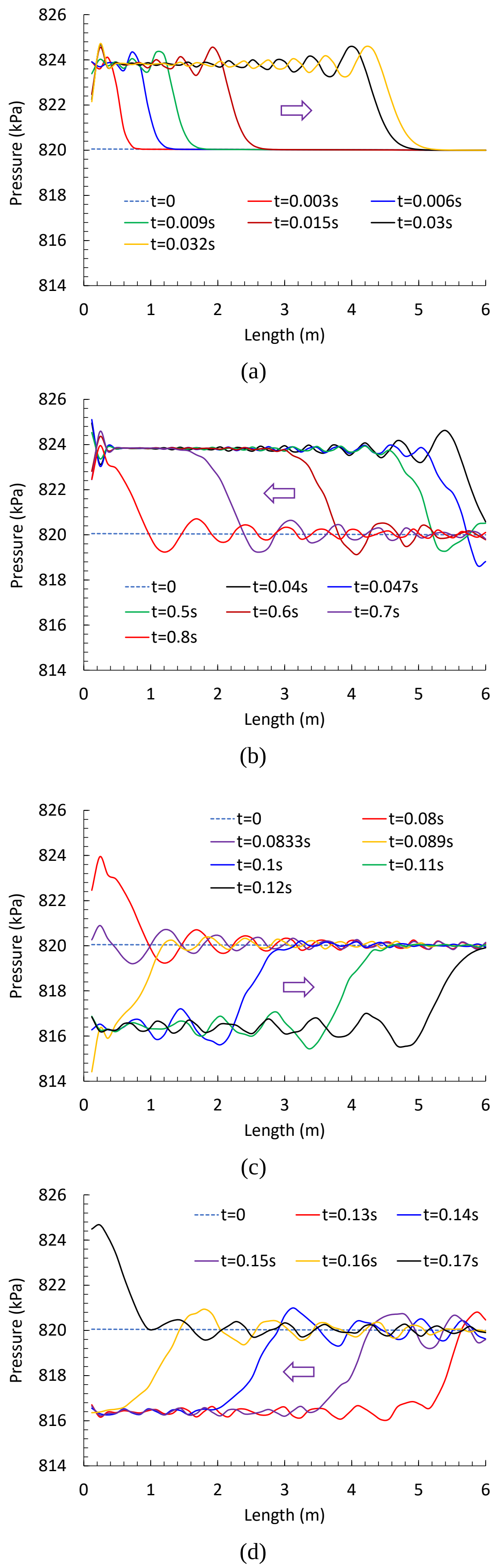
(a)

(b)

(c)

(d)

Figure 7. Flow profiles along flow channel (Case 2)



(a)

(b)

(c)

(d)

Figure 8. Pressure profiles along flow channel (Case 2)

DOI
10.3384/ecp204327

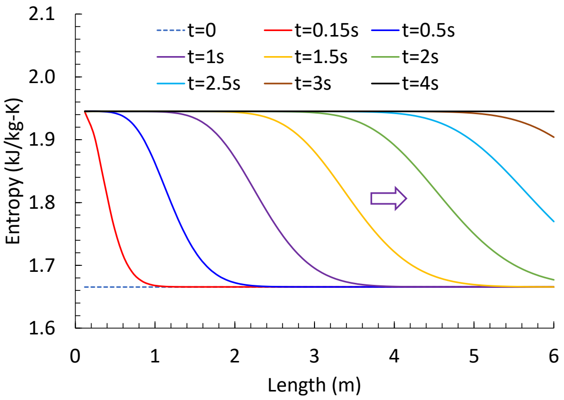


Figure 9. Entropy profiles along flow channel (Case 2)

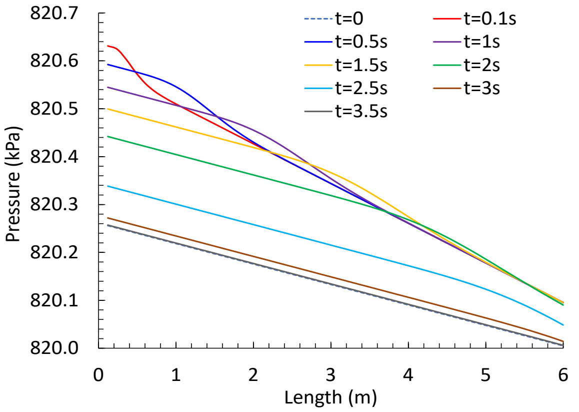


Figure 10. Pressure profiles along flow channel (Case 3)

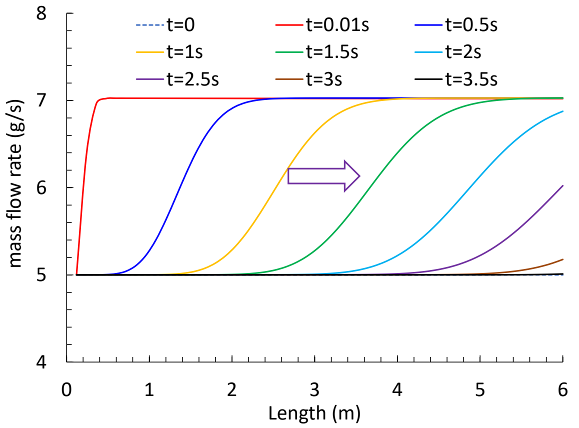


Figure 11. Flow profiles along flow channel (Case 3)

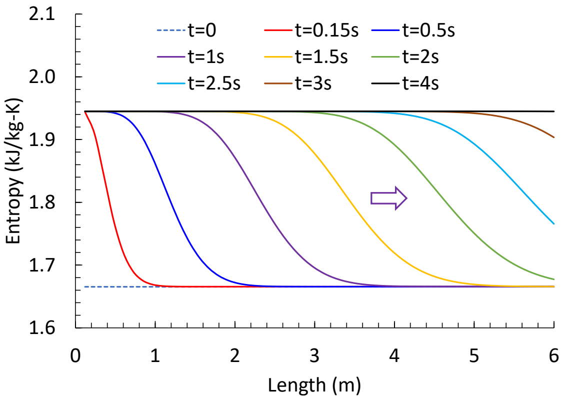


Figure 12. Entropy profiles along flow channel (Case 3)

The profiles of pressures, mass flow rates and entropies along the flow channel at various time instants for Case 4 were manifested in Fig. 13 to 15. This was the simplest case with only one characteristic speed. Again, entropy wave and mass flow wave traveled along with the fluid and did not reflect at the boundaries. Compared with Case 3, no rippling effect was observed in the pressure profiles.

Comparison of simulation speed between these cases indicates that Case 1 was the slowest, followed by Case 2 and then Case 3, while Case 4 was the fastest. For 15 sec simulation, CPU time of these cases was 240 sec, 180 sec, 7 sec and 1 sec, respectively. At the end of simulation, acoustic wave effect was still in place for both Case 1 and Case 2, whereas in Case 3 and 4 steady-state conditions arrived at around t = 4 sec. Profiling showed that pressures and mass flows were the dominating states for Case 1 and 2, whereas pressures and specific enthalpies were the dominating states for Case 3 and 4.

The choice of solver did not exhibit noticeable effect on the results as long as the solutions were stable, but the computational speeds could vary significantly. With the Euler solver, the simulation speeds were substantially slower than the DASSL solver. In comparison, the CPU time for these cases were 42894 sec, 21215 sec, 649 sec, and 604 sec, respectively. This indicated that the BDF methods were much more numerically efficient than the explicit fixed time step methods.

The above analysis indicated that both dynamic momentum term and acceleration pressure loss term imposed significant impact over simulation speed and accuracy. In the applications where acoustic effect is not a great concern, these terms can be neglected in models to achieve much higher computational performance at the expense of accuracy.

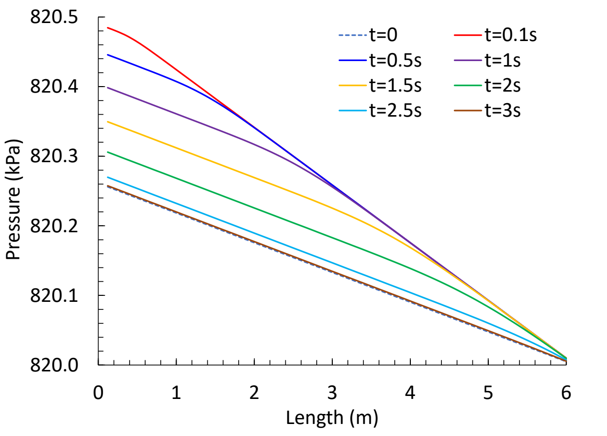


Figure 13. Pressure profiles along flow channel (Case 4)

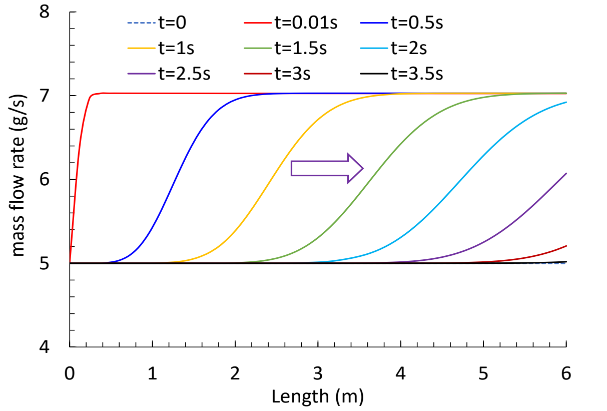


Figure 14. Flow profiles along flow channel (Case 4)

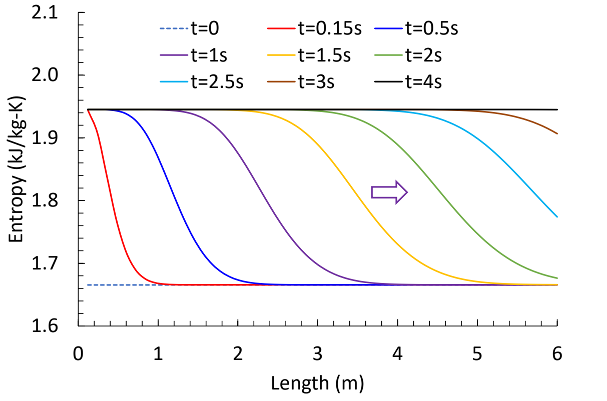


Figure 15. Entropy profiles along flow channel (Case 4)

4 Conclusions

This paper explored the effect of dynamic momentum and acceleration pressure loss on the characteristics of one-dimensional compressible flow. Eigen decomposition for the governing equations was shown to obtain their characteristic form and analytical solution for the wave speeds was derived. Simulation studies were performed to provide evidence to the theoretical analyses. It was found that acoustic effect arising from the dynamic momentum greatly affected the simulation speed. Meanwhile, it was shown that both dynamic momentum and acceleration pressure loss may be neglected in models to speed up simulations in applications where energy transfer is more important than momentum transfer.

References

Christopher Laughman and Hongtao Qiao. On closure relations for dynamic vapor compression cycle models. American Modelica Conference, 2018.

Christopher Laughman and Hongtao Qiao. Patch-based thermodynamic property models in the subcritical region. 18th International Refrigeration and Air Conditioning Conference at Purdue, 2021.

Dassault Systemes, AB. Dymola 2023x, 2023.

Hongtao Qiao and Christopher Laughman. Comparison of approximate momentum equations in dynamic models of vapor compression systems. In Proceedings of the 16th International Heat Transfer Conference, 2018.

Hongtao Qiao, Vikrant Aute and Reinhard Radermacher. Transient modeling of a flash tank vapor injection Heat Pump System - Part I: Model Development. *International Journal of Refrigeration*, No. 49, pp. 169–182, 2015.

Joost Brasz and Kenneth Koenig. Numerical methods for the transient behavior of two-phase flow heat transfer in evaporators and condensers. *Numerical Properties and Methodologies in Heat Transfer*, pp: 461-476, 1983.

Knight Doyle. Elements of Numerical Methods for Compressible Flows. Cambridge Aerospace Series, Cambridge: Cambridge University Press, 2006.

LargeTESModelingToolkit: A Modelica Library for Large-scale Thermal Energy Storage Modeling and Simulation

Michael Reisenbichler-S.[1,2] Franz Wotawa[2] Keith O'Donovan[1,3] Carles Ribas Tugores[1] Franz Hengel[1]

[1]AEE - Institute for Sustainable Technologies, Austria, m.reisenbichler@aee.at
[2]Institute of Software Technology, Graz University of Technology, Austria, wotawa@ist.tugraz.at
[3]Modelon Deutschland GmbH, Germany, keith.odonovan@modelon.com

Abstract

This paper introduces the LargeTESModelingToolkit, a novel Modelica library for modeling and simulation of large-scale pit and tank thermal energy storage. This first comprehensive Modelica library in the field provides the flexibility and tools needed to develop new storage models tailored to the desired application. It also offers researchers and industrial users pre-built storage models for simulation studies to answer the relevant questions for an optimized design at storage and system level.

In this paper, we present the library's key features and structure and introduce the underlying physical and mathematical foundations and modeling approaches. Moreover, we discuss the validation of the models, present the first results, and show the library's applicability using an exemplary simulation case study.

Keywords: Modelica library, Large-scale thermal energy storage, Pit TES, Tank TES

1 Introduction

The integration of large-scale underground hot-water tank and pit thermal energy storage systems offers a high potential to considerably increase the share of renewable energy in future local and district energy systems. These large-scale thermal energy storage (TES) technologies can provide the flexibility needed to store volatile renewable energy sources for a few days as well as on a seasonal basis, bridging the natural gap between supply and demand (Schmidt et al. 2018). At the same time, they also offer a high economic attractiveness for storing large amounts of heat due to economies of scale and a certain flexibility in site selection due to attractive underground integration without free-standing tall structures. In contrast, the large volumes involved lead to high investment costs, which require fundamental planning at the component, storage, and system level. Experimental investigations in the design phase are limited due to the size of these storage technologies and the long time periods in question. Therefore, numerical simulations from the component to the system level are used throughout the whole design process, from the feasibility phase to the detailed design (Dahash, Ochs, Janetti, et al. 2019).

To date, for large-scale tank (TTES) and pit (PTES) thermal energy storage systems TRNSYS[1] is the most widely used simulation tool for storage and system design questions as well as for scientific studies. For example, an overview of past studies and the used TRNSYS models can be found in Xiang et al. (2022).

Modelica TES models are up to now mainly focused on free-standing models (i.e., without modeling of the surrounding ground) (Leoni et al. 2020). These models are primarily used to simulate hot water storage tanks for domestic applications in small size ranges (i.e., up to a few cubic meters) and are, for instance, included in the Modelica Buildings library (Wetter et al. 2014). In recent years, dedicated TTES and PTES models have been developed incorporating modeling of the surrounding ground. Dahash, Ochs, and Tosatto (2020) demonstrated a model with cylindrical geometry and conducted a cross-comparison with generic boundary conditions between the Modelica model and a model developed in COMSOL Multiphysics[2]. Moreover, Reisenbichler et al. (2021) also developed a TTES model with cylindrical geometry in Modelica. To assess the accuracy of the developed model, a validation case study has been conducted by comparing the simulation results with real measurement data of a Danish PTES in Dronninglund, alongside a cross-comparison against other numerical models. A similar Modelica model with cylindrical geometry was developed by Fournier (2022) and also validated against the Dronninglund PTES measurement data. Recently, Kirschstein (2022) developed a Modelica PTES model with a conical geometry, which is available as part of the Modelica Solar District Heating (MoSDH) library (Formhals 2022).

Modelica's equation-based, object-oriented, and multi-domain modeling approach inherently facilitates high reusability, expandability and adaptability of the produced models. Therefore, with Modelica's features and capabilities, large-scale TES modeling on the storage and system level can be taken to the next level.

However, currently available models are still scattered, limited in functionality and flexibility (e.g., in the choice of geometries), focused on specific applications, or have

[1] https://trnsys.com/ (accessed: June 01, 2023)
[2] https://comsol.com/ (accessed: June 02, 2023)

DOI
10.3384/ecp204337

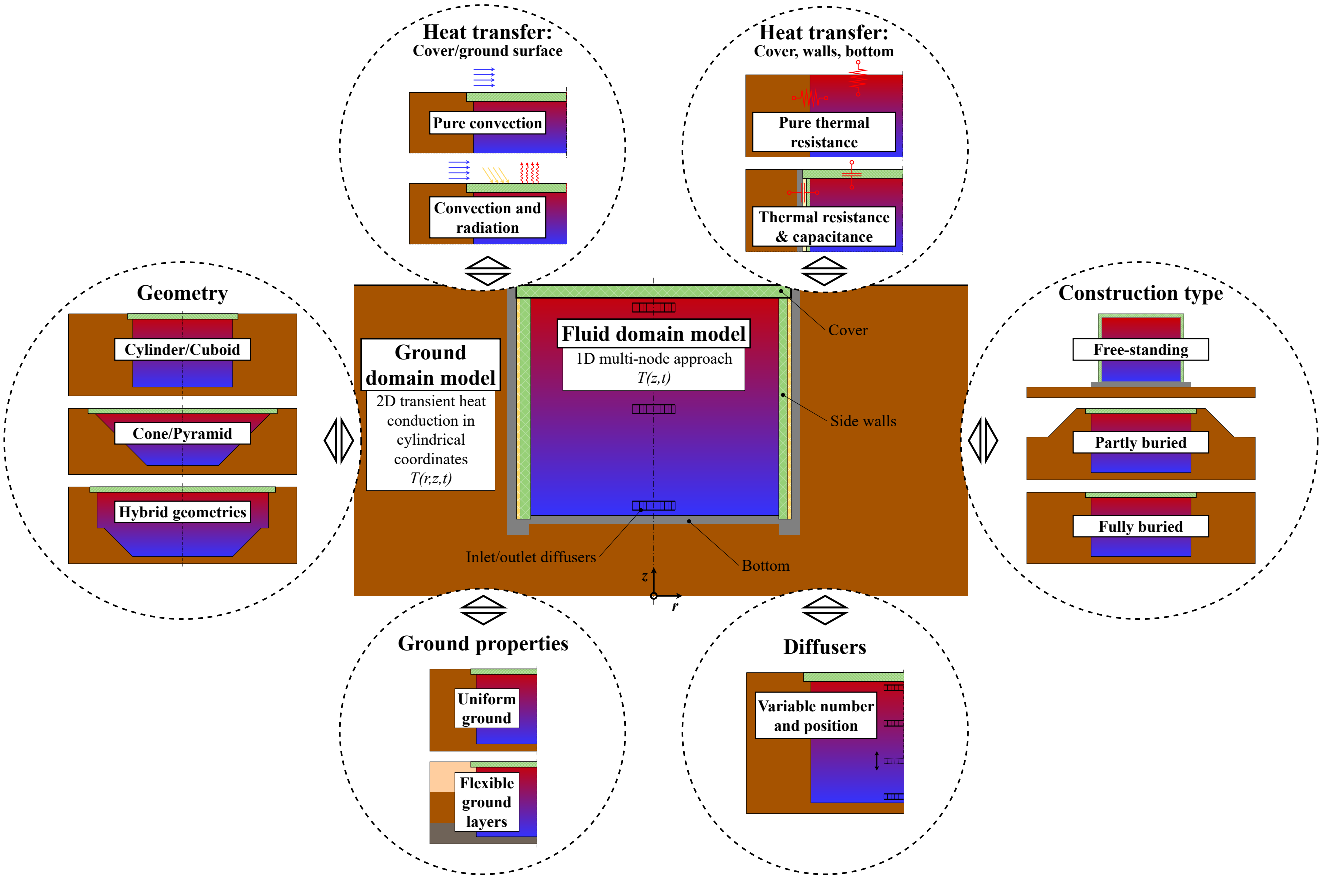


Figure 1. Overview and features of the *LargeTESmtk*.

not yet undergone a comprehensive validation process.

Consequently, our overall goal is to enable more efficient modeling and simulation of large-scale TES in multi-annual dynamic system and storage simulations. Therefore, with the development of the Modelica LargeTESModelingToolkit *(LargeTESmtk)* library, we aim to provide a comprehensive toolkit for the modeling and simulation of large-scale pit and tank TES. In addition to an easy-to-use library with scientifically proven, pre-built storage models for researchers and industrial users, the library is also intended to provide the foundation and tools for the development of new storage models customized to the wanted application.

This paper focuses on the presentation of this Modelica library. We start with an overview of the main features, the implementation in Modelica, and the modeling approaches of the main models. Then we give a brief insight into the models' ongoing validation process and show the library's application in an exemplary simulation case study.

2 The LargeTESModelingToolkit library

This section introduces the *LargeTESmtk* by giving an overview of its main features and models. Afterwards, we describe the corresponding Modelica library structure.

2.1 Overview

Figure 1 provides an overview of the library. Displayed in the center is the basic structure of each storage model, consisting of the two main models for the fluid and ground domain, surrounded by an extract of available model configuration options. Along with other features, the library should provide a wide range of model configuration options in terms of geometry (e.g., cylindrical, conical, or hybrid geometries), heat transfer mechanisms (e.g., pure convection or combined convection and radiation) or ground properties (e.g., uniform ground or specific ground layers) that can be tailored specifically to the wanted application and level of detail.

Underground TTES are typically surrounded by vertical retaining walls to withstand the horizontal loads (e.g., lateral earth pressure of the enclosing ground or hydrostatic water pressure) and have cylindrical or cuboidal geometries. PTES, on the other hand, typically have conical or pyramidal geometries with slopped walls and hence do not require retaining walls to accommodate the horizontal loads (Pauschinger et al. 2020). Both PTES and TTES can be either fully buried (below original ground level) or partly buried with the excavated soil as embankments. In addition, the integration of large-scale TTES in district heating grids as free-standing steel tanks for mainly short-term heat storage is widely used (Dahash, Ochs, Janetti, et al. 2019). As shown in Figure 1, the library provides all the necessary building blocks to model the mentioned storage types and geometry options.

Eventually, the *LargeTESmtk* library should offer the following key features and benefits:

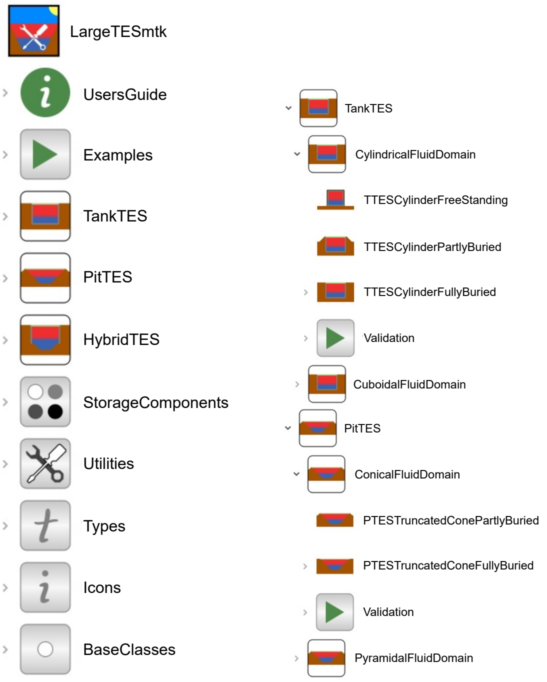
- High adaptability, extensibility, and reusability of the models and sub-models
- Large portfolio of configuration options for initial model generation and later customizations (e.g., concerning geometry, ground properties, heat transfer mechanisms)
- Adaptable level of detail, enabling an application-oriented adjustment between accuracy and calculation performance
- Broad range of application of the models from preliminary design to detailed system and storage design studies
- Simple integration and coupling with other relevant system components (e.g., solar thermal systems) for holistic investigations at system level

The developed models with the *LargeTESmtk* are to be applied in parameter studies, sensitivity, and techno-economic analyses for optimized design on storage and system level. This may include addressing important storage design questions regarding the volume, geometry, insulation quality of the cover, side walls, and bottom, or the number and position of inlets and outlets (i.e., diffusers). In addition, for instance, the investigation of long-term effects (e.g., the development of storage performance in the first years of operation during the heat-up of the surrounding ground) of different system integration concepts (e.g., post-heating concepts via large heat pumps) or storage operation strategies is possible. Finally, the developed models can be used in conjunction with appropriate case studies and methods to obtain relevant techno-economic key performance indicators for decision-making and project planning, such as storage efficiency, thermal losses, stratification quality, investment costs, levelized cost of storage and heat, primary energy consumption, CO_2 emissions or savings.

It is also necessary to point out current limitations of the library models. Mainly to reduce the numerical effort, the ground domain model is restricted to axisymmetric geometries. Thus, non-axisymmetric strorage geometries (e.g., pyramids or cuboids) cannot be modeled directly, but are implemented by corresponding parametrizations and geometry transformations. Furthermore, this approach leads to limitations in the modeling of occurring groundwater flows.

2.2 Modelica library structure

Figure 2a shows the top-level library structure of the *LargeTESmtk*. The Modelica library reflects the models and features described in the overview in subsection 2.1. The structure is oriented around the Modelica Standard Library (MSL) (Modelica Association 2020) and therefore contains common packages such as `Utilities`, `Types`, `Icons` or `BaseClasses`.



(a) Top-level structure **(b)** Sub-packages

Figure 2. Modelica *LargeTESmtk* library structure.

The `Examples` package is intended for models demonstrating the application of the library.

The `TankTES` and `PitTES` packages contain basic pre-built storage models for different fluid domain geometries (e.g., cylindrical or conical geometries) and construction types (free-standing, partly buried, fully buried) as shown in Figure 2b. Also included are `Validation` sub-packages providing validation examples for the individual storage models.

The `HybridTES` package is intended to include storage models for hybrid geometries (e.g., a combination of a cylinder at the top and a truncated cone at the bottom).

The `StorageComponents` package contains the models from which the pre-built models are assembled and contains various fluid and ground domain models for different geometries and component models for covers or side walls. As such, this package also contains the core components for building new storage models.

3 Methods

This section explains the main modeling approaches of the fluid and ground domain models applied in the *LargeTESmtk*. For both models, the finite difference method (FDM) is used to solve the governing basic partial differential equations (PDE). The so-called "method-of-lines" is applied to replace the spatial derivative terms in the PDEs with algebraic approximations (finite differences), leading to a system of ordinary differential equations (ODE) and differential algebraic equations (DAE), including only time-dependent functions, which can be solved with the common ODE-solvers in Modelica-based simulation environments (Fritzson 2015). Furthermore, instead of deriving the corresponding finite difference formulations directly from the basic PDEs, we will use the energy balance approach here since it is considered more intuitive (Çengel and Ghajar 2015). The energy balance approach is based on subdividing the respective calculation domain into a sufficient number of volume elements (control volumes) and subsequently forming energy and mass balances for each element. Accordingly, in the middle of each volume element are the nodal points (nodes) at which the temperatures are to be determined.

3.1 Fluid domain model

Applying the energy balance method, the fluid region is subdivided along its axial direction into equidistant volume elements with a uniform temperature per element. Then, the energy balances are established and the resulting ODEs for each element are solved. This approach is widely used in dynamic system simulation tools and in the literature often referred to as 1D multi-node model or approach (Untrau et al. 2023). Since we assume that the storage fluid is incompressible and the storage is always fully filled, the formation of the mass balance can be omitted (Powell and Edgar 2013). Figure 3 shows a schematic representation of the fluid domain modeling approach with the respective geometrical parameters and energy flows.

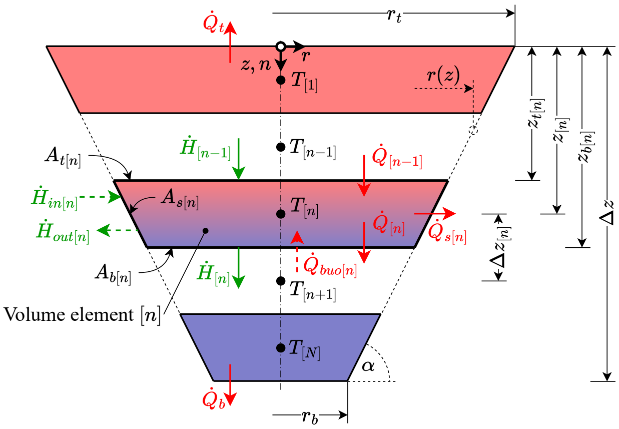


Figure 3. Fluid domain modeling approach.

The energy balance for each volume element $[n]$ can be expressed as the sum of all incoming and outgoing enthalpy flows $\dot{H}$ and heat flow rates $\dot{Q}$ equal to the change in internal energy U or, in case of constant thermophysical properties of the storage fluid (water), the temperature change T of the element with time t:

$$\frac{dU_{[n]}}{dt} = \sum_{[n]} \dot{H} + \sum_{[n]} \dot{Q} \tag{1}$$

$$V_{[n]} \cdot \rho \cdot c_p \cdot \frac{dT_{[n]}}{dt} = \sum_{[n]} \dot{H} + \sum_{[n]} \dot{Q} \tag{2}$$

where ρ and c_p are the density and the specific heat capacity of the storage fluid.

The following equations describe the calculation of the necessary geometrical parameters for a conical fluid domain geometry:

$$r(z) = r_b + \frac{\Delta z - z}{\tan \alpha} \tag{3}$$

$$A(z) = r(z)^2 \pi \tag{4}$$

$$V_{[n]} = \frac{1}{3} \Delta z_{[n]} \pi \left(r_{t[n]}^2 + r_{t[n]} r_{b[n]} + r_{b[n]}^2 \right) \tag{5}$$

$$A_{s[n]} = \frac{\Delta z_{[n]}}{\sin \alpha} \pi \left(r_{t[n]} + r_{b[n]} \right) \tag{6}$$

The top and bottom surface areas of the volume element $A_{t[n]}$ and $A_{b[n]}$ are obtained by inserting the respective coordinates $z_{t[n]}$ and $z_{b[n]}$ in Equation 4. A similar approach can be used for other fluid domain geometries. For instance, simply choosing an angle of $\alpha = 90°$ will result in a cylindrical fluid domain.

The sum of all enthalpy flows for each volume element is the result of the induced volume flows during charging and discharging of the storage, where each enthalpy flow follows the basic equation $\dot{H} = \dot{V} \cdot \rho \cdot c_p \cdot T$, with the corresponding volume flow $\dot{V}$ and temperature T:

$$\sum_{[n]} \dot{H} = \dot{H}_{[n-1]} - \dot{H}_{[n]} + \left(\dot{H}_{in[n]} - \dot{H}_{out[n]} \right) \tag{7}$$

The internal enthalpy flows $\dot{H}_{[n]}$ and $\dot{H}_{[n-1]}$ result from the interaction between the adjacent upper and lower volume elements and depend on whether a downward flow (typically during charging) or an upward flow (typically during discharging) in the storage occurs. Additional enthalpy flows $\dot{H}_{in[n]}$ or $\dot{H}_{out[n]}$ may occur if the respective fluid element is also used for the external volume flows for charging and discharging the storage:

$$\dot{H}_{[n]} = \dot{V}_{[n]} \cdot \rho \cdot c_p \cdot \begin{cases} T_{[n]} & \text{if } \dot{V}_{[n]} > 0 \\ T_{[n+1]} & \text{if } \dot{V}_{[n]} < 0 \end{cases} \tag{8}$$

$$\dot{H}_{[n-1]} = \dot{V}_{[n-1]} \cdot \rho \cdot c_p \cdot \begin{cases} T_{[n-1]} & \text{if } \dot{V}_{[n-1]} > 0 \\ T_{[n]} & \text{if } \dot{V}_{[n-1]} < 0 \end{cases} \tag{9}$$

$$\dot{H}_{in[n]} = \dot{V}_{in[n]} \cdot \rho \cdot c_p \cdot T_{in[n]} \tag{10}$$

$$\dot{H}_{out[n]} = \dot{V}_{out[n]} \cdot \rho \cdot c_p \cdot T_{out[n]} \tag{11}$$

whereby $T_{out[n]}$ equals $T_{[n]}$.

The sum of all heat flow rates of each element results from the heat conduction between the adjacent elements, the heat losses to the surroundings, and a buoyancy-induced heat flow rate that may arise:

$$\sum_{[n]} \dot{Q} = \dot{Q}_{[n-1]} - \dot{Q}_{[n]} - \dot{Q}_{s[n]} + (\dot{Q}_{buo[n]}) \tag{12}$$

The prevailing heat flow rates due to heat conduction between the adjacent elements $\dot{Q}_{[n]}$ and $\dot{Q}_{[n-1]}$ are calculated with the thermal conductivity of the storage fluid k, the heat transfer area A, the distance between the fluid nodes $\Delta z_{[n]}$ and the corresponding temperature difference ΔT. The lateral heat losses $\dot{Q}_{s[n]}$ are derived with the overall heat transfer coefficient $U_{s[n]}$, composed of the inner convective heat transfer coefficient and the thermal resistance of the wall, the heat transfer area $A_{s[n]}$ and the temperature difference between the fluid element $T_{[n]}$ and the surrounding ground $T_{s,g[n]}$:

$$\dot{Q}_{[n-1]} = k \cdot \frac{A_{t[n]}}{\Delta z_{[n]}} \cdot (T_{[n-1]} - T_{[n]}) \tag{13}$$

$$\dot{Q}_{[n]} = k \cdot \frac{A_{b[n]}}{\Delta z_{[n]}} \cdot (T_{[n]} - T_{[n+1]}) \tag{14}$$

$$\dot{Q}_{s[n]} = U_{s[n]} \cdot A_{s[n]} \cdot (T_{[n]} - T_{s,g[n]}) \tag{15}$$

Similar to $\dot{Q}_{s[n]}$, additional heat losses to the top $\dot{Q}_t$ and the bottom $\dot{Q}_b$ occur for the first and the last fluid element.

A buoyancy model is applied to account for buoyancy-induced natural convection in the storage when temperature inversion occurs (i.e., a higher fluid layer has a lower temperature than the layer below). Instead of the buoyancy-induced volume flow into the adjacent fluid layer above that occurs in reality, this volume flow is emulated by adding a corresponding heat flow rate $\dot{Q}_{buo[n]}$ to the fluid element and can, for instance, be expressed as (Wetter et al. 2014):

$$k_{buo[n+1]} = V_{[n+1]} \cdot \rho \cdot c_p \cdot \frac{1}{\tau} \tag{16}$$

$$\Delta T_{[n]} = T_{[n+1]} - T_{[n]} \tag{17}$$

$$\dot{Q}_{buo[n]} = \begin{cases} k_{buo[n+1]} \cdot \Delta T_{[n]}^2 & \text{if } \Delta T_{[n]} > 0 \\ 0 & \text{if } \Delta T_{[n]} \leq 0 \end{cases} \tag{18}$$

where k_{buo} is a proportionality constant since a heat flow rate is used instead of a volume flow, and τ is a time constant that determines how fast the temperature compensation between the fluid layers occurs.

The `Fluid.Storage.Stratified` model of the Modelica IBPSA library (IBPSA 2018) served as the basis for the Modelica implementation of the fluid domain model in the *LargeTESmtk*. However, as described above, multiple extensions and adjustments to the model (e.g., extension to conical geometry, coupling with ground model) were made.

3.2 Ground domain model

The basic mathematical description and the governing equations for the ground domain model follow the partial differential equations of two-dimensional transient heat conduction in cylindrical coordinates with constant thermophysical properties. Again, we use the energy balance approach to derive the corresponding ordinary differential equations for each element, which are then solved. Figure 4 shows a schematic representation of the ground domain modeling approach with the respective geometrical parameters and energy flows.

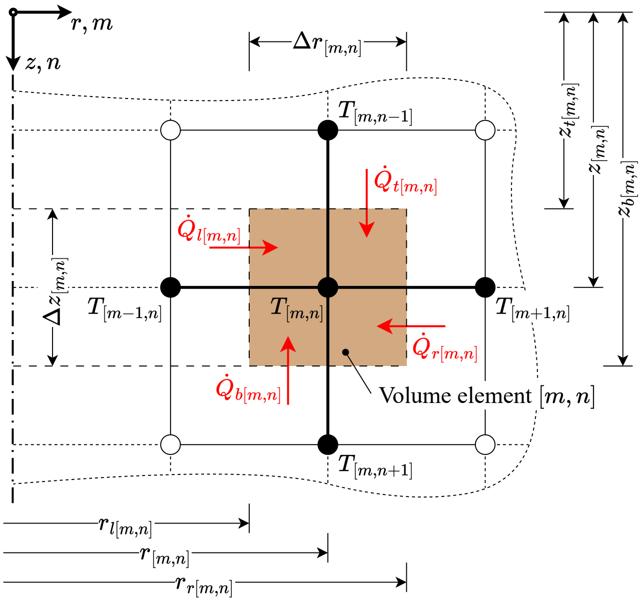


Figure 4. Ground domain modeling approach.

The energy balance for each volume element $[m,n]$ can be expressed as the heat flow rates $\dot{Q}$ into the element from the top, bottom, left and right surface equal to the change in internal energy U or, in case of constant thermophysical properties, the temperature change T of the element with time t:

$$\frac{dU_{[m,n]}}{dt} = \sum_{[m,n]} \dot{Q} \tag{19}$$

$$\left(V \cdot \rho \cdot c_p \cdot \frac{dT}{dt} \right)_{[m,n]} = \left(\dot{Q}_t + \dot{Q}_b + \dot{Q}_l + \dot{Q}_r \right)_{[m,n]} \tag{20}$$

where ρ and c_p are the density and the specific heat capacity of the ground. The volume of the individual ground elements results from:

$$V_{[m,n]} = \pi (r_{r[m,n]}^2 - r_{l[m,n]}^2)(z_{b[m,n]} - z_{t[m,n]}) \tag{21}$$

with the respective geometrical parameters illustrated in Figure 4.

The heat flow rates follow the basic equation $\dot{Q} = G \cdot \Delta T$, where G is the thermal conductance of the ground, the product of the thermal conductivity k and the corresponding geometrical relationship, and ΔT the temperature difference between the adjacent volume element and the volume element under consideration:

$$\dot{Q}_{t[m,n]} = k\frac{\pi(r^2_{r[m,n]} - r^2_{l[m,n]})}{z_{[m,n]} - z_{[m,n-1]}} \cdot (T_{[m,n-1]} - T_{[m,n]}) \quad (22)$$

$$\dot{Q}_{b[m,n]} = k\frac{\pi(r^2_{r[m,n]} - r^2_{l[m,n]})}{z_{[m,n+1]} - z_{[m,n]}} \cdot (T_{[m,n+1]} - T_{[m,n]}) \quad (23)$$

$$\dot{Q}_{l[m,n]} = k\frac{2\pi(z_{b[m,n]} - z_{t[m,n]})}{\ln(r_{[m,n]}/r_{[m-1,n]})} \cdot (T_{[m-1,n]} - T_{[m,n]}) \quad (24)$$

$$\dot{Q}_{r[m,n]} = k\frac{2\pi(z_{b[m,n]} - z_{t[m,n]})}{\ln(r_{[m+1,n]}/r_{[m,n]})} \cdot (T_{[m+1,n]} - T_{[m,n]}) \quad (25)$$

4 Validation

We are going to present in this paper only a part and excerpts of the validation and cross-comparison studies that have already been carried out. For details, please refer to the respective publications mentioned below. Furthermore, not all of the studies conducted have been published yet, but we will give a brief insight into the ongoing work.

Mainly at the beginning of the development of the models and the library itself, certain sub-models were compared with analytical solutions (e.g., steady-state one-dimensional heat conduction) and with available similar models of other Modelica libraries. These examples are not presented here, but are included in the respective `Validation` sub-packages in the Modelica library.

To validate and assess the accuracy of the developed TTES model with a cylindrical fluid geometry, a validation case study was conducted comparing the simulation results with real measurement data of the Danish PTES in Dronninglund (Reisenbichler et al. 2021). Figure 5 shows an excerpt of this study with the comparison between the simulated and measured storage temperatures. In summary, the validation case study revealed that the storage temperatures as well as the charged and discharged energies (with deviations in the range of 1%) could be accurately represented compared to the measurement data. Somewhat higher deviations from the measured data (in the range of 10%) were only seen in the simulated total thermal losses, particularly in the side and bottom thermal losses. Presumably, this is due to the differences between the cylindrical model geometry and the actual pyramidal geometry of the real storage, despite adjusting certain model parameters to the geometry of the real storage (e.g., assuming constant thermal conductance values of the top, side and bottom surfaces between both geometries and corresponding adjustment of the overall heat transfer coefficients). This deviation in thermal losses must be considered in detailed design studies, in particular when the model cannot accurately represent the actual storage geometry.

Furthermore, the cylindrical TTES model of the *LargeTESmtk* was part of a cross-comparison study of various large-scale TES models from different simulation tools (COMSOL Multiphysics, TRNSYS, Modelica/Dymola[3] and MATLAB/Simulink[4]) (Ochs et al. 2022). In this study, all models were examined in a scenario with generic boundary conditions, in which the system was neglected and emulated by a simplified charging and discharging profile, and with different insulation levels (e.g., insulated and non-insulated). The results were compared in terms of storage and ground temperatures, charged and discharged energy, and thermal losses. Overall, a good agreement between the *LargeTESmtk* TTES model and the other models with respect to the compared temperatures as well as energy values could be demonstrated.

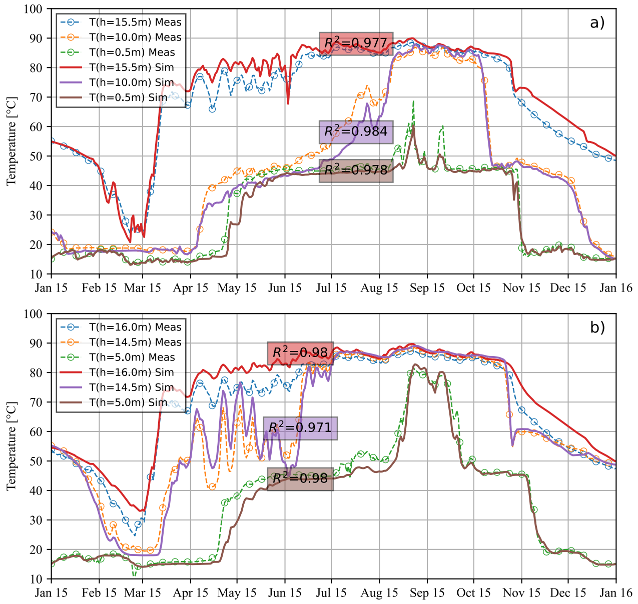


Figure 5. Comparison between simulated and measured storage temperatures of the PTES in Dronninglund for the year 2015 in daily resolution; a) at the three inlet and outlet diffuser heights; and b) at the top, between top and middle, and middle and bottom diffuser (Reisenbichler et al. 2021). In the corresponding colored boxes, the coefficient of determination values (R^2) for the entire year are shown.

The same study was extended to a cross-comparison with PTES models. In addition, a similar cross-comparison study is conducted in the course of the IEA ES TCP Task 39 "Large Thermal Energy Storages for District Heating"[5]. In both studies, the *LargeTESmtk* PTES model with conical geometry is included and the results show good agreement with the other involved models. However, the results of both studies have not yet been published.

Further validation studies of the models from the *LargeTESmtk* (including further comparisons with measurement data from real TES systems) are in progress.

[3] `https://www.3ds.com/products-services/catia/products/dymola/` (accessed: June 09, 2023)

[4] `https://www.mathworks.com/products/simulink.html` (accessed: June 09, 2023)

[5] `https://iea-es.org/task-39/` (accessed: June 06, 2023)

5 Application

A main application area of the storage models is the integration in (multi-annual) system simulations. Thereby, the interaction of the storage together with other system components (e.g., large heat pumps, district heating systems) is evaluated on system level. For example, the storage models of the *LargeTESmtk* have already been used in case studies of large-scale TES with volumes up to millions of cubic meters integrated into DH systems for storing heat from geothermal and solar thermal plants (O'Donovan 2020).

However, in this paper, we demonstrate the application of the library models through an exemplary simulation case study on storage level. The study compares the performance of tank and pit TES with cylindrical and conical fluid geometry under different operation modes ranging from short-term to seasonal storage operation. For this purpose, we will use the prebuilt storage models `TTESCylinderFullyBuried` and `PTESConeFullyBuried` of the *LargeTESmtk*.

5.1 Case study description

As the focus is on the storage level, we neglect other system components such as heat supply units and heat consumers (e.g., the district heating system). Instead of the system components, simplified generic charging and discharging profiles are applied for the different operation modes. Yet, the operation modes are based on real storage application scenarios. The seasonal operation is based on the application of the storage in a solar district heating (SDH) system and the short-term operation on the application for the optimization of a combined heat and power (CHP) plant (Pauschinger et al. 2020).

The number of nominal storage cycles per year resulting from the different operation modes is specified in Table 1. Each full storage cycle (t_{cycle}) consists of a charging phase t_{ch}, a storage phase (TES in the charged state) t_{store}, a discharging phase t_{dis} and an idle phase (TES in the discharged state) t_{idle}. During the charging phase, the inlet volume flow rate $\dot{V}_{ch,in}$ and temperature $T_{ch,in}$ at the top diffuser and, during the discharge phase, $\dot{V}_{dis,in}$ and $T_{dis,in}$ at the bottom diffuser (implemented as step functions) are applied. $T_{ch,in}$ and $T_{dis,in}$ are assumed to be 95 °C and 55°C, across all operation modes.

Table 2 shows the applied model parameters in terms of storage dimensions, fluid, ground, and cover properties, and the applied heat transfer coefficients (HTC). The thermophysical properties of the storage fluid (water) are assumed to be constant at a temperature of 75 °C based on the prevailing storage temperatures (Kretzschmar and Wagner 2019). The ground properties were derived from general values for unconsolidated rocks of gravel, sand, and clay/silt, in both dry and water-saturated conditions (Verein Deutscher Ingenieure 2000). Both investigated storage types are assumed to be insulated only at the top (with a slight extension of the insulation layer beyond the storage edges), while the side walls and bottom remain uninsulated. The applied properties of the storage cover are based on the currently deployed cover constructions (after their revision) of the Danish PTES in Marstal and Dronninglund (Bobach 2020). Only convective heat transfer is considered for the cover and ground surface with the ambient air. The corresponding ambient temperatures of a typical meteorological year (TMY) for the time period between 2015 and 2020 for a generic Central European location (in this case, Vienna) were obtained from the PVGIS online tool (Huld, Müller, and Gambardella 2012). Radiative heat transfer mechanisms are not considered.

Table 1. Investigated operation modes of the simulation case study.

		No. of nominal storage cycles per year				
		120	60	12	4	1
		Short-term				Seasonal
t_{ch}	[h]	24	50	240	720	2,880
t_{store}	[h]	12.5	23	125	375	1,500
t_{dis}	[h]	21	43	185	515	1,800
t_{idle}	[h]	15.5	30	180	580	2,580
$\boldsymbol{t_{cycle}}$	**[h]**	**73**	**146**	**730**	**2,190**	**8,760**
$\dot{V}_{ch,in}$	[m³/h]	2,200	1,100	250	90	27
$\dot{V}_{dis,in}$	[m³/h]	2,200	1,100	250	90	27

To ensure a quasi-stationary storage operation and to neglect the effect of the heat-up phase, the simulation time is five years and the results are only evaluated for the last simulation year. Accordingly, the extent of the ground domain is chosen with a distance of 50 m in axial and radial direction from the fluid domain. Thus, the ground domain is sufficiently large so that the outer nodes remain unaffected by the fluid domain and adiabatic boundary conditions can be applied for the lateral and bottom ground domain boundaries. Moreover, we shifted the simulation start time to the beginning of May to start directly with a charging phase for the seasonal operation. Consequently, the ambient temperature profile is also considered with a corresponding time shift.

5.2 Results and discussion

Various definitions of energy and exergy efficiencies (e.g., with or without consideration of the difference between internal energy content or specifically for the application on seasonal storage) are used and discussed in the literature (Dahash, Ochs, Janetti, et al. 2019; Sifnaios et al. 2022). However, mainly to give a first indication, this study uses the simple definition of the annual storage efficiency η as the annual discharged energy Q_{dis} divided by the annual charged energy Q_{ch}:

$$\eta = \frac{Q_{dis}}{Q_{ch}} \tag{26}$$

As identical charging and discharging profiles are continuously repeated for each nominal storage cycle and the

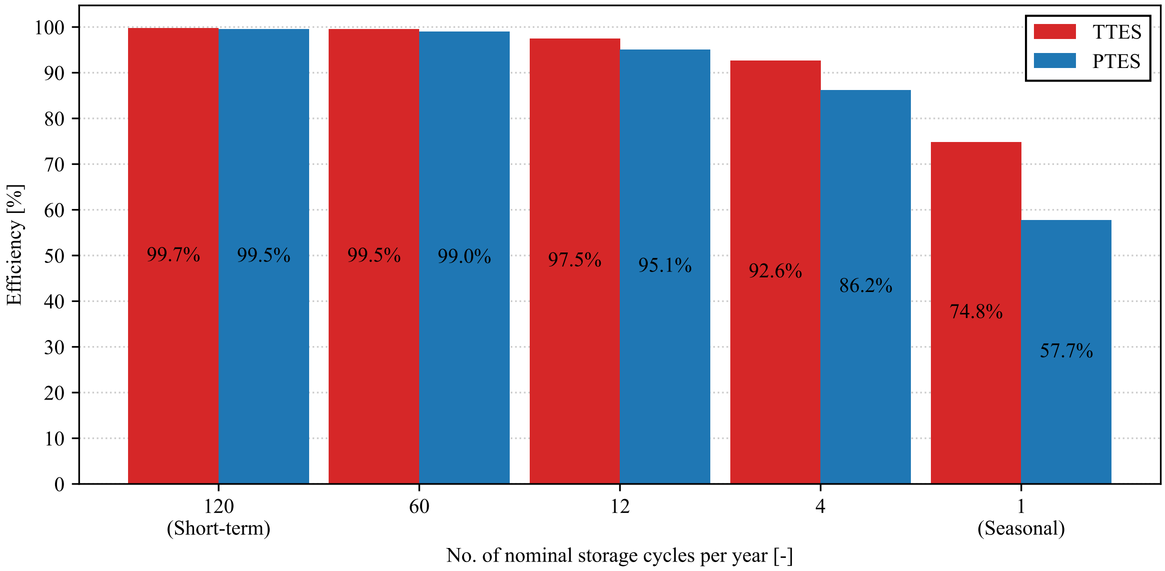


Figure 6. Annual storage efficiencies ranging from short-term to seasonal storage operation.

Table 2. Applied model parameters of the simulation case study.

Parameter		**TTES**	**PTES**
Dimensions			
Volume	[m³]	50,000	50,000
Top diameter	[m]	42	92.5
Bottom diameter	[m]	42	30.5
Height/depth	[m]	36	15.5
Slope angle	[°]	90	26.6
Top diffuser height	[m]	35.5	15.0
Bottom diffuser height	[m]	0.5	0.5
Fluid			
Density	[kg/m³]	974.86	
Thermal conductivity	[W/(m·K)]	0.66	
Specific heat capacity	[J/(kg·K)]	4,192	
Initial temperature	[°C]	10	
Ground			
Density	[kg/m³]	2,700	
Thermal conductivity	[W/(m·K)]	1.2	
Volume-related specific heat capacity	[kJ/(m³·K)]	2,000	
Initial temperature	[°C]	10	
Cover			
Density	[kg/m³]	40	
Thermal conductivity	[W/(m·K)]	0.04	
Specific heat capacity	[J/(kg·K)]	741	
Layer thickness	[m]	0.3	
Initial temperature	[°C]	10	
Insulation extension	[m]	1.5	
HTCs			
Convective HTC ground/cover surface	[W/(m²·K)]	25	
Overall HTC top	[W/(m²·K)]	0.133	
Overall HTC side	[W/(m²·K)]	90	
Overall HTC bottom	[W/(m²·K)]	90	

simulation time is five years, the influence of the change in internal energy content is very small. Furthermore, this definition of storage efficiency allows for a comparison across all operational modes and can be interpreted simply as the ratio of energy recovered to energy stored, regardless of the observation period between the longest and shortest nominal storage cycle duration.

Figure 6 shows the resulting storage efficiencies depending on the operation mode (i.e., number of nominal storage cycles) ranging from short-term to seasonal storage operation. The expected trend that a higher number of storage cycles leads to higher storage efficiency is clearly evident for both storage types. Thus, the short-term operation with a number of 120 nominal storage cycles shows the highest efficiencies with values above 99%. With a lower number of storage cycles, the storage utilization decreases and the actual storage phases become longer, resulting in the lowest storage efficiency for the seasonal operation, whereby the storage efficiency only falls below 90% starting from a number of four storage cycles. The lowest efficiency is about 58% for the PTES case, which is in the range of the measured storage efficiencies for the PTES in Marstal (Sifnaios et al. 2022).

As expected, the TTES generally performs better than the PTES. One main reason for that is the lower surface-to-volume ratio of the cylindrical geometry compared to the conical geometry. At high storage cycle numbers, there is only a slight difference between the storage types because of the short storage phases and the high storage utilization. However, for seasonal storage operations, the difference between TTES and PTES is considerable at about 17%.

As the computational performance is an important factor for the applicability of the models, especially for parameter studies with numerous simulation runs, a first insight of the needed calculation time will be given here. Since the calculation time depends on many factors (e.g., model discretization, number of other system components, solver settings), detailed analyses are the subject of further studies. The simulations required approx. 30 minutes for each single TTES case and 45 minutes for each PTES case[6]. Considering the rather long simulation time of five years per case, these calculation times are considered to be within an acceptable range for more extensive parameter studies.

It is important to mention that the main purpose of this rather small simulation case study was to show the application of the storage models of the library. Storage parameters such as volume, height-to-diameter ratio, ground properties or insulation level, which can have a high impact on the storage efficiency, were not evaluated. However, this simulation case study can serve as a basis for these broader parameter studies. Furthermore, with the provided models and flexibility of the *LargeTESmtk*, the study may also be extended to other storage geometries or combined with additional system components for evaluations on system level.

6 Conclusion and outlook

This study introduced the *LargeTESmtk*, a Modelica library for the modeling and simulation of large-scale pit and tank TES at storage and system level. This comprehensive Modelica library provides pre-built storage models for multi-annual dynamic system simulations for researchers and industrial users. In addition, the provided sub-models and the wide range of model configuration options (e.g., in terms of geometry, heat transfer mechanisms, or ground properties) allow high flexibility in the modeling process and facilitate the development of new models specifically tailored to the desired application.

The library models are to be applied in simulation studies to address relevant storage design questions (e.g., regarding the proper storage geometry or insulation quality) or to investigate different system concepts to achieve an optimized design at storage and system level. These simulation studies also form the basis for techno-economic evaluations to obtain the relevant key performance indicators, such as storage efficiency and levelized cost of heat, for decision-making and project planning.

The accuracy of selected library models has been shown with excerpts from completed and ongoing validation case studies, including measurement data of real TES, and cross-comparison studies with other storage models. Further studies in this regard are underway.

To demonstrate the application of the *LargeTESmtk*, an exemplary simulation case study was conducted comparing the performance of pit and tank TES under different operation modes ranging from short-term to seasonal storage operation. As expected, the simulation case study revealed that the storage efficiency drops from above 99% for the short-term operation to around 58% for seasonal operation for the PTES case and that TTES generally performs better than the PTES. With the simulation study, the good applicability of the models with reasonable calculation times suitable for large parameter studies was shown. Moreover, the *LargeTESmtk* provides the flexibility and models to extend the study to other storage geometries or studies on system level.

The Modelica library is currently still undergoing a continuous development process. As shown in this paper, many features and models are already available, but some of the presented models and features (e.g., hybrid geometries) still need to be added and subjected to validation studies. Furthermore, it is intended to make the library available to everyone soon. So far, Dymola has been this library's main development and simulation environment. However, it is planned to test and ensure compatibility of the library with other simulation environments such as OpenModelica[7].

Acknowledgements

This work was mainly funded by the Austrian Research Promotion Agency (FFG) in the framework of the program “Forschungspartnerschaften Industrienahe Dissertationen 2020 - 6. AS Forschungspartnerschaften“ as part of the project “DevMoTES” (Project No. 878848).

References

Bobach, Morten Vang (2020-10-28). “PTES the next generation of storing energy: New cover solution implemented in Marstal”. Danish HEATSTORE Theme Day (2020). Online. URL: https://www.heatstore.eu/documents/20201028_DG-temadag_Aalborg%20CSP.pdf (visited on 2023-06-08).

Çengel, Yunus A. and Afshin J. Ghajar (2015). *Heat and Mass transfer: Fundamentals & Applications*. Fifth Edition. OCLC: ocn870517093. New York (US): McGraw-Hill Education. 968 pp. ISBN: 978-0-07-339818-1.

Dahash, Abdulrahman, Fabian Ochs, Michele Bianchi Janetti, et al. (2019-04-01). “Advances in seasonal thermal energy storage for solar district heating applications: A critical review on large-scale hot-water tank and pit thermal energy storage systems”. In: *Applied Energy* 239, pp. 296–315. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2019.01.189. URL: http://www.sciencedirect.com/science/article/pii/S0306261919301837 (visited on 2019-03-28).

Dahash, Abdulrahman, Fabian Ochs, and Alice Tosatto (2020-10). “Simulation-based Design Optimization of Large-scale Seasonal Thermal Energy Storage in Renewables-based District Heating Systems”. In: *BauSIM 2020 - 8th Conference of IBPSA Germany and Austria*. BauSIM 2020. ZSCC: NoCitationData[s0]. Graz (AT), 23-25 September 2020: Verlag der Technischen Universität Graz, pp. 112–119. ISBN: 978-3-85125-786-1. (Visited on 2020-12-15).

[6] **PC specifications:** Virtual machine: Windows Server 2012 R2 (Hyper-V); Intel(R) Xeon(R) CPU E5-2420 v2 @2.20GHz (5 logical cores); 8-32 GB RAM (dynamically allocated); MS Win 10 Pro (64-bit)

[7] `https://openmodelica.org/` (accessed: August 06, 2023)

Formhals, Julian (2022-09-01). *MoSDH - Modelica Solar District Heating library*. Version v1.1. URL: https://github.com/MoSDH/MoSDH (visited on 2023-06-14).

Fournier, Nathan (2022). "Numerical Modelling of a Pit Thermal Energy Storage in Modelica". Master Thesis. Copenhagen (DK): Department of Civil and Mechanical Engineering, Technical University of Denmark (DTU).

Fritzson, Peter (2015). *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. 2. ed. Piscataway, NJ: IEEE Press. 1223 pp. ISBN: 978-1-118-85912-4.

Huld, Thomas, Richard Müller, and Attilio Gambardella (2012-06-01). "A new solar radiation database for estimating PV performance in Europe and Africa". In: *Solar Energy* 86.6, pp. 1803–1815. ISSN: 0038-092X. DOI: 10.1016/j.solener.2012.03.006. URL: http://www.sciencedirect.com/science/article/pii/S0038092X12001119 (visited on 2021-01-10).

IBPSA (2018-09-27). *Modelica IBPSA library*. Version v3.0.0. URL: https://github.com/ibpsa/modelica-ibpsa (visited on 2023-06-14).

Kirschstein, Xenia (2022). "Modellierung, Simulation und Analyse der Einbindung saisonaler Untergrundwärmespeicher in das Fernwärmenetz Campus Lichtwiese". Masterarbeit. Darmstadt (DE): Technische Universität Darmstadt. DOI: 10.26083/tuprints-00022118. URL: https://tuprints.ulb.tu-darmstadt.de/22118/ (visited on 2023-04-25).

Kretzschmar, Hans-Joachim and Wolfgang Wagner (2019). "D2.1 Thermophysikalische Stoffwerte von Wasser". In: *VDI-Wärmeatlas*. Ed. by Peter Stephan et al. Series Title: Springer Reference Technik. Berlin, Heidelberg (DE): Springer Berlin Heidelberg, pp. 201–218. ISBN: 978-3-662-52988-1 978-3-662-52989-8. DOI: 10.1007/978-3-662-52989-8_12. URL: http://link.springer.com/10.1007/978-3-662-52989-8_12 (visited on 2021-01-10).

Leoni, Paolo et al. (2020-09). *Large-scale thermal energy storage systems to increase the ST share in DHC*. IEA SHC TECH SHEET 55.A.3.3. IEA SHC. URL: https://task55.iea-shc.org/Data/Sites/1/publications/IEA-SHC-T55-A-D.3.3-FACT-SHEET-Large-Scale-storages-for-ST-share-increase.pdf (visited on 2021-03-13).

Modelica Association (2020-06-04). *Modelica Standard Library*. Version v4.0.0. URL: https://github.com/modelica/ModelicaStandardLibrary (visited on 2023-06-14).

O'Donovan, Keith (2020-05-25). "Giga-Scale Pit Storage as an Essential Part of District Heating Systems: A Simulation based case study". 14th International Renewable Energy Storage Conference 2020 (IRES 2020). Online. URL: https://www.gigates.at/images/Artikel/IRES_Pres_KOD_v6.pdf (visited on 2023-06-11).

Ochs, Fabian et al. (2022-03-03). "Comprehensive Comparison of Different Models for Large-Scale Thermal Energy Storage". In: *Proceedings of the International Renewable Energy Storage Conference 2021 (IRES 2021)*. International Renewable Energy Storage Conference 2021 (IRES 2021). ISSN: 2589-4943. Atlantis Press, pp. 36–51. ISBN: 978-94-6239-546-6. DOI: 10.2991/ahe.k.220301.005. URL: https://www.atlantis-press.com/proceedings/ires-21/125971003 (visited on 2022-03-28).

Pauschinger, Thomas et al. (2020-03). *Design Aspects for Large-Scale Aquifer and Pit Thermal Energy Storage for District Heating and Cooling*. Final Report IEA DHC/CHP Annex XII. Stuttgart (DE): Solites, p. 111. URL: https://www.iea-dhc.org/the-research/annexes/annex-xii/annex-xii-project-03.

Powell, Kody M. and Thomas F. Edgar (2013-12-01). "An adaptive-grid model for dynamic simulation of thermocline thermal energy storage systems". In: *Energy Conversion and Management* 76, pp. 865–873. ISSN: 0196-8904. DOI: 10.1016/j.enconman.2013.08.043. URL: https://www.sciencedirect.com/science/article/pii/S0196890413005177 (visited on 2023-06-10).

Reisenbichler, Michael et al. (2021-09-01). "Towards more efficient modeling and simulation of Large-scale Thermal Energy Storages in future Local and District Energy Systems". In: *Proceedings of the 17th IBPSA Conference*. Building Simulation Conference 2021. Bruges, Belgium: International Building Performance Simulation Association, pp. 2155–2162. DOI: 10.26868/25222708.2021.30911. URL: https://publications.ibpsa.org/conference/paper/?id=bs2021_30911 (visited on 2022-10-21).

Schmidt, Thomas et al. (2018-09-01). "Design Aspects for Large-scale Pit and Aquifer Thermal Energy Storage for District Heating and Cooling". In: *Energy Procedia*. 16th International Symposium on District Heating and Cooling, DHC2018, 9–12 September 2018, Hamburg, Germany 149, pp. 585–594. ISSN: 1876-6102. DOI: 10.1016/j.egypro.2018.08.223. URL: https://www.sciencedirect.com/science/article/pii/S1876610218305198 (visited on 2021-04-05).

Sifnaios, Ioannis et al. (2022-08-15). "Performance comparison of two water pit thermal energy storage (PTES) systems using energy, exergy, and stratification indicators". In: *Journal of Energy Storage* 52, p. 104947. ISSN: 2352-152X. DOI: 10.1016/j.est.2022.104947. URL: https://www.sciencedirect.com/science/article/pii/S2352152X22009537 (visited on 2022-07-05).

Untrau, Alix et al. (2023-03-01). "A fast and accurate 1-dimensional model for dynamic simulation and optimization of a stratified thermal energy storage". In: *Applied Energy* 333, p. 120614. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2022.120614. URL: https://www.sciencedirect.com/science/article/pii/S0306261922018712 (visited on 2023-04-30).

Verein Deutscher Ingenieure (2000-12). *VDI 4640-Part 1 - Thermal use of the underground: Fundamentals, approvals, environmental aspects*.

Wetter, Michael et al. (2014-07-04). "Modelica Buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. ISSN: 1940-1493, 1940-1507. DOI: 10.1080/19401493.2013.765506. URL: http://www.tandfonline.com/doi/abs/10.1080/19401493.2013.765506 (visited on 2021-01-09).

Xiang, Yutong et al. (2022-11-25). "A comprehensive review on pit thermal energy storage: Technical elements, numerical approaches and recent applications". In: *Journal of Energy Storage* 55, p. 105716. ISSN: 2352-152X. DOI: 10.1016/j.est.2022.105716. URL: https://www.sciencedirect.com/science/article/pii/S2352152X22017042 (visited on 2022-12-09).

Dynamic Modeling and Experimental Validation of Dishwasher with Heat Pump System

Erdoğan Mert Şeren[1] Mutlu İpek[1]

[1]Arçelik Global, Istanbul 34950, Turkey erdoganmert.seren@arcelik.com mutlu.ipek@arcelik.com

Abstract

Integration of heat pump systems with conventional dishwashers or household water heaters using electric heaters offers a promising solution to significantly reduce expected energy consumption. In this study, a comprehensive approach was undertaken to develop sub-models for each component of the heat pump dishwasher. These sub-models were subsequently integrated to form a complete cycle model of the heat pump dishwasher. The specific components modeled included the compressor, evaporator, condenser, and capillary tube. Furthermore, an algorithm model was devised to ensure the proper functioning of all the individual models, in accordance with the operational principles of the dishwasher. To validate the model, the temperature variation within the dishwasher during the heating and cooling phases was compared against experimental data. The maximum deviation observed in the cabinet temperature of the dishwasher was found to be ±1 °C, with a corresponding deviation of 0.5 minutes in the cycle duration. Moreover, the maximum deviation in power consumption amounted to 2.4%, while a maximum deviation of 2.9% was noted in energy consumption. The results obtained from the model closely aligned with the experimental outcomes, thereby confirming its accuracy and reliability.

Keywords: Modeling of Heat Pump System, Dishwasher with Heat Pump, Algorithm, Modeling Validation, Dynamic Model, Dymola, Modelica

1 Introduction

Global warming has gained international attention, leading to climate change with altered weather patterns, rising sea levels, and increased extreme weather events. As the world economy recovers from the pandemic, greenhouse gas emissions are expected to rise. To tackle these challenges, the United Nations has set Sustainable Development Goals, and the European Union aims to reduce domestic greenhouse gas emissions by at least 40% by 2030, aligning with the Paris Agreement's goal of limiting global temperature rise to below 2 °C (European Commission, 2016).

The widespread use of household appliances such as washing machines, dishwashers, and refrigerators has resulted in significant environmental consequences due to their high per-unit emissions. To tackle this issue partially, integrating heat pump systems to household water heaters that employ electric heaters like dishwasher or washing machines presents a viable approach for substantially reducing energy consumption and mitigating the environmental impact associated with these appliances. To illustrate, Flück et al. (2017), investigated the heat pump system integrated in dishwashers and found out that it can lead to a significant reduction of up to 50% in electricity consumption compared to conventional electricity heaters. Atasoy et al. (2022) conducted a study where a heat pump water heater system was employed to replace a conventional electricity water heater for heating 4 liters of water to 50°C. The aim was to integrate it with household appliances such as dishwashers. The system was optimized by adjusting compressor speed and air flow rate through the evaporator. The optimized system was then compared to the conventional electricity water heater system, revealing a 17% decrease in energy consumption. On the other hand, the system modeling provides a means to understand system behavior and optimize its performance. It reduces the need for extensive experimentation, making it a valuable tool for system analysis and optimization. For instance, Caglayan et al. (2021) developed a comprehensive representation of a household refrigerator using Modelica, encompassing the intricate details of various parts such as the cabinet, compressor, capillary-tube, and control algorithm. Another research widening this study was conducted by Husain et al. (2023). In the study, a model created for a double-door refrigerator with a top-mounted configuration, featuring a serial refrigeration system. In addition to these, Ipek et al. (2023) proposed a dynamic model created in Modelica on upright freezer of refrigerator by investigating the periodic door opening cases with maximum 6% deviation on energy consumption value.

DOI
10.3384/ecp204347

In the context of dishwasher models specifically, Caskey et al. (2018) built a dishwasher model on Modelica simulating an external hot water circulation instead of a conventional electric heater to demonstrate the potential energy savings. A novel methodology for the development and validation of a comprehensive model for a heat pump dishwasher was presented in this paper. The study introduced by constructing a detailed model of the dishwasher cabinet, capturing its structural and thermal characteristics. Subsequently, individual models of all heat pump components, including the compressor, condenser, evaporator, and capillary tube, were built to create a complete cycle model. Moreover, an algorithm model was developed, taking into account the dishwasher's cycle algorithm and its impact on energy consumption and performance and all models were associated with each other. Finally, the developed model was validated through experimentation under standard conditions. By means of this, the dynamic model serves as a valuable way for comprehending system dynamics and enhancing operational efficiency. By minimizing the reliance on extensive experimentation, it emerges as a potent tool for both analysis and optimization of systems.

2 System Design and Experimental Procedure

The heat pump system integrated into the dishwasher, located within the bottom chassis volume, comprises four key components: compressor, condenser, capillary tube, evaporator and R600a is used as the refrigerant. The main objective of the heat pump system is to heat the dishwasher's washing water. Upon activation, the refrigerant, at high temperature and high pressure, is discharged from the compressor and directed towards the condenser. The condenser facilitates efficient heat transfer, creating a hot surface for the purpose of heating. The secondary loop of the dishwasher, responsible for water circulation, is interconnected with the condenser through the heat pump system. In other words, the water passes continuously through the condenser, where it is efficiently heated. On the other hand, the refrigerant exits the condenser and flows through the capillary tube. Within this component, the refrigerant undergoes a decrease in pressure and temperature. It then proceeds to the evaporator, which is situated in front of an opening at the rear of the bottom chassis compartment of the dishwasher. It is in direct contact with the environment in which the dishwasher is placed. Through of fans located behind it, air drawn from the bottom chassis compartment is passed over the evaporator and released into the environment. This process enables the evaporator to draw heat from the environment. Having absorbed heat in the evaporator, the refrigerant is subsequently directed back to the compressor, thus completing the cycle. Depending on the algorithm, the heat pump system remains active until the water reaches the desired temperature. Overall, the integration of the heat pump system in the dishwasher allows for efficient heating of the washing water, contributing to enhanced energy efficiency and performance. Table 1 provides the specific details of the dishwasher with a heat pump used in this study.

Table 1. Dishwasher with heat pump details.

Dishwasher with Heat Pump	
Dimensions (H x W x D)	818 x 598 x 550 mm
Place Settings	16 pcs
Water Consumption	9.9 L
Energy Consumption	470 Wh
Program Duration	190 min.
Compressor	Reciprocating
Condenser	Helical Tube
Evaporator	Fin and Tube
Refrigerant	R600a
Charge Amount	30g

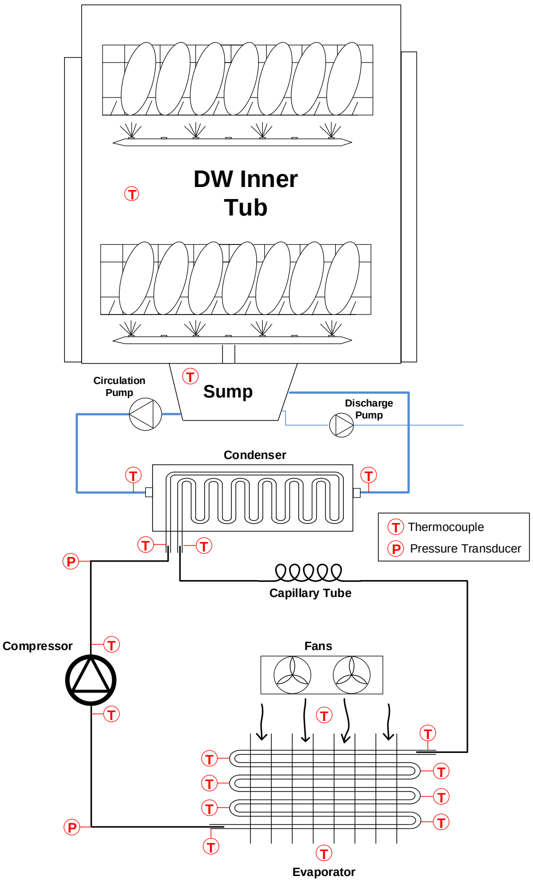


Figure 1. Schematic illustration of heat pump dishwasher.

With respect to the experimental procedure, the experiments are proceeded according to European dishwasher test standard EN 60436 (European Standard, 2020). Also, in order to find out system behavior, any number of thermocouple and two refrigerant high-low pressure transducers are added to the system. The system schematic is illustrated in Figure 1.

The experiments are conducted in a testing chamber in which the temperature and humidity are controlled fairly. When the dishwasher placed reaches the equilibrium temperature with the chamber then, it is operated, and the heating is taken place only by the heat pump system. Temperature, pressure, power, and energy consumption information are collected. The experiment is completed when the cycle is end off at which the water temperature reaches predetermined temperature. The accuracy of experimental results can be influenced by errors arising from measurement devices and the experimental setup. To assess the error values associated with calculated parameters derived from experimental data, various approaches have been suggested. One such method is uncertainty analysis, which was pioneered by Kline and McClintock (1953). Experimental accuracy values for temperature, voltage, current, power and energy consumption calculated by McClintock uncertainty approach are represented in Table 2.

Table 2. Uncertainties of experimental system

Sensor	Uncertainty
Temperature	± 0.5 °C
Voltage	± 0.25%
Current	± 0.25%
Power	± 0.5%
Energy	± 0.75%

3 Modeling Methodology

To create a dynamic model of a heat pump dishwasher, it is essential to model each component individually. These modeling studies are carried out using the Dymola software (Dassault Systems, 2021), which uses the Modelica language. While the dishwasher cabinet and its algorithm are modeled using the object-oriented Modelica language, the heat pump system is modeled from the TIL library with specific modifications. The library is developed by TLK-Thermo GmbH (2020) in Modelica modeling language that includes a created library containing various sub-libraries, such as fluid, gas, and refrigerant components. The complete model, represented by Figure 2, demonstrates the integration of information-carrying components within their respective models, establishing interconnections between each model for a coherent system.

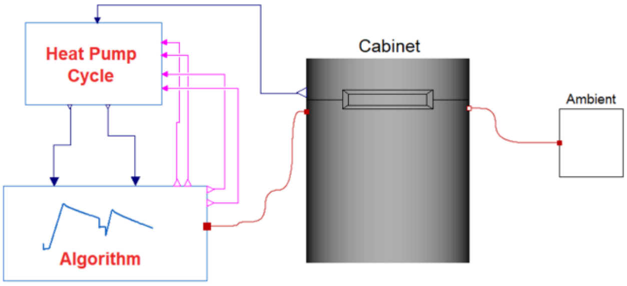


Figure 2. Dishwasher with heat pump model.

The heat pump system receives inputs from the cabinet and outputs to the algorithm model. Through the algorithm model, the water is heated, resulting in the cabinet being warmed via the connected heat ports. The algorithm's outputs are then communicated to the heat pump model, allowing the system to be controlled and operated according to the algorithm's instructions. Furthermore, the ambient model considers the ambient temperature and outer convection coefficient to simulate the heat transfer between the cabinet and the external environment. In the next sections, a comprehensive description of each component of the complete cycle model is provided in detail.

3.1 Cabinet Model

The cabinet of a dishwasher with a heat pump is of utmost importance due to its significant role in heat losses and energy consumption. The model's primary aim is to accurately estimate the heat loss through the cabinet walls to the surrounding environment and track the cabinet's air temperature over time. To reduce heat loss from the dishwasher's cabinet to the surroundings, the cabinet is constructed with multiple layers like insulation materials and bitumen. The heat transfer within the cabinet is modeled using a thermal resistance structure approach. Each material and component are divided into smaller segments to enhance the accuracy of the model.

The Figure 3 depicts the thermal resistance structure of dishwasher front wall. In this figure, the black resistance structures represent heat transfer through conduction, while the green resistances represent heat transfer through convection. Specifically, the "Outer Convection" resistance structure simulates the heat loss from the dishwasher cabinet to the surrounding environment, while the "Bottom Volume Convection" convection resistance represents the heat transferred to the bottom chassis volume.

Since one of the main objectives of the model is to observe the temperature inside the cabinet over time, the heat transfer mechanism needs to be calculated in a time-dependent manner.

To achieve the desired outcome, the one-dimensional time-dependent heat transfer equation along the wall is supposed to be solved. This equation is shown in Equation 1.

$$\frac{\partial^2 T}{\partial x^2} = \frac{\rho c_p}{k} \frac{\partial T}{\partial t} \tag{1}$$

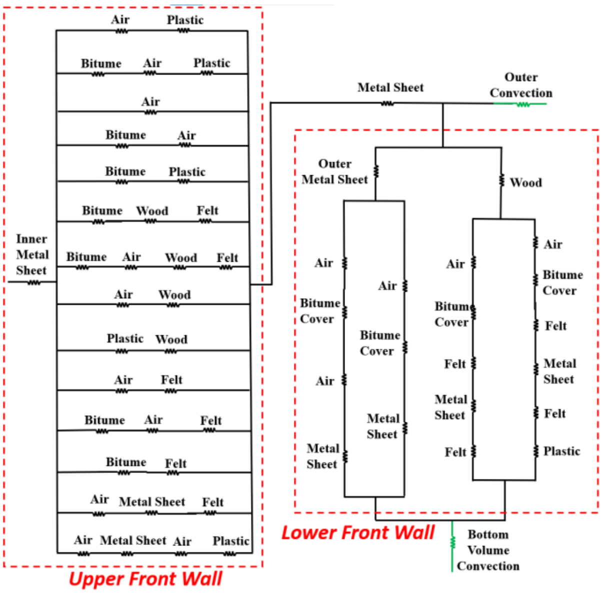
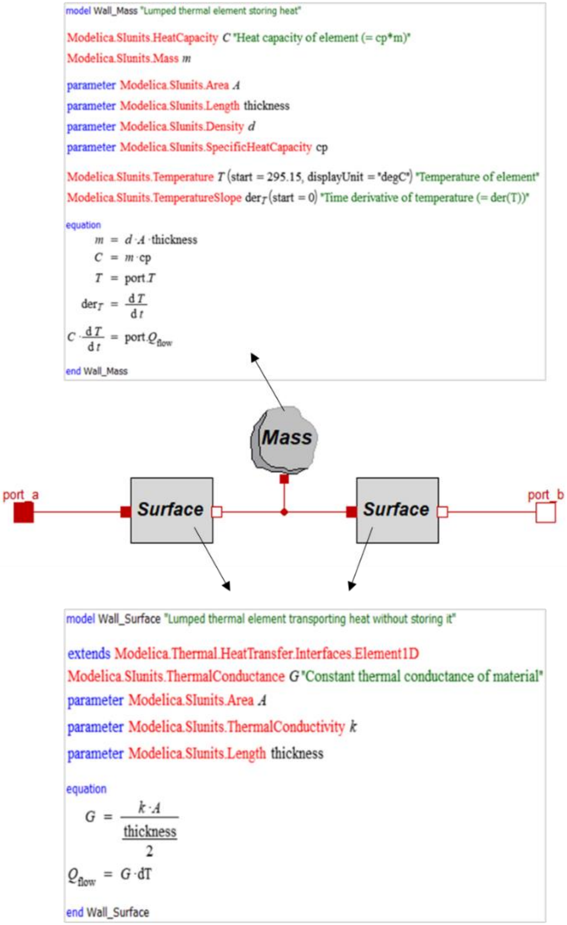


Figure 3. Thermal resistance structure of dishwasher’s front wall.

A single resistance structure created in Dymola is demonstrated in Figure 4. This structure is used to construct each wall by connecting each other thermally, either in parallel or in series, based on the resistance structure created for each wall. Subsequently, properties such as thickness, area, density, specific heat capacity, and thermal conductivity are assigned to each resistance, depending on the material. By means of this approach, cabinet model was created and illustrated in Figure 5.

After constructing each wall structure, in addition to the resistance structures, dishes and other components that generate thermal loads within the cabinet are created as lumped masses in models indicated by numbers 3 and 4, respectively. These masses are thermally connected to the structure based on their materials and masses. On the other hand, since the heat pump system is located in the bottom chassis volume of the dishwasher, the operating conditions of the system are significantly influenced by this volume. This is because the evaporator is directly fed by the air present in this volume. Therefore, the bottom chassis volume is also modeled within the bottom wall structure and provides feedback to the heat pump system model through the air temperature represented by number 5. Finally, the heat port indicated by number 1 is connected to the algorithm model to associate the heated water with the dishwasher cabinet, while the heat port represented by number 2 is thermally connected to the ambient model to simulate the heat loss from the cabinet to the surroundings.

To sum up, each individual resistance in the front wall resistance structure is represented in the model using a wall element. After creating the thermal resistance structures of all walls, like the front wall, they are individually created in the model using wall elements. Subsequently, each wall model is thermally connected to one another.


```
model Wall_Mass "Lumped thermal element storing heat"

Modelica.SIunits.HeatCapacity C "Heat capacity of element (= cp*m)"
Modelica.SIunits.Mass m

parameter Modelica.SIunits.Area A
parameter Modelica.SIunits.Length thickness
parameter Modelica.SIunits.Density d
parameter Modelica.SIunits.SpecificHeatCapacity cp

Modelica.SIunits.Temperature T (start = 295.15, displayUnit = "degC") "Temperature of element"
Modelica.SIunits.TemperatureSlope der_T (start = 0) "Time derivative of temperature (= der(T))"

equation
```

$$\begin{aligned} m &= d \cdot A \cdot \text{thickness} \\ C &= m \cdot \text{cp} \\ T &= \text{port}.T \\ \text{der}_T &= \frac{dT}{dt} \\ C \cdot \frac{dT}{dt} &= \text{port}.Q_{\text{flow}} \end{aligned}$$

```
end Wall_Mass
```

```
model Wall_Surface "Lumped thermal element transporting heat without storing it"

extends Modelica.Thermal.HeatTransfer.Interfaces.Element1D
Modelica.SIunits.ThermalConductance G "Constant thermal conductance of material"
parameter Modelica.SIunits.Area A
parameter Modelica.SIunits.ThermalConductivity k
parameter Modelica.SIunits.Length thickness

equation
```

$$\begin{aligned} G &= \frac{k \cdot A}{\frac{\text{thickness}}{2}} \\ Q_{\text{flow}} &= G \cdot \text{dT} \end{aligned}$$

```
end Wall_Surface
```

Figure 4. A wall element model created in Modelica Standard Library.

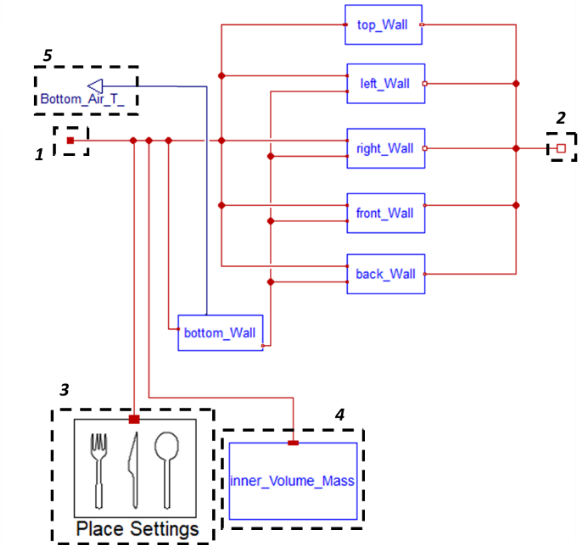


Figure 5. Dishwasher Cabinet Model in Dymola.

3.2 Algorithm Structure in Dymola

The complex algorithm of the dishwasher is primarily composed of two main steps, depending on the predetermined temperature value and time step. For example, during the main wash cycle, the water heating step is completed when the water reaches the desired temperature, while the cold rinse step operates based on the specified duration since there is no heating or specific water temperature set value. These two fundamental steps are reflected in the model by revising the "Timer" and "Trigger" models available in the logical library of Dymola shown in Figure 6.

The code revision allows the timer to run when the input is true and return false when the specified time period is over, providing flexibility in achieving the desired period independently of the simulation time. The trigger structure outputs a predetermined start value if the Boolean input is false and outputs the given input if it is true, enabling logical sequencing in the algorithm model. The water to be heated is modeled as a lumped mass, and different water models are used for each washing step.

The heating systems are controlled using the “timer” and “trigger” structures and associated with the cabinet structure. If the algorithm model is connected to a heat pump system, it can use the condenser capacity as the water heating capacity from that model. On the other hand, if a conventional heating analysis is desired, the heater capacity in the algorithm is inputted, and the water is heated accordingly based on this capacity. The algorithm model is communicated to both the cabinet and the heat pump model.

Power consumption by components such as circulation pump and compressor are known, allowing for dynamic power consumption and total energy calculation. The algorithm model is designed to be user-oriented by changing the time, temperature, and power parameter values out of the model, enabling simple parameter customization without interfering the code structure.

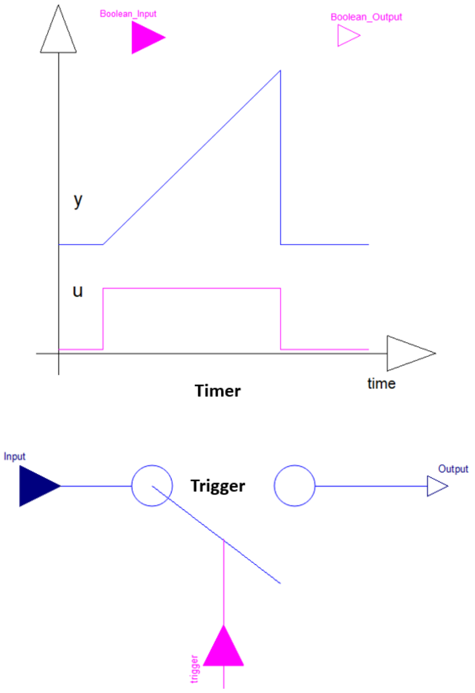


Figure 6. Algorithm components in Modelica Standard Library

3.3 Heat Pump System

The heat pump system located in the bottom chassis of the dishwasher enables much more efficient heating of the water compared to conventional heaters. To optimize the efficiency, it is crucial to ensure that all components of the heat pump system operate under optimal conditions. However, achieving this through experimental methods can involve significant workload and time. Therefore, modeling the heat pump system is vital, and system optimization, along with operating conditions, may be determined through parametric analysis on the model, which is much faster compared to experiments. In this context, each component of the heat pump system used in the dishwasher was individually modeled to create the heat pump system model by means of TIL Library introduced by TLK-Thermo GmbH.

3.3.1 Compressor Model

Reciprocating compressor model represented in the library is evaluated. The compressor is not considered as isentropic due to certain losses. For this reason, the model needs certain physical parameters simulating losses from the compressor. To illustrate, in order to simulate friction losses, friction coefficient parameters must be determined. They can be found by semi-empirical approach. Proceeding to the determination of these parameters, to begin with, compressor performance tests are completed in calorimeter system experimentally. In this test, input parameters which are condensation, evaporation, subcool, superheat and speed are assigned according to the working conditions of compressor in the system. Then outputs are achieved as cooling capacity, power consumption, mass flow rate and discharge temperature depending on working condition.

In parallel with this process, the compressor model tester is created in Modelica, therefore, input and output parameters are correlated with compressor model equations without any parameter assignation. Then the whole tester model is turned into FMU standardizing interface to be used in computer simulations to develop complex cyber-physical systems. Then it is imported to the ModelFitter (2022). Besides, the performance inputs and outputs obtained experimentally are introduced in ModelFitter. Thanks to statistical analysis, the physical parameters are estimated with R-square value of 0,95. Thus, physical parameters of the compressor model (heat losses, friction losses and so on) are calibrated depending on calorimetric measurements. So, the compressor model can calculate compressor power, mass flow rate, discharge temperature and capacity between at certain pressure levels.

3.3.2 Condenser Model

To create condenser model, the tube model in the TIL library is used and it is discretized as finite volume cells. For heat transfer coefficient governing heat transfer rate, correlations are used. In single phase region Dittus-Boelter (1985) correlation is used.

$$Nu = 0{,}023Re^{4/5}Pr^{1/3} \qquad (2)$$

However, for two phase, convenient correlation is not found in the library and literature for helical type coil condenser. For this reason, two phase heat transfer coefficient is determined experimentally from water side by using thermal resistance approach and LMTD method (see in Figure 7). To attain water side heat transfer coefficient Gnielinski (1976) correlation is used. In detail, during the calculation process, the condenser capacity of the heat pump system is determined by evaluating the enthalpy difference between the inlet and outlet of the refrigerant and multiplying it by the refrigerant flow rate. Subsequently, the capacity contributed by the single-phase region is subtracted from the total capacity to determine the capacity associated with the two-phase region. The convection coefficient on the water side is calculated using the Gnielinski correlation with known values of water inlet and outlet temperature, as well as water flow rate. The capacities of the water and two-phase refrigerant sides are equalized with each other and in this case, the two-phase heat transfer coefficient of refrigerant becomes the only unknown value. Also, the friction factor is determined based on the Moody diagram. The Gnielinski correlation and friction factor are shown in Equation 3 and 4.

$$Nu = \frac{\left(\frac{\zeta}{8}\right)(Re - 1000)Pr}{1.07 + 12{,}7\sqrt{\frac{\zeta}{8}}\left(Pr^{2/3} - 1\right)} \qquad (3)$$

$$\zeta = 0.184Re^{-0.2} \qquad (4)$$

On the other hand, geometrical parameters such as tube diameter, length and wall thickness are introduced to the tube model. In the cycle model obtained condenser capacity is considered for heating process of water.

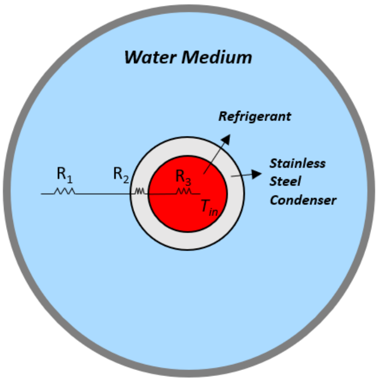


Figure 7. Condenser structure of dishwasher's heat pump.

3.3.3 Capillary Tube Model

Capillary tube is also a component presented in the TIL library. The pressure drop in the capillary tube can be examined in two region, single phase and two phase. Since the refrigerant is in liquid phase at the inlet of capillary tube, pressure drop takes place linearly depending on only friction. After certain point the first vapor bubble starts to form, and the refrigerant becomes two-phase state. In this region the pressure drop is non-linear and depends on both friction and momentum because of continuous density change of refrigerant. Pressure drop characteristic is represented by Navier Strokes equation and the friction term can be determined Swamee-Jain (1976) correlation. For fully developed flow, one dimensional Navier Strokes equation is shown in Equation 5.

$$\frac{dp}{dx} - \mu \frac{d^2u}{dy^2} = 0 \tag{5}$$

To illustrate, for two-phase region it is integrated along the tube, and it is described as in the Equation 6.

$$p.\pi.\frac{D^2}{4} - (p + \Delta p).\pi.\frac{D^2}{4} - \tau_w.\pi.D.\Delta L = m.\Delta V \tag{6}$$

Also, the total pressure drop in the capillary tube is expressed as in the Equation 7.

$$\Delta p = \left[\left(\frac{f}{2D}\right)\Delta L + \frac{\rho_{in} - \rho_{out}}{\rho}\right].\frac{G^2}{\rho} \tag{7}$$

Lastly, Swamee-Jean correlation offering explicit solution to determine the friction factor in the pressure drop equation. The Swamee-Jean correlation is given as in the Equation 8.

$$f = \left[-2\log\left(\frac{\varepsilon/_D}{3.7} + \frac{5.74}{Re^{0.9}}\right)\right]^{-2} \tag{8}$$

In addition, for single phase pressure drop inertia terms are neglected and pressure drop equation does not contain density change.

3.3.4 Evaporator Model

The heat exchanger in the form of a fin and tube structure, which is used as an evaporator in the system is also available in the TIL library similar to a tube structure. In this model, the Dittus-Boelter correlation is used when the refrigerant is in single-phase, while the Shah (1979) correlation shown in the Equation 9 is utilized for two-phase region. Additionally, for the air side correlation, the equation developed by Wang et al. (2002) specifically for the wavy fin structure is evaluated. Also, whole geometrical parameters such as number of serial and parallel tube distance, fin pitch, length and so on are introduced to the model depending on heat exchanger structure.

$$h = h_{liq}\left[(1-x)^{0.8} + \frac{3.8x^{0.76}(1-x)^{0.04}}{{p_r}^{0.38}}\right] \tag{9}$$

3.4 Dishwasher with Heat Pump Model

After individually modeling all the components of the heat pump system as mentioned in the previous sections, they are connected to each other to create a complete heat pump model. As seen in Figure 8, the compressor speed is controlled by a boundary condition. A tube model is used to simulate the heat transfer between the discharge, suction tubes and the surrounding environment.

Similar to the condenser, the dynamic temperature of the bottom chassis volume, which is generated in the cabinet model and increases during the experiment, is associated with the tube model through a resistance. Similarly, the bottom chassis temperature is used as the suction air for the evaporator, as it strongly influences the operating conditions of the evaporator and thus the heat pump system. Additionally, boundary conditions such as air flow rate are defined for the air-side boundary associated with the evaporator.

The behavior of the entire system is revealed through sensors, and the thermophysical properties of the refrigerant and air side are assigned using the System Information Manager model.

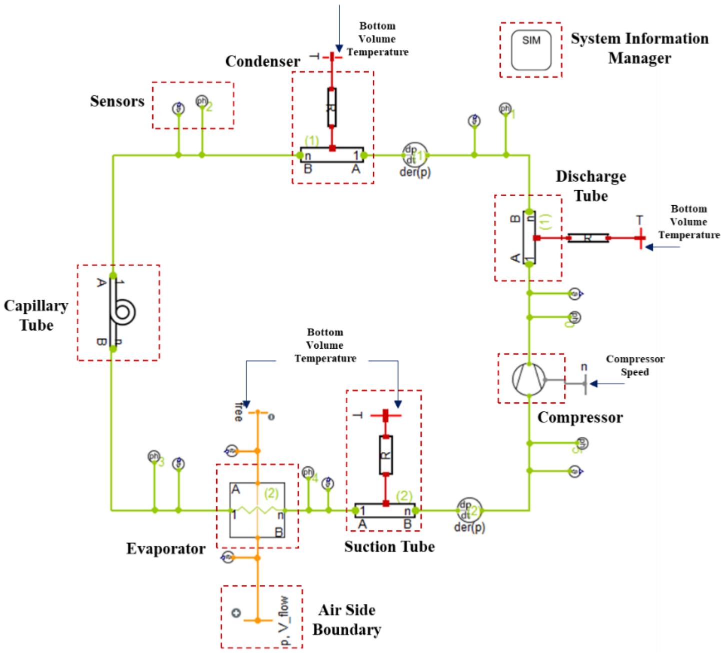


Figure 8. Whole heat pump model illustration in TIL Library.

4 Experimental Validation

After building the entire model, fine-tuning processes were conducted through experimental testing under standard conditions, European dishwasher test standard EN 60436. The fine-tuning parameters used included the thermal loss parameter, which simulated heat losses from the condenser surface to the cabinet, and the "tube roughness" in the capillary tube, which was adjusted to match the system pressure levels. In other words, each component should be fine-tuned before creating whole model. Because the real cases have always included chaotic circumstances. To illustrate, while validating heat loss from the dishwasher cabinet, the heat transfer between water circulated by spray arms hitting to the cabinet walls and cabinet walls cannot be predicted and calculated. On the other hand, the capillary tube in the heat pump system consists of tube roughness parameter excepting geometrical parameters, this parameter should be defined by far and away to converge pressure levels. For this reason, the model is in need of fine-tuning parameters like them. The fine-tuning primarily focused on optimizing the cabinet temperature and heating time. By adjusting these parameters, the model's accuracy and performance were improved to align with the experimental results. Subsequently, a comparative analysis was conducted between the model and experimental results of the base experiment, focusing on the temperature of the dishwashing cabinet, power consumption, energy consumption, and low-high pressure values. This assessment aimed to ascertain the degree of agreement between the model's predictions and the actual measurements obtained during the experimental testing. The behavior of experimental and simulated cabinet temperatures is illustrated in Figure 9. Maximum experimental-simulation difference is ±1 °C. In addition, the difference between the cycle time of the model and experiment is 0.5 minutes. The behavior of the model in the cabinet temperature is in line with the experimental outcomes.

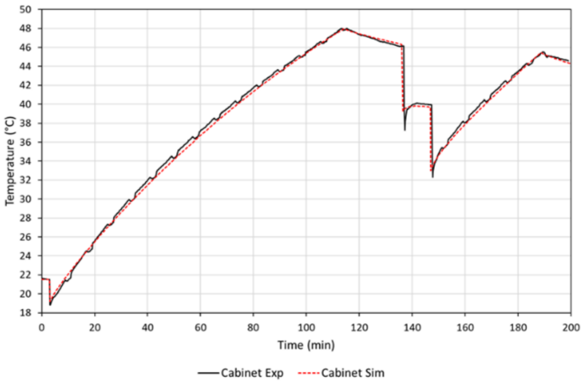


Figure 9. Cabinet temperature comparison.

Figure 10 illustrates the time-dependent power consumption during the dishwasher cycle. The power consumption is primarily influenced by the compressor, circulation pump, and drain pump. The circulation pump operates throughout most of the cycle and its power consumption varies depending on the algorithm-driven spray arm positions and speeds. This variability leads to oscillations in power consumption. However, since the model calculates the average pump power consumption, this oscillation is not observed in the model. The average power consumption deviation during the cycle is 2.4%.

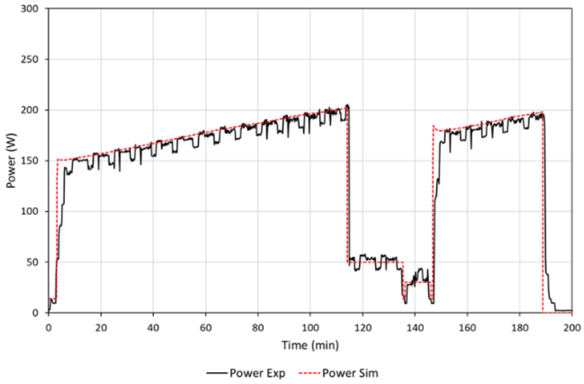


Figure 10. Power consumption comparison.

When comparing the model and experiment, it is observed that the model consumes more power than the experiment during the initial operation of the compressor. This difference can be attributed to the model's assumption of instantaneous evaporation temperature and the absence of accounting for the gradual increase in compressor speed for lubrication. In reality, it takes time for the refrigerant to reach the evaporation temperature, and the compressor gradually increases its speed. These factors result in a noticeable disparity in power consumption at the start of compressor operation. However, despite these variations, the overall trend in power consumption remains consistent between the model and experiment. The deviation in power consumption leads to a 2.9% difference in the overall energy consumption value.

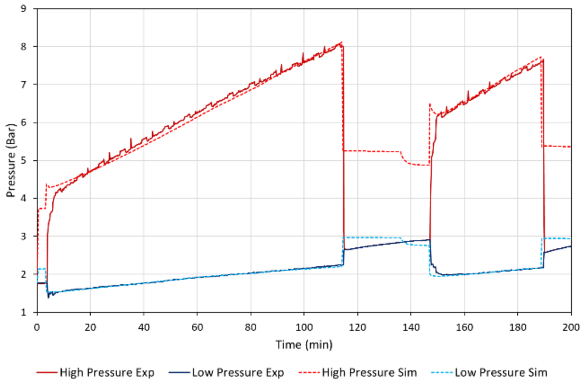


Figure 11. High- and low-pressure comparison.

The comparison between the model and experiment in Figure 11 shows that the model accurately predicts pressure drops and compressor performance along the capillary tube. However, there is a maximum inconsistency in the condenser pressure, with a deviation of approximately 0.23 bar, while it is 0.1 bar in the evaporation side. This difference is attributed to the use of fixed superheat values using the compressor model due to calorimeter limitations, while actual operating conditions may vary. Despite these discrepancies, the model fits the data well and allows for parametric analysis. It is important to note that numerical errors can explain significant deviations in the high-pressure results when the compressor switches on and off.

Following all validation procedures, the comparison between the model and experimental results for an entire cycle is presented in Table 3, providing the maximum deviations in cabinet temperature, power, energy, and high and low-pressure line values.

Table 3. Validation deviation results.

Cabinet Temperature	± 1 °C
Cycle Time	0,5 min
Power	2,4%
Energy	2,9%
High Pressure	0,23 bar
Low Pressure	0,1 bar

5 Conclusion

In this study, an object-oriented modeling methodology using the Modelica Programming Language has been presented for a dishwasher with heat pump . The study involves creating a sub model for each component of the heat pump dishwasher and then integrating all the models to obtain a comprehensive dynamic cycle model for the heat pump dishwasher. The cyclic behavior of the dishwasher has been experimentally validated. As a result, the developed dynamic model is able to respond parametric changes, by means of this, it could be used for optimizing processes. A prime example of this is that optimization studies for component and cabinet insulation sizing to increase system efficiency and reduce experimental effort.

Acknowledgement

The authors would like to express their sincere gratitude to Arçelik Global for their invaluable support throughout the course of this research. The contribution has been instrumental in the successful completion of this study.

References

European Commission, “Communication from the Commission to the European Parliament and the Council Brussels. No. COM (2016) 110 final”, 2016, https://eur-lex.europa.eu/ accessed on August 16, 2022.

Flück, S., M. Kleingries, E. Dober, I. Gau, P. Bon, A. Loichinger and B. Wellig, “Household Dishwasher with a Monovalent Heat Pump System”, paper presented at the 12th IEA Heat Pump Conference, Rotterdam, 2017.

Atasoy, E., B. Cetin and O. Bayer, “Experiment-based Optimization of an Energy-Efficient Heat Pump Integrated Water Heater for Household Appliances”, Energy, Vol. 245, p. 123308, 2022. doi: https://doi.org/10.1016/j.energy.2022.123308

Caglayan, A., S. M. Husain, M. Ipek, T. N. Aynur, S. Cadirci, “Dynamic Modeling and Experimental Validation of a Domestic Refrigeration Cycle”, Journal of Thermal Science and Engineering Applications, Vol. 14, p. 071007, 2021. doi: https://doi.org/10.1115/1.4052453

Husain, S. M., M. Ipek, A. Caglayan, T. N. Aynur, S. Kocaturk, H. Bedir, “Dynamic Modelling of The Steady State and Load Processing Operation of A Domestic Refrigerator Cooled Through Natural Convection”, International Journal of Refrigeration, Vol. 146, pp. 15-27, 2023. doi: https://doi.org/10.1016/j.ijrefrig.2022.08.015

Ipek M., I. Dincer, Experimental Investigation and Dynamic Modelling of an Upright Freezer Under Periodic Door Opening Conditions” International Journal of Refrigeration, 2023. doi: https://doi.org/10.1016/j.ijrefrig.2023.06.004

Caskey S. L., E. A. Groll, “Dishwasher Modelica Model Analysis with an External Heat Loop” paper presented at the International High Performance Buildings Conference, Purdue, 2018. URL https://docs.lib.purdue.edu/ihpbc/331/

EN60436, “Electric Dishwashers for Household Use – Methods for Measuring the Performance,” European Standard, 2020.

S. Kline and F. McClintock, “Describing Uncertainties in Single-Sample Experiments”, Mechanical Engineering, Vol. 75, pp. 3-8, 1953. doi: https://doi.org/10.1016/0894-1777(88)90043-X

Dittus, F. W. and L. M. K. Boelter, “Heat Transfer in Automobile Radiators of the Tubular Type”, International Communications in Heat and Mass Transfer, Vol. 12, pp. 3-22, 1985. doi: https://doi.org/10.1016/0735-1933(85)90003-X

Gnielinski, V., “New Equations for Heat and Mass Transfer in Turbulent Pipe and Channel Flow”, International Chemical Engineering, Vol. 16, pp. 359-368, 1976.

Swamee, P. K. and A. K. Jain, "Explicit Equations for Pipe Flow Problems", Journal of Hydraulics Division, Vol. 102, pp. 657-664, 1976.

Shah, M. M., "A General Correlation for Heat Transfer During Film Condensation Inside Pipes", International Journal of Heat and Mass Transfer, Vol.22, pp. 547-556, 1979. doi: https://doi.org/10.1016/0017-9310(79)90058-9

Wang, C., Y. Hwang and Y. Lin, “Empirical Correlations for Heat Transfer and Flow Friction Characteristics of Herringbone Wavy Fin-and-Tube Heat Exchangers”, International Journal of Refrigeration, Vol. 25, pp. 673-680, 2002. doi: https://doi.org/10.1016/S0140-7007(01)00049-4

TLK-Thermo GmbH. TIL3.9.1, 2020. URL http://www.tlkthermo.com/index.php

TLK-Thermo GmbH. ModelFitter, 2022. URL https://www.tlk-thermo.com/index.php/en/software/60-modelfitter

Dassault Systems, Dymola: Multi-Engineering Modeling and Simulation based on Modelica and FMI, 2021. URL https://www.3ds.com/products-services/catia/products/dymola/

A Library to Simulate Processes in the Factory Hall

Julia Gundermann   Torsten Blochwitz

ESI Germany GmbH, Dresden, Germany {julia.gundermann, torsten.blochwitz}@esi-group.com

Abstract

The Modelica language is well suited to model systems with coupled discrete and continuous dynamics. This feature is crucial, if one wants to model the flow of items through manufacturing steps such as preparation, mounting, or transport in the shop floor. The library *ProcessSimulation* can be used to model such processes. By default, it omits the technical details of the process steps, and focuses on the flow of material items through the process steps. In addition to that, a base model to calculate the energy consumption in the different manufacturing steps is provided. It can be enriched with technical details of the components. The library can be used for the calculation of (net) energy consumption, but also for task planning.

Keywords: process simulation, energy consumption, Modelica library

1 Introduction

There are dedicated and advanced commercial tools for plant or manufacturing simulation and beyond, such as *Tecnomatix* (2023) (Siemens), *FlexSim* (2023) (FlexSim Software Products, Inc.) or *Arena Simulation Software* (2023) (Rockwell Automation). They base on discrete event simulation and cover a wide range of applications. To support modelling activities, some of these programs also integrate sophisticated 3D visualisation. There is also a Modelica library which can model discrete event simulation, since it adopts the Discrete EVents System (DEVS) formalism (Sanz, Urquia, and Dormido 2009; Sanz, Urquia, Cellier, et al. 2012; *DESLib* 2023). It offers a rich but complex functionality. The purpose of the library presented here is to evaluate the capabilities of simulating processes in the factory hall by means of the Modelica language with a simpler approach. It is shown that there are applications for which this way of modelling is sufficient. The Modelica language can cover both discrete and time-continuous processes. It is suited, if machines and transport means should not only be considered as event sequences, but can also be enriched with models of physical processes, i.e. mechanics and electrics which for example contribute to the consumption of energy. This publication introduces the library and its components and outlines two applications.

2 The Library

2.1 General

This library contains models of distinct groups to describe material flow through processes in the factory hall. These are

- storages to store items,
- machines to process items,
- transport devices to transport items,
- a tasks supervisor to model tasks and their preconditions on machines,
- an energy meter to observe energy consumption in the system.

The *material* which flows through the manufacturing line is identified by its amount as integer quantities. This is in contrast to the continuous material amount, which is used in the *Business Simulation Library* (2023). On the other hand, *material* is also not a set of single-wise identifiable items, like in DEVSlib (DEVSlib is a subpackage of *DESLib 2.0* (2023)). The Modelica language is not able to handle the generation and disappearance at simulation time. For that reason, DEVSlib implements an External Object in programming language C to handle a variable number of messages and entities. In our approach we only count the number of items. The item flow is handled by the definition of pairs with active and passive partners using specific connector types. There is always one active and one passive partner in the connection. The passive partner indicates its availability (`freeCapa`) and free/available `capacity`, the active partner triggers the `handOver` and defines the handed quantity (`handedCapa`). The *active* connector is defined as follows:

```
connector MaterialA
  "Active Material Handover"
  input  Boolean freeCapa;
  output Boolean handOver;
  input  Integer capacity;
  output Integer handedCapa;
end MaterialA;
```

The *passive* connector `MaterialP` is defined in the same way with input and outputs exchanged. Figure 1 shows the connectors. With the Boolean parameter `handOver` the events are triggered in which one or more items are handed over from one component to another. In this way,

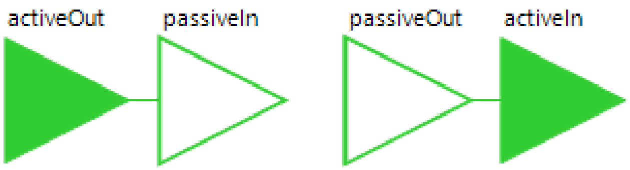


Figure 1. Connector pairings of the process simulation library. There is always one active and one passive partner in the connection.

the flow of material is as fast/efficient as locally (i.e. in a sequence of two consecutive components) possible. It should be mentioned that this modelling technique creates a lot of events during simulation. The number of variables which change as a consequence of these events is low, and their values are calculated from simple equations. Hence, the duration of a single event iteration is determined by the solver's performance and in general short, however the high number of events in total might affect simulation time.

2.2 Storages

There is a generic component named `Storage` which models the storage of items, e.g. in a storage room, but also on a wagon, or a dedicated place upfront a machine. It can be parametrized with an initial capacity and a maximal capacity. The storage is passive regarding material flow. This means it indicates its free capacity, but the process of handing over material is triggered from the connected transport device (see next section). There exists a variant of the storage named `MergeStorage`. This can be used, if a certain ratio of components of two different types (e.g. four wheels and a chassis) has to be available before processing the next step (e.g. transport to an assembling machine). Figure 2 shows the icons.

2.3 Transport Devices

There are three types of transport devices. They share the commonality that all of them actively trigger the loading and unloading of items. Items are loaded if the preceding storage has enough items to transport, and they are unloaded if the target has enough free capacity. Otherwise, the transport stops.

Plain Transport

This transport device loads a parametrizable number of items within a `loadTime`, transports them within a `transportTime` and unloads them at a connected target. The transport runs either up to a maximum defined number, or indefinitely as long as there are items to transport.

Conveyor

This transport device models a conveyor belt. It needs `processTime` to transport one item from start to target, and loads up to a parametrized `maxCapa` of items (all separated by `processTime/maxCapa`).

Shared transport device (e.g. automated guided vehicle AGV)

This device models the transport as defined in connected `TransportTasks`. All the information about the transport task is defined in the connected device. This includes

- the number of items to carry
- the number the device can carry at once
- the time to load the device
- the time to transport the loaded items
- a condition when the transport task is prepared to be run (e.g. to model that a transport task will only be started if there is enough free capacity in the target storage).
- a unique id

Each transport task is connected to a `SharedResource` by a specific connector pair, in which the shared resource receives each task's `id` as well as its status (prepared, finished), and sends the `currentTaskNr` to all transport tasks.

The shared resource exists in two variants - one runs the tasks "AS PLANNED", i.e. as defined by the sequence of task IDs in the shared resource. The other runs the tasks "ON DEMAND", i.e. whenever a transport task becomes status "prepared", it will be scheduled as the next transport task. If several tasks become prepared at the same time, they will be scheduled in order of increasing task id.

Figure 3 shows the icons of the transport devices. All transport devices display the number of currently transported items. In addition to that, each transport task shows

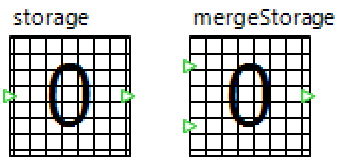


Figure 2. Storage and merge storage component. The number on the icon displays the number of (merged) items which are currently in the storage.

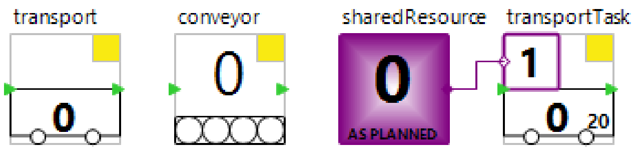


Figure 3. Transport devices of the process simulation library. From left to right: transport, conveyor, and shared resource with transport task.

in its top left corner the unique task ID, which is highlighted in violet if the task is currently running. The bottom right corner shows the total number of items to transport. The icon of the shared resource shows the ID of the currently active transport task.

2.4 Machines

The library contains a plain `Machine`, which processes items. This refers to any type of process - mounting, drilling, packing, sawing, ... The plain machine runs down a `setupTime` to prepare the machine, and a `cycleTime` to process one item. It processes `nMax` items, then it stops. An item is taken and processed as soon as it is available in the connected input storage, and handed to the output storage if there is free capacity. Two output variables indicate the progress of the machine `setup`, and the `progress` of processing the current item, respectively. A `machine` is always located between two storages, one from which the items are taken, and a second one which is filled with items. Figure 4 shows two storages connected by a machine which processes items. The numbers displayed in the storages and machine vary over time and show the number of stored or processed items, respectively.

2.4.1 Machines with Tasks

Whereas this plain `Machine` runs only one process to its end, there are four more advanced machine types in the library (cf. Figure 7). They all share the following base structure: they all are vectorized versions of the plain `Machine` connected with start and target storages (configuration as in Figure 4). This structure is used to define and simulate a sequence of tasks. For each of these tasks material is taken/delivered from/to its dedicated storage. Each task has its own parameter values for number of items, setup time and cycle time.

The storages before and after the machine are empty, and have to be filled/emptied by other active processes (i.e. transport, or predecessor machine). For example, in Figure 5, transport devices have been connected. The passive connectors to the input and output storages have the size `n1`, `n2`. The values of these integer parameters are zero by default, and grow with the number of connected components (due to the Dialog annotation `connectorsizing=true`). The machine needs the same number of connections on both sides, an assert is thrown, if `n1<>n2`. The sequence of connecting the components is important. The first connected transports "belong" to the first set of task parameters, and so on.

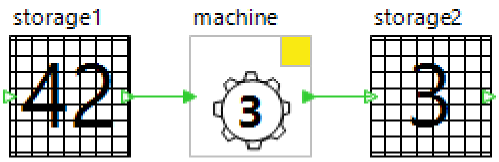


Figure 4. The Machine component, connected with an input storage and a target storage.

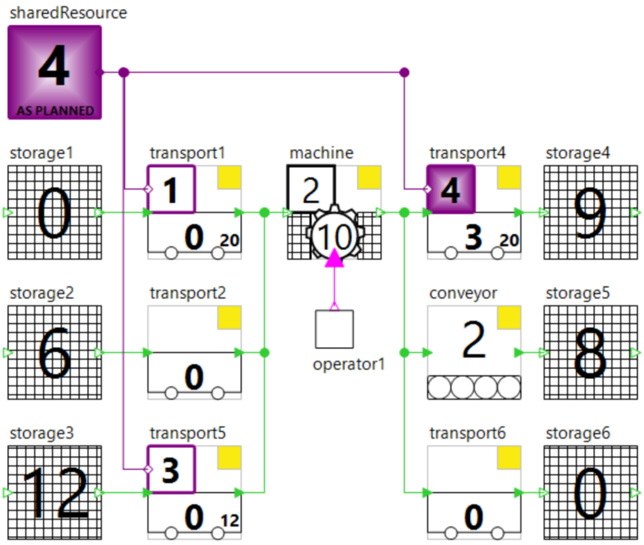


Figure 5. The Machine with local tasks. The example contains three tasks on the machine, hence three input and output processes (all transport) have been connected. The numbers on the machine icon indicate that it currently runs task 2, in which 10 items have been processed.

There are two additional features to highlight:

- The model of the machine with tasks contains an array-parameter `taskIDsC`, with which one can define positive integer IDs for all tasks (C stands for connected). In addition to that, there is a second parameter `taskIDsO` which is a reorder of the task IDs (O = ordered). This parameter allows to change the order of the tasks. Since the start or finalization of tasks can depend on transport processes, a reordering could improve total process time.
- The machine has an additional connector, to which an `Operator` must be connected. The machine runs only if the operator is available. With this additional condition shift durations or breaks can be modelled.

The machine variant with all this functionality is named `MachineWithTasks`. Further variants are extensions of this type and are explained below.

2.4.2 Machines with global tasks

If one wants to model tasks on different machines, which can depend on each other, there is a dedicated type named the `MachineWithGlobalTasks`, which is an extension of the `MachineWithTasks`. A task on this machine type will only start when all preconditioned tasks (on the same or other machines) have finished. Therefore, this machine type contains an `outer GlobalTaskList tasks`, which parametrizes each task's preconditions and controls their fulfilment. To do so, the global task list contains the number `n` and IDs of all tasks. It also contains a matrix parameter to define each task's preconditions[n,p], to denote up to p preceding tasks per task. Figure 6 shows the icons of the machine and the tasks component.

2.4.3 Machines with prefilled storages

There is a variant of the machine with tasks, which exists for both the machine with local tasks and global tasks. At simulation start, this variant has all start-storages in the machine filled with the capacity as defined in the parameter vector nMax. The maximum capacities of the target storages are set to the same values, respectively. This machine model no longer contains any (passive) material connectors. This model can be used, if only planning the processes on the machines is of interest. To reflect the auto-filled storages in the types, the type names have an "SC"-suffix (for self-contained), i.e. they are named MachineWithTasksSC and MachineWithGlobalTasksSC, respectively.

The icons of these four variants, which are all possible combinations of the features "local or global tasks" and "with connectors or prefilled storages", are displayed in Figure 7. Section 3.2 shows a small example of this functionality.

2.5 Energy Meter

An additional feature of this library is the calculation of total power and energy consumption. Most of the components in the factory hall consume or provide power. This is reflected in the library, all transport devices and the machine model contain variables and equations to calculate their power consumption. We define a global EnergyMeter, to easily sum them up to determine the total power and energy consumption, and use the Modelica inner/outer connection to collect all power terms from all components. This avoids the manual connection of all consumers or producers of power with the EnergyMeter, which would reduce clarity on the diagram view of the model. To facilitate the collection of power terms, the library defines an EnergyContributor, which is the base type of the contributing components. Between all energy contributors and the outer meter, a connection with a flow variable is created. Each energy contributor adds its power P to this connection automatically, the value of the energy meter's connector is the negative sum of all these contributions (since flow connections establish a sum-to-zero coupling), hence the total power consumption is determined by the negative value of the connector. Listing 1 shows part of the energy contributor's and energy meter's definition. Figure 8 shows the icon of the energy meter.

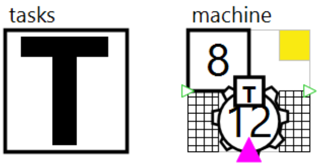


Figure 6. The Machine with global tasks, and the global task list. The machine has the same functionality as the machine with local tasks, but it needs an outer global task list, in which task's preconditions can be parametrized.

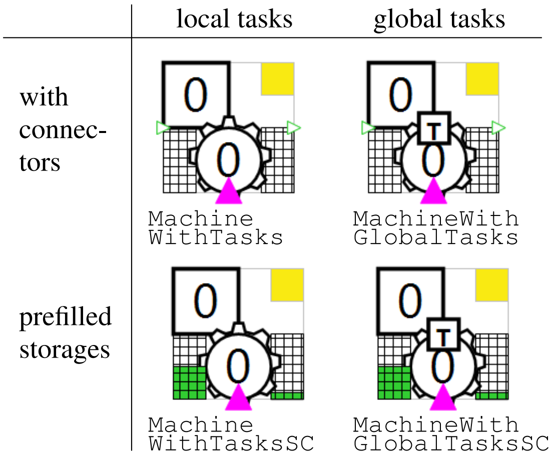


Figure 7. Variants of machines with tasks

Listing 1. Modelica code snippet outlining the energy meter and contributor

```
connector FlowCtr
        "Flow Connector for Meters"
 flow Real i "Flow Variable";
end FlowCtr;

model EnergyMeter "Energy Meter"
  Real P "Power";
  Real E "Energy";
 protected
  FlowCtr pc "Power Collector";
 equation
  P = - pc.i;
  der(E) = P;
end Energy;

partial model EnergyContributor
        "Energy Contributor"
  outer EnergyMeter energyMeter;
  Real P "Power
        - eqn. defined by derived type";
 protected
  FlowCtr power;
 equation
  power.i=P;
  connect(power, energyMeter.pc);
end EnergyContributor;
```

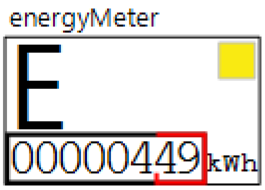


Figure 8. The Energy Meter component, displaying the total energy consumption like an electricity meter.

It shall be mentioned that the defined connector

`FlowCtr` is not conform with the requirement as stated in the Modelica specification, section 9.3.1. (Modelica Association 2023): The connector is unbalanced, i.e. the numbers of flow variables is not equal to the number of variables that are neither parameter, constant, input, output, or stream. According to (Olsson et al. 2008) this prevents any model using this connector from being locally balanced. However, this request was derived for systems with multiple components of the same type. When using the `ProcessSimulation.EnergyMeter` and (extended) `EnergyContributor`s, any allowed configuration contains exactly one energy meter and zero or more energy contributors. Any such combination has a balanced number of unknowns and equations. Furthermore, the `FlowCtr`s `pc` in the meter and `power` in the contributor are declared as `protected`, which at least produces a warning in some Modelica compilers. Here it is used to prevent any component which is neither meter nor contributor from being connected to the power balance. Defining the connectors as public but omitting annotations for the Placement could have the same effect, at least in the diagram view of the model.

2.6 Comparison to existing libraries

As mentioned, there are Modelica libraries which cover applications that are also adressed by the Process Simulation library (PSL): the *DESLib* (2023) (or *DESLib 2.0* (2023)) and the *Business Simulation Library* (2023). In this section, the differences are outlined. This is not meant to be a comprehensive summary of the other libraries' functionality, merely only supposed to help figuring out for which application the Process Simulation library is advantageous or sufficient, or when to rely on existing libraries.

The Business Simulation Library (BSL) contains - beside many others - classes which are comparable to those in the PSL - e.g., `Oven` (as a variant of a machine), `Conveyor` or `MaterialStock`. Different to the PSL, it uses real-valued material flow rates instead of integer numbers of material items. This avoids events due to handovers between components. Besides, the details of class parametrization differ. To name a few differences: the BSL `Oven` prepares batches of parametrizable size, there are also parameters for the setup and process of the batch, and a loading time. This is different from the PSL `Machine`, with has a "batch-wise"(`nMax`) setup time, but element-wise process times. The BSL `Conveyor` uses the Modelica `delay` operator which creates a time lapse between the inflow and outflow of material. This delay is triggered within a `when sample(..)` statement with a model-wide sampling frequency which is by default 16/s, i.e. it creates 16 time events per seconds, independent of the flow rate. This differs from the PSL `Conveyor` which creates (2 `maxCapa/processTime`) state events per second.

The DEVSlib (sub-package of *DESLib 2.0* (2023)) implements the Parallel Discrete Event System Specification (PDEVS) formalism (Zeigler, Prähofer, and Kim 2000), which is powerful to describe (parallel) discrete event systems together with continuous state systems. To create such a system, an *Atomic DEVS* is defined. The package provides an `atomicDraft` model, which implements this Atomic DEVS, and a guide how to create own models from a duplicate of this, to model a DEVS system. Due to the underlying formalism the application range is big, among the provided examples are controllers coupled to physical systems, game of life, and a supermarket model.

In the process oriented DEVS formalism, "systems are described from the point of view of the entities that flow through them using the available resources" (Sanz, Urquia, and Dormido 2009). For assembly line models this means that it is possible to track and identify each manufactured item during its flow through the line. This differs from the approach in the Process Simulation library, where the manufactured items are counted as integer numbers. There are applications where this is sufficient, e.g. to determine the energy consumption in an assembly line.

Furthermore, DESlib provides mechanisms to introduce stochasticity into the models (by the package RandomLib). As stated in (Sanz, Urquia, and Dormido 2009), process-oriented models are usually stochastic, which is why the generation of random numbers is necessary. Models created with the Process Simulation library are fully deterministic, material in assembly lines or tasks on machines run as planned. In the existing library components, random effects could modify the conveyor belt's velocity, machine/task process times, or others, which cause delays or accelerations in different manufacturing steps, and would result in variations of total process times. Since no systematic analysis of the random effects on the system's behaviour can be provided, and the result of single simulations with random effects are hard to interpret, such effects were not implemented in the current version. However, the result of the "deterministic" simulation provides all information to determine mean process times or energy consumption.

The currently available *DESLib 2.0* (2023) library version does not provide an example for an assembly line, i.e. no class models such as machines, stocks, conveyors. Hence a comparison of functionality or performance is not immediately possible.

3 Examples/Use Cases

In this section, two examples which outline the different facets of the Process Simulation library are presented.

3.1 Example 1: Energy analysis of the manufacturing of a three-wheeler

Figure 9 shows a screenshot of the (simplified) production line of a three-wheeler. As several of the components are `EnergyContributor`s, the component `inner Energy energy` is needed in the model. The contributions to the

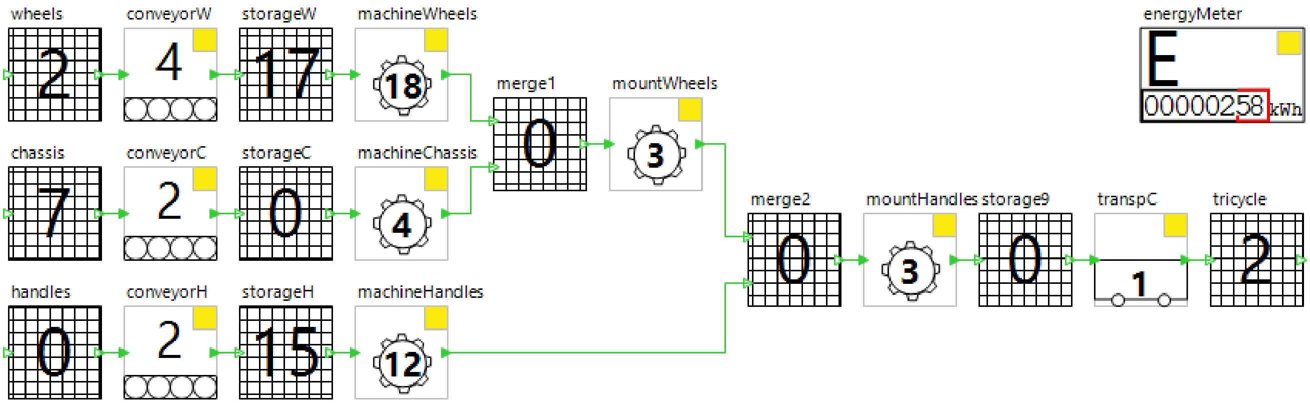


Figure 9. Simplified production line of a three-wheeler to illustrate the usage of the energy consumption. It contains pre-processing of wheels, handles and chassis, followed by two mounting machines. The transports to and from the machines are by either conveyors or plain transport (e.g. carrying). All components with a yellow box in the top right corner contribute to the overall power and energy consumption.

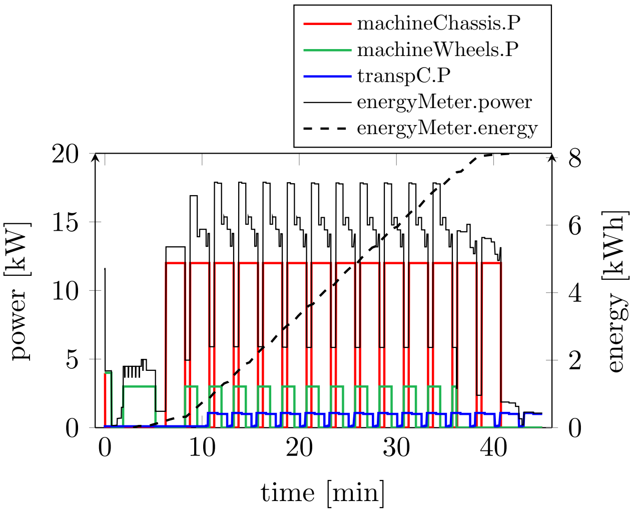


Figure 10. Power consumption of the different components of the three-wheeler production line. Shown are only the consumptions of one machine, one transport and the conveyor. Besides that the plot contains the total power and total energy consumption.

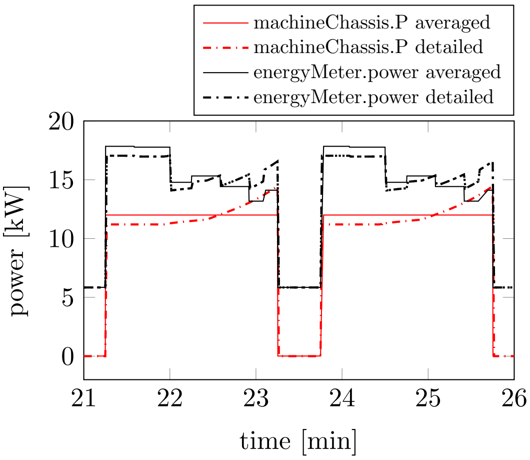


Figure 11. Power consumption of the three-wheeler manufacturing line with a detailed power modelling in the `machineChassis` component in comparison to the averaged original version. The figure shows two cycles of the machine only.

power (and energy) consumptions are parametrized within the different components. Figure 10 shows the power consumption of selected components and the entire manufacturing line. One can see that the power consumption has peaks of 17.8 kW, the main contribution stemming from the `machineChassis`. The manufacturing of 14 three-wheelers consumes an energy of 8.13 kWh in total.

One possibility to examine the power consumption in more detail is to extend the elementary components with more comprehensive models of the underlying processes. In the example considered here, the main energy contributor of Figure 9 is `machineChassis`. To enhance the calculation of the power consumption for this component, a Functional Mockup Unit (FMU) which models a detailed manufacturing process with non-constant power consumption, was imported to an extended copy of the `Machine` class. The original `machineChassis` component is replaced by a component of the new class.

Figure 11 shows the variation of power consumption of the modified machine in contrast to the previously constant value. This affects the peak power consumption, it reduces to 17.1 kW, but has no influence on the total energy consumption of the production line.

3.2 Example 2: Planning of tasks on machines and shared transport devices

This second example is created sequentially to illustrate the different machine variants, used together with other library types.

a) We start with the simple scenario of a single machine, on which three tasks have to be run in sequence. We do not care for the transport to and from the machine, we simply assume the material is there. We want to know how long it takes to run these tasks. The component to model this is the `MachineWithTasksSC`. Table 1 shows the details of the tasks. Figure 12 shows the diagram view of this example, and a result plot with the number of processed items. It should be mentioned that as soon as the last cycle of a task is completed the machine switches to the next task, hence the variable displaying the processed items per task peaks at (`nMax`-1) items in each task (in the example 4, 2, 9). From the graph one can read that the three tasks are finished after 7.5 hours.

Task ID	nr. of items	cycle time*	setup time*	machine	preconditions
1	5	45	20	1	-
2	3	15	20	1	-
3	10	12	20	1	1
4	50	2	10	2	2
5	35	3	10	2	-

(*in minutes)

Table 1. Details of the tasks in Example 2.

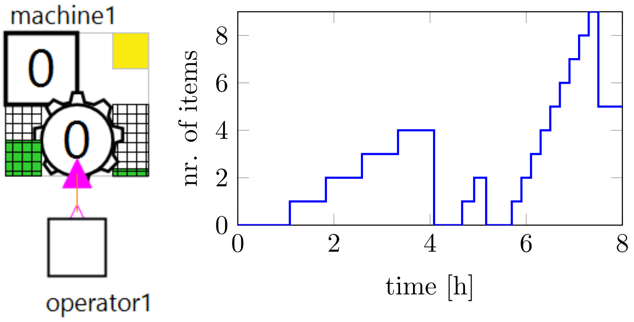


Figure 12. Example 2a: One machine with a sequence of three tasks, and a result window showing the processed items per task.

b) Now assume there are more tasks on a second machine, and there are some preconditions. Task 3 can only be run after task 1, task 4 only after task 2. Again, refer to Table 1 for the task details. To model this, we have to use the type `MachineWithGlobalTasksSC`, and the `inner GlobalTaskList tasks`, the latter parametrizes and controls the preconditions to the tasks. Figure 13 shows the model.

In this example the orders of the tasks on the machines influence the total process time. With this configuration one could test various task orders to figure out which one is the fastest. Technically this is done by modifying the value of the parameter `taskIDsO` on each machine to define the order of the tasks. With this small number of tasks and precondition one can find optimal solutions by choosing sensitive orderings. It makes sense to run task 5 on machine 2 first, since task 4 has to wait for the finalization of task 2. Furthermore, task 3 should be executed as last one on machine 1, since tasks 1 and 2 are preconditions to other tasks. With a small number of simulations, one finds that the following task orders result in the same and fastest total simulation time, which is 7.5 hours.

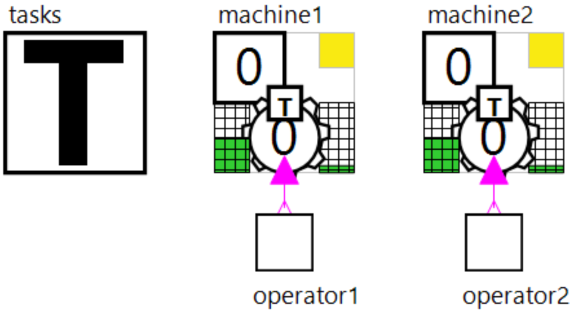


Figure 13. Example 2b: Two machines with five tasks in total.

	Machine 1	Machine 2	Total time
Variant 1	{1, 2, 3}	{5, 4}	7 h 30 min
Variant 2	{2, 1, 3}	{5, 4}	7 h 30 min

c) As a third step we want to consider not only the processes on the machines, but also transport processes to and from them. Figure 14 shows the model. The machine types have been changed to `MachineWithGlobalTasks`, and have passive material connectors. The transports to the machines are all provided by an AGV, five transport tasks have been created. The IDs of the transport tasks (displayed in the top left corner of the `transportTask` components) are equal to the machine tasks which follow - this is merely an (intended) coincidence. Transport task 1 takes 11 minutes per component, whereas the others are all faster (3.5 minutes). It should be determined which transport task sequence and machine task sequence leads to the fastest finalization of all machine tasks. To answer this question, we analyse reasonable transport task orders together with the two fastest machine task variants. The following considerations help excluding some of the 120 variants: Firstly, the preconditions of machine tasks should be the same for the transport tasks (this transfer of numbers works here since we chose the same numbering, and only have transport tasks *to* machines). Secondly, the order of machine tasks should be kept in the order of transport tasks (i.e., transport task 1 before 2 before 3, and 5 before 4, if to test with machine tasks variant 1). Taking these restrictions into account there are 7 transport task orders to test for variant 1, and 9 for variant two.

From simulation results one can read that the transport task order {1,5,2,3,4} together with machine task variant 2 leads to the same total time to finalize all machine tasks, which is 7.5 hours. See Figure 15. The first task (task 1) on machine 1 is not delayed by transport task 1, since the transport of the first item has finished before the machine setup. Machine task 5 is delayed but is not precondition

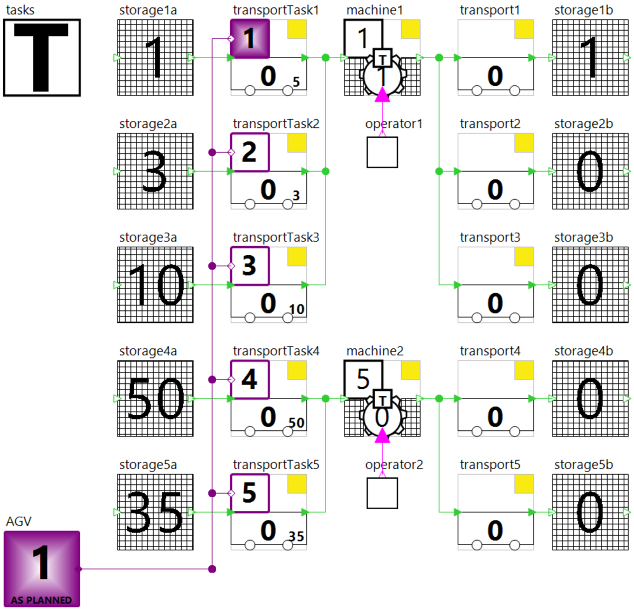


Figure 14. Example 2c: Two machines with five tasks, and transport processes to and from the machine. The transport to the machine is provided by an AGV.

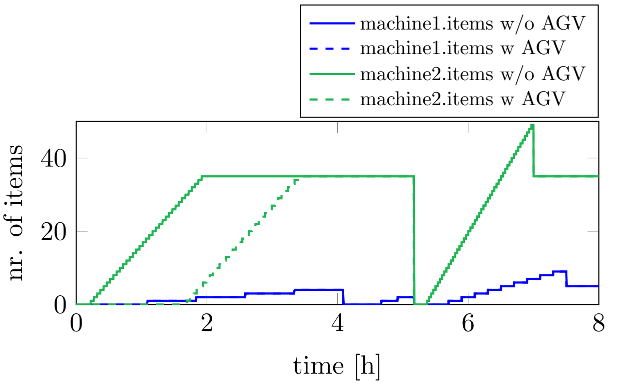


Figure 15. Example 2 b/c: Processed items on machines 1 and 2 in the variants of the model without transport and with transport (AGV) included.

to any other task. By the time machine task 5 has finished on machine 2, the necessary machine task 2 on machine 1 has finished such that machine task 4 can start right away. In total, the AGV transport does not affect the machine task finalization for this combination, and all tasks can be finished within an eight hour shift.

With this example the different variants of the machines were illustrated, in combination with the usage of the shared resource as a transport device. A combination with other transport means, like in subsection 3.1 is also possible. Of course, for more complex scenarios finding the optimal sequence of machine and/or transport tasks becomes challenging. In such cases, a dedicated optimizer should be used, it is beyond the scope of this library. The simulations are very fast, and a reordering is only a reparametrization of the model. Model creation, modification, simulation, and request of results is all possible via a scriptable interface, which eases the connection to other tools like optimizers.

One realization of such a task plan simulation in the configuration as described in example 2b was developed in the DIMOFAC project[1], where the challenge was to optimize tasks on several machines. The information about tasks and machines is provided in asset administration shells (AASs). A dedicated optimization tool for task planning (Kousi et al. 2019; Evangelou et al. 2021) reads this information, determines candidates of task orders and generates simulation requests for all of them. Based on these requests, a Modelica model is created (in *SimulationX* (2023)), which reads the task and machine specific information from their AASs. After the simulation, metrics like net machine utilization or process time are returned to the optimization tool, which evaluates this information to create new simulation requests and finally identify the optimal task schedule.

4 Summary and Outlook

In this publication the *ProcessSimulation* library was presented. It can be used as a low-level entry point to model material flow through a production line, to evaluate the performance of a manufacturing system in terms of energy and machine or transport task order. The library can be extended if needed, e.g. by multiple-merge storages, or machines with multiple material outputs. Besides, the consideration of random effects on the process times remains an open issue. Regarding the machine types the library can flexibly be extended such that the calculation of the energy consumption becomes more fine-grained. One could use this to identify/reduce maximum demand loads to the power station which provides the energy in a manufacturing line, which helps to save costs. The concept of an energy meter which monitors the consumption (or production) of energy can be transferred to other electricity net types (230V, 400V, high voltage), heat, water, or compressed air consumption.

In 3.2, a scenario to optimize production plans was illustrated. A separate optimization tool was used to determine the optimal schedule of tasks. For a closer integration to the simulation environment (*SimulationX*), it remains future work to develop python scripts (e.g., using dedicated libraries, such as *Python MIP (Mixed-Integer Linear Programming) Tools* (2023)) for the optimization of tasks on machines and transport devices. The necessary python interface exists in *SimulationX*.

The modelling of the material transport between components was realized by connectors which trigger the handover immediately when material and capacity is available, and all components work with pre-planned velocity. A remaining task is to extend this concept with signal interfaces to externally control the processes - handover, status of machines, transport/manufacturing velocities. This

[1] https://dimofac.eu/ (accessed on June 1, 2023)

could be used for virtual commissioning of a manufacturing line, to evaluate the reliability and robustness of control signals for the line on a simulation model of it.

Acknowledgements

This research has been supported by the European Union's Horizon 2020 research and innovation programme under the grant agreement No 870092, the project DIMOFAC (Digital and Intelligent MOdular FACtories).

References

Arena Simulation Software (2023). URL: https://www.rockwellautomation.com/en-us/products/software/arena-simulation.html (visited on 2023-05-23).

Business Simulation Library (2023). URL: https://github.com/modelica-3rdparty/BusinessSimulation (visited on 2023-05-23).

DESLib (2023). URL: http://www.euclides.dia.uned.es/DESLib/ (visited on 2023-08-15).

DESLib 2.0 (2023). URL: http://www.euclides.dia.uned.es/vsanz/files/DESLib-2.0web.zip (visited on 2023-08-15).

Evangelou, George et al. (2021). "An approach for task and action planning in Human–Robot Collaborative cells using AI". In: *Procedia CIRP* 97. 8th CIRP Conference of Assembly Technology and Systems, pp. 476–481. DOI: https://doi.org/10.1016/j.procir.2020.08.006.

FlexSim (2023). URL: https://www.flexsim.com/manufacturing-simulation/ (visited on 2023-05-23).

Kousi, Niki et al. (2019). "AI based combined scheduling and motion planning in flexible robotic assembly lines". In: *Procedia CIRP* 86. 7th CIRP Global Web Conference – Towards shifted production value stream patterns through inference of data, models, and technology (CIRPe 2019), pp. 74–79. DOI: https://doi.org/10.1016/j.procir.2020.01.041.

Modelica Association (2023). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.6*. Tech. rep. Linköping: Modelica Association. URL: http://www.modelica.org.

Olsson, Hans et al. (2008). "Balanced Models in Modelica 3.0 for Increased Model Quality". In: *Proceeedings of the 6th International Modelica Conference*. Linköping University Electronic Press. LiU Electronic Press, pp. 21–33. URL: https://elib.dlr.de/55892/.

Python MIP (Mixed-Integer Linear Programming) Tools (2023). URL: https://pypi.org/project/mip/ (visited on 2023-06-05).

Sanz, Victorino, Alfonso Urquia, François E. Cellier, et al. (2012). "Modeling of hybrid control systems using the DEVSLib Modelica library". In: *Control Engineering Practice* 20, pp. 24–34.

Sanz, Victorino, Alfonso Urquia, and S. Dormido (2009-10). "Parallel DEVS and Process-Oriented Modeling in Modelica". In: *Proceedings of the 7th International Modelica Conference*. Linköping University Electronic Press. LiU Electronic Press. DOI: 10.3384/ecp09430104.

SimulationX (2023). URL: www.SimulationX.com (visited on 2023-06-07).

Tecnomatix (2023). URL: https://plm.sw.siemens.com/en-US/tecnomatix/products/plant-simulation-software/ (visited on 2023-05-23).

Zeigler, Bernard, Herbert Prähofer, and Tag Gon Kim (2000-01). *Theory of Modeling and Simulation 2nd Edition*. San Diego, CA: Academic Press.

Modelica 3.6 - Changes, Benefits and Implementation

Hans Olsson[1]

[1] Dassault Systèmes, Sweden, hans.olsson@3ds.com

Abstract

The latest release of the Modelica Language Specification version 3.6 brings several benefits to users, and this paper will discuss the changes and the benefits for the clearer parameter defaults, clearer start-value priority, selective model extension, and multi-lingual support. The benefits only occur when the features are implemented in Modelica tools, and to facilitate that, the paper will discuss the design choices when implementing the new standard in Dymola 2023x Refresh 1 and 3DEXPERIENCE 2023x FD03.

Keywords: Modelica, model variation, initialization

1 Introduction

The Modelica Language, (Olsson (editor), 2023) and (MAP-Lang 2023), is developed by the Modelica Association Project MAP-Lang on GitHub, using LaTeX and HTML (Miller 2023).

Modelica 3.6 adds a number of new features, corrections, and improvements. New major features are organized as Modelica Change Proposals (MCPs), specifically for this release:

- Undefined modification (MCP-0009).
- Selective Model Extension (MCP-0032).
- Multilingual support of Modelica (MCP-0035).

Their rationales can be found in the directory RationaleMCP in (MAP-Lang 2023). In the specification they can be found in sections 7.2.7, 7.4, and 13.6.

An MCP must be test-implemented in at least one tool before being added to the specification, and the design documents include that experience.

However, fully implementing, documenting, and testing an MCP may reveal new issues (especially when done in other tools than the tools used for the test implementations) and may also find interesting use-cases; and thus the experience in this paper will add additional insights beyond the MCP-design.

This paper will focus on the new major features (MCPs), and some of the minor features and corrections; especially concerning parameters (that go together with the MCP-0009 Undefined modification), start-value priority, and connection restrictions.

Modelica 3.6 was completed on February 28th, the document branch built March 9th, and accepted by Modelica Association March 23rd, 2023.

2 Clear setting of parameters

2.1 Background

MCP-0009 (Undefined modification) goes together with some minor improvements (and corrections) related to parameter values and defaults. The MCP was proposed by ESI (previously ITI) whereas the corrections and minor improvements roughly correspond to what was already implemented in Dymola.

Prior to Modelica 3.6 the specification had the following issues:

- Once you had set a parameter value you could only change it, but not remove the setting completely. Removing setting is useful when:
    - A model parameter has a badly chosen default value and there is no obvious generic correct value (e.g., capacitance).
    - A model parameter has a value, but it is desired to implicitly compute the parameter from an initial equation instead. (Note: in this case it is also necessary to set `fixed=false`.)
- If you declared a parameter without modifying any of its attributes (i.e., no start-value) it was unclear what tools should do; whereas if you did set the start-value it was clear that it could be used with a warning.
- And if you declared a parameter (or variable) and only set the min-attribute to e.g., `2` it was even less clear what tools should do.

The reason for the latter two problems were that the default value for the start-attribute of a Real/Integer was `0`, and it was not clear whether that should be used as default.

A model with all of these issues is given below:

```
model M
  parameter Integer k=1;
  parameter Integer l(start=1)
  parameter Integer m;
  parameter Integer n(min=2);

  Real x[:]=fill(0.0, n-1);
end M;

model M2
  extends M;
…
end M2;
```

Prior to Modelica 3.6 we could not remove the default for `k` when constructing `M2` and it was unclear if `m` should use a default start-value of `0` or not, and clearly using a default start-value of `0` for `n` would be problematic: by violating the min-restriction and generating a negative sized array.

2.2 Changes

MCP-0009 (Undefined modification) solves the first issue by allowing the modifier `=break` to remove the parameter setting. The other issues were solved by clearly specifying a priority for default values and erasing the default values for the start-attributes in the specification and instead introducing the fallback value.

The fallback value is the value closest to "zero" that is consistent with any potential min and max-attribute (for a Boolean it is `false` and for a String it is `""`); adding this restriction ensures internal consistency so that if a variable has a min-attribute of `2` we do not attempt to use a value of `0` (which solves the last issue).

Thus Modelica 3.6 allows us to give a clear precedence for different values for a parameter (unless they have fixed=false) as follows:

1. Value (unless Undefined).
2. Value of the start-attribute (unless Undefined).
3. Fallback value during check.

The default for both the Value and the Value of the start-attribute is now Undefined (which means that the next item in the precedence list is used), and MCP-0009 enables a user to restore them to Undefined by a modification of the form `=break`. A diagnostic is required when the Value is Undefined in a simulation model.

Note that `break` is just for modifiers, and it is not possible to use it to handle other variants of Undefined. Thus it does not correspond to Not-A-Number in IEEE floating point arithmetic, the Maybe-monad in functional languages, or std::optional in C++ (ISO/IEC 2020).

For a non-parameter that is used in an initial non-linear system of equations the starting point for the first iteration is:

1. Value of the start-attribute.
2. Fallback value.

2.3 Implementation aspects

Undefined modification can be fairly trivially implemented in the translator by treating `break` as a normal modification with the special rule that if the resulting modification after merging is `break` the setting of the value is just skipped.

Supporting it in the Graphical User Interface was not complicated, but required care to keep existing options as before *and* clearly specify the new option. If there is a modifier for a parameter `p` like `p=2` the parameter dialog shall support both removing the modifier giving no modification at this level or setting `p=break`. Both could be described in similar ways, which would not be helpful.

The solution was to call the first "Remove modifier" (as earlier) and the second "Set to No Value"; where "No Value" was preferred over the specification word "Undefined" and the syntactic "break". Obviously the user can also write these as modifiers in the parameter dialog.

For backwards compatibility there is an issue, not with Modelica 3.5 but going back to Modelica 2.0 where a component could be named `break` and thus `p=break` could mean that `p` is modified to have the value of the parameter `break`. That was solved when supporting Modelica 2.1 by handling existing components named `break` with a warning (that was possible in Modelica 2.1 to 3.5 since `break` was then only used to break inside loops), and prevent the creation of new components named `break`. That compatibility feature was removed after about 11 years (consistent with the mission plan of MAP-Lang where models should run for at least 10 years), and before Modelica 3.6 was released.

Using the fallback value when checking a model is consistent with the intended use – zero, or close to zero, and consistent with min- and max-attributes. In practice it is almost always the min-attribute that introduces the restriction, since if a variable cannot have both signs the preference is to have only non-negative values (e.g., temperatures, array sizes) and even if further restricted the min-attribute is the allowed value closest to zero.

3 Clearer precedence for start-values

3.1 Background

Modelica is equation based and in order to efficiently handle the resulting systems of equations tools have to make a number of choices for start-values in systems of equations and during initialization – this indirectly also influences which variables are torn out. For a general introduction to tearing see (Elmqvist & Otter 1994).

Consider Modelica.Fluid.Examples.HeatingSystem:

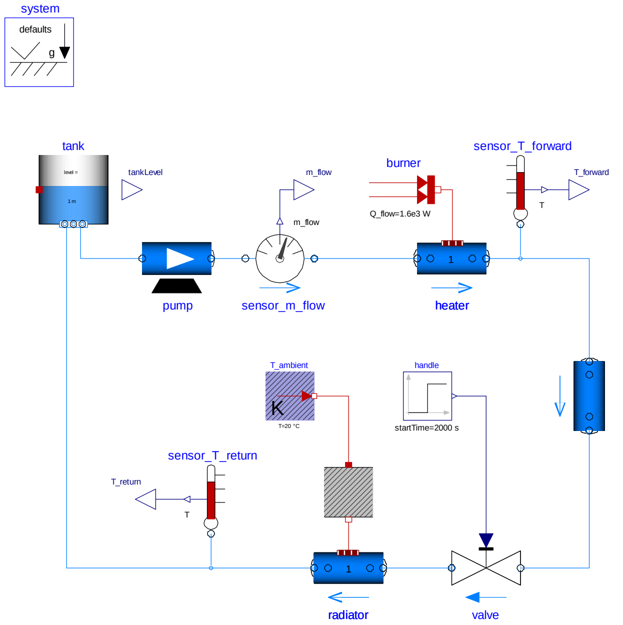


Figure 1 HeatingSystem

There are 4 pressure states and 5 temperature states in this model, and for each of them there are between 7 and 13 different variables that are equal to the state, but have their own start-value. Setting all of them to the same value is tedious and error-prone, but partially done in this model:

```
Pipes.DynamicPipe radiator(
    …
    p_a_start=110000,
    state_a(p(start=110000)),
    state_b(p(start=110000)))
```

These choices matter as different choices can lead to different results or even no results, when reusing a Modelica model in a different tool, different version of the same tool, or even as a sub-model in a different model.

Using the average of the values does not work (e.g., for this model the default temperature in the components was 15°C and the goal was to change the hot part to 80°C – not to have an average of them), and having an explicit priority system was deemed too complicated so instead the idea is to prioritize the existing start-values in some logical way.

The goal with that priority is both to make the choice more predictable (so that the same model gives the same choice) and controllable for users (to get the start-values they want).

3.2 Start-value priority

Modelica 3.3 introduced a priority between start-values with the clear goal that values set "later" (closer to the root of the instance hierarchy) should have precedence. This was based on existing heuristics in Dymola (Dassault Systemes 2023 section 5.8.3; Casella 2011). The rationale with preferring a "later" value is that if there is a problem the user should introduce a new start-value when using the component and that will naturally be seen as "later".

However, it was found wanting in some cases – in particular start-values are often bound to parameters that are then propagated, e.g., in the model above the start-value for the temperature of the heater is the parameter `T_start`. The priority for such cases was based on where the parameter was introduced (and propagated to the start-value) – ignoring whether the parameter was modified later on. In practice that often meant that multiple values had the same priority.

Modelica 3.6 extended the priority to consider where the parameter is set (in this case `T_start`) to break ties in such cases. By only using it as a tie-breaker it adds more detailed priorities without modifying the priority of existing unambiguous cases, reducing the risk of breaking backwards compatibility.

3.3 Implementation aspects

Implementing a more detailed priority for guess-values was fairly straightforward (there were some existing special cases that had to be removed as well). And even if designed with backwards compatibility in mind it is also possible to disable the new feature in Dymola. (Having a standardized way of disabling the new feature was not considered. It would add unnecessary complexity as it is always possible to resolve issues in specific models by adding new start-values with higher priority.)

However, even if the priority is predictable and controllable for users an additional requirement is that the choice is explainable. Thus the logging of start value priority was improved to provide those priorities and all considered start-values. In this model enabling logging gives:

```
The iteration variable heater.mediums[1].T has been
selected to have the guess value 353.15.
    •  353.15, the start value of heater.mediums[1].T
       given as heater.T_start. At level 1. Original start-
       value at level 2.
    •  288.15, the start value of
       heater.flowModel.states[2].T given as 288.15.
    •  …
```

The place where T_start was modified gives the level, whereas the original start-value level is where heater.mediums[1].T.start was modified. Levels are counted up from the current model and thus a lower level has precedence.

4 Selective Model Extension

MCP-0032 introduces selective model extension as a way of selectively deciding what to inherit from another model, and uses the same keyword `break` as MCP-0009 with similar considerations.

4.1 Goal of Selective Model Extension

The goal of selective model extension was originally to enable unforeseen structural variation by giving the possibility to exclude components and connections when inheriting, and doing it in a traceable and well-defined way, (Bürger 2019).

It has also been found useful when the structural variations *could be foreseen*, but supporting all possible foreseen structural variations would create a too complicated model.

Automatically generated models e.g., for Mechanical systems (Elmqvist et al 2009) and Fluid, is thus an additional use-case where a model can be reproducibly generated (so that the physical models always have the correct parameters), and then some connections selectively deselected and control components added. As an example if the previously shown HeatingSystem had been automatically generated from a physical model all of the sensors and actuator had likely been missing. Adding the flow-sensor would be a typical case where it is necessary to deselect a connection to insert a new component (and two new connections).

4.2 Implementation aspects

4.2.1 Deselecting connections

The original test-implementation of MCP-0032 handled deselections in the translator and when showing classes in the Graphical User Interface, but the deselections were written textually (despite the idea naturally being described in terms of changes of the diagram).

What was missing was the User Experience of actually deselecting graphically. This was done by adding "Deselect" option to the context menu of inherited elements (currently with a warning), similarly as the "Delete" option.

That revealed an unforeseen case – a user might first deselect a connection and then later deselect one of its endpoint components (or attempt to deselect both the connection and the component at the same time). That is an error according to the specification (since it does not make sense to write that textually), and can be avoided by adding an extra step removing redundant deselections of connections after any change of the deselections.

4.2.2 Automatic connector sizing

Applying deselections to Fluid models revealed that it interacted with connector sizing in unforeseen ways.

Fluid models have arrays of connectors with automatic sizing (introduced in Modelica 3.1) which ensures that different connections to the array are treated as multiple independent connectors, and the array is adapted in size. Treating them as independent connectors allows correct mixing for stream-connectors (Franke et al 2009); and is normally not necessary in other domains.

Note it is possible to use automatic sizing for other domains – one well-known use is multi-input logical And/Or-blocks; another use is to add parameter-attributes to each array element for a physical array of connectors; similar considerations apply in those cases.

Before Modelica 3.6 automatic sizing always created a dense array of connectors, and if you removed the connection to an element in the middle of the array the array was shrunk and elements re-numbered. Note that the component with automatic sizing connector can be inherited and local connections added – but the inherited connections always precede them.

However, when deselecting it is possible to remove an inherited connection to an element in the middle of an array of connections – and it is not straightforward to re-number the remaining inherited connections. More importantly it is usually not *desirable* to re-number them, as the intended use of Selective Model Extension is usually to re-introduce another connection to the same connector element (after adding/removing some component in the path of the fluid).

As an example look at the reusable HeatingSystem model, and consider deselecting the connection between radiator and tank to add an additional radiator.

The new model uses deselections:

```
model HeatingSystem2
  extends HeatingSystem(break
     connect(radiator.port_b,
             tank.ports[1]));
  Modelica.Fluid.Pipes.DynamicPipe
                      radiator1(…);
   …
equation
  connect(radiator1.port_b, tank.ports[1]);
  …
end HeatingSystem2;
```

In the diagram we can see an additional radiator (with sensor and wall-components), and that the new connection to the tank replaces the existing one.

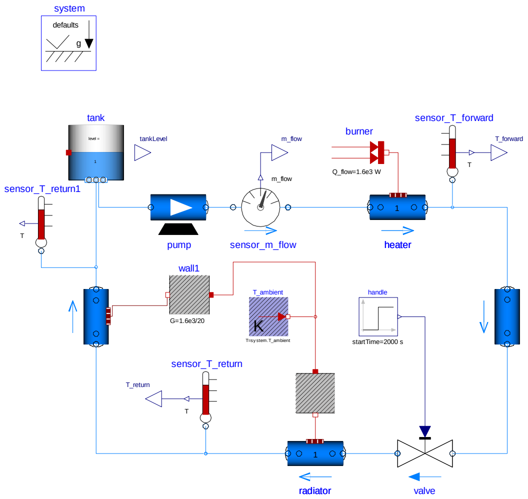


Figure 2 HeatingSystem with extra radiator.

Instead of renumbering the connections this is accomplished by modifying the User Experience of graphically connecting to a connector with automatic sizing to check if there are any holes due to deselected connections (or components) and suggest connecting to the missing element(s) instead of automatically adding it to the end and resizing the connector.

4.2.3 Possible extensions

The current selective model extension works for graphical objects: connections and components.

Deselecting non-connect equations is not possible as equations are not named (a potential extension) and deselecting a component used in such equations does not work either. This is not entirely trivial as one problem with even deselecting connect-equations is that the deselection are *by design* sensitive to the exact syntax used in the equations; and for non-connect equations this problem gets worse. However, when models are structured with large textual equations it may be useful. Automatically deselecting equations would also be useful for removing e.g., initial equations for de-selected components – but it may be possible to find another solution for that.

Additionally even for graphical objects there are some possible extensions. When filtering a signal it is currently necessary to de-select the connection, add the filter, and then reconnect it on both sides. Similarly when using selective model extension to replace a non-replace component it is necessary to first deselect the component and then add a new component and connect it. Simplifying that would be possible (a tool might possibly add this without modifying the language). On the other hand making such operations too easy might risk errors – and could lead to under-use of replaceable component, and relying on replacing them in this way.

4.2.4 Implementation variants

The flattening in Modelica is a hierarchical tree traversal where modifications are propagated downwards, and the resulting tree is then transformed into a hybrid DAE that is simulated.

The deselection is in the MCP seen as deselecting (or pruning) sub-trees after they have been built. That ensures that the deselection actually deselects something and that the pre-deselection elements are correct; and can also be implemented in a straightforward way in the translator.

Propagating deselections downwards similarly as modifications and preventing them from being built does not easily allow similar checks, but was implemented for the Graphical User Interface.

The benefit is that it allows the components to efficiently directly draw their graphics, instead of generating an intermediate graphical representation that is later pruned, and it also allows treating deselected components uniformly with normal components. A uniform treatment of all components in the component browser allows a toggle for deselecting components – to both show the current status and revert deselections. Something similar may be implemented for connections in the future.

5 Connection Restrictions

Causal connectors (with input and/or output) have restrictions to ensure that any input must be given a value - they normally imply that if there is an input component in the connector it must be connected exactly once from the outside (Olsson et al 2008); before Modelica 3.0 this rule only applied to entire connectors declared as input.

Modelica 3.3 added the restriction that conditional physical connectors (i.e., connectors with at least one flow variable) must be connected if enabled. The idea was that if you set a Boolean parameter to enable that connector it would not make sense to leave it unconnected, and the default semantics for unconnected connectors (zero flow) do not always make sense.

The intended case was the optional support connectors in the rotational library where many components have an optional support connector. The default (top part of diagram) is that it is disabled and instead there is an implicit connection to ground (giving zero position instead of zero flow) – but if enabled (bottom part) there is a new connector for the support of the component. The connector is marked with a red circle and the connections are red and dashed.

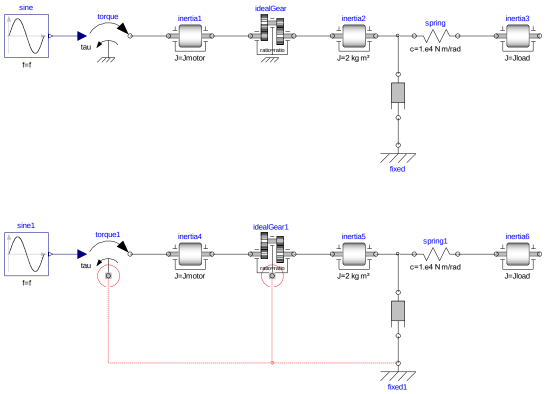


Figure 3 Driveline with implicit grounding (top) and with explicit grounding (bottom).

If the dashed connection to the ideal-gear is missing the model would simulate, but generate incorrect results (if the torque-generator connection to ground is missing the model is singular). The intent of the restriction was to catch such cases early and generate good diagnostics.

However, when checking the Modelica Standard Library according to those semantics it was revealed that the situation was more complicated – in particular several Electrical Machine models had one Boolean parameter controlling *multiple* components including a connector, and in those cases leaving the connector unconnected was used and normal.

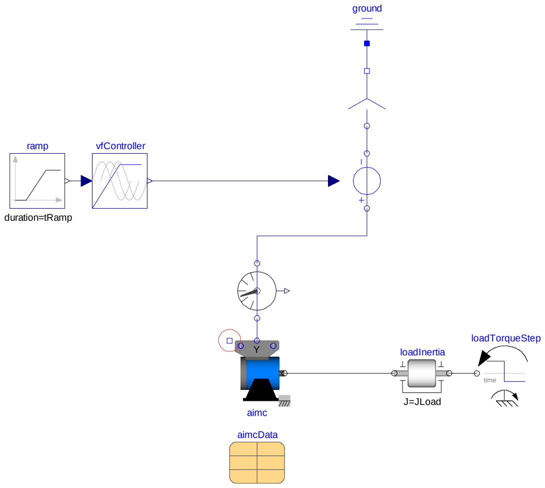


Figure 4 Machine model with unconnected conditional connector.

The red circle marks the single phase (“star”) connector of the terminalBox, it is only available in the Delta-configuration, but as shown here it is not always connected. On the other hand it was known that several models with unconditional connectors had assertions to ensure that they were connected (from the outside). Note, there are also some causal connectors with redundant assertions to check that the connectors are connected, those assertions can just be removed – and predate the improvements in Modelica 3.0.

Thus MAP-Lang in Modelica 3.6 decided to replace the connection restriction based on whether the connection was conditional or not by an annotation indicating whether it must be connected, `mustBeConnected`, (and additionally one saying that it may only be connected once, `mayOnlyConnectOnce`). Both of them are given as a string indicating the reason – and for a conditional connector the restrictions are only active when it is conditionally active.

In this case the optional support flange could have:

```
Support support(
      phi=phi_support,
      tau=-flange.tau) if useSupport
    "Support/housing of component"
annotation (
  mustBeConnected="If the optional support
flange is enabled it must be connected",
Placement(transformation(extent=
  {{-10,-110},{10,-90}})));
```

This ensures that the previous correct examples work, and if the connect was missing a specific error message is given, e.g. in Dymola

The connector torque1.support was not connected from the outside, and it must be connected since: "If the optional support flange is enabled it must be connected"

The `mayOnlyConnectOnce` can be used in combination with automatic sizing to ensure that there is only one connection to each array element.

5.1 Implementation details

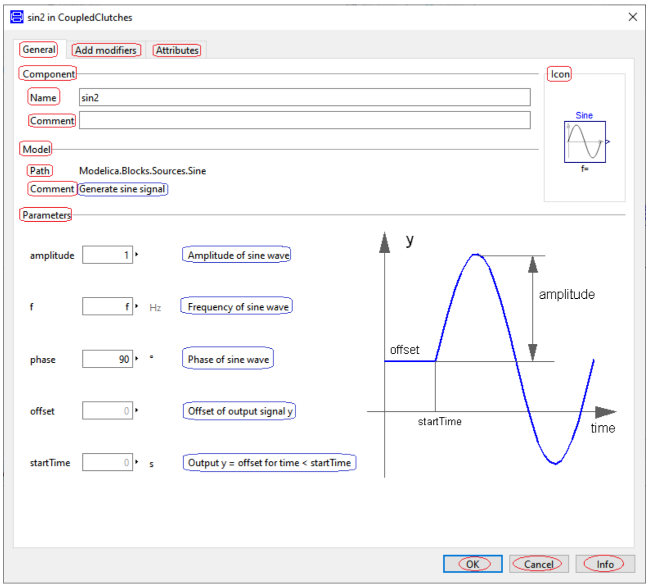
The connection restriction in Modelica 3.3 was not originally implemented in Dymola, and shows that even seemingly obvious improvements should be fully test-implemented before being added to the specification.

Note that one could think of multiple possible interpretations of “connected”: an active connect-statement involving that connector (used for `mustBeConnected`), or that its elements are part of a connection-set with additional elements (used for `mayOnlyConnectOnce` with specific restrictions for streams-connectors). The latter ensures that redundant connections are ignored, and the special rules for streams-connectors imply that sensor components are ignored, and thus one can, e.g., add a temperature-sensor without violating the restriction.

6 Multilingual support of Modelica

The documentation of the Modelica Standard Library is only written in English, whereas many tools support translation of their User Experience to different natural languages – in order to ease the use for non-English users.

Modelica 3.6 allows Modelica libraries (including the Modelica Standard Library) to provide translations without modifying the actual Modelica source code of the library; this was proposed and (test-)implemented by ESI. However, actually providing a localized User Experience requires both that the tool support using the translation and that the translation exists for the specific library – thus getting the benefit of this addition may take longer.


sin2 in CoupledClutches

Component
Name: sin2
Comment:

Model
Path: Modelica.Blocks.Sources.Sine
Comment: Generate sine signal

Parameters
amplitude: 1 — Amplitude of sine wave
f: f Hz — Frequency of sine wave
phase: 90 ° — Phase of sine wave
offset: 0 — Offset of output signal y
startTime: 0 s — Output y = offset for time < startTime

Figure 5 Dymola parameter dialog for Sine-block. Multilingual support means translating the texts in blue ovals. Texts in red ovals are already translated as part of tool settings (currently only for Japanese).

6.1 Implementation feedback

The multilingual support in Dymola 2023x Refresh 1 is only partial, but it revealed two important issues to consider.

The first is that a library maintainer for e.g., the Modelica Standard Library should update the English texts even if they normally work in another natural language. The simplest way to handle that is make it easy to disable the translations to keep the possibility of directly modifying the description and documentation in the library.

The second is that the descriptions often directly and indirectly reference the name of components, which makes translation more complicated. Note that the names of parameters and classes are deliberately not translated (and the Modelica language itself also uses English keywords).

Consider a parameter named “startTime”, with a description “Output y = offset for time < startTime”. This indicates two problems – first “offset” and “y” are other component names and should likely not be translated, and second that the description does not say that it is a “start time” (or “Time Sine Wave Starts”) since it is implied by its name. It is obviously possible to handle during the translation – but it means that it is not just a matter of merely directly translating the existing descriptions.

Additionally the existing documentation often also uses images for showing how the model works, including the names of parameters. Thus the user should at least be able to recognize the English parameter names.

The partial support in Dymola is intended to give library authors the possibility to start providing the translation, and thus it is possible to generate the entire translation template – and use the translation of a few key impacting items like class and component descriptions.

7 Conclusions

This paper demonstrates that Modelica 3.6 has new powerful improvements. Dymola 2023x Refresh 1 and 3DEXPERIENCE 2023x FD03 supports these features (clearer parameter defaults, clearer start-value priority, connection restrictions, selective model extension, and multi-lingual support), and other tools have also released or are working on support for these features – and the goal of this paper is to improve portability by helping other implementers support these features. In particular, the selective model extensions considerations for deselecting connections and automatic connector sizing; and user experience for undefined modification and start-value precedence.

Acknowledgements

The work on Modelica 3.6 has been carried out by MAP-Lang. The author (who is also the chair of the group) acknowledges the hard work of the members of the group and the interesting discussions.

References

Bürger, Christoff (2019): Modelica language extensions for practical non-monotonic modelling: on the need for selective model extension. In: *Proceedings of 13th International Modelica Conference* 277-288 http://dx.doi.org/10.3384/ecp1915727

Casella, Francesco (2011): Selection of missing initial equations and of start attributes for alias variables URL: https://github.com/modelica/ModelicaSpecification/issues/561

Dassault Systèmes. (2023) Dymola 2023x Refresh 1: *Dymola, Dynamic Modeling Laboratory, User Manual.* Dassault Systèmes AB, Lund, Sweden.

Elmqvist, Hilding, and Martin Otter. 1994. Methods for Tearing Systems of Equations in Object-Oriented Modeling. Proceedings ESM'94, European Simulation Multiconference, Barcelona, Spain, June 1-3, pp.326--332.

Elmqvist, Hilding, Sven Erik Mattsson, and Christophe Chapuis (2009): Redundancies in Multibody Systems and Automatic Coupling of CATIA and Modelica. In: *Proceedings of 7th International Modelica Conference* 551-560 http://dx.doi.org/10.3384/ecp09430113

Franke, Rüdiger, Francesco Casella, Martin Otter, Michael Sielemann, Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson (2009): Stream Connectors – "An Extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena". In: *Proceedings of 7th International Modelica Conference* 108-121 http://dx.doi.org/10.3384/ecp09430078

ISO/IEC. (2020). ISO International Standard ISO/IEC 14882:2020(E) – Programming Language C++. Geneva, Switzerland: International Organization for Standardization (ISO). URL: https://isocpp.org/std/the-standard

MAP-Lang (2023): Modelica Language Specification. URL: https://github.com/modelica/ModelicaSpecification/

Miller, Bruce (2023): A LATEX to XML/HTML/MathML Converter URL: https://math.nist.gov/~BMiller/LaTeXML/

Olsson Hans, Martin Otter, Sven Erik Mattsson, and Hilding Elmqvist (2008): Balanced Models in Modelica 3.0 for Increased Model Quality. In: *Proceedings of 6th International Modelica Conference* 21-33 https://modelica.org/events/modelica2008/Proceedings/sessions/session1a3.pdf

Olsson, Hans (editor) (2023): Modelica - A Unified Object-Oriented Language for Systems Modeling Language Specification Version 3.6. URL: https://specification.modelica.org/maint/3.6/MLS.pdf

Modelica Models in SSP

Dag Brück

Dassault Systèmes, Lund, Sweden, dag.brueck@3ds.com

Abstract

This is a proposed optional extension for SSP 2.0 that defines how Modelica models can be referenced in SSP. It specifies the mapping of key Modelica concepts to SSP, which necessitates a few small extensions. The purpose is to broaden the scope of SSP to embrace the more powerful modeling concepts of Modelica, for environments that can support it.

Keywords: Modelica, SSP, extension, standardization

1 Introduction

Using System Structure and Parameterization (SSP) (Mai et al. 2019) as the high-level architecture description combined with parameter sets, there is a natural desire to be able to reference not only FMUs but also Modelica component models in SSP. Modelica (Elmqvist 2014; MLS36 2023) offers strong modeling capabilities and usually yields a more efficient and accurate simulation of complex systems compared to component-based modeling with FMUs.

Given the example in Figure 1, We see two control-oriented blocks defined as FMUs, and then a large and more complex physical model with tightly-coupled elements that, experience has proven, are not well represented by interconnected FMUs. In this example, the hybrid driveline uses pre-defined electrical, mechanical and hydraulic components from Dymola's VeSyMA library. Templates with replaceable components offers great flexibility for exploring design alternatives. Because of Modelica's inherent properties with equations, acausal connections, etc., efficient simulation code can be generated without sacrificing modular flexibility.

However, SSP has strong capabilities for combining the system topology with meta-data (Heinkel et al. 2022), multiple parameter sets, name mapping and basic simulation setup (Hällqvist et al. 2021). In total, it yields an attractive high-level package for a credible simulation experiment (Heinkel and Steinkirchner 2022).

SSP is attractive for packaging interconnected simulation modules with parameter data into a single unit that is suitable for simulation. For many tools, connection of causal FMUs are sufficient, but as the example shows, structures that are more complex can be efficiently manipulated and simulated if SSP is extended to the Modelica domain. This proposal adds those capabilities, without adding undue complexity to tools that chose to forego the benefits of Modelica. The purpose of this proposal is to provide a minimal solution for mapping Modelica models into an SSP context, without any attempt to cover advanced Modelica concepts.

The scope of this document is to define how components and connectors can be specified as Modelica models, and the mapping of Modelica modifiers to an SSP syntax. Further constraints are presented in the Discussion.

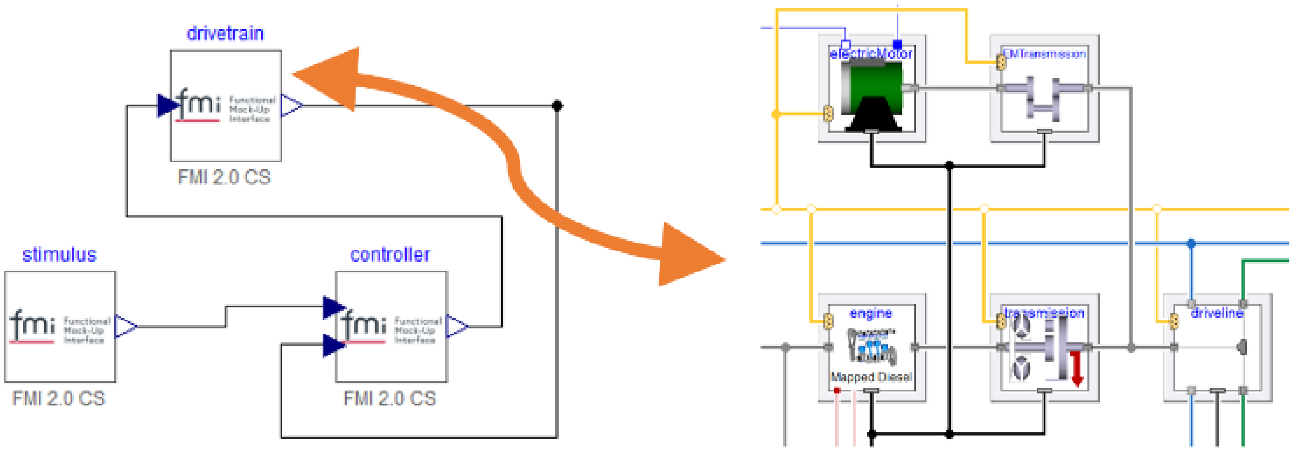


Figure 1. SSP system description using a complex component with Modelica representation.

2 Identification of the new SSP format

Within the constraints of the existing version of the SSP specification, it is possible to make extensions using a layered standard. Such a layered standard is limited to adding annotations and new media (MIME) types.

Because this proposal defines additional attributes, for reasons discussed below, a new version number is required. In the intermediate period until a future standardization, the SSP design group has decided that SSDs (System Structure Descriptions) using this extension shall use the version number *2.0-alpha.1*.

3 Representation of components

For SSP to support Modelica component models, the proposed encoding is:

- The component source attribute contains the path of the Modelica model. The source URI *modelica:* designates a model in the namespace of the Modelica environment.
- A media type (formerly known as a MIME type) is a two-part identifier for file formats and format contents. The media type used for Modelica is `"text/x-modelica"`.

Possibly an optional version number of the Modelica library should be added, to support on-demand loading.

An example of such a Modelica component element is shown in Listing 1.

4 Representation of connectors

Modelica connectors for built-in input and output types are easily mapped to SSP connectors, see Table 1. Doing this mapping facilitates connections with FMUs and nested SSDs to Modelica components with fundamental connector types.

Modelica Type	SSP Type	SSP Kind
RealInput	ssc:Real	input
RealOutput	ssc:Real	output
IntegerInput	ssc:Integer	input
IntegerOutput	ssc:Integer	output
BooleanInput	ssc:Boolean	input
BooleanOutput	ssc:Boolean	output
StringInput	ssc:String	input
StringOutput	ssc:String	output

Table 1. Mapping of standard connector types in Modelica.Blocks.Interfaces to SSP.

In SSP 2.0 we expect a richer set of types because of the adaption to FMI 3. This offers the opportunity to represent additional Modelica types, for example arrays and perhaps even hierarchically structured types.

Modelica connectors of more advanced types are mapped the same way as Modelica component models.

- The connector type is `ssc:Binary`.
- The connector source attribute contains the full path of the Modelica model.
- The media type is `"text/x-modelica"`.
- Acausal Modelica connector types are in SSP of `kind="acausal"`. This is a generalization compared to SSP 1.0.

For all connector types, the following extension is made:

- ConnectorGeometry is amended: An optional attribute `rotation` of type `xs:double` may be specified. It should be noted that this extension is of general interest to SSP, regardless of Modelica support.

Two such Modelica connectors are shown in Listing 2.

5 Representation of modifiers

Modelica modifiers are mapped to SSP parameter sets as follows:

- Literal modifiers of Modelica types Real, Integer, Boolean and String are mapped to their corresponding types in SSP.
- Other modifier expressions are mapped to `ssv:Enumeration`, with the value attribute containing the Modelica text of the modifier value. This is needed to handle expressions, which are common in Modelica models.

An example of a parameter binding is shown in Listing 3. A more complete example is shown in Listing 4.

6 Discussion

After presenting the actual proposal for extension in Sections 3-5, some of the design choices can be discussed.

6.1 The scope of the proposal

If we intend to work actively with Modelica models in the SSP context (not only display the architecture), we can assume that a full Modelica environment (Brück 2023), and hence any needed Modelica library, is available. For that reason, the SSD only needs to store references to the Modelica models.

Storing the model text of a component model and all dependent models would be a major effort (corresponding to “Save Total” in Dymola), but doable. If needed, a tool could for example store the Modelica text as an SSP annotation. It should be noted that this approach would fail for many libraries that are deployed in encrypted form, or use a license mechanism.

```
<ssd:Elements>
  <ssd:Component name="sin"
        type="text/x-modelica" source=
  "modelica://Modelica.Blocks.Sources.Sine">
    <ssd:Connectors>
      ...
    </ssd:Connectors>
    <ssd:ElementGeometry x2="-110" x1="-130"
        y1="-10" y2="10"/>
  </ssd:Component>
</ssd:Elements>
```

Listing 1. Representation of a Modelica component.

```
<ssd:Connectors>
  <ssd:Connector name="y" kind="output"
    description="Connector of Real output
signal">
    <ssc:Real/>
    <ssd:ConnectorGeometry x="0.5" y="0.0"
      rotation="90" />
  </ssd:Connector>
  <ssd:Connector name="flange_a"
    kind="acausal"
    description="Flange of left shaft">
    <ssc:Binary mime-type="text/x-modelica"

source="modelica://Modelica.Mechanics.Rotati
onal.Interfaces.Flange_b"/>
    <ssd:ConnectorGeometry x="0" y="0.5"/>
  </ssd:Connector>
</ssd:Connectors>
```

Listing 2. Representation of Modelica connectors.

```
<ssd:ParameterBindings>
  <ssd:ParameterBinding
     type="application/x-ssp-parameter-set">
    <ssd:ParameterValues>
      <ssv:ParameterSet
                   name="DefaultParameters"
                   version="1.0">
        <ssv:Parameters>
          <ssv:Parameter
             name="peak">
             <ssv:Real value="1.1"/>
          </ssv:Parameter>
          <ssv:Parameter
             name="startTime">
             <ssv:Enumeration value="2*T2"/>
          </ssv:Parameter>
        </ssv:Parameters>
      </ssv:ParameterSet>
    </ssd:ParameterValues>
  </ssd:ParameterBinding>
</ssd:ParameterBindings>
```

Listing 3. Example of a parameter biding.

Advanced Modelica concepts such as inheritance, replaceable components/models, re-declarations and expandable connectors are intentionally left out because there is no natural mapping to SSP concepts.

6.2 Tools without Modelica capabilities

We can expect that most tools supporting SSP will not have capabilities to simulate Modelica models. Such tools can partially support SSP files that use Modelica components with little additional effort. The information to display components and their connectors, as well as connections, is identical. The tool must ignore what it cannot process, such as any component source of type "text/x-modelica". Simple editing operations are possible, assuming that the new attributes described in this document are ignored but maintained.

Note that in either case, SSPs that do not include Modelica components are completely unchanged compared to SSP 1.0. In that sense, this is an unobtrusive extension.

6.3 Enumeration expressions

The proposal to handle parameter expressions as enumerations can be questioned. Reusing the concept of enumeration values can be perceived as a creative abuse of the rules, but appears to fall within the constraints of SSP. A cleaner solution would be to introduce a new kind of value for this case, but that adds another incompatibility with SSP 1.0. A further generalization would be to use Modelica's full modifier syntax, which would allow e.g. redeclaration.

It has been proposed to generalize the allowed string for e.g. Real values to include an arbitrary expression. This is not a good idea because it defeats the possibility to syntax-check purely numeric parameters sets.

6.4 Change proposal or layered standard

When developing a proposed extension of SSP, one is faced with the choice of two approaches.

The first is to make an extension that is as close as possible to SSP 1.0 with minimum disruption for existing tools and users. SSP is designed to manage this, using the concept of Layered Standard that uses a general extension mechanism in the form of annotations. Using such annotations, it is possible to make a layered standard that can (in limited form) be processed by conforming tools that know nothing at all about new features.

The second is to make an extension proposal designed to cleanly extend SSP 1.0 into SSP 2.0 with new concepts that naturally fit into SSP. In this case, we need to add attributes that are not present in SSP 1.0 instead of using annotations. Examples are the proposed rotation attribute and the notion of acausal connectors. The drawback is that tools that strictly conform to SSP 1.0 will not be able to read the new format, which for that reason should be identified with a unique version number.

After due consideration we have respectfully opted for the second approach, a non-layered extension. The key reason is the future growth path with a potentially wide application. If this feature will be a permanent part of SSP 2.0, we want it to be “natively” integrated and not be implemented with annotations. If we started with a layered standard based on SSP 1.0, the native representation in SSP 2.0 would require yet another migration effort. Some aspects of the proposal, such as the rotation attribute for connectors, are not inherently tied to Modelica.

Acknowledgements

The author wants to thank the members of the SSP design group, in particular Pierre Mai, Robert Hällqvist and Peter Lobner, for constructive feedback on the proposal.

References

Brück, Dag (2023). "SSP in a Modelica Environment" in Proc. 15th International Modelica Conference, Aachen, Germany.

Elmqvist, Hilding (2014). "Modelica Evolution - From My Perspective" in Proc. 10th Modelica Conference, Lund, Sweden.

Heinkel, Hans-Martin, P. R. Mai, R. Aue, J. Bou, C. Bühler, C. Franke and A. Puetz (2022). "SRMD format and classifications for metadata". https://gitlab.setlevel.de/open/processes_and_traceability/traceability_data/-/blob/main/SETLevel_SRMD_and_classifications_for_metadata.pdf.

Heinkel, Hans-Martin and K. Steinkirchner (2022). "Credible Simulation Process". https://gitlab.setlevel.de/open/processes_and_traceability/credible_simulation_process_framework/-/blob/main/Credible-Simulation-Process-v1-2.pdf.

Hällqvist, Robert, R. C. Munjulury, R. Braun, M. Eek and P. Krus (2021). "Engineering Domain Interoperability Using the System Structure and Parameterization (SSP) Standard" in Proc. of the 14th International Modelica Conference, Linköping, Sweden.

Mai, Pierre R. et al. (2019). "System Structure and Parameterization". https://ssp-standard.org/publications/SSP10/SystemStructureAndParameterization10.pdf.

MLS36 (2023). “Modelica – A Unified Object-Oriented Language for Systems Modeling, Language Specification Version 3.6”.

```
<?xml version="1.0" encoding="UTF-8"?>
<ssd:SystemStructureDescription fileversion="4.0.0" generationDateAndTime="2023-08-15T14:16:21Z"
     generationTool="Dymola by Dassault Systemes" name="CoupledClutches" version="2.0-alpha.1" >
  <ssd:System description="Drive train with 3 coupled clutches" name="CoupledClutches">
    <ssd:Connectors>
      <ssd:Connector description="Frequency to invoke clutch1" kind="parameter" name="f">
        <ssc:Binary mime-type="text/x-modelica"
   source="modelica://Modelica.Units.SI.Frequency"/>
      </ssd:Connector>
      ...
    </ssd:Connectors>
    <ssd:Elements>
      <ssd:Component name="sin2" source="modelica://Modelica.Blocks.Sources.Sine" type="text/x-
   modelica">
        <ssd:Connectors>
          <ssd:Connector description="..." kind="output" name="y">
            <ssc:Real/> <ssd:ConnectorGeometry x="1.05" y="0.5"/>
          </ssd:Connector>
        </ssd:Connectors>
        <ssd:ElementGeometry rotation="270" x1="-40" x2="-20" y1="30" y2="50"/>
        <ssd:ParameterBindings>
          <ssd:ParameterBinding type="application/x-ssp-parameter-set">
            <ssd:ParameterValues>
              <ssv:ParameterSet name="DefaultParameters" version="1.0">
                <ssv:Parameters>
                  <ssv:Parameter name="amplitude"> <ssv:Real value="1"/> </ssv:Parameter>
                  <ssv:Parameter name="f"> <ssv:Enumeration value="f"/> </ssv:Parameter>
                  <ssv:Parameter name="phase"> <ssv:Real value="1.570796326794897"/>
                  </ssv:Parameter>
                </ssv:Parameters>
              </ssv:ParameterSet>
            </ssd:ParameterValues>
          </ssd:ParameterBinding>
        </ssd:ParameterBindings>
      </ssd:Component>
      ...
    </ssd:Elements>
    <ssd:Connections>
      <ssd:Connection endConnector="flange_a" endElement="J1"
          startConnector="flange" startElement="torque"> </ssd:Connection>
      ...
    </ssd:Connections>
    <ssd:ParameterBindings>
      <ssd:ParameterBinding type="application/x-ssp-parameter-set">
        <ssd:ParameterValues>
          <ssv:ParameterSet name="DefaultParameters" version="1.0">
            <ssv:Parameters>
              <ssv:Parameter name="f"> <ssv:Real value="0.2"/> </ssv:Parameter>
              <ssv:Parameter name="T2"> <ssv:Real value="0.4"/> </ssv:Parameter>
              <ssv:Parameter name="T3"> <ssv:Real value="0.9"/> </ssv:Parameter>
            </ssv:Parameters>
          </ssv:ParameterSet>
        </ssd:ParameterValues>
      </ssd:ParameterBinding>
    </ssd:ParameterBindings>
    <ssd:SystemGeometry x1="-140" x2="140" y1="-100" y2="100"/>
    <ssd:Annotations>
      <ssc:Annotation type="com.3ds.ssp">
        <smmd:ModelicaMetaData>
          <UserAnnotation key="description" value="Drive train with 3 coupled clutches"/>
          <UserAnnotation key="name" value="CoupledClutches"/>
          <UserAnnotation key="version" value="4.0.0"/>
          <UserAnnotation key="versionDate" value="2020-06-04"/>
          <UserAnnotation key="revisionId" value="6626538a2 2020-06-04 19:56:34 +0200"/>
        </smmd:ModelicaMetaData>
      </ssc:Annotation>
    </ssd:Annotations>
  </ssd:System>
  <ssd:DefaultExperiment startTime="0" stopTime="1.5"/>
</ssd:SystemStructureDescription>
```

Listing 4. CoupledClutches example from MSL with certain parts removed for brevity.

Beyond FMI - Towards New Applications with Layered Standards

Christian Bertsch[1] Matthias Blesken[2] Torsten Blochwitz[3] Andreas Junghanns[4]
Pierre R. Mai[5] Benedikt Menne[2] Kevin Reim[2] Markus Süvern[2] Klaus Schuch[6]
Torsten Sommer[7] Patrick Täuber[2]

[1]Robert Bosch GmbH, Germany, Christian.Bertsch@de.bosch.com
[2]dSPACE GmbH, Germany, {PTaeuber, MBlesken, MSuevern, BMenne, KReim}@dspace.de
[3]ESI ITI, Germany, Torsten.Blochwitz@esi-group.com
[4]Synopsys, Germany, Andreas.Junghanns@synopsys.com
[5]PMSF IT Consulting, Germany, pmai@pmsf.eu
[6]AVL List GmbH, Austria, klaus.schuch@avl.com
[7]Dassault Systems, Germany torsten.sommer@3ds.com

Abstract

The FMI standard — just like any other standard — faces the challenge of balancing generality with enabling specific use cases. Including every domain or use-case specific extension in the core standard would significantly increase its length, making it unreadable and unimplementable. To allow for extensions of the core standard for specific use cases, the Modelica Association developed the concept of layered standards, first in the SSP standard and later in FMI.

This paper presents the concept of layered standards and describes the layered standards currently under development by the FMI Project: XCP support of FMUs, network communication, and structured variables and n-D lookup tables in FMI 3.0.

Keywords: FMI, layered standard, XCP, network communication, regular maps

1 Introduction

1.1 Motivation for Extension Mechanisms of Standards

The versions 1.0 and 2.0 of the FMI standard (Blochwitz 2011) (Blochwitz 2012) already contain many optional features, and FMI 3.0 (Junghanns 2021) has increased their number even more. If additional optional features were continually added to address specific use cases and usage domains, the standard would significantly increase in length and become unreadable and unimplementable.

1.2 Requirements

The layered standards approach is based on a hierarchical structure of standards that meet the following requirements:

- The core standard remains generic to ensure broad usage and tool support.

- The extensions (layered standards) depend on the core standard, but not vice versa. This allows for flexible extension not only by the FMI Project but also by other organizations, independent of the release cycle of the FMI Standard.

1.3 Extension Mechanisms for Standards

A layered standard allows specific, new use cases to be handled, without violating the core standard, but rather building on it. It is realized by using extension points contained in the base standard, that were either already intended for extension via layered standards, or can be used, even if not originally so intended.

Some standards are generally intended to be extended by layered standards, providing just frameworks for this extension: For example, URIs defined in RFC-3986[1] (Berners-Lee, Fielding, and Masinter 2005) gain their expressive power through the extension via scheme definitions, like the file or http schemes.

Other standards provide extension via layered standards on top of a core standard that already provides a robust set of functionality. The HTTP standard RFC-2616 (Nielsen et al. 1999) provides the core functionality behind the web, while allowing among others for extension via additional headers, which have been used to provide, e.g., RFC-2965 (Montulli and Kristol 2000) for HTTP State Management Mechanism — Cookies, or RFC-6797 (Hodges, Jackson, and Barth 2012) for HTTP Strict Transport Security. Similarly, the SMTP standard RFC-2821 (Klensin 2001) provides the core functionality behind email, allowing extension via additional options or services, e.g. RFC-3207 (Hoffman 2002) for STARTTLS.

And some standards can be extended via layered standards even though the base standard only contains very limited extension points: FMI 2.0, for example, provided annotations as a user-defined mechanism in the core XML, and did not prohibit additional files to appear in the FMU archive, thus allowing extension, while not necessarily

[1] Here and in following references to IETF standards, not always the newest incarnation of the RFC is cited, but rather the relevant versions from a historical perspective of layered standard development.

DOI
10.3384/ecp204381

having layered standards in mind. The OSI Sensor Model Packaging (OSMP) (ASAM OSI Project 2022) specification is an example of a layered standard that was layered on top of FMI 2.0.

1.4 The Concept of Layered Standards in the Modelica Association

For Modelica Association standards, the concept of layered standards was first introduced for the System Structure and Parameterization (SSP) Standard version 1.0 (MAP SSP 2019)

They can be defined and released

- by third parties, completely independent from the Modelica Association Project (MAP),
- by third parties that are endorsed by the MAP, or
- by the MAP project itself, making them a MAP layered standard.

1.5 The Concept of Layered Standards to the FMI Standard

In FMI, the concept of layered standards was introduced in the version 3.0 (MAP FMI 2022b), and later backported to FMI 2.0.4 (MAP FMI 2022a). The following specific provisions for layered standards were made, see Figure 1:

- The ZIP structure now contains an "extra/" folder where additional files can be placed without disturbing the rest of the FMI mechanisms.
- The XML schema allows extensions to elements to add further information, via Annotation elements.
- For both of these mechanisms the standard provides suggested rules on naming using reverse domain notation to ensure that separately defined extensions do not clash.
- New values for *matchingRule* or *terminalKind* for Terminals can be defined.
- Potentially, layered standards could add new functions to the API of the FMU, however no rules to avoid name clashes have yet been devised as part of FMI 3.0.
- FMI 3.0.1 introduces a recommendation for a standardized fmi-ls-manifest.xml file that allows importing tools to detect which layered standards are supported by the FMU (and which version). This way, the additional capabilities of the FMU can be used more easily and a list of supported layered standards can be displayed to the user.

For FMI 3.0, the standard specifies that an FMU supporting a layered standard on top of FMI 3.0 must at the same time still be a valid FMI 3.0 FMU. This requirement puts certain constraints on a layered standard:

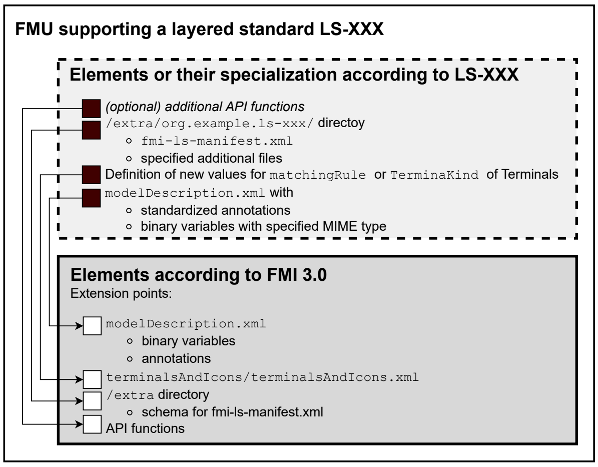

FMU supporting a layered standard LS-XXX
Elements or their specialization according to LS-XXX
(optional) additional API functions
/extra/org.example.ls-xxx/ directoy
fmi-ls-manifest.xml
specified additional files
Definition of new values for matchingRule or TerminaKind of Terminals
modelDescription.xml with
standardized annotations
binary variables with specified MIME type
Elements according to FMI 3.0
Extension points:
modelDescription.xml
binary variables
annotations
terminalsAndIcons/terminalsAndIcons.xml
/extra directory
schema for fmi-ls-manifest.xml
API functions

Figure 1. An FMU supporting a layered standard LS-XXX using the extension mechanisms of FMI 3.0.

- The layered standard can add optional features to the FMU, like additional files inside the FMU's zip file, or it can extend XML files where the base standard schema allows it.
- On the other hand, the layered standard can place additional restrictions on XML elements (e.g. only allow the use of certain Variable types), or mandate an optional FMI 3.0 feature to be required.

In FMI 3.0, some new features that are "orthogonal" to the core FMI functionality (namely *TerminalsAndIcons* and *BuildConfiguration*), were already developed with some features of layered standards in mind, e.g., by having their own XML files. However, as they are of most general interest and considered very important, it was decided to include them in the core FMI standard. This has the benefit of possibly being supported by many tools, but the drawback that updates of these features are limited to the release cycle of the core FMI 3.0 standard.

A layered standard could also extend beyond the core standard, e.g., by introducing additional API functions or side channels, or by removing limitations on the calling sequence of API functions of the FMI state machine of a certain FMI kind. An example would be setting states of a Co-Simulation FMU after initialization, e.g., in order to realize nonlinear Kalman filters.

In the next sections we will give examples of layered standards that are currently in development by the FMI Project and other institutions or companies.

2 Current Development of Layered Standards for FMI by the FMI Project

Currently three layered standards are in development by the FMI Project. They comprise extensions to FMI that are of general interest. Additionally, they demonstrate and standardize how FMI 3.0 and its mechanism can be used for new important application domains. Furthermore, the provision of these layered standards by the FMI Project will promote the concept and validate the extension mechanisms of FMI, potentially leading to further improvements.

2.1 FMI Layered Standard for XCP (LS-XCP)

2.1.1 Motivation

XCP (Universal Measurement and Calibration Protocol (ASAM e.V. 2017)) is a standardized protocol used in the automotive industry to measure variables and adapt control parameters inside an electronic control unit (ECU) through buses like CAN. When wrapping a virtual ECU (vECU) into an FMU, supporting XCP-based measurement and calibration is required for many simulation use cases. Before this standardization effort, already several tools implemented proprietary XCP support for FMUs. As these are incompatible to each other, the need for standardization became apparent.

2.1.2 Approach

The main idea is to ship an A2L file (see (ASAM e.V. 2018) ASAP2) in a standardized location inside the FMU and to describe the capabilities of the FMU w.r.t. the XCP protocol.

The layered standard describes two alternative implementations depending on the use case and data availability (MAP FMI 2023b):

- The FMU implements an XCP slave which provides access to measurement and calibration variables of the vECU and handles the communication protocol with the XCP master in the MCD tool. The necessary information for an MCD tool is given in a description file which follows the ASAM MCD-2 MC standard (aka A2L, also ASAP2) and customarily carries the file extension .a2l. Figure 2 shows a typical design with an XCP service contained in the FMU.

- An external XCP slave implementation accesses the memory of the vECU to expose the XCP protocol to the MCD tool. In this case, the A2L file is still shipped with the FMU but the importer needs to provide the XCP slave implementation. *fmi3IntermediateUpdateCallback* calls or the Clocks mechanism could be used to synchronize DAQ lists. Figure 3 illustrates a typical design utilizing an external XCP service.

The following extension mechanisms to FMI are used:

- The **"extra/" directory** is used to provide additional files: An fmi-ls-manifest.xml provides information about the capabilities of the FMU and an A2L file for each supported platform describes the memory layout.

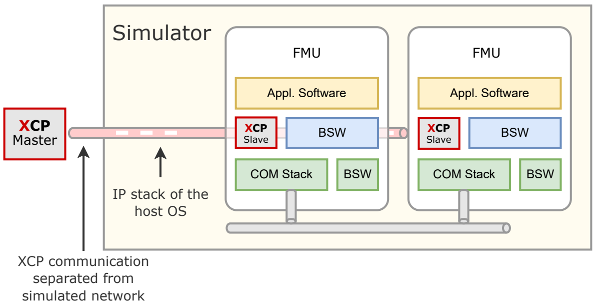


Figure 2. Direct communication of XCP master and XCP slave via the IP stack of the host OS.

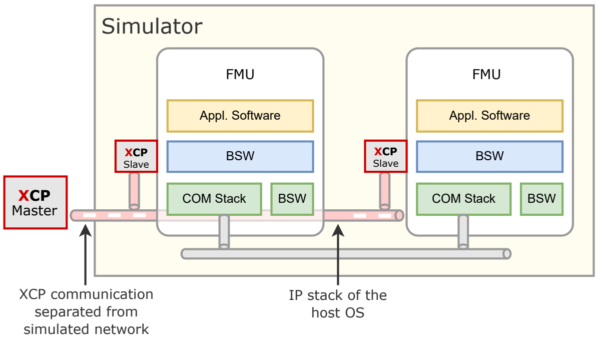


Figure 3. Communication of XCP master and external XCP slave via the IP stack of the host OS.

- A set of **(structural) parameters** for the configuration of the XCP service is introduced, e.g., to change the TCP port the XCP service is listening on. FMUs that provide an XCP service should also provide these parameters that follow a specified naming convention.

- It is specified when the XCP service should be started and stopped in order to have an early access to the XCP calibration parameters.

This layered standard will have no effect on the FMU interface, nor the C-API behavior. While measurement of FMU internal variables does not have a numeric effect on the FMU, so called calibration does. Calibration is the tuning of FMU internal parameters. Such changes will affect the numeric behavior of the FMU. If the FMU contains controller code, numeric stability or energy preservation laws are of lesser concern. On the other hand, plant models offering XCP access for parameter calibration may introduce surprising numerical effects in solvers that might require proper handling, like resetting solvers with every XCP write action.

Therefore, it is necessary to synchronize XCP variable access (read and write) with the state of the FMU.

2.1.3 Status and Outlook

A proposal for such a layered standard is being developed on GitHub (https://github.com/modelica/fmi-ls-xcp).

With small limitations, this layered standard can also be used with FMI 2.0.4.

Most mechanisms have been agreed upon and first prototype implementations have been realized. The cross-check of generated FMUs has been started as of writing this paper by companies such as dSPACE GmbH, ETAS GmbH, PMSF IT Consulting and SYNOPSYS.

2.2 FMI Layered Standard for Network Communication (LS-BUS)

2.2.1 Motivation

Simulation of modern automotive systems requires network communication between vECUs. Traditional simulations according to FMI 2.0 or other formats deal either with continuous signals or with non-standardized and proprietary solutions for exchanging network messages. In practice, such proprietary solutions often lead to interoperability issues when creating simulation systems with vECUs provided by different suppliers.

To minimize the resulting effort, the FMI layered standard for Network Communication (MAP FMI 2023a) was introduced. By using FMI 3.0 core standard features such as Co-Simulation, Clocks, clocked variables and terminals, the layered standard specifies a common bus interface and defines how to emulate a transport layer for several bus types in detail. While this layered standard has been initiated with automotive use cases in mind (with automotive network technologies such as CAN, LIN, FlexRay, CAN FD, CAN XL and Ethernet), the used concepts are kept general. This way the layered standard could also be applied to other domains such as industrial automation.

2.2.2 Approach

General concepts: The proposal on GitHub (https://github.com/modelica/fmi-ls-bus) provides two abstraction layers for different use cases:

- Physical signal abstraction ("high cut"): Use individual, clocked signal variables to transport logical, unit-based values between vECUs, ignoring transport layer-specific properties. The layered standard for this abstraction basically defines how bus signals have to be described in the model description file. It should be noted that creating FMUs with this abstraction layer typically requires a network description.
- Network abstraction ("low cut"): This abstraction allows the implementation of virtual bus drivers within FMUs on the level of the hardware abstraction layer. It uses clocked binary variables to exchange bus operations between FMUs based on a lightweight protocol defined by the layered standard. Bus operations are used to transmit bus messages as well as bus events like acknowledge or error events. This enables both ideal and more realistic bus simulations, depending on the capabilities of the FMU and importer. These capabilities can include timing, arbitration, error handling, status monitoring and other effects.

It is assumed that FMUs will provide only one of these abstraction layers although it would be technically feasible to support both.

System composition: In the simplest use case, the importer does not need to provide specific bus semantics of certain variables of an FMU; it simply forwards variable values between two FMUs according to the FMI standard. Such a simulation is shown in Figure 4. In this case, however, the bus simulation is idealized, i.e., effects like transmission time, arbitration or any other bus-specific behavior are not taken into account.

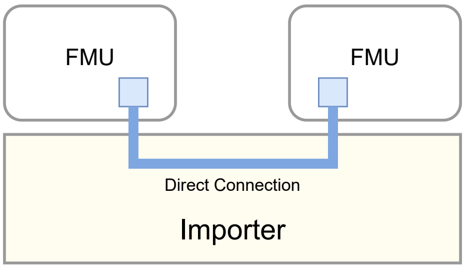


Figure 4. Direct communication of two FMUs, e.g., vECUs, on a common importer.

Only if more than two FMUs should be connected to a single network or a detailed bus simulation is desired in the "low cut" case, a dedicated bus simulation component is required. This bus simulation component then forwards bus operations between multiple senders and receivers and emulates the bus behavior. This type of communication allows the simulation of complex bus features, such as arbitration, the simulation of timing or the injection of bus failures. The supported bus features cannot be specified explicitly, but refer to a specific implementation of the bus simulation component and depend on the requirements of the bus simulation. The implementation of the bus simulation component could be done either by special capabilities of the importer, or by the provision of a Bus Simulation FMU. See Figure 5 for the simulation with such a Bus Simulation FMU. Here, the importer does not require any special features for bus simulation.

It should be explicitly noted that the FMUs integrated in the respective use case do not necessarily have to be different, which means that the same FMU can be integrated across all system compositions. The interface of the FMU to the importer is always the same, but a different subset of the features may be used.

Used FMI 3.0 concepts: The following FMI 3.0 concepts are used by this layered standard:

- **FMI for Co-Simulation**: The layered standard is currently being described in the context of the Co-

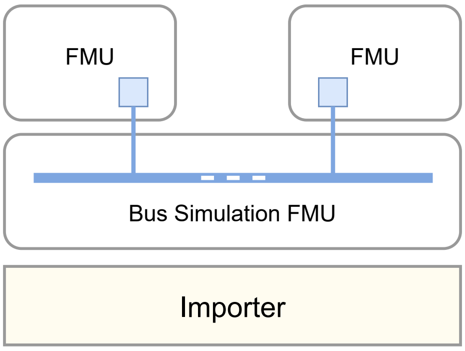


Figure 5. Realization of complex bus simulations by using a Bus Simulation FMU in addition to the communicating FMUs (e.g. vECUs).

Simulation mode, as this is best suited for the identified use cases.

- Hierarchical **Terminals** are used for grouping of variables. For the "high cut", the signals are nested hierarchically in several Terminals. The signals, together with their associated Clocks, are grouped based on the PDU and frame to which they belong. In the final aggregation, all frame terminals are combined in a Bus Terminal depending on their specific bus type. Within the "low cut" two binary variables and two Clocks are coupled into a common bus interface aggregated by a Terminal.
- **Clocks** and **clocked variables** are used to synchronize the exchange of network data among all FMUs in the simulated network.
- **Binary variables** with a dedicated MIME-type are used for the "low cut" to define the data exchanged for a specific bus type such as CAN.
- In the "high cut" variant, the **"extra/" directory** can optionally be used to ship network description files such as ARXML, DBC, LDF, Fibex, or others.

Timing aspects: Clocks are used to inform the importer about new signal values ("high cut") or binary bus operations ("low cut") to be exchanged. FMUs that want to send out those values exactly in time can use time-based Clocks and should support a variable communication step size. Periodic (fixed-time) FMUs are also supported, but in this case, multiple sends might fall into one communication step. While "high cut" signal variables will miss all but the last value sent, in the "low cut" case, all bus operations will be buffered in the binary variables.

Example for a "low cut" network interface: Figure 6 shows an example FMU with two binary variables `Rx_Data` and `Tx_Data`, and two Clock variables `Rx_Clock` and `Tx_Clock` that are aggregated to a `Bus Terminal`. Independent of the bus feature `Input` and `Output` represent exemplary additional FMI variables of the example FMU.

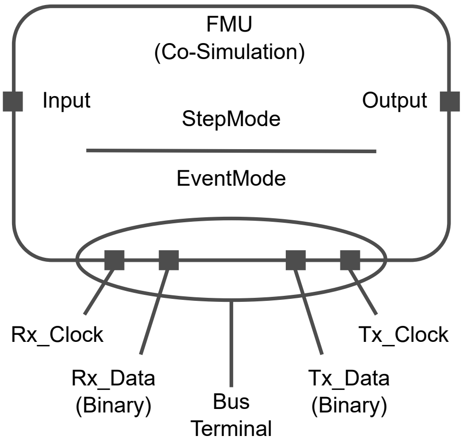


Figure 6. Example for a "low cut" Bus Terminal.

Based on this generic bus interface, an FMU can either be connected to a bus simulation component or directly to another FMU via an importer.

2.2.3 Status and Outlook

The first iteration primarily focuses on the basic concepts of the layered standard and the support for simulating CAN, CAN FD and CAN XL buses. Adding support for Ethernet, LIN and FlexRay is planned for upcoming iterations. Other bus systems from various domains can be specified in the future as required.

2.3 FMI Layered Standard for Structuring of Variables, Maps and Curves (LS-Struct)

2.3.1 Motivation

Grouping of variables, especially parameters: For many use cases, grouping of several parameters is very important. In FMI 1.0, 2.0 and 3.0, this can be partially realized with the "structured naming convention" (MAP FMI 2022a) which can be used, e.g., to represent array variables and/or hierarchical structures of variable. However, FMI 1.0 and 2.0 define scalar variables only, so for arrays this is not efficient, and generally the "structured naming convention" is not flexible enough for many applications.

Realizing n-D lookup tables: A special case of grouping variables is the representation of n-D lookup tables. In the context of the layered standard, an n-D lookup table is a sampled representation of a function of n input variables $y = F(x_1, x_2, x_3, \ldots, x_n)$ sampled on the vertices of a rectlinear grid. Such an n-D lookup table could be also called a map from the n-dimensional domain to a codomain. In (ASAM e.V. 2018) a 1-D lookup table is called `CURVE`, a 2-D lookup table is called `MAP` (see Figure 7), and a 3-D lookup table is called `CUBOID`. 4-D and 5-D lookup tables are called `CUBE_4` and `CUBE_5`, respectively. Higher dimensional lookup tables are not defined in (ASAM e.V. 2018).

FMI 3.0 introduces array variables (optionally with sizes that can be changed via structural parameters). However, an n-D lookup table is more than just one array parameter but it can be represented as a combination of several array variables.

Some tools started to implement proprietary solutions to represent lookup tables with FMI, e.g., through vendor-specific annotations or naming conventions, and the need for standardization became apparent.

2.3.2 Approach

Grouping of variables: FMI 3.0 includes a mechanism for grouping variables through the definition of Terminals ((MAP FMI 2022b). This concept was originally designed to allow for the connection of a group of variables, typically inputs or outputs, but also for parameter propagation. However, the semantics of Terminals are kept general, and thus they can be used just for grouping of variables. In this case connecting these "terminals" isn't the main focus.

n-D lookup tables: Their representation is a specialization of the more general parameter grouping concept. An exposed n-D lookup table within an FMU contains the following information:

- Domains: For each of the n dimensions of the lookup table an array variable (typically a parameter or a constant) with the sampling points (along this dimension) of the lookup-table must be referenced.

- Codomain: The sampled function values are stored in this references n-dimensional array.

- Optionally, for each dimension a variable (typically an input or a local variable) can be referenced that represent the current operating point (along this dimension)

- Optionally, additional variables containing related information can be referenced. (e.g., the interpolation algorithm in between the sampling points)

It is intended to use the concept of Terminals for this purpose by defining specific values for the terminalKind, variableKind and matchingRule attributes. Note that, e.g., CombiTable1D or CombiTable2D blocks of the Modelica standard library used within an FMU can be exposed with this approach as well.

2.3.3 Status and Outlook

At the time of writing this paper, this layered standard is in an early stage. Its development can be followed on GitHub (MAP FMI 2023c).

3 Layered Standards by other Organizations and Companies

The concept of layered standards is especially suitable for specific extensions to FMI that will not become part of the FMI Core standard and that are not developed by the FMI Project. Thus, companies and other organizations are encouraged to develop their own layered standards.

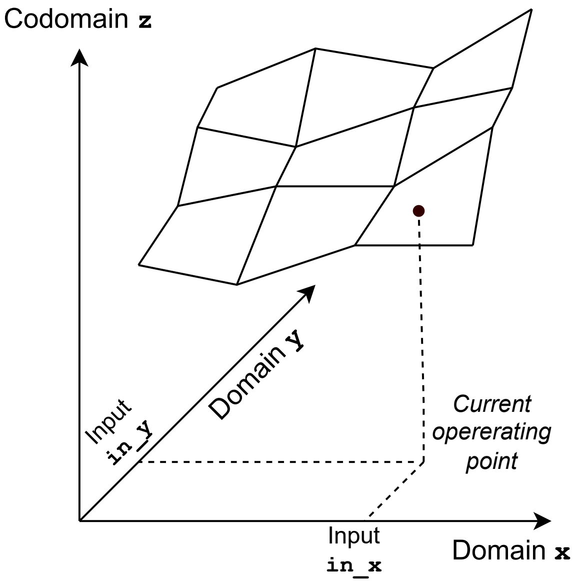


Figure 7. A 2-D lookup table with the current operating point

First drafts for such layered standards developed have been already created, e.g., for exchangeable binary codecs and the realization of binary variables in FMI 2.0 via string variables (Bosch 2023).

4 Summary and Outlook

The FMI 3.0 standard is currently in a phase of rapid adoption by tool vendors. However, many existing use cases for FMI-based simulation can already be handled with FMI 2.0. The switch to FMI 3.0 will be driven by new use cases with new requirements, such as detailed simulations of vECUs. For their realization with FMI, additional information and extended capabilities for certain domains are beneficial and necessary. In this paper we presented the concept of layered standards to FMI. We expect their broad adoption in the near future, especially after the publication of the layered standard examples currently being created by the FMI Project. During their development the concept of layered standards and the foreseen extension mechanisms in FMI 3.0 have already proven very effective.

Acknowledgements

We would like to thank all contributors to the Modelica Association standards, particularly FMI and SSP. The concept of layered standards for FMI is based on their valuable work and contributions.

References

ASAM e.V. (2017-11). *ASAM MCD-1 XCP v1.5.0.* URL: https://www.asam.net/standards/detail/mcd-1-xcp/.

ASAM e.V. (2018-03). *ASAM MCD-2 MC (aka ASAP2) v1.7.1.* URL: https://www.asam.net/standards/detail/mcd-2-mc/.

ASAM OSI Project (2022-07). *ASAM OSMP (Open Simulation Model Packaging) V1.3.0, part of ASAM OSI (Open Simulation Interface) v3.5.0*. URL: https://opensimulationinterface.github.io/osi-documentation/#_osi_sensor_model_packaging.

Berners-Lee, Tim, Roy T. Fielding, and Larry M Masinter (2005-01). *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. DOI: 10.17487/RFC3986. URL: https://www.rfc-editor.org/info/rfc3986.

Blochwitz, Torsten et al. (2011). "The Functional Mockup Interface for Tool independent Exchange of Simulation". In: *8th International Modelica Conference*. URL: http://www.ep.liu.se/ecp/063/013/ecp11063013.pdf.

Blochwitz, Torsten et al. (2012). "Functional Mockup Interface 2.0: The Standard for Tool Independent Exchange of Simulation Models". In: *8th International Modelica Conference*. URL: https://lup.lub.lu.se/search/ws/files/5428900/2972293.pdf.

Bosch (2023-05). *FMI Layerd Standard Drafts by Robert Bosch GmbH*. URL: https://github.com/boschglobal/dse.standards/tree/main/modelica.

Hodges, Jeff, Collin Jackson, and Adam Barth (2012-11). *HTTP Strict Transport Security (HSTS)*. RFC 6797. DOI: 10.17487/RFC6797. URL: https://www.rfc-editor.org/info/rfc6797.

Hoffman, Paul E. (2002-02). *SMTP Service Extension for Secure SMTP over Transport Layer Security*. RFC 3207. DOI: 10.17487/RFC3207. URL: https://www.rfc-editor.org/info/rfc3207.

Junghanns, Andreas et al. (2021). "The Functional Mock-up Interface3.0 - New Features Enabling New Applications". In: *14th International Modelica Conference*. URL: https://doi.org/10.3384/ecp2118117.

Klensin, Dr. John C. (2001-04). *Simple Mail Transfer Protocol*. RFC 2821. DOI: 10.17487/RFC2821. URL: https://www.rfc-editor.org/info/rfc2821.

MAP FMI (2022a-05). *Functional Mock-up Interface (FMI) Standard 2.0.4*. URL: https://github.com/modelica/fmi-standard/releases/download/v2.0.4/FMI-Specification-2.0.4.pdf.

MAP FMI (2022b-05). *Functional Mock-up Interface (FMI) Standard 3.0*. URL: https://fmi-standard.org/docs/3.0/.

MAP FMI (2023a). *Layerd Standard for Bus (unreleased)*. URL: https://modelica.github.io/fmi-ls-bus/main/.

MAP FMI (2023b). *Layered Standard for XCP (unreleased)*. URL: https://modelica.github.io/fmi-ls-xcp/main/.

MAP FMI (2023c). *Layered Standard Structuring of Data (unreleased)*. URL: https://modelica.github.io/fmi-ls-struct/main/.

MAP SSP (2019-03). *System Structure and Parameterization (SSP) Standard 1.0*. URL: https://ssp-standard.org/publications/SSP10/SystemStructureAndParameterization10.pdf.

Montulli, Lou and David M. Kristol (2000-10). *HTTP State Management Mechanism*. RFC 2965. DOI: 10.17487/RFC2965. URL: https://www.rfc-editor.org/info/rfc2965.

Nielsen, Henrik et al. (1999-06). *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. DOI: 10.17487/RFC2616. URL: https://www.rfc-editor.org/info/rfc2616.

Low-order aquifer thermal energy storage model for geothermal system simulation

Alessandro Maccarini[1] Michael Wetter[2] Davide Varesano[3] Martin Bloemendal[4] Alireza Afshari[1] Angelo Zarrella[3]

[1]Department of the Built Environment, Aalborg University, Denmark, amac@build.aau.dk
[2]Building Technology and Urban Systems Division, Lawrence Berkeley National Laboratory, USA
[3]Department of Industrial Engineering, University of Padova, Italy
[4]Department of Water Management, Delft University of Technology, and KWR water research, The Netherlands

Abstract

This paper presents a low-order aquifer thermal energy storage (ATES) model for simulation of combined subsurface and above-surface energy systems. The model is included in the Modelica IBPSA Library, which is a free open-source library with basic models for building and district energy and control systems. The model uses a lumped-component method, in which the transient conductive-convective heat and mass transfer equation is radially discretized. To verify the accuracy of the model, we present an inter-model comparison from a simulation test suite. Results show that the Modelica ATES model is in good agreement, with a normalized mean bias error for yearly variation of aquifer temperatures of 1.6×10^{-2} and 9×10^{-5} at 1 m and 10 m distance from the well.

Keywords: Aquifer thermal energy storage, Thermodynamics, IBPSA library

1 Introduction

Buildings account for approximately 30% of global energy consumption and are responsible for a significant portion of greenhouse gas emissions (IEA 2022). As populations and urbanization continue to grow, the energy demands of buildings are also increasing. Hence, to limit climate change, it is important to move away from using fossil fuels for heating and cooling of buildings.

Thermal energy storage is a key technology to increase renewable energy utilization. Thermal storage refers to the temporary storage of thermal energy, which can be used later to meet heating and cooling demands. By integrating thermal storage into building energy systems, it is possible to shift the energy demand daily from peak hours to off-peak hours, when energy is cheaper and often more sustainable, or seasonally to shift heat availability from summer to winter. This leads to significant reductions in carbon emissions, as well as lower energy costs for building owners and occupants.

Aquifer Thermal Energy Storage (ATES) is a technology that improves sustainability of space heating and cooling for buildings through seasonal storage of heat in aquifers. An ATES system consists of at least two wells which are coupled via a heat exchanger to exchange heat with the associated building. The basic operation of an ATES system is shown in Figure 1. In summer, cool groundwater is extracted from one well and circulated through the building system via heat exchangers or heat pumps. The water, which is heated during the process, is then injected into the other well (Dinçer et al. 2010). In winter the process is reversed. Since all the water from one well is re-injected into the other well, there is no net groundwater extraction.

ATES systems are a relatively modern technology. After engineering feasibility had been demonstrated in various projects, ATES technology was successfully established in Europe for both heating and cooling of buildings. Nowadays, more than 3000 ATES are implemented worldwide with 85% of the installations in the Netherlands and 10% in Sweden, Belgium and Denmark. Worldwide, ATES systems transfer a total amount of heat and cold that is estimated to exceed 2.5 TWh per year (Fleuchaus et al. 2018).

In early research studies on ATES, the focus revolved around solving technical, geochemical, and engineering obstacles. In subsequent years, the focus shifted towards improving system performances including the energy use of the above-surface plant facilities such as heat pumps, heat exchangers and cooling machines. Their performance is influenced by physical and chemical processes occurring underground. Predicting the behavior of these processes within a specific project often proves challenging with analytical calculations. Therefore, mathematical modeling took a pivotal role in the research and design of ATES systems.

Several ATES models and simulators have been presented in the literature. These are mostly developed using computational fluid dynamics software such as SHEMAT (Keller et al. 2020), FEFLOW (Trefry et al. 2007), MT3DMS (Bedekar et al. 2016), MODFLOW (Hughes et al. 2017). Such software enable an accurate prediction of underground conditions, but they have limited capabilities to integrate building simulation tools such as EnergyPlus, TRNSYS and Modelica, as only sophisticated co-simulations techniques can be used. Thus, a detailed

DOI
10.3384/ecp204389

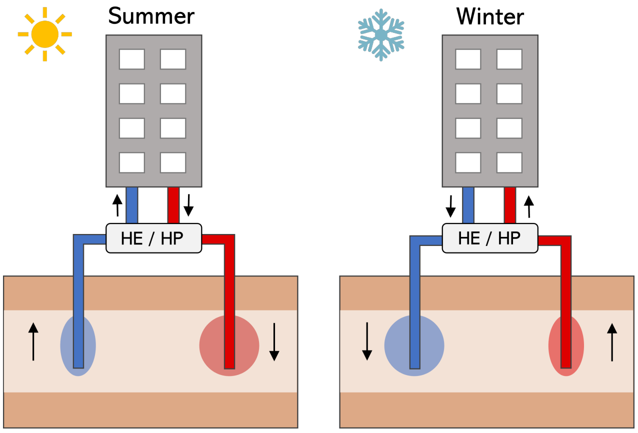


Figure 1. Basic working principle of an ATES system. The terms HE and HP stand respectively for heat exchanger and heat pump

performance analysis of an ATES connected to a building system typically requires knowledge in co-simulation, is computationally expensive, and is time consuming (Beernink et al. 2022; Bloemendal et al. 2018; Scalco et al. 2022).

Despite the fact that building systems (e.g. heat pumps, circulation pumps) dominate the primary energy use of ATES systems, only a few studies investigated the performance of ATES in conjunction with a detailed building system model. Bozkaya et al. (2018) developed a co-simulation method that combines COMSOL, MATLAB and TRNSYS. In this approach, COMSOL was used to model the ATES, TRNSYS to model the building and its heating, ventilation and air-conditioning (HVAC) system, and MATLAB acted as mediator to exchange information between the tools. Other studies (Kranz et al. 2013; Drenkelfort et al. 2015) used TRNAST, an ATES model developed for TRNSYS. The model is based on a finite-difference method and it relies on the assumptions that wells are thermally decoupled from each other. Tugores et al. (2015) describe a Modelica-based ATES model integrated into a district energy system model. The model was developed using a finite-volume approach and it was validated against a model simulated in COMSOL. However, this ATES model is not publicly available.

The literature review indicates that there is a lack of open-source, low-order ATES models that can enable fast and accurate geothermal system simulations integrating both above- and sub-surface system models.

This paper presents the development of an ATES model implemented in Modelica and to be included in the Modelica IBPSA Library (`https://github.com/ibpsa/modelica-ibpsa`). The Modelica IBPSA Library is a free open-source library with basic models that codify best practices for the implementation of models for building and district energy and control systems (Wetter, Treeck, et al. 2019). It is used as the basis of the four Modelica libraries AixLib (Müller et al. 2016), Buildings (Wetter, Zuo, et al. 2014), BuildingSystems (Nytsch-Geusen et al. 2012) and IDEAS (Jorissen et al. 2018). Making available a free and open-source ATES model is expected to accelerate the uptake of ATES-based energy system solutions towards the decarbonization of the heating and cooling sector.

The paper is structured as follows: Section 2 presents the modeling approach, Section 3 presents an inter-model comparison for a test simulation case and the paper ends with concluding remarks.

2 Methodology

2.1 Modeling approach

To calculate aquifer temperature at different locations over time, the model solves simplified heat and mass transfer. The following assumptions were made to simplify the numerical modeling of ATES:

- The computational domain is homogeneous, i.e., material properties and physical parameters are constant across the entire domain.
- The aquifer is confined by two impermeable layers. Therefore, the vertical infiltration of water is neglected, i.e., the movement of water is only radial.
- All heat transfer is axial-symmetric, and there is no vertical heat transfer.
- Natural ground water flow is neglected. Movement of water is only driven by artificial pumping energy.

The model is based on the partial differential equation (PDE) for 1D conductive-convective transient radial heat transport in porous media

$$\rho c \frac{\partial T}{\partial t} = \lambda \frac{\partial^2 T}{\partial r^2} - u \rho_w c_w \frac{\partial T}{\partial r}, \tag{1}$$

where ρ is the density, c is the specific heat capacity, T is the temperature, r is the radius, λ is the thermal conductivity and u is the velocity. The subscript w indicates water. The first term on the right hand side of (1) describes the effect of conduction, while the second term describes convection.

The soil and its water content are assumed to be in thermal equilibrium, i.e. they can be described by the same temperature. With this assumption, the properties of the aquifer are calculated as a weighted average of the values for dry soil and water

$$\rho c = \phi \rho_w c_w + (1-\phi)\rho_s c_s, \tag{2}$$

$$\lambda = \phi \lambda_w + (1-\phi)\lambda_s, \tag{3}$$

where ϕ is the porosity [1] and subscript s denotes dry soil.

The geometric representation of the model is illustrated in Figure 2. The aquifer around the well is modeled as a radially symmetric disc.

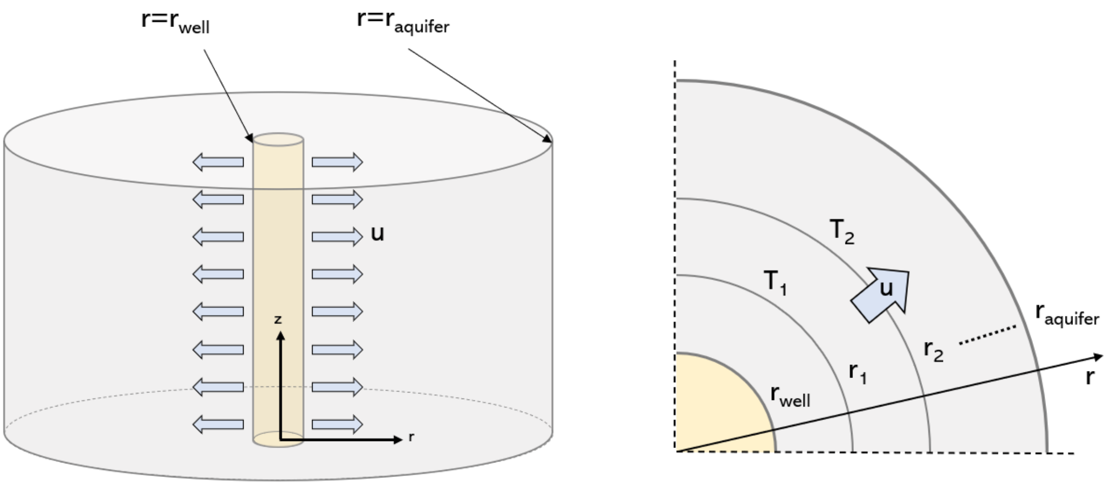


Figure 2. Geometric domain and discretization approach.

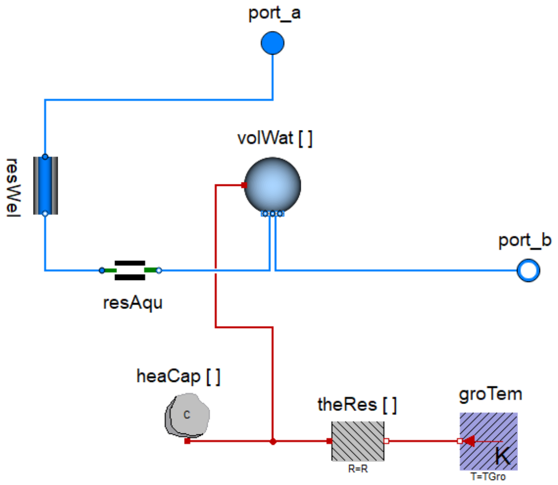


Figure 3. Modelica diagram of the ATES model.

2.2 Modelica implementation

The model implementation in the Modelica language is based on a lumped-component approach as Modelica does not support describing PDEs (i.e., equations that involve the derivative of variables in spatial directions). Therefore, we spatially discretized the domain and implemented the heat transfer process in the aquifer using a series of thermal capacitances and resistances along the radial direction. The implementation uses an array of `HeatCapacitor` and `ThermalResistor` models, as shown in Figure 3. The fluid flow was modelled by adding a series of fluid volumes, which are connected to the thermal capacitances via heat ports. The fluid stream was developed using the model `IBPSA.Fluid.MixingVolumes.MixingVolume`. This model represents an instantaneously mixed volume, where potential and kinetic energy at the port are neglected.

The thermal capacitance of each cylindrical layer is

$$C_i = \rho c V_i, \tag{4}$$

where V_i is the volume of the i^{th} cylindrical layer. The thermal resistances are calculated as

$$R_i = \frac{log(\frac{r_{c,i}}{r_{c,i-1}})}{2\pi\lambda h}, \tag{5}$$

where $r_{c,i}$ is the radius to the center of the i^{th} cylindrical layer, $r_{c,i-1}$ is the radius to the center of the $i-1^{th}$ cylindrical layer, and h is the thickness of the aquifer.

Based on this lumped-component approach, the heat balance for the i^{th} cylindrical layer is

$$\begin{aligned} C_i \frac{dT_i}{dt} &= \frac{T_{i+1}-T_i}{R_{i+1}} + \frac{T_{i-1}-T_i}{R_i} \\ &+ \dot{m}_w c_w \begin{cases} (T_{i-1}-T_i), \text{if } \dot{m}_w \geq 0, \\ (T_i - T_{i+1}), \text{otherwise,} \end{cases} \end{aligned} \tag{6}$$

where $\dot{m}_w$ is the water mass flow rate. The first and second term on the right hand side of (6) represent the heat conduction to neighboring layers, while the third term represents the heat convection caused by the movement of water.

Each cylindrical layer is spaced according to an expansion coefficient. This number is used to increase the size of the cylindrical layers from the well to the outer boundary of the aquifer.

The friction losses are computed using a power law model expressed by

$$\dot{m}_w = k \Delta p^m, \tag{7}$$

where k is a flow coefficient, Δp is the pressure difference, and m is the flow exponent. The flow coefficient is based on one data point of mass flow rate and pressure difference, and a given flow exponent.

3 Numerical experiments

3.1 Simulation setup

Ideally, the validation of a mathematical model would include comparison against measurements. However, measuring an ATES system, retrieving data, and analyzing them is a complex and costly task, and we had no such data available. Therefore, we conducted an inter-model comparison, using a simulation test suite developed by Mindel et al. (2021). This test suite comprises a set of cases to assess the thermo-hydraulic modelling capabilities of various geothermal simulators.

The comparison was carried out with respect to the test case called "TC2 - well-test comparison". The main goal of TC2 is to compare aquifer temperatures under a typical operation of an ATES system consisting of injection, falloff, drawdown, and build-up. Such a sequence represents an idealized operation of a seasonal storage system.

The injection phase represents the charging period, while the drawdown phase represents the discharge period. Intermediate phases of falloff and build-up represent periods of storage or inactivity. The overall operational period is one year, and the sequence of the different phases is the following:

1. **Injection**: Water is pumped at $\dot{m}_w$ = 1 $kg\,s^{-1}$ and T_{inj} = 120°C for 120 days.
2. **Falloff**: Well is shut-in, $\dot{m}_w$ = 0 $kg\,s^{-1}$, for 60 days.
3. **Drawdown**: Water is pumped at $\dot{m}_w$ = -1 $kg\,s^{-1}$ for 120 days.
4. **Build-up phase**: Well is shut-in, $\dot{m}_w$ = 0 $kg\,s^{-1}$, for 65.25 days.

A list of relevant input parameters used for the simulations are shown in Table 1.

Table 1. Simulation input specification summary.

Parameter	*Value*	*Unit*
Porosity	0.2	[1]
Soil density	2680	$kg\,m^{-3}$
Soil thermal conductivity	2.8	$W\,m^{-1}\,K^{-1}$
Soil specific heat capacity	833	$J\,kg^{-1}\,K^{-1}$
Wellbore radius	0.1	m
Domain radius	2400	m
Domain height	200	m
Initial temperature	34	$°C$
Injection temperature	120	$°C$

For comparative purposes, temperature values were recorded via virtual probes located at $r = \{1, 10\}[m]$ for $t = \{0, 50, 100, 150, 200, 250, 310, 365.25\}[days]$.

3.2 Result comparison

Figure 4a shows the temperature vs. time comparison for the first probe, located at $r = 1$ m. Overall, the simulation results of the Modelica model (named *IBPSA_Mod* in the graph) are in good agreement with the other simulators. As stated in Mindel et al. (2021), the discrepancy between the initial value of temperature for *Tough3*, *MOOSE* and the other simulators is due to the different injection conditions. These simulators do not use an enthalpy source-type or a Neuman boundary condition but they use a Dirichlet boundary condition. It can also be observed that the tool *CODE_BRIGHT* presents a time delay.

Figure 4b shows the temperature vs. time comparison for the second probe, located at $r = 10\,m$. Also in this case, the Modelica model provides simulation results that are in good agreement with the other simulators. Data from *Tough3* have been omitted from Figure 4b as they largely differ from the other tools.

Note that the test suite includes temperature comparisons also for probes located further away from the well ($r > 10\,m$). However, at such locations, only a small numerical fluctuation can be observed, which is about ±0.01°C around the initial temperature of 34°C. Therefore, these test cases have not been included in this work.

To quantitatively compare the simulation results, we normalize the temperatures by their driving potential, and use the Mean Bias Error (MBE), applied to the point-to-point difference between the model and a reference value. We normalized the temperatures using

$$\theta = \frac{T - T_{min}}{T_{max} - T_{min}}, \tag{8}$$

where T is the temperature of the simulation, T_{min} is equal to the initial conditions $T_{min} = 34$°C, and $T_{max} = 120$°C is the temperature at which the water is injected.

The mean bias error is

$$MBE = \frac{\sum_{i=1}^{n}(\theta_{m,i} - \theta_{r,i})}{n}, \tag{9}$$

where the subscript m denotes the model value and r the reference value. For the reference values, we used the average of all other simulators, but removed the results from *Tough3*, *MOOSE* and *CODE_BRIGHT*, as they are seen as outliers due to different boundary conditions and time delay.

The results are shown in Table 2 and Table 3. For the probe located at $r = 1$ m, the normalized MBE is 1.6×10^{-2}, while for the probe at $r = 10\,m$, it is 9×10^{-5}. Since both values are positive, it can be deduced that the Modelica model globally over-predict the results. The higher normalized MBE for the probe at $r = 1$ m is mostly due to the fact that the simulation results of the Modelica model lie on the high end of the temperature range across tools.

4 Conclusions

ATES is a technology that enables seasonal storage of thermal energy in the groundwater. Several ATES models exist in literature, but these typically use computational fluid dynamics software, making it difficult to analyze the interactions between sub-surface and above-surface energy systems.

DOI
10.3384/ecp204389

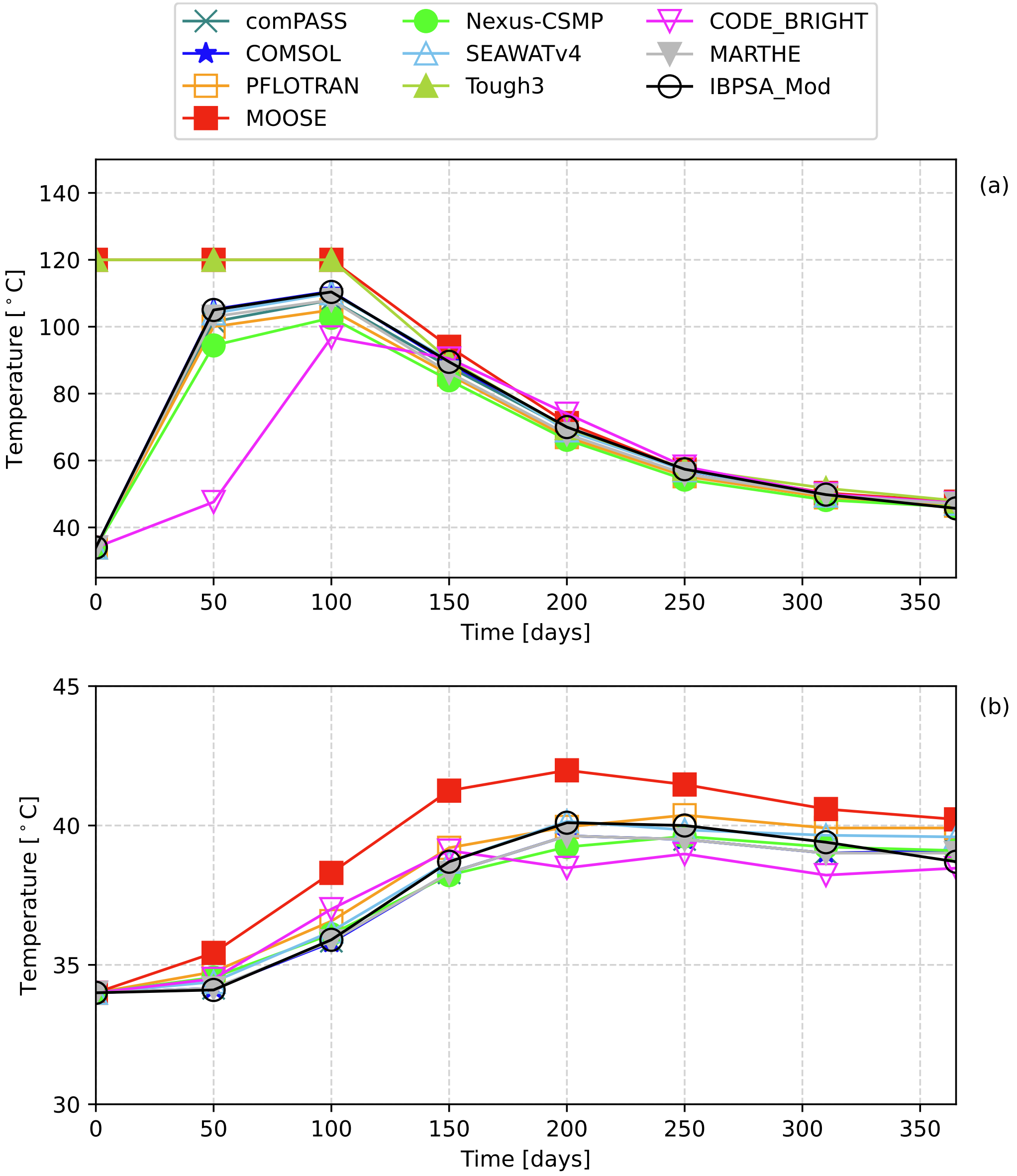


Figure 4. Temperature variation over time for the probes located at r=1 m (a) and r=10 m (b).

Table 2. Quantitative comparison for $r = 1\ m$.

Time [d]	θ_m	θ_r	*Error*
0	0	0	0
50	8.3×10^{-1}	7.8×10^{-1}	4.2×10^{-2}
100	8.9×10^{-1}	8.5×10^{-1}	3.5×10^{-2}
150	6.5×10^{-1}	6.2×10^{-1}	2.7×10^{-2}
200	4.2×10^{-1}	4×10^{-1}	2×10^{-2}
250	2.7×10^{-1}	2.6×10^{-1}	1.5×10^{-2}
310	1.8×10^{-1}	1.8×10^{-1}	5.8×10^{-3}
365.25	1.4×10^{-1}	1.5×10^{-1}	-1.4×10^{-2}
			MBE=1.6×10^{-2}

Table 3. Quantitative comparison for $r = 10\ m$.

Time [d]	θ_m	θ_r	*Error*
0	0	0	0
50	1.2×10^{-3}	4×10^{-3}	2.8×10^{-3}
100	2.2×10^{-2}	2.4×10^{-2}	-1.9×10^{-3}
150	5.5×10^{-2}	5.2×10^{-2}	2.3×10^{-3}
200	7.1×10^{-2}	6.6×10^{-2}	4.9×10^{-3}
250	7×10^{-2}	6.6×10^{-2}	3.6×10^{-3}
310	6.3×10^{-2}	6.2×10^{-2}	6.4×10^{-3}
365.25	5.5×10^{-2}	6.1×10^{-2}	-6.4×10^{-3}
			MBE=9×10^{-5}

We presented the development of a simplified ATES model to be included in the Modelica IBPSA Library. This is expected to facilitate the design, operation and control of ATES-based energy systems, as the ATES model can be directly connected to above-surface energy system models, typically by use of a heat exchanger or a heat pump.

The accuracy of the model was validated by performing an inter-model comparison using results from a simulation test suite. Results showed that the Modelica ATES model can predict aquifer temperatures with a good degree of accuracy. The normalized mean bias error at $r = 1\ m$ and $r = 10\ m$ distance from the well was 1.6×10^{-2} and 9×10^{-5} respectively.

Further work will focus on the validation of the model against measurements retrieved from a real-life ATES system. Moreover, a detailed application case study consisting of a full coupling between the ATES model and building energy system model will be developed and the model will be included in the Modelica IBPSA Library. In addition, a multi-well model will be developed to enable the connection of multiple wells within the same ATES system.

5 Data availability statement

The model is available from `https://github.com/ibpsa/modelica-ibpsa`, commit `https://github.com/ibpsa/modelica-ibpsa/commit/84f9135e737147fee779962a3b3f33ea40479657`. The validations were performed using commit `https://github.com/ibpsa/modelica-ibpsa/commit/b073350fe7952dc70ff3f60a014f4431ad3f5d43`.

Acknowledgements

This work was supported by Innovation Fund Denmark (grant no. 1158-00005B) and the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy (contract no. DE-AC02-05CH11231) as part of the project GOES - Geothermal-based Optimized Energy Systems, which is conducted under the H2020 GEOTHERMICA ERA-Net research program.

In addition, this work was supported by the Danish Energy Agency under the Energy Technology Development and Demonstration Programme (grant no. 64019-0575).

This work emerged from the IBPSA Modelica Working Group whose work is conducted under the umbrella of the International Building Performance Simulation Association (IBPSA). The IBPSA Modelica Working Group develops the IBPSA Modelica Library and coordinates the needs between the IBPSA and the Modelica communities.

References

Bedekar, V. et al. (2016). “MT3D-USGS version 1: A U.S. Geological Survey release of MT3DMS updated with new and expanded transport capabilities for use with MODFLOW”. In: *U.S. Geological Survey - Techniques and Methods 6-A53*. DOI: 10.3133/tm6A53.

Beernink, Stijn et al. (2022). “Maximizing the use of aquifer thermal energy storage systems in urban areas: effects on individual system primary energy use and overall GHG emissions”. In: *Applied Energy* 311, p. 118587. DOI: 10.1016/j.apenergy.2022.118587.

Bloemendal, Martin, Marc Jaxa-Rozen, and Theo Olsthoorn (2018). “Methods for planning of ATES systems”. In: *Applied Energy* 216, pp. 534–557. DOI: 10.1016/j.apenergy.2018.02.068.

Bozkaya, Basar, Rongling Li, and Wim Zeiler (2018). “A dynamic building and aquifer co-simulation method for thermal imbalance investigation”. In: *Applied Thermal Engineering* 144, pp. 681–694. DOI: 10.1016/j.applthermaleng.2018.08.095.

Dinçer, Ibrahim and Marc A. Rosen (2010). *Thermal Energy Storage : Systems and Applications*. John Wiley and Sons.

Drenkelfort, Gregor et al. (2015). “Aquifer thermal energy storages as a cooling option for German data centers”. In: *Energy Efficiency* 8, pp. 385–402. DOI: 10.1007/s12053-014-9295-1.

Fleuchaus, Paul et al. (2018). “Worldwide application of aquifer thermal energy storage – A review”. In: *Renewable and Sustainable Energy Reviews* 94, pp. 861–876. DOI: 10.1016/j.rser.2018.06.057.

Hughes, Joseph D, Christian D Langevin, and Edward R Banta (2017). “Documentation for the MODFLOW 6 framework”. In: *U.S. Geological Survey - Techniques and Methods 6-A57*. DOI: 10.3133/tm6A57.

IEA (2022). *Buildings*. IEA, Paris. URL: https://www.iea.org/reports/buildings.

Jorissen, Filip et al. (2018). “Implementation and Verification of the IDEAS Building Energy Simulation Library”. In: *Journal of Building Performance Simulation* 11 (6), pp. 669–688. DOI: 10.1080/19401493.2018.1428361.

Keller, Johannes et al. (2020). “SHEMAT-Suite: An open-source code for simulating flow, heat and species transport in porous media”. In: *SoftwareX* 12, p. 100533. DOI: 10.1016/j.softx.2020.100533. URL: https://www.sciencedirect.com/science/article/pii/S2352711020301357.

Kranz, Stefan and Stephanie Frick (2013). “Efficient cooling energy supply with aquifer thermal energy storages”. In: *Applied Energy* 109, pp. 321–327. DOI: 10.1016/j.apenergy.2012.12.002.

Mindel, Julian E. et al. (2021). “Benchmark study of simulators for thermo-hydraulic modelling of low enthalpy geothermal processes”. In: *Geothermics* 96, p. 102130. DOI: 10.1016/j.geothermics.2021.102130.

Müller, Dirk et al. (2016). “AixLib - An Open-Source Modelica Library within the IEA-EBC Annex 60 Framework”. In: *Proceedings of BauSim Conference 2016: 6th Conference of IBPSA-Germany and Austria*, Dresden, Germany, 3–9 September.

Nytsch-Geusen, Christoph et al. (2012). “Modelica BuildingSystems - Eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme”. In: *Proceedings of BauSim Conference 2012: 4th Conference of IBPSA-Germany and Austria*, Berlin, Germany, 26–28 September.

Scalco, Elisa et al. (2022). “An aquifer thermal energy storage model for efficient simulations of district systems”. In: *Proceedings of CLIMA 2022 Conference*, Rotterdam, The Netherlands, May 22–25. DOI: 10.34641/clima.2022.346.

Trefry, Mike G. and Chris Muffels (2007). “FEFLOW: A Finite-Element Ground Water Flow and Transport Modeling Tool”. In: *Groundwater* 45.5, pp. 525–528. DOI: 10.1111/j.1745-6584.2007.00358.x.

Tugores, Carles Ribas et al. (2015). “Coupled modeling of a District Heating System with Aquifer Thermal Energy Storage and Absorption Heat Transformer”. In: *Proceedings of the 11th International Modelica Conference*, 197–206, Versailles, France, Sep 21–23. DOI: 10.3384/ecp15118197.

Wetter, Michael, Christoph van Treeck, et al. (2019). “IBPSA Project 1: BIM/GIS and Modelica framework for building and community energy system design and operation - ongoing developments, lessons learned and challenges”. In: *IOP Conference Series: Earth and Environmental Science* 323. DOI: 10.1088/1755-1315/323/1/012114.

Wetter, Michael, Wangda Zuo, et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Heat Consumer Model for Robust and Fast Simulations of District Heating Networks

Johannes Zipplies[1] Janybek Orozaliev[1] Klaus Vajen[1]

[1]Institute for Thermal Engineering, University of Kassel, Germany, {j.zipplies, solar}@uni-kassel.de

Abstract

Dynamic thermo-hydraulic simulations of district heating networks are an essential tool to investigate concepts for their sustainable design and operation. The way the numerous heat consumers are modeled has crucial impact on the simulation performance. The proposed model for heat consumers is designed to require low computational effort by using a simplified modeling approach, avoiding state events and limiting its dynamics, while still reproducing their main characteristics. It is tested for a demonstration network, showing its ability to yield plausible results throughout the whole range of operational states including undersupply situations. The results show that the heat consumer model itself requires little time to simulate but significantly influences the simulation time for the district heating network. Fast dynamics and including a bypass in the model increase the simulation time, so that users should sensibly choose how to use these options. Furthermore, heat consumer models triggering many state events result in the highest computational effort.

Keywords: Modelica, District Heating Network, Heat Consumer, Simulation Performance

1 Introduction

In light of the man-made climate crisis, a fast decarbonization of heating has to be achieved. Within this transition of the heat sector, district heating is a recommended solution, especially for densely populated areas, as it facilitates a combination of various renewable heat sources, excess heat usage and heat accumulators (even seasonal) and coupling to the electricity sector to reach a economically viable sustainable heat supply system, so called 4th generation district heating (Lund et al. 2014).

Models for dynamic thermo-hydraulic simulations of the district heating networks (DHN) are an essential tool within the transformation towards 4th generation DH. Their purpose is to investigate how decentralized or fluctuating renewable heat supply units and heat accumulators may be integrated into DHNs (understand dynamic effects, develop control strategies), how and to what extent the DHN itself acts as a heat storage and which side-effects, such as pipe fatigue through temperature cycles or hydraulic bottlenecks, new units and operating strategies entail.

A major challenge in the simulation of DHNs is to find a compromise between model accuracy and computational effort. Within a DHN the heat consumers (HC) are very numerous so that the effort for simulating them is crucial for the overall simulation time. Moreover, the HCs have a major impact on dynamics of mass flows and temperatures within the network and thus determine the effort to compute the fluid and temperature propagation in the pipes.

Within this contribution a HC model for simulations of DHNs is described and evaluated. The goal of the proposed HC model is to provide plausible behavior throughout the whole range of possible operation states (including undesirable situations, such as too low supply line temperatures or differential pressures) while keeping the computational effort for simulations as low as possible.

2 Simulation of DHN Using Modelica

Modelica, being an acausal multi-physics modeling language, is generally well suited for dynamic simulations of DHNs with their thermal and hydraulic effects and the option of flow-reversals.

2.1 Models for DHN and HC

The *Modelica Standard Library* (Modelica Association 2019) provides a large number of base models (such as the fluid-connector) and component models for thermo-fluid systems. Moreover, van der Heijde et al. (2017) developed and validated a dynamic plug-flow pipe model, that is freely available via the *Modelica IBPSA Library* (International Building Performance Simulation Association 2018) and which is used within other libraries with models specialized for the simulation of DHN.

2.1.1 AixLib

AixLib is an open-source Modelica library for the simulation of energy systems on building to district scale developed at RWTH Aachen University (Müller et al. 2016; Maier et al. 2022; RWTH-EBC 2021). It extends the *Modelica IBPSA Library* and has a section *DistrictHeatingCooling* with models specialized for the simulation of DHN.

Within this section, the library provides so called "open-loop" models for HCs, which are similar to the HC model presented in this contribution. The open-loop design means, that the models do not contain a fluid model that connects flow and return line, which allows to decouple the respective equation systems for fluid flow and pressures in the DHN. Stock et al. (2023) state that open-loop models reduce the computational effort and yield valid results when the research focus is on heat distribution and not on control of the HCs or heat sources. They successfully evaluate the hydraulic effects of the integration of a waste heat source into an existing DHN at different temperature levels.

The HC models determine the required mass flow based on a heat load input and the temperatures. The return line temperature is either a constant value or set to achieve a constant temperature difference to the supply line. A bypass that maintains a minimum mass flow may be included. It is active whenever the HC mass flow would drop below a threshold (irrespective temperatures and pressures), sets the heat flow to zero and triggers state events, whenever activated or deactivated.

2.1.2 DisHeatLib

Leitner et al. (2019) describe a method to assess the operation of coupled heat and power networks and published their Modelica models within the library *DisHeatLib* (AIT-IES 2022), which builds upon the *Modelica IBPSA Library* and contains a variety of models for DHN Simulations.

To model HCs, the library provides models for *demand* (intended as a simple representation of a heat load) and *substation* (modeling heat transfer from the network to the HC). The *substation* models provide a variety of technical configurations (with or w/o heat exchanger, optional storages, optional bypass), so that these technical options and their behavior within a DHN can be examined. However all HC models in *DisHeatLib* include control loops, fluid models that connect supply to return line and some have a high degree of detail as the various components are explicitly modeled, which results in high computational effort to simulate a DHN with numerous HCs.

2.1.3 DHNSim

At the Department of Solar and Thermal Engineering of the University of Kassel, Modelica models for long-term simulation of whole DHNs have been developed within the in-house library *DHNSim*. The pipe model in *DHNSim* builds upon the plug-flow pipe model by van der Heijde et al. (2017). Furthermore, the library contains models for supply units, the HCs (described in this contribution, see section 3) and the required environment to easily build a consistent DHN model. Zipplies, Orozaliev, and Vajen (2023, in press) give an overview on the structure, goals and general implementation of the models.

2.2 Strategies for Fast Simulations of DHN

Figure 1 illustrates a general consideration of the drivers for computational effort of DHN simulations. On the one hand, the pipe network model results in a large system of equations that has to be solved for each simulation step and numerous states to integrate. Thus, it determines the effort to calculate one simulation step. On the other hand, the models of the supply unit and HCs will not cause much computational effort themselves, if they are simple. However, as they determine the mass flows, temperatures and pressures in the network (and their derivatives), they will have a crucial impact on the number of steps that a variable step size solver will have to calculate. Given this consideration, the following subsections describe general strategies for fast simulations of DHN that apply to the proposed HC model, which is described in detail in section 3.

2.2.1 Simplified Modeling Approach of HCs

The simulation of a large branched or even meshed pipe network is a complex task that requires high computational effort. Therefore it is recommendable, or even absolutely necessary, to limit the degree of detail of the models of the supply unit and the HCs in the DHN to a minimum extent that still leads to valid results. This applies especially to the HCs, as they are numerous and determine the mass flows and return temperatures for the pipe network. Thus, the proposed HC model does not contain detailed physical models for the actual components of substation and secondary side (pipes, valves, heat exchangers, pumps, heat storage, radiators, floor heating etc). This simplified modeling approach allows to follow the open-loop design implemented in *AixLib* (see section 2.1.1).

The HC model simply uses a prescribed heat flow (or optionally mass flow) as input and uses the actual temperature in the supply line and a prescribed return line temperature (constant value or as additional variable input) to calculate the mass flow. While this seems to be a very simple modeling task at first glance, some more details and features are needed to obtain fast, stable and valid simulations with such a HC model. These are described in section 3 and include major improvements compared to the open-loop models in *AixLib*.

2.2.2 Avoiding Events

Simulation models may include equations or algorithms that abruptly change the model behavior. Examples are flow reversals (mass flow changes sign) or switching units on and off (boolean variable changes value). Within Modelica these moments are called "events" and whenever the integration algorithm detects such an event, the integrator tries to determine the exact point of time, when this abrupt change occurs, and restarts the simulation with the changed model behavior from this point, so that the transition from one state to the other is simulated correctly.

While this approach avoids inaccurate results or even failures of the simulation that might occur otherwise, it also adds computational effort to the simulation. Thus, models should generate events only if necessary and high numbers of events should be avoided in the use case of long-term simulations of large DHNs.

If a variable is continuous at an event, it is possible to prevent the event using the Modelica built-in function `smooth()` (Fritzson 2015). Furthermore

DOI
10.3384/ecp204397

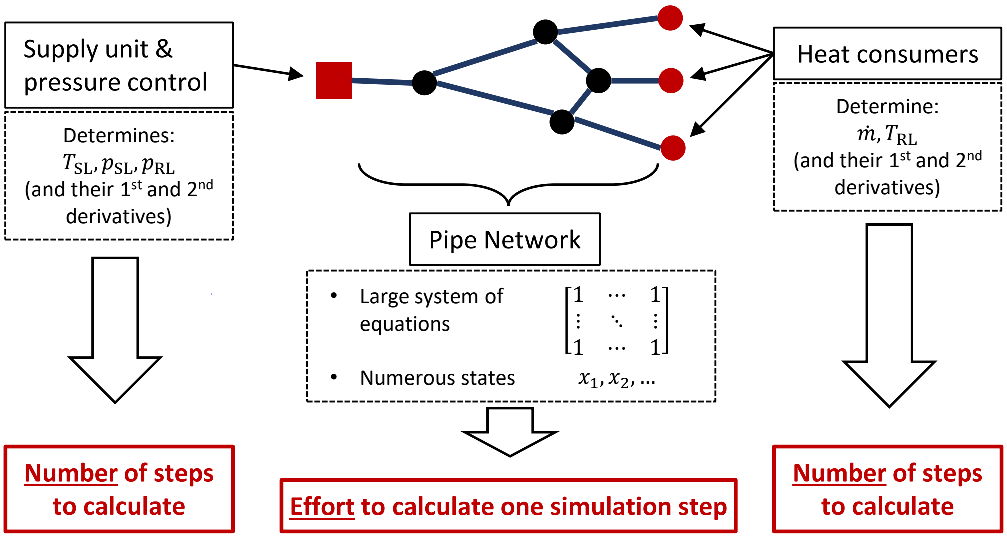

Supply unit & pressure control
Determines:
T_{SL}, p_{SL}, p_{RL}
(and their 1st and 2nd derivatives)
Heat consumers
Determine:
$\dot{m}, T_{RL}$
(and their 1st and 2nd derivatives)
Pipe Network
Large system of equations
Numerous states
$x_1, x_2, \dots$
Number of steps to calculate
Effort to calculate one simulation step
Number of steps to calculate

Figure 1. Drivers for computational effort of DHN simulations: While the model of the pipe network dominates the effort to calculate one step, models of the supply unit and even more of the HCs determine the number of steps, that the simulation requires.

the *Modelica Standard Library* provides the function `Modelica.Fluid.Utilities.regStep()` to approximate a step by a smooth transition, that is once continuously differentiable and prevents events (Modelica Association 2019). Both functions are very useful to avoid events in the HC model and are used in the implementation wherever applicable.

2.2.3 Limiting Dynamics of the Models

When modeling DHNs, it is useful to define which time scale of dynamic effects is within scope and which not. Then, the dynamics of the models can be restricted to this time scale, so that the effects out of scope are not modeled and simulated to avoid computational effort. In fact, preventing the HC model from imposing instant changes of mass flows is not a limitation of the model, but a realistic feature, because the actuators need some time to react to changing set-points (e.g. valve opening/closing time, usually seconds to a few minutes).

3 Description of the Proposed Heat Consumer Model

The implementation of the proposed HC model follows the previous considerations to keep computational effort for the simulation of the DHN low.

3.1 Heat Consumer Model Design

Figure 2 gives an overview on the design of the proposed HC model. The open-loop design (more detail in section 2.2.1) is obvious as there is no fluid model connecting the supply line with the return line. The model is split into a bypass and a load part, which both are modeled via a control block that calculates set-points for mass flow (and return temperature in the latter case) and two mass flow sources that generate the prescribed in- and outflows. The calculation of the mass flows is based on the input load time series (connected via a data bus), the measured differential pressure in the load and an input value of the supply line temperature, which is connected to the end of the supply line pipe model right before the HC to provide a valid temperature value during zero flow periods. The differential pressure signal is also connected to the data bus for further processing by the network's differential pressure control.

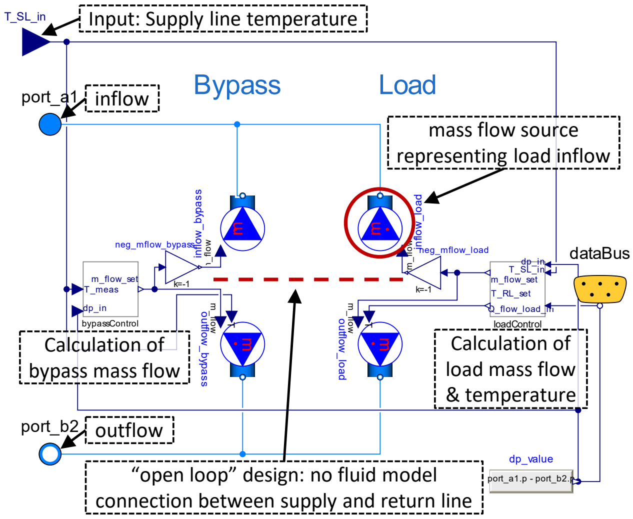


Figure 2. Diagram layer of the proposed HC model.

3.2 Determining Load Mass Flow

The determination of the load mass flow $\dot{m}_{\text{load}}$ within the load control block deserves special attention. It is calculated from the prescribed heat flow $\dot{Q}_{\text{load}}$, the heat capacity of water c_{p} and the temperatures in supply T_{SL} and return line T_{RL} according to equation 1.

$$\dot{m}_{\mathrm{load}} = \frac{\dot{Q}_{\mathrm{load}}}{c_{\mathrm{p}} \cdot (T_{\mathrm{SL}} - T_{\mathrm{RL}})} \tag{1}$$

However, a robust implementation of this simple equation according to the previously described goals and strategies requires some more details.

First, $\dot{m}_{\mathrm{load}}$ is limited to a meaningful range between 0 and a maximum mass flow m_flow_max.

Second, there may be situations, when the supply line temperature is close to or even below the set-point return line temperature, causing equation 1 to yield infinite or negative values. In such cases, it can be assumed that the set point temperature for the supply line of the secondary side of the substation is not reached, causing the controller and regulator of the substation to increase the primary mass flow as much as possible. Furthermore, it is assumed that in these situations the amount of heat extracted from the mass flow is negligibly small. The increase in primary mass flow is modeled as a smooth transition between normal and undersupply operation with a regStep() formula, increasing the mass flow from $\dot{m}_{\mathrm{load}}$ (calculated according to equation 1) to m_flow_max. The prescribed return line temperature changes to the actual supply line temperature when the difference between flow and return line temperature crosses zero using a smooth() operator to avoid events.

Third, the HC model is intended to be used within a DHN model with a differential pressure control that assures a minimum differential pressure. In cases where the heat supply unit is not able to provide sufficient differential pressure at HCs (below 90 % of the rated minimum differential pressure), the model reduces m_flow_max with another regStep() formula, finally reaching 0 when the differential pressure is 0 or below. This simple and computationally light implementation allows to detect such pressure undersupply situations and provides insight into which units would be affected to what extent. However, it is not a physically exact representation of the mass flow and pressure loss conditions in such a situation. This feature might add an algebraic loop to the model, as it introduces an interdependence of differential pressure and mass flows at the HCs. The resulting nonlinear equation system would be solved during each time step at high computational effort. This is avoided by the next feature.

Fourth, in line with section 2.2.3, the mass flow variable has a time constant tau_m_flow. This is implemented by introducing two mass flow variables: m_flow_fast is calculated according to equation 1, while the mass flow to be set in the model m_flow_set is delayed by using the time constant tau_m_flow, as shown in listing 1 (implementation adapted from Lawrence Berkeley National Laboratory (2023, Section 3.3.4)). This feature introduces state variables into the mass flow calculation, so that the algebraic loop mentioned in the previous paragraph is avoided.

Listing 1. Implementation of the mass flow time constant

```
der(m_flow_set) = (m_flow_fast-
    m_flow_set) / tau_m_flow;
```

Finally, as an optional feature, the consumer model is able to include a hysteresis: Whenever the prescribed heat flow falls below the switch-off threshold, $\dot{m}_{\mathrm{load}}$ is set to zero until the value rises again above the switch-on threshold. This feature may reduce computational effort if the time series of the prescribed load value includes longer periods of negligibly low values: Instead of simulating them in detail, they are omitted. However, this feature will trigger events whenever the thresholds are crossed at the cost of additional computational effort so that the simulation time may even increase.

3.3 Determining Bypass Mass Flow

The bypass is intended to maintain a certain minimum temperature in the supply line before the HC. To that end, the bypass control sets the bypass mass flow depending on this temperature: When it is high enough, the mass flow is 0. Once the temperature approaches the set-point temperature, that the bypass shall maintain, the mass flow is gradually increased within a bandwidth around the set-point temperature until it finally remains at a maximum mass flow, if the supply line temperature is at or below the lower end of the bandwidth. Once again, this behavior is implemented using regStep(), as this yields a smooth characteristic and does not trigger events.

In addition, alike in load control, the maximum bypass mass flow is reduced in cases of pressure undersupply.

The return temperature of the bypass mass flow is simply set to the actual supply line temperature, as it is assumed that no heat is extracted from this mass flow.

4 Evaluation of the Heat Consumer Model

To evaluate the HC model concerning the results and its effects on simulation performance, a demonstration network is modeled and simulated in Dymola using the models of the in-house library DHNSim. The simulations are run with the same network and heat load data for different HC model configurations to investigate their effects on simulation results and performance. Additionally, simulations are also performed with the two open-loop demand models from AixLib.Fluid.DistrictHeatingCooling (VarTSupplyDp and VarTSupplyDpBypass, constant return temperature) and the most simple configuration of DisHeatLib.Demand.Demand (constant return temperature, linearized flow characteristic in the flow unit). For the last, it was a difficult task to obtain stable operation of the HCs due to oscillations in the internal control loops. Table 1 gives an overview on the simulation runs and their specifications.

DOI
10.3384/ecp204397

Table 1. Overview on the simulation runs

Name	Specifications (HC model and other)
main	*DHNSim*, constant return temperature, with bypass, `tau_m_flow` = 180 s, no hysteresis
fastDynamics	alike *main* , but `tau_m_flow` = 30 s
noBypass	alike *main* , but no bypasses
hysteresis	alike *main* , but with hysteresis to swith load mass flow off
onePipe	alike *main* , but pipe network contains only one pipe
AixLib	*AixLib* open-loop demand model, constant return temperature
AixLibBypass	*AixLib* open-loop demand model, constant return temperature, with bypass
DisHeatLib	*DisHeatLib* demand model, constant return temperature, linearized flow characteristic

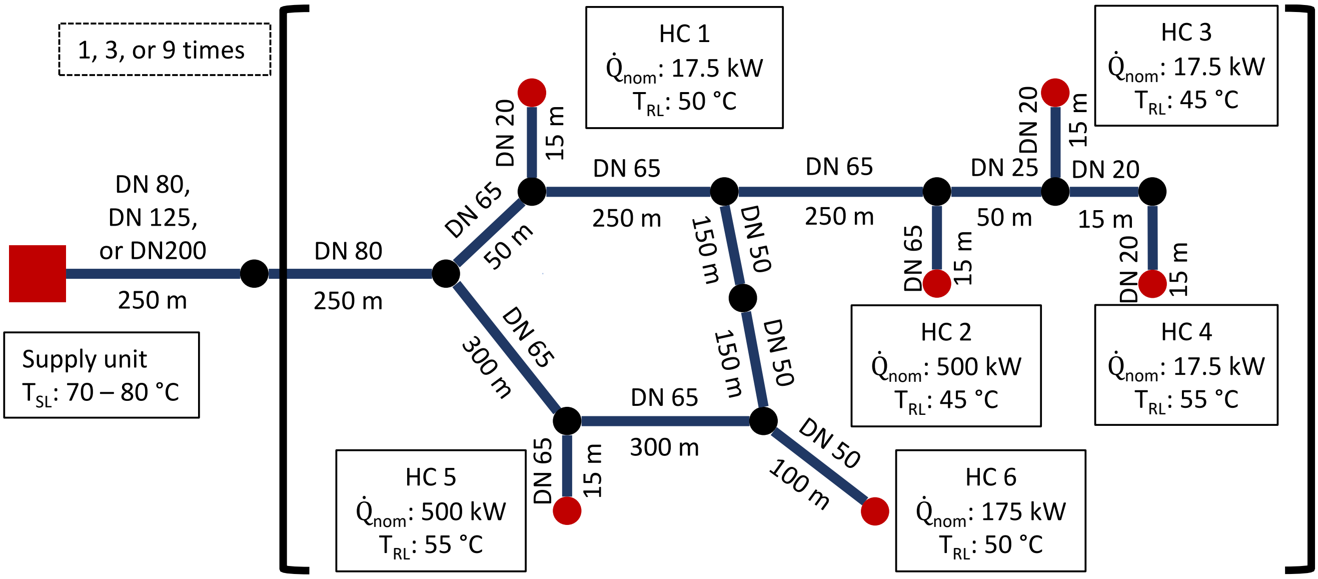


Figure 3. Layout of the demonstration network with the main parameters: Nominal heat load and constant return line temperature of HCs (HC 1 to HC 6), supply line temperature at the supply unit and nominal diameter and length for the pipe segments. Simulations were run for different network sizes where the network part in brackets exists 1, 3 or 9 times after the first pipe.

4.1 Demonstration Network

The basic demonstration network is a fictional, simple DHN with one supply unit and 6 HCs. The pipe network consists of 17 pairs of pipes (supply and return line), including house lead-in pipes, and contains one loop to introduce a certain degree of complexity (the loop results in a non-linear system of equations for the mass flows and pressures). The pipe closing the loop (DN 50) has been split into two parts to obtain a temperature value in the middle of the pipe for analysis. To analyze the effect of different network sizes, this basic layout ("small") was repeated 3 times ("medium") and 9 times ("large"), branching off after the first network pipe. The layout and the main parameters are depicted in Figure 3.

The HCs are simulated with 6 real, measured heat load profiles from an existing DHN with a resolution of 15 min. For the "medium" and "large" simulation, the profiles were reused with random variations (normal distribution, standard deviation 10 %), so that the peaks and valleys do not perfectly coincide. The heat load profiles consist of exemplary periods for high load, medium load, low load (each three days) and an undersupply situation (two days, with temporary drop of supply line temperature to 50 °C). The supply line temperature at the supply unit is set via a temperature curve between 70 - 80 °C, apart from the undersupply situation, where the actual measured supply temperatures are used. During the simulation, these values are interpolated using smooth splines with `Modelica.Blocks.Sources.CombiTimeTable`.

The heat load profiles, being real measurement data, show higher dynamics (frequent peaks and valleys) than common synthetic heat load profiles. This is most pronounced during medium and low load period at HC 5 and HC 6, which show frequent switching between zero and substantial load values. Furthermore, most of the load profiles include periods with zero load for some hours. Thus, these heat load profiles are challenging, but yet realistic, examples of heat load profiles that may be used in the simulation of DHNs.

4.2 Evaluation of Simulation Results and Performance

To check if the demonstration network is configured realistically, some general indicators are estimated from the results of the *main* simulation run (low load = winter, medium load = spring and autumn, high load = summer). For the basic small demonstration network and the *main* simulation run, the estimation yields a total annual heat demand of 3.2 GWh/a, relative heat losses of 12 % and a relative hydraulic energy for the circulation of 0.22 %. Given the route length of 2.2 km, the linear heat density is 1.3 MWh/(m a). The mass flow weighted mean temperatures at the supply unit are 72 °C in the supply line and 48 °C in the return line. These values are considered plausible for a medium sized DHN with network temperatures as low as possible while still supplying old buildings and preparation of domestic hot water.

4.2.1 Comparison of General Simulation Results

In general, the simulation results of the different HC models should be similar. In the following, the results are compared to the *main* simulation run and major differences are reported and explained.

The total heat from the supply unit does not differ more than 3 % compared to the *main* result for all models and periods, which indicates a good agreement of the models.

During the low load period, it makes a major difference whether the HC model includes a bypass. Compared to *main*, models without bypass (*noBypass*, *AixLib* and *DisHeatLib*) result in about 9 % less heat losses, because the network is not kept hot and lower return temperatures occur. Furthermore, they yield a 20 to 30 % higher maximum heat flow due to mass flow peaks after the supply line temperature had cooled down. In addition, the maximum pressure difference at the supply unit is 13 to 19 % lower, due to less mass flow in the network. Accordingly, the total hydraulic energy at supply unit is about 30 % less than with bypasses in this period.

Furthermore during the low load period *AixLibBypass* has 7 % less heat losses than *main*, as the constant bypass flows are not sufficient to keep the network hot (but also should not be tuned to the necessary value, because too much load would be omitted then).

In the undersupply period, the *AixLib* models have 13 % lower maximum pressure differences, as they assume a constant minimum temperature difference (set to 5 K in this case), while *DHNSim* models set mass flows to a maximum allowed value. Furthermore, the hydraulic energy is 20 to 30 % less without bypasses (*noBypass*, *DisHeatLib*) and 60 % less for the *AixLib* models, due to lower mass flows in both cases.

Another difference is that the maximum differential pressure at the supply unit is 16 % higher for the *AixLib* models during the high load period, due to a single, probably faulty, data point in the heat load profile of HC 4 (critical path), with a prescribed heat flow of 35 kW (although rated to 17.5 kW). The *DHNSim* models limit the mass flow to twice the nominal mass flow (parameter may be changed to other values) which limits the heat load in this case to 25 kW.

4.2.2 Effect of Mass Flow Time Constant

The comparison of the heat and mass flows at HC 6 for the simulation runs *main* and *fastDynamics* in Figure 4 demonstrates, how a heat load peak is delayed and has a reduced peak value compared to the input signal due to the added dynamics. The smaller the value of `tau_m_flow`, the more immediate is the reaction of the HC model to the input signal. Depending on the goals and available input data, the user of the model shall choose a sensible value for `tau_m_flow`. For input time series at a resolution of 15 min to 1 h a value of 180 s has proven to be suitable in previous simulation studies.

Figure 4 also shows, that `tau_m_flow` has an impact on bypass operation. After 130 h, the heat flow signal, and subsequently the mass flow, drops to zero. However, after a short zero flow period, the supply line temperature (not shown for clarity), drops below the set point of the bypass, causing it to increase the mass flow. The bypass in *fastDynamics* reacts earlier, so that the cooled house lead-in pipe gets flushed within less time than in *main*. The slower dynamics in *main* finally cause a slightly higher peak mass flow, while in *fastDynamics* the slopes of the mass flow are steeper. However both configurations maintain the required supply line temperature at the HC.

4.2.3 Bypass Behavior

Bypasses are intended to maintain a small mass flow through HCs during zero load periods so that the supply line temperature does not drop too much. Figure 5 shows the results for temperatures and mass flows at HC 6 during a period without heat load for *main*, *noBypass* and *AixLibBypass*. In *main*, the bypass starts to operate once the supply line temperature approaches 65 °C. The mass flow shows an decreasing oscillation, that is caused by the interplay of the thermostatic control approach and the delay due to the dwell time of the water in the house lead-in pipe. As long as the bypass operates and the heat load is zero, the return line temperature equals the supply line temperature. The bypass successfully maintains the required temperature of about 65 °C. Once the heat load rises (at 133.5 h), the return temperature smoothly drops.

In contrast, in *noBypass* the mass flow is zero during the period without heat load and the supply line temperature continuously drops. As a consequence, a mass flow peak occurs afterwards until the supply line temperature rises, causing steep slopes of mass flow and temperature. Nevertheless, the implementation of the HC is robust also without bypass, due to the limited dynamics and maximum value of mass flow and its reduction, if the differential pressure is too low. This prevents the HC model from imposing too high mass flows after zero flow periods, that might cause a simulation failure.

DOI
10.3384/ecp204397

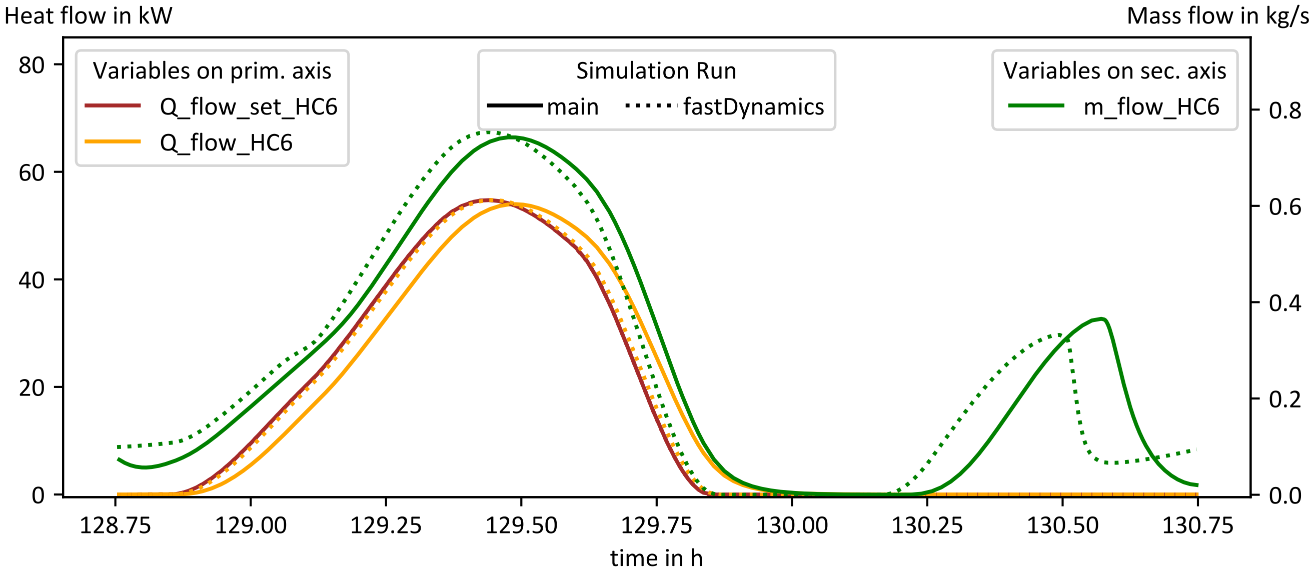


Figure 4. Example of the effect of the mass flow time constant at HC 6.

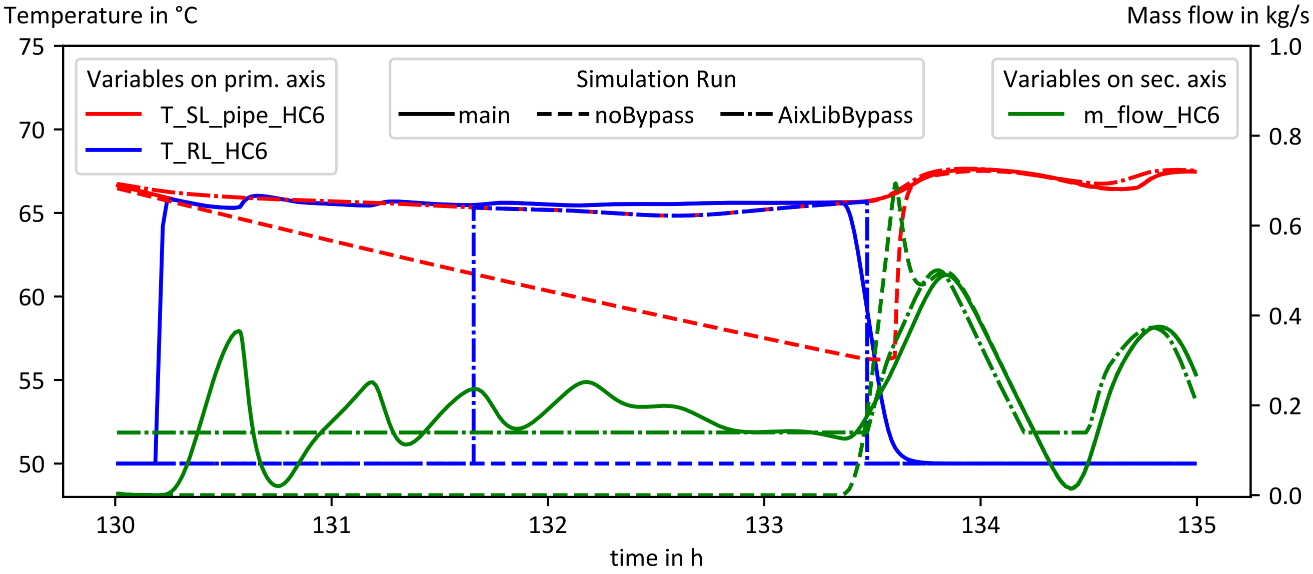


Figure 5. Demonstration of the bypass part of the consumer model.

AixLibBypass maintains a constant minimum mass flow. If tuned properly, this approach succeeds to maintain a sufficient supply line temperature. However, the bypass is active irrespective the supply line temperature, whenever the load mass flow reaches the set point, as can be seen at 134.5 h. The implementation of the return line temperature is not robust (switches at an undetermined time instant, here 131.7 h) and causes abrupt changes.

For the demonstration network, the *noBypass* implementation requires substantially less time to compute (see section 4.2.6), which indicates that the reduced effort (no state for and calculation of bypass mass flow) outweighs the computational effort to simulate the higher dynamics after zero flow periods. In the end, it is up to the user, if a bypass shall be included, depending on whether it is intended and realistic to have them.

In general, the proposed bypasses work as intended: In the *main* simulation run, only HC 6 has supply line temperatures below 64 °C in the three days low load period, in total during 1 h, affecting a heat consumption of 9 kWh. In contrast, without a bypass, at all HCs supply line temperatures below 64 °C occur, with the strongest effect at HC 6 during the low load period for 30 h and 300 kWh.

The bypass implementation of AixLib does not reduce the duration of temperature undersupply significantly for two reasons. First, bypass operation does not depend on temperature but on heat load, so that in some periods the bypass would not act, although the supply line temperature is low. Second, and more important, it was not possible to tune the bypasses of critical consumers to a value that always maintains the supply line temperature, because it was chosen to limit the maximum allowed bypass mass

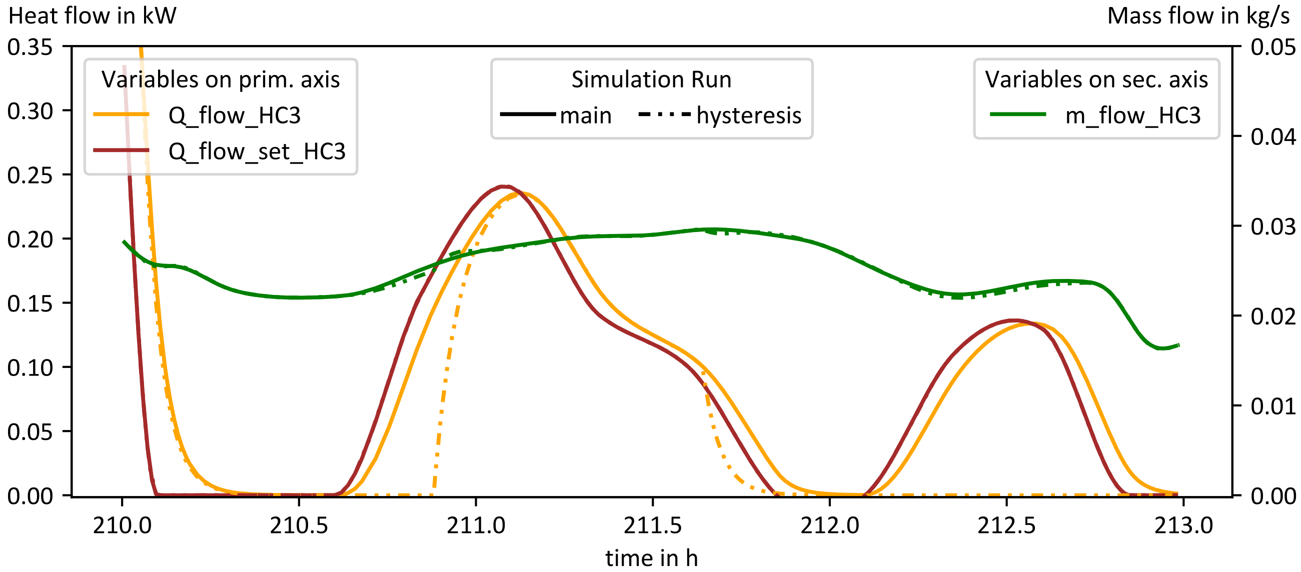


Figure 6. Demonstration of the effect of hysteresis at very small heat load peaks.

flow to 10 % of nominal mass flow, as too much heat load was omitted otherwise.

4.2.4 Load Hysteresis

The hysteresis feature explained in section 3.2 affects the behavior of the HC model when the prescribed load value is close to zero. Figure 6 shows an example for HC 3, comparing the results for heat and mass flows from *main* and *hysteresis*. In the period, two very low heat load peaks occur. The first (210.5 - 212 h) reaches values above the hysteresis thresholds. While the *main* result shows a smooth rise of heat flow following the set-point, *hysteresis* has a zero heat flow until the threshold is reached (right before 211 h), followed by a steep rise of heat flow until the required values is reached. At the falling slope, the heat flow suddenly falls to zero, once the switch-off threshold of the hysteresis is crossed (at 211.7 h). The second heat load peak (212 - 213 h) never crosses the switch-on threshold, so that it is completely ignored in *hysteresis*. The mass flows are very similar, as they are dominated by the bypass mass flow that is similar for both results.

This example shows, that the hysteresis approach may avoid the calculation of negligible heat flows. However, it imposes additional computational effort due to the events that are triggered whenever thresholds are crossed and the steep slopes that occur right after every switching.

4.2.5 Undersupply

The proposed HC model is designed to provide plausible results during undersupply situations (supply temperature and/or differential pressure too low). Figure 7 shows the simulation results in such a period for *main*, *noBypass* and *AixLib* at HC 4, which is at the end of the critical path. The supply line temperature (upper graph) falls steadily and approaches the return line temperature, and as a consequence, the proposed HC model increases the mass flow (lower graph) to reach the needed heat flow.

In *main* the first phase of undersupply starts at about 281 h when the mass flow reaches its maximum (first vertical dotted line). From this point onward, the set point heat flow is not covered.

The second phase, starting after 282 h is marked by insufficient differential pressure: Due to the enormous increase of mass flows in the network, pressure losses in the pipes rise, causing high differential pressures to be provided by the heat supply unit. At a certain point, the upper limit of differential pressure is reached so that the required minimum differential pressure at the HC (here 0.6 bar) is no longer maintained. As a consequence, the load model reduces the mass flow.

Finally, after 284 h (third dotted line) the supply line temperature even drops below the set-point return temperature, so that the heat flow is zero and the return line temperature equals the supply line temperature.

Once the supply line temperature rises substantially at 287 h, the required differential pressure is restored and the HC returns to normal operation.

DisHeatLib behaves similar to *main*, as the flow unit in the model limits the mass flow to a maximum value according to the available pressure difference. The parameterization is derived from nominal values and results in rather low maximum mass flows and more undersupply than *main*.

The *AixLib* model however deals differently with the situation: The model assumes a minimum temperature difference between flow and supply line (here 5 K), that is used whenever the supply line temperature is low. This implementation leads to smaller mass flows compared to the proposed HC model and lets the model follow the set point heat flow, so that no undersupply occurs. However, the return temperature drops to 45 °C, and might even drop further, which would be unrealistic if the secondary return temperature of the actual HC would be higher than that.

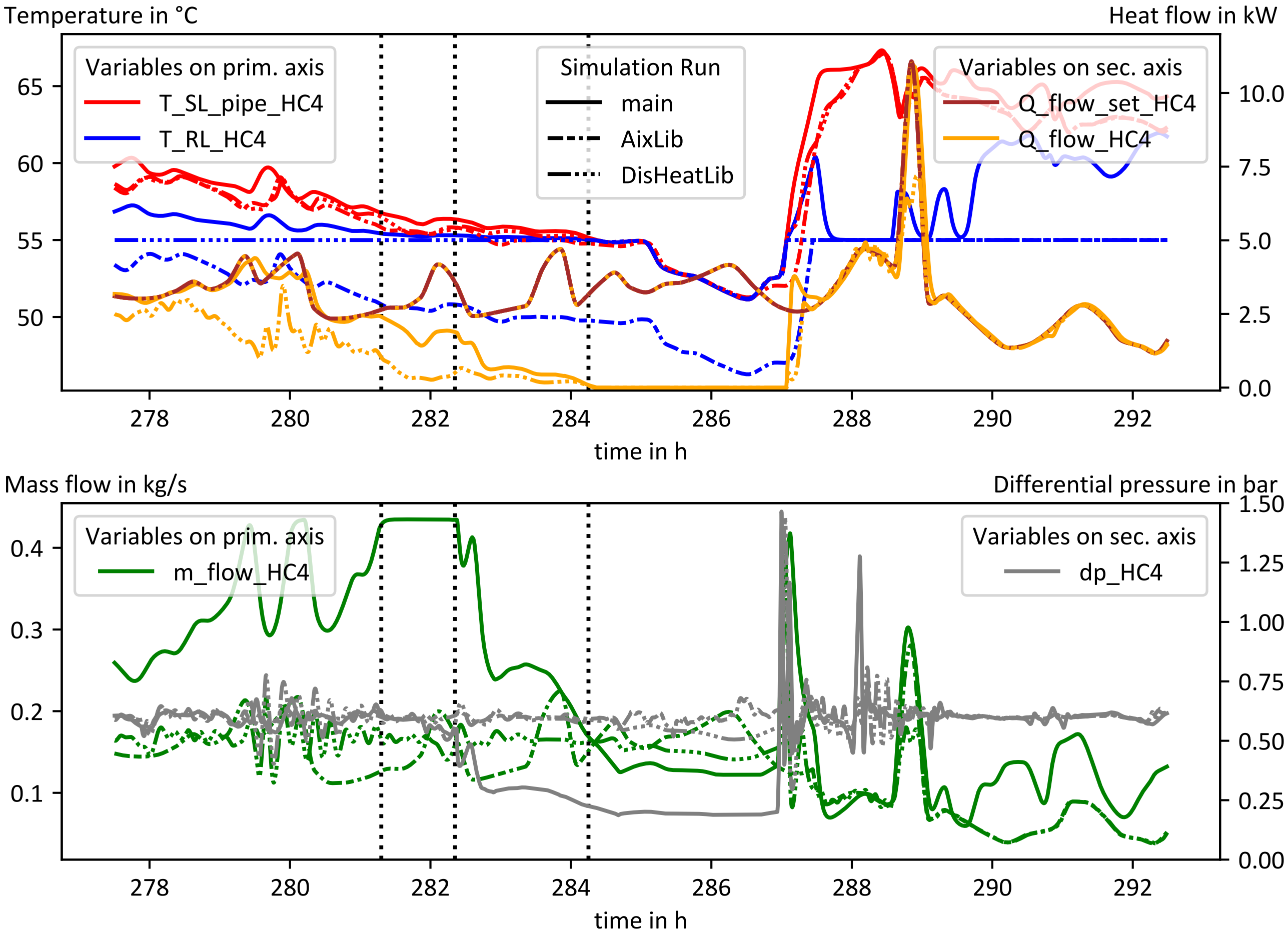


Figure 7. Demonstration of how the consumer models deal with the undersupply situation.

4.2.6 Simulation Performance

For a profound analysis of the influence of the different implementations of HC models and network sizes on simulation performance, the CPU-time for integration is evaluated (mean of 3 runs, integration algorithm *Dassl*, tolerance 1×10^{-4}, on a machine with CPU Intel i5-4300U @ 4x1.9 GHz, RAM 8 GB).

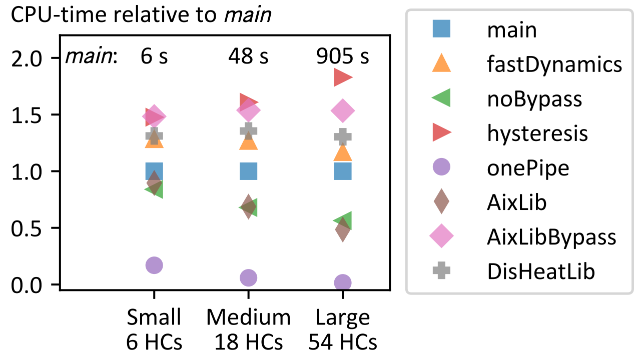


Figure 8. Comparison of CPU-time of the different HC models at different network sizes. Values are shown relative to *main*.

Figure 8 shows the CPU-time relative to the *main* model. *OnePipe* requires only a small fraction of CPU-time compared to *main*, which proves, that the HC model itself does not require much computational effort. However, the implementation of the HC model causes major variations of CPU-time for the same network, with an increasing importance for larger models (more than factor 3 for the large model). The models with open loop design and without bypass (*noBypass* and *AixLib*) have the lowest and very similar CPU-times. The proposed model *main* is the fastest with a bypass. *FastDynamics* lead to a minor increase in CPU-time. *DisHeatLib* requires 30 % and *AixLibBypass* 50 % more computational effort. The *hysteresis* implementation causes the longest CPU-times.

The absolute values (indicated in Figure 8 as well) show that CPU-time scales non-linear with model size, reaching about 900 s (*main*) for the large demonstration network. Assuming a linear dependence on simulated time, an annual simulation would take about 8 hours, which is acceptable but substantial, which stresses the importance of a careful design of the HC models.

The results also reveal, that both, the number of result points and the number of events have a direct, and almost linear impact on CPU-time: A simple linear fit of CPU-time as a function of these two quantities yields a high coefficient of determination of 0.72. Figure 9 shows both, measured and fitted CPU-times for the large model.

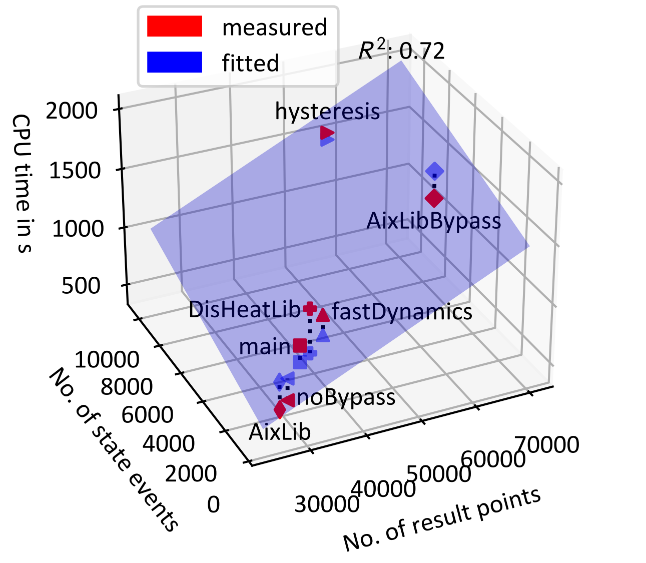


Figure 9. Dependence of CPU-time on number or state events and result points for the large model (54 HCs).

The *AixLib* implementation and the proposed HC model without bypass *noBypass* yield the fastest simulation with a low number of state events, followed by *main* (with bypasses). Increasing the dynamics of the proposed model (*fastDynamics*) increases the number of steps to be calculated and subsequently the CPU-time. *AixLibBypass* has a high CPU-time due to more events and an increase of time steps, while the *hysteresis* triggers by far most events and has thus highest CPU-times. Therefore, hysteresis should be used with care, as it may substantially increase the simulation time. *DisHeatLib* sticks out with almost 50 % higher CPU-time measured than according to the fit, because it is the only HC model that does not follow the open loop design which leads to more complex systems of equations to be solved.

5 Conclusion

The implementation of the HC model has crucial impact on the computational effort in DHN simulations. The proposed HC model yields plausible results for the whole range of load situations, including undersupply, and requires 50 % less CPU-time than the equivalent HC model from *AixLib* and 30 % less CPU-time than the most simple demand model from *DisHeatLib* (without bypass). Users of the model may choose whether their use case requires to use the thermostatic bypass that maintains the supply line temperature and if fast dynamics of the HC model are needed, as both options increase CPU-time. Load hysteresis is not recommendable for the used load profiles, as it triggers many state events, which causes a substantial increase of simulation time.

The main weakness of the proposed HC model is the assumption of a constant return temperature or temperature difference. Future work will focus on an implementation that is more realistic and reflects different operation states.

Acknowledgements

This work has been supported by the German Federal Ministry for Economic Affairs and Energy (SINTEG project Csells, grant number 03SIN119).

References

AIT-IES (2022). *DisHeatLib*. https://github.com/AIT-IES/DisHeatLib. (Visited on 2023-06-05).

Fritzson, Peter (2015). *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Piscataway NJ: IEEE Press.

International Building Performance Simulation Association (2018). *Modelica IBPSA Library*. https://github.com/ibpsa/modelica-ibpsa.

Lawrence Berkeley National Laboratory (2023). *User Guide of the Modelica Buildings Library*. URL: https://simulationresearch.lbl.gov/modelica/userGuide/ (visited on 2023-06-07).

Leitner, Benedikt et al. (2019). "A method for technical assessment of power-to-heat use cases to couple local district heating and electrical distribution grids". In: *Energy* 182, pp. 729–738. ISSN: 03605442. DOI: 10.1016/j.energy.2019.06.016.

Lund, Henrik et al. (2014). "4th Generation District Heating (4GDH): Integrating smart thermal grids into future sustainable energy systems". In: *Energy* 68, pp. 1–11. ISSN: 03605442. DOI: 10.1016/j.energy.2014.02.089.

Maier, Laura et al. (2022). "AixLib: An open-source Modelica library for compound building energy systems from component to district level with automated quality management: Preprint submitted November 17, 2022". In: *Journal of Building Performance Simulation*. ISSN: 1940-1493. URL: https://www.researchgate.net/publication/365475776_AixLib_An_open-source_Modelica_library_for_compound_building_energy_systems_from_component_to_district_level_with_automated_quality_management (visited on 2023-06-05).

Modelica Association (2019). *Modelica Standard Library*. Linköping. URL: www.Modelica.org.

Müller, D. et al. (2016). "AixLib - an Open-Source Modelica Library within the IEA-EBC Annex 60 Framework". In: *Conference Proceedings of Central European Symposium on Building Physics CESBP*. URL: http://www.ibpsa.org/proceedings/bausimPapers/2016/A-02-3.pdf (visited on 2023-06-05).

RWTH-EBC (2021). *AixLib*. https://github.com/RWTH-EBC/AixLib. (Visited on 2023-06-05).

Stock, Jan et al. (2023). "Modelling of waste heat integration into an existing district heating network operating at different supply temperatures". In: *Smart Energy* 10.5, p. 100104. ISSN: 26669552. DOI: 10.1016/j.segy.2023.100104.

van der Heijde, B. et al. (2017). "Dynamic equation-based thermo-hydraulic pipe model for district heating and cooling systems". In: *Energy Conversion and Management* 151, pp. 158–169. ISSN: 01968904. DOI: 10.1016/j.enconman.2017.08.072. (Visited on 2018-01-04).

Zipplies, J., J. Orozaliev, and K. Vajen (2023, in press). "Development of Models for Long-Term Simulations of District Heating Networks at High Temporal and Spatial Resolutions". In: *Proceedings of EuroSun 2022*. Ed. by Klaus Vajen. Freiburg, Germany: International Solar Energy Society.

DOI
10.3384/ecp204397

Convection of Chemicals and Other Substances with ThermoSysPro

Giorgio Simonini   Arnaud Duval   Sarah Hocine-Rastic   Mathilde Praud

EDF Lab Chatou, France,
{giorgio.simonini, arnaud.duval, sarah.hocine-rastic, mathilde.praud}@edf.fr

Abstract

Digital twins are a powerful support tool for plant operation: they provide further understanding on ongoing phenomena and allow realistic projection of the current plant state into the future. Among other twins, EDF is developing a digital twin of the chemistry of the secondary circuits of its nuclear plants. Such a tool will give access to the *pH* in any point of the circuit and in any operating condition (e.g. partial load, power transients...), outperforming the current, limited, monitoring techniques. It is expected to help operators and engineers to better monitor the circuit (e.g. for erosion corrosion) and anticipate the consequences on equipment of different operating strategies (e.g. for amines' injection pumps maintenance).

ThermoSysPro, the EDF R&D's thermal-hydraulic library, is the bedrock of the tool under development. To meet the needs of the target application, modeling of amines convection and some related chemistry, allowing the computation of *pH*, are introduced in a new version of the library. Moreover, the presented approach aims at proposing a general framework allowing the convection of custom substances (i.e. easily customized by the end user following its needs). This will open the door for a wide range of other applications: radioactive substances, pollution (e.g. salted water ingress coming from a heat-exchanger leak), just to cite a few, could be modeled in ThermoSysPro to augment the scope of the digital twins.

Keywords: Digital Twin, Secondary Chemistry, Substance Convection, Amines, ThermoSysPro

1 Introduction

Nowadays, a increasing number of Digital Twins, virtual copies of industrial units, are developed all over the world and in every industries including nuclear power plants. Physical and statistical models, fed by plant data in real time, allow, for example:

- to investigate otherwise inaccessible phenomena, in particular where no (or too few) measures are available;
- to diagnose safety or performance issues;
- to project the future plant state for decision making support.

In EDF R&D, our team and its partners have worked for several years on digital twins for thermal-hydraulic applications (see for example (Girard 2014; Corona Mesa-Moles et al. 2019; Schwartz 2023; Gerrer and Girard 2020)), mainly supported by our Modelica library: ThermoSysPro (El Hefni and Bouskela 2019). The application presented in this paper however, goes beyond the strict thermal-hydraulic domain and concerns secondary circuit chemistry of French nuclear power plants.

1.1 Industrial Application

The main objective of secondary chemistry is to limit various types of corrosion. By doing so, it ensures the integrity of the equipment (and boosts their lifetime) and reduces the corrosion product source term responsible for fouling - which leads to plant performances limitation - and tube support plates clogging of steam generators - which may lead to tube instability and vibration and its rupture; both these phenomena require expensive treatments or component replacements. Secondary chemistry mainly consists in implementing appropriate chemical conditioning and pollution monitoring. It is a compromise between safety, performance, environmental releases and wastes, and operating and maintenance costs.

To identify the best compromise, it is necessary to have an online, i.e. at different loads and during transients, view of the chemistry at each point of the circuit. However, currently, this is not possible by only using available measures, due to very limited measuring points or a too low sampling frequency.

1.2 Needs for a ThermoSysPro Evolution

To go beyond the current limitations, the idea is to develop a digital twin of the secondary chemistry, combining the few (both in space and in time) available measures with thermal-hydraulic and *simple* chemistry modeling. *Simple* since for our application the main phenomena of interest are:

- The convection, in the several components of the secondary circuit, of the conditioning amines, which are the chemicals used to control the *pH*.
- The evaluation of the *pH* (at the desired temperature, i.e. the fluid or the ambient one).

DOI
10.3384/ecp204407

Such a digital twin will provide nuclear power plant chemical engineers with a complete view of the secondary circuit chemistry at any point and continuously. It will also allow to project the impact of a specific manoeuvre on the secondary chemistry to optimize the operation of the plant. This paper presents the recent developments of the ThermoSysPro library to make it able to deal with such phenomena. These developments are enough general to open the door for other application combining thermal-hydraulics and convection of whatever substances of interest (not just conditioning amines). The modeling approach will be detailed in chapter 2, while chapter 3 will present some simple application examples. The conclusions and perspectives will be discussed in chapter 4.

2 Modeling Approach

2.1 Main Modeled Phenomena

The present application concerns the convection of a few amines (bases) in a water biphasic circuit. However, it is quite easy and convenient to generalize the problem so that any kind and any number of substances can be convected in such a circuit. Specifically, the mass balance equation has to be respected for any substance *i*:

$$\frac{dMi}{dt} = Qi_{IN} - Qi_{OUT} \tag{1}$$

where Mi is the mass of the substance i in a control volume and Qi_{IN} (respectively Qi_{OUT}) the inlet (outlet) mass flow rate for that substance at the boundary of the volume.

Equation 1 supposes that there is no mass source S_{IN} or mass sink S_{OUT} inside the control volume. This is a normal assumption for thermal-hydraulic modeling where the fluid is considered as a unique media; however, this may not be the case when substances are taken individually: chemicals may react and *mutate*, i.e. a substance may disappear and another one appear; the same happens for radioactive substances that naturally disintegrate. Amines are chemicals and sources/sinks may exists in the circuit, depending on the nature of the amine and on the thermodynamic conditions. However, as a first modeling approach, their effect is neglected.

Assumption 1 *Inner sources and sinks of convected substances are out of the scope of the current work.*

Knowing that the global mass balance is already resolved, it is also possible to use the mass concentration of each substances $Ci = \frac{Mi}{M}$,where M is the total mass in the volume, instead of its mass Mi. The equation 1 becomes:

$$\frac{d(M*Ci)}{dt} = Ci_{IN} * Q_{IN} - Ci * Q_{OUT}$$

$$V * \rho * \frac{Ci}{dt} + Ci * \frac{dM}{dt} = Ci_{IN} * Q_{IN} - Ci * Q_{OUT}$$

$$V * \rho * \frac{Ci}{dt} + Ci * (Q_{IN} - Q_{OUT}) = Ci_{IN} * Q_{IN} - Ci * Q_{OUT} \tag{2}$$

where V is the constant volume of the control volume, ρ the fluid density, Q_{IN} and Q_{OUT} the inlet and outlet total mass flow and Ci_{IN} the mass concentration of the substance i in the inlet flow.

The equation 2 can easily be adapted in case of multiple inlets/outlets:

$$V * \rho * \frac{Ci}{dt} + Ci * (\sum Q_{IN_k} - \sum Q_{OUT_k}) = \sum(Ci_{IN_k} * Q_{IN_k}) - \sum(Ci * Q_{OUT_k}) \tag{3}$$

with obvious notation.

Equation 3 make the hypothesis that the concentration Ci is the same for all the outputs. This is generally the case when the convected substance is homogeneously dissolved in the fluid. While this is a quite common assumption for monophasic conditions, it is far from true in a component where liquid and gas phases are split. A steam dryer or a steam generator with blowdown are typical examples of component where this assumption is refuted.

In this case Ci should be split in Ci_{GAZ} and Ci_{LIQ} which are directed to the corresponding outlet(s). Additional equation should then be added to evaluate how the Ci entering a control volume is split in gaz and liquid phase. This depends on the nature of the substances. For amines, the equilibrium between gas and liquid concentrations is mainly controlled by:

- A *distribution coefficient* defined as the ratio of the concentrations (in molality, mol/kg) of the species in the vapor phase to the undissociated species in the aqueous phase.
- An *association constant* governing the equilibrium, in the aqueous phase, of the undissociated and the dissociated species.

Besides amines, some *generic* substances can be conceived, such as:

- A *homogeneous* substance who make no distinction between gas and liquid phases. In this case $Ci = Ci_{GAS} = Ci_{LIQ}$.
- A *non-volatile* substance who cannot pass to the gas phase. In this case $Ci_{GAS} = 0$ and $Ci = Ci_{LIQ} * (1 - x)$, where x is the vapor quality in the control volume. [1]

[1] It has to be noticed that, with this definition, the liquid concentration tends to infinite when the vapor quality tend to 1. It corresponds to the expected behaviour since the substance cannot migrate to the gas phase and accumulate in the liquid one; however, such a substance would probably crystallize, so *leaving* the liquid phase, above some threshold of liquid concentration; to take into account such a phenomena, a sink should be added to the model (out of the current scope, see Assumption 1).

These generic substances can be used to model the convection of several substances with similar behaviour (for example the 2.1 could be used to model salt in water). These generic substances will be used in chapter 3 for illustration purpose.

Within the framework of this work, the concentration of the transported substances is supposed to be low, of some *ppm* or even *ppb*. Such values does not impact the thermal-hydraulic properties of the fluid, such as the density or the viscosity.

Assumption 2 *The concentration of the transported substances does not impact the thermal-hydraulic properties of the fluid.*

Once the concentration of the substances (amines in this specific case) are defined, the *pH* can be computed. *pH* is the *potential of hydrogen* and it is defined as follows:

$$pH = -log_{10}([H^{+}]) \tag{4}$$

$[H^{+}]$ being the equilibrium molar concentration in mol/L of hydrogen ions in the solution.

Equation 4 could be rewritten as a function of $[OH^{-}]$ (hydroxide ions concentration), to which the concentration of hydrogen ions is related via the self-ionization constant of water, Kw:

$$pOH + pH = -log_{10}(Kw) \tag{5}$$

It is worth noticing that the value of Kw depends on water temperature and pressure (see for example Marshall and Franck (1981)).

The OH^{-} formulation is preferred since the hydroxide ions concentration also appears in the equations governing the biphasic equilibrium (see above).

2.2 Code Implementation: the `ConvectedQuantities` Package

One of the main objectives of this development is to ease the reuse of the substances' convection models: in the framework of the target application, 4 chemical species have to be convected (to compute the *pH*); more generally, *None* or other sets of substances may have to be convected in the future. For this reason, the proposed approach consists in developing two subpackages, `Components` and `Substances`.

2.2.1 The `Components` sub-Package

The `Components` package is dedicated to the modeling of the convection of the substances, following the equation 3 (module `MassBalance`) or its declination for heterogeneous liquid/gaseous outlet (module `MassBalance_HeterogeneousPhases`).

The equations in these modules are *vectorized* so that they can deal with any number of substances (general approach).

2.2.2 The `Substances` sub-Package

Substances have then to be *vectorised* too. The concentrations of a set of substances is store as an enumeration as suggested by Tiller (2023) for chemicals. The other information to be provided for each set is how the concentrations of each substance distribute in biphasic fluid. Here follow the example code for *None*, *Homogeneous* and *NonVolatile* substances:

Listing 1. *None* definition

```
  package None
    replaceable type Concentrations =
        enumeration(:);

    replaceable block PhasesSeparation
      import ThermoSysPro.Units.SI;

      input SI.Temperature T "Fluid
          Temperature";
      input SI.Density rho_liquidPhase "
          Fluid Density";
      input Real x "Title";
      input Real SubC[None.Concentrations]
          "Total Species Concentrations";

      output Real Cl[None.Concentrations] "
          Species Concentration in the
          Liquid Phase";
      output Real Cg[None.Concentrations] "
          Species Concentration in the Gas
          Phase";

    equation
      Cl = fill(0,size(Cl,1));
      Cg = fill(0,size(Cl,1));

    end PhasesSeparation;
```

Listing 2. *Homogeneous* definition

```
package Homogeneous "Substance with
    homogeneous distribution in gas and
    liquid phases"
  extends None(
        redeclare type Concentrations =
            enumeration(substance "Gas/
            Liquid homogeneous substance"),
        redeclare block PhasesSeparation =
            PhasesSeparation_internal);

  block PhasesSeparation_internal

    import      ThermoSysPro.Units.SI;

    input SI.Temperature T "Fluid
        Temperature";
    input SI.Density rho_liquidPhase "Fluid
         Density";
    input Real x "Title";
    input Real SubC[HomogeneousSubstance.
        Concentrations] "Total Species
        Concentrations";

    output Real Cl[HomogeneousSubstance.
        Concentrations] "Species
        Concentration in the Liquid Phase";
    output Real Cg[HomogeneousSubstance.
        Concentrations] "Species
        Concentration in the Gas Phase";

  equation
    Cl = SubC;
    Cl = Cg;

end Homogeneous;
```

Listing 3. *NonVolatile* definition

```
package NonVolatile "Substance tending to
    remain in liquid phase"
  extends None(
    redeclare type Concentrations =
        enumeration(substance "Gas/Liquid
        homogeneous substance"),
    redeclare block PhasesSeparation =
        PhasesSeparation_internal
  );

  block PhasesSeparation_internal

    import  ThermoSysPro.Units.SI;

    input SI.Temperature T "Fluid
        Temperature";
    input SI.Density rho_liquidPhase "Fluid
         Density";
    input Real x "Title";
    input Real SubC[NonVolatile.
        Concentrations] "Total Species
        Concentrations";

    output Real Cl[NonVolatile.
        Concentrations]
      "Species Concentration in the Liquid
          Phase";
    output Real Cg[NonVolatile.
        Concentrations]
      "Species Concentration in the Gas
          Phase";

  equation
    Cl * (1-x) = SubC;
    Cg = fill(0,size(Cl,1));

  end PhasesSeparation_internal;

end NonVolatile;
```

As with *Homogeneous* and *NonVolatile* sets (of only one substance), any other set of substances can be defined by extending *None* and redeclaring the `Concentration` enumeration and `PhasesSeparation` according to the characteristic of the that specific set.

2.3 Application to `WaterSteam` or `Fluid` Packages of ThermoSysPro

The code presented in the previous paragraph deals with the convection of generic substances, but still have to be "connected" with existing ThermoSysPro modules, which deal with "everything else" (i.e. fluid mass balance, energy balance, fluid properties...). Since the target application concern chemical water conditioning, the connection have been done to the ThermoSysPro packages[2] `WaterSteam` and `Fluid`. However, the following fundamentals for linking `ConvectedQuantities` apply for other fluids/-packages.

The general required modifications are listed hereafter:

- The *replaceable* definition of the convected species (see Listing 4) is added. *None* is used as default so that nothing is convected if useless (no trailing equations/variables *polluting* or increasing the size of the system of equations). The `size` of `Concentration` is 0 in this case.

- Fluid connectors are replaced with new connectors which contains also a new `SubC` variable which is defined to host the convected `Species.Concentration`.

Listing 4. Definition of Convected *Species*

```
// In the declaration part
replaceable package Species = ThermoSysPro.
    ConvectedQuantities.Substances.None ;
```

Subsequent modifications depend on the nature of module: *biports* (Singular and Pipe Pressure Losses, Stodola Turbines, Control Valves, Pumps...) or *volumes* (or junctions).

2.3.1 Biports

In this components, the convected substances concentration is transmitted from the inlet to the outlet: inlet and outlet concentration have to be equalized.

Listing 5. Connection of input and output concentration in a *biport*

```
// In the equation part
C1.subC = C2.subC ;
```

2.3.2 Volumes

These components require the use of the additional modules developed in the `ConvectedQuantities` package. In particular the `MassBalance[...]` module (with specific binding equations) has to be used to calculate the distribution of the substances between the different outputs and to model their dynamic behaviour. An example is given in Listing 6.

Listing 6. Connection of `MassBalance` to a *volume*

[2] The ThermoSysPro library is composed by several packages, needed to model any type of power plant; for example: `InstrumentationAndControl`, `Combustion`, `ElectroMechanics` and some packages dedicated to different fluids (e.g. `WaterSteam`); since the version 4.0 of ThermoSysPro, the `Fluid` package merges them all (e.g. oils, flues gases, refrigerants...).

DOI
10.3384/ecp204407

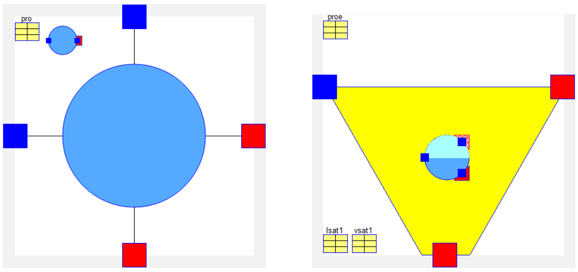


Figure 1. Modified Diagrams of *VolumeA* (left) and *SteamDryer* (right). The icons corresponding to the used `MassBalance[...]` components are visible.

```
ThermoSysPro.ConvectedQuantities.
    Components.MassBalance
    sub_massBalance(
 redeclare package Species = Species,
 n_in=2, n_out=2, #Number of inlets and
     of outlets
 dynamic_mass_balance=
     dynamic_mass_balance,
 V=V,
 Qin = {Ce1.Q,Ce2.Q}, #Binding the inlet
     mass flow rates
 Qout = {Cs1.Q,Cs2.Q}, #Binding the inlet
      mass flow rates
 rho = rho)
```

3 Examples

3.1 Homogeneous Substances

The first simple example concerns the mixing of two water flows with different concentrations of a single generic substance. Figure 3 shows the ThermoSysPro model, composed by:

- a main water line (the horizontal one at the bottom of the diagram, "Source A") with null concentration at the beginning of the simulation;
- a secondary line (on the top, "Source B") that injects water with 100 *ppm* of such a substance when a control valve is opened (from 5 to 11 *seconds*).

Figure 2 shows the resulting concentration of the water reaching the sink: as expected, once the valve is opened (@ 6 *seconds*) the concentration increases.

The increase does not end at 11 *seconds* (when the valve is fully opened) because of the *inertia* due to the mixing volume where a perfect mixing is assumed. For reproduction purpose: the volume is of 100 m^3, to be compared to a mass flow rate of 5000 kg/s on the main line and 2716 kg/s on the secondary line.

3.2 Amines and pH

The second example is related to the target application: the convection of amines and the evaluation of *pH* in a secondary circuit. To focus on the heterogeneous behaviour of amines in biphasic conditions, in this example a small portion of a classical secondary circuit is modeled (cf. Figure 4) : the steam dryer following a high pressure turbine (not in the model's scope). The dryer module separates liquid flow (going down in the diagram), which feeds the preheaters (out of the scope of the example), from gaseous flow (going right in the diagram), which is reheated before feeding the low pressure turbine. It is then a relevant example to see test the `MassBalance_HeterogeneousPhases` module.

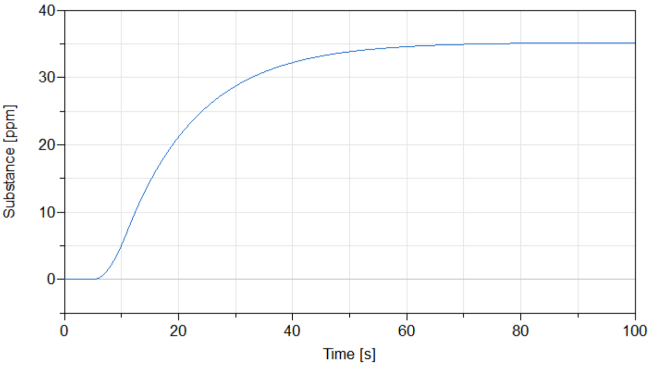


Figure 2. Evolution of the concentration of a generic convected substance: effect of the *inertia* due to a mixing volume.

In this application, the `Concentration` array is of length 4, since 4 different amines are used:

- Ethanolamine
- Morpholine
- Ammonia
- Hydrazine

These amines are characterized by a very different behaviour in diphasic conditions: Ammonia concentrates in the gaseous phase, while Hydrazine and Ethanolamine concentrates in the liquid phase; as to Morpholine, it tends to slightly concentrates in diphasic conditions.

In the example, arbitrary but reasonable values are set for the concentration of the amines upstream and see how they split in the dryer: the results are illustrated by Table 1.

[ppm]	*upstream*	*liquid*	*gas*	*liq/gas*
Ethanolamine	4	11.3	0.891	12.6
Morpholine	8	11.3	6.58	1.72
Ammonia S.	3	0.398	4.12	0.097
Hydrazine	0.2	0.609	0.025	24.5

Table 1. Concentrations of amines *upstream* of the dryer and in its *liquid* or *gaseous* outlets.

Results show that Ethanolamine is almost 13 times more concentrated in the liquid phase than in the gaseous one; this ratio reaches 25 for Hydrazine. For Morpholine this effect (greater concentration in the liquid phase)

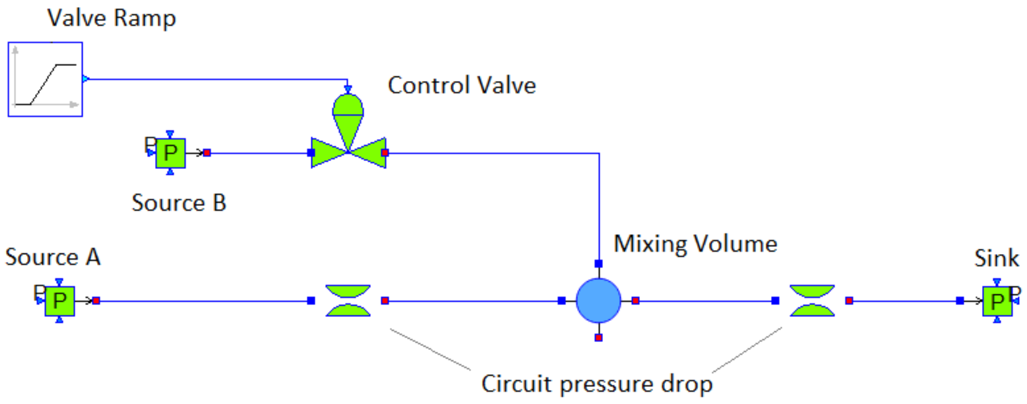


Figure 3. Simple Example for homogeneous substances (Volume)

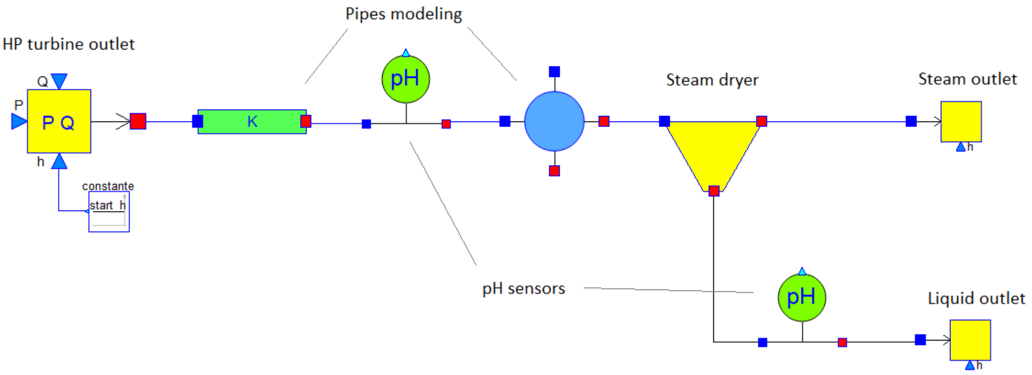


Figure 4. Simple Example for Amines (Steam Dryer)

is far lower while for Ammonia the opposite is true: this amine concentrates itself in the gaseous phase (about 10 times more than in the liquid phase). These results are consistent with the experts' expectation (historical measurements, cf. § 4) and some *back-of-the-envelope* verification (hand-calculations in simplified configurations).

Concerning *pH*, two *sensors* are included in the model, as shown in Figure 4. *pH* can be measured at fluid temperature or at ambient temperature (in the latter case the diphasic fluid is condensed). The results are displayed in Table 2.

pH	*upstream*	*liquid*
@ Fluid T	7.71	7.71
@ Ambient T	9.86	9.89

Table 2. *pH upstream* and in the *liquid* outlet. *pH* is provided at the fluid or at ambient Temperatures.

At fluid temperature, the *pH* upstream and in the liquid line are the same. This corresponds to the expected behaviour since the *pH* calculation are mainly based on the amines' concentrations in the liquid phase, which are the same in this two measurement points.

When the fluid is brought to ambient conditions (pressure and temperature), the *pH* is significantly higher. In this case, the amines' concentrations are different in the two measurement points: while in the *liquid* one there only is liquid, in the *upstream* one gas is also present; it condensate in the cooling process and mixes with the previous liquid phase. As a result, the *pH* values are different.

Some more quantitative and extensive tests are ongoing (see the perspectives in the § 4) and the preliminary results confirm the consistency of the computed values.

4 Conclusions and Perspectives

In this paper are illustrated the fundamentals of an approach to implement the convection of substances in the thermal-hydraulic library ThermoSysPro. The target application for these recent developments is building a dynamic digital twin for our power plant, with a particular focus on the secondary chemistry. One of the main interests of using ThermoSysPro as the base thermal-hydraulic layer is the opportunity of being able to reuse already developed thermal-hydraulic models. Thanks to the Modelica language it was possible to develop a generic approach, with some basic substances *hard-coded* in ThermoSysPro and easy customization for the end user: amines in the target application; but the user can easily develop specific set of substances (any number of simultaneous substances) by defining specific biphasic equilibrium. The developed modules allow for taking into account dynamic evolution (dilution/concentration) of substances concentration and heterogeneous biphasic behaviour.

The next steps includes the continuation of the verification and validation phase: in particular, the model output will be compared to experimental data (*pH* values and amines' concentrations) coming from several EDF nuclear power plants. In fact, some experimental campaigns have been realized in the last decades and provide a pertinent database for the validation of our models (data is not public). The comparison between computed and measured values will provide an estimation of the accuracy of the digital twin.

Moreover, the modifications listed in paragraph 2.3 still need to be applied to all the `Fluid` and/or `WaterSteam` packages of ThermoSysPro. The developements have been realized on Dymola (*Dymola* 2023); tests on other softwares, OpenModelica in particular (Fritzson et al. 2020), have still to be performed. All this work will hopefully lead to a new major ThermoSysPro version which will be made available in the GitHub repository of ThermoSysPro (*ThermoSysPro* 2023). Currently, the v4.0, which does not include the features presented in this paper, is publicly available.

Then, further studies will be dedicated to overcome some current limitations:

- ThermoSysPro deals with flow reversal without using the *stream* concept (El Hefni and Bouskela 2019). The ThermoSysPro's way of dealing with flow reversal still has to be applied to substance convection.
- As per Assumption 1, current developments do not take into account the possible creation or destruction of convected substances. However, such phenomena may be of interest, for example for chemical species who would react and mutates in other species: it is the case of some amines in particular conditions. Radioactive disintegration is another relevant application.

Finally, a comparison with other solutions devoted to substance convection, such as the Modelica Standard Library, should be performed to identify potential improvements for our developments.

Acknowledgements

The authors would like to thank Daniel Bouskela for his last teachings and for his legacy.

References

Corona Mesa-Moles, Luis et al. (2019-02-01). "Robust Calibration of Complex ThermosysPro Models using Data Assimilation Techniques: Application on the Secondary System of a Pressurized Water Reactor". In: 13th International Modelica Conference, pp. 553–560. URL: https://ep.liu.se/en/conference-article.aspx?series=ecp&issue=157&Article_No=56 (visited on 2023-06-08).

Dymola (2023). URL: https://www.3ds.com/fr/produits-et-services/catia/produits/dymola/avantages-cles/ (visited on 2023-06-09).

El Hefni, Baligh and Daniel Bouskela (2019). *Modeling and Simulation of Thermal Power Plants with ThermoSysPro: A Theoretical Introduction and a Practical Guide*. Cham: Springer International Publishing. ISBN: 978-3-030-05104-4 978-3-030-05105-1. DOI: 10.1007/978-3-030-05105-1. URL: http://link.springer.com/10.1007/978-3-030-05105-1 (visited on 2023-05-23).

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Gerrer, Claire-Eleutheriane and Sylvain Girard (2020). "Health monitoring using statistical learning and digital twins". In: NAFEMS 2020.

Girard, Sylvain (2014). *Physical and Statistical Models for Steam Generator Clogging Diagnosis*. SpringerBriefs in Applied Sciences and Technology. Cham: Springer International Publishing. ISBN: 978-3-319-09320-8 978-3-319-09321-5. DOI: 10.1007/978-3-319-09321-5. URL: https://link.springer.com/10.1007/978-3-319-09321-5 (visited on 2023-06-08).

Marshall, William L. and E. U. Franck (1981-04). "Ion product of water substance, 0–1000 °C, 1–10,000 bars New International Formulation and its background". In: *Journal of Physical and Chemical Reference Data* 10.2, pp. 295–304. ISSN: 0047-2689, 1529-7845. DOI: 10.1063/1.555643. URL: https://pubs.aip.org/aip/jpr/article/10/2/295-304/240967 (visited on 2023-06-09).

Schwartz, Aurélien (2023). *Metroscope*. URL: https://www.metroscope.tech/ (visited on 2023-06-08).

ThermoSysPro (2023). GitHub. URL: https://github.com/ThermoSysPro (visited on 2023-08-09).

Tiller, Michael M. (2023). *Modelica by Example*. Modelica by Example. URL: https://mbe.modelica.university/ (visited on 2023-06-08).

Using the DLR Thermofluid Stream Library for Thermal Management of Fuel Cell Systems in Aviation

Niels Weber[1] Camiel Cartignij[2] Dirk Zimmer[1]

[1]German Aerospace Center (DLR), Germany {niels.weber, dirk.zimmer}@dlr.de
[2]Eindhoven University of Technology, Netherlands c.j.g.cartignij@student.tue.nl

Abstract

For more environmental friendly aircraft, different propulsion systems are considered. Either fuel cell or fully electrically driven aircraft come along with challenging heat dissipation tasks. An intelligent thermal management system is essential to prevent failures and to ensure a reliable operation of the propulsion system. The exploration space for appropriate cooling systems seems endless, hence it is vital to rely on robust modeling libraries that enable a quick design and simulation of different architectures. The open source DLR Thermofluid Stream Library (TFS) forms such a basis and proved to be expedient in that sense. This paper gives an overview of a complete fuel cell system for future aircraft that covers the most essential subsystems and is modeled solely of components contained in the TFS. The focus is on different cooling systems and methods that can be quickly investigated in the context of the overall fuel cell system throughout an entire flight mission.

Keywords: Thermofluids, thermal management, robust modeling, fuel cell systems, open source software, prototypical control

1 Introduction

In order to reduce greenhouse emissions, intensive research is currently being carried out into new types of propulsion systems for aircraft. Using fuel cells to provide the required energy to power the propellers of an aircraft is one concept among others. Different architectures were investigated for example in the DLR internal EXACT project (*EXACT – Conceptual study for future climate-neutral flight* n.d.) and one possible design of a liquid hydrogen (LH2) driven fuel cell aircraft is shown in Figure 1.

The idea is to integrate the fuel cell directly into the nacelle that is mounted on the wing, with multiple nacelles forming the total propulsion system of the aircraft. A sketch of a possible nacelle-integrated setup is shown in Figure 2. As the total required power is in the order of several Megawatts and with an efficiency of modern proton exchange membrane (PEM) fuel cell stacks of approximately 50 %, a huge amount of heat is generated during the process. PEM fuel cell stacks typically operate between 60 °C and 90 °C. Low grade heat in such temperature regions is challenging to remove due to the small temperature difference to the ambient in comparison to conventional combustion engines. Throughout a flight profile, the ambient temperature can vary from 40 °C at take-off to −50 °C in cruise conditions. This variation results in a substantial increase in the temperature difference between the fuel cell and the ambient air during a single mission. In order to dissipate the heat at take-off conditions, a large, heavy heat exchanger is required. Contrary, in cruise conditions significantly smaller components would be sufficient and most of the cooling system becomes unnecessary dead weight. Therefore it is beneficial to investigate advanced and novel cooling system concepts to minimize the impact of the cooling system on fuel consumption.

Figure 1. LH2 driven Fuel Cell Aircraft from EXACT project.

In addition to the fuel cell, there are other heat sources that need to be cooled like the motor and the converter of the power train or a battery. Additionally to cooling, the fuel cell has to be supplied with a sufficient oxygen and hydrogen flow. For optimal operation, the conditions of the supply flows have to match very precise requirements regarding temperature, pressure and humidity. All the mentioned subsystems need some sort of cooling that can be realized in many different ways. The mechanisms vary from simple liquid cooling to direct evaporation in the fuel cell and vapor-compression systems. A dedicated number of ram air channels is used as the main heat sink for each cooling task. Using alternative heat sinks like the LH2 supply lines are also under consideration to be included in the overall thermal management.

This paper will give an overview on the capabilities of the Thermofluid Stream Library (TFS) (Zimmer, Weber, and Meißner 2021) to model the main domains of a fuel cell system and its subsystems. The strength of the library is the avoidance of large non-linear equation systems when setting up thermal networks, due to a new approach on modeling thermofluid streams. It enables an easy set up and initialization of the system models and leads to high robustness and performance. The library and its modeling approach will not be further presented in this paper, detailed explanations can be found in (Zimmer, Bender, and Pollok 2018) and (Zimmer 2019). After introducing the overall system model, the focus will be on different cooling systems and methods. Common interfaces enable a quick and easy configuration of different cooling systems that can be tested in a mission simulation in the environment of the overall system. Nevertheless, the prototypical control design that is necessary to guide the systems through the fast changing operating points of a flight mission are only briefly discussed in this paper. As a conclusion, results of an exemplary flight mission will be compared and discussed.

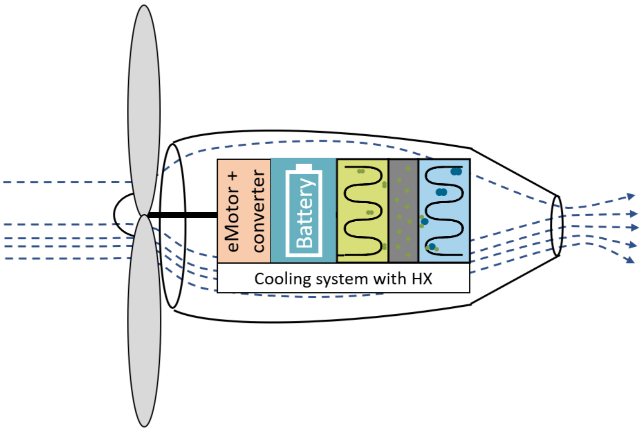


Figure 2. Sketch of a nacelle-integrated fuel cell propulsion system consisting of a propeller, electric motor, converter, battery, fuel cell stack and cooling system.

2 Fuel Cell System

The top level of the overall fuel cell system and its subsystems is shown in Figure 3. It consists of a fuel cell model, air and LH2 supply, components of the power electronics and the cooling system. The ambient and boundary conditions are provided by an environmental bus from a flight mission block. As this paper mainly focuses on the cooling system, the other subsystems are only briefly explained in the following.

The central heat load of the system is the fuel cell. Our model contains the main reaction equations to convert chemical energy into electrical energy. The resulting heat load is calculated from an energy balance across the whole fuel cell component and is essentially determined by a fixed efficiency. The power input of the fuel cell stack is connected to a converter and a motor. Both power electronic components are efficiency-based models. This means, that the fuel cell will always deliver the power that is requested from the shaft, taking the losses of the converter and the motor into account. In critical flight phases as take-off or overshoot with very high power demand, a battery can serve as a booster in order to relieve the fuel cell. However, in the current state of our models, the battery is not yet included.

The air supply provides the required oxygen mass flow to the fuel cell. Outside air is taken from a separate inlet and is fed through a compressor and heat exchanger to be prepared for the required conditions of the fuel cell. Optimal operating conditions of PEM fuel cells in aircraft were derived in a detailed study by (Schröder et al. 2021).

In our model, the supply air is controlled to a constant temperature near stack conditions at a pressure of 1.6 bar and a relative humidity of 75 %. During a flight mission, the optimal set points may vary depending on the flight phase but for the sake of simplicity, they are initially kept constant. To control the humidity, the required amount of water is extracted from the fuel cell exhaust gas and injected into the air supply. In practice, this task would be carried out by a membrane humidifier. At the current state, a combination of idealized water extractor and injector models are used to fulfill this task. Integrating a component model of a membrane humidifier is planned in future research activities.

3 Fuel Cell Cooling System

The cooling system enables temperature control of the fuel cell and power electronics, by rejecting heat to either the LH2 supply or to the ambient air using a ram air channel. To obtain optimal fuel cell efficiency, the LH2 needs to be warmed up from its cryogenic storage temperature to approximately the same temperature of the fuel cell itself. This provides a heat sink, capable of absorbing approximately 10 % of the generated heat, without causing any drag on the aircraft. The detailed modelling of LH2 as a heat sink is planned to be investigated in future research activities, however, in this paper, the focus is on using the ambient air as the only heat sink. A potential cooling system interface is shown in Figure 4. In this case the cooling loops are separated and the heat from the fuel cell is rejected to a different ram air channel than the heat from the power electronics. This enables an independent control of the heat loads and increases the total cooling capacity.

For the thermal management of fuel cell systems, many different cooling strategies can be applied. A detailed review of different cooling methods for PEM fuel cells was carried out by (Chen et al. 2021). In the following, a selected variety of different cooling concepts is presented. For each system architecture, ambient air is utilized as heat sink. To this end, the outside air is captured and guided through the system by a dedicated number of ram air channels. Different kinds of heat exchangers inside the channels reject the heat into the ram air flow. By leveraging the high robustness and easy usability of the TFS,

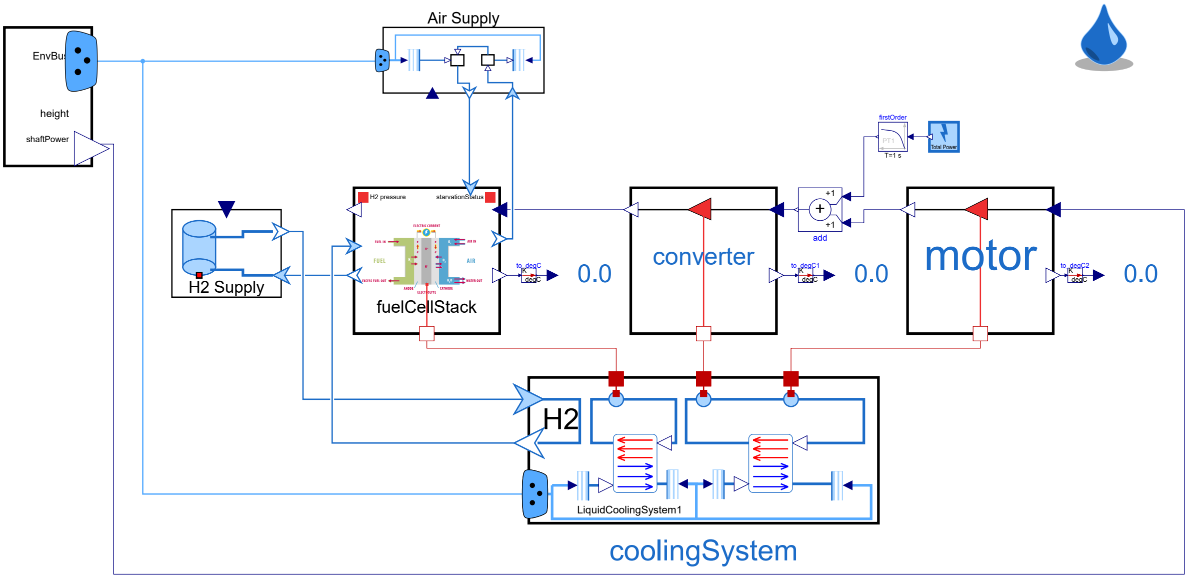


Figure 3. Modelica model of the overall fuel cell system.

different cooling systems can be rapidly set up, compared and optimized. For all concepts, the main objective is to maintain a constant fuel cell operating temperature of 80 °C.

3.1 Ram air channel

For all cooling systems, the common ram air channel design shown in Figure 5 is used. In this system, the valve represents the ram air door at the intake with variable opening, which is mainly used in cruise to control the amount of airflow through the channel. In ground operations, the door is fully open and sufficient airflow is generated by the ram air fan. Depending on the flight phase, the control split feeds the input signal of the controller to either the valve or the fan to ensure sufficient cooling airflow. The geometry of the ram air channel is modeled by the dynamic pressure boundaries to convert the air velocity into dynamic pressure.

As shown in Figure 4, distinct ram air cooling systems are used for the fuel cell and the power electronics. For the sake of simplicity, the following description of cooling systems is tailored specifically to the fuel cell, while a liquid cooling system is used for the power electronics in each case.

3.2 Liquid Cooling

The liquid cooling system shown in Figure 6 utilizes a single-phase medium, which directly transfers heat between the heat source and the heat exchanger within the ram air channel. This means that the sensible heat of the coolant is used and the heat transfer results in a temperature increase of the coolant. Typically, a mixture of deionized water and propylene or ethylene glycol is used as a coolant to lower the freezing point. This type of cooling system is commonly used in the automotive industry for fuel cell electric vehicles.

In this system, the pump is controlled in a way to maintain a constant temperature difference of the coolant across the fuel cell, while the ram air channel controller directly controls the temperature of the fuel cell itself. While being simple in the design, this system possesses a fundamental limitation, due to the relatively small temperature difference between the coolant and the ambient air at take-off conditions. This requires a large heat exchanger in the ram air channel, inducing significant drag and weight on aircraft level, which is carried through the entire flight profile.

3.3 Evaporative Cooling

In order to improve the efficiency of the heat exchange between the fuel cell, coolant and ram air, a phase-changing coolant like methanol can be used. For two-phase coolants and refrigerants, we use the media models from TILMedia Suite (*TILMedia Suite - Software package for calculating the properties of thermophysical substances* n.d.) developed by TLK Thermo GmbH. An internal wrapper was implemented to match the interfaces of the TFS to enable the usage of the external media models.

By evaporating methanol inside the fuel cell and by condensing it inside the heat exchanger in the ram air channel (condenser), heat exchanger efficiency can be improved, since the much larger latent heat instead of the sensible heat is utilized and the heat exchange occurs at a constant temperature. The most simple version of a system following this concept can be seen in Figure 7, where the methanol is simply circulated by a pump. In order to prevent damages, it is necessary to ensure that the methanol reaches the pump in a liquid, subcooled state. If the coolant could not completely be condensed in the ram air channel, the phase separator after the heat exchanger serves as a buffer element and only feeds liquid methanol to the pump. The points 1 and 2 correspond to the operat-

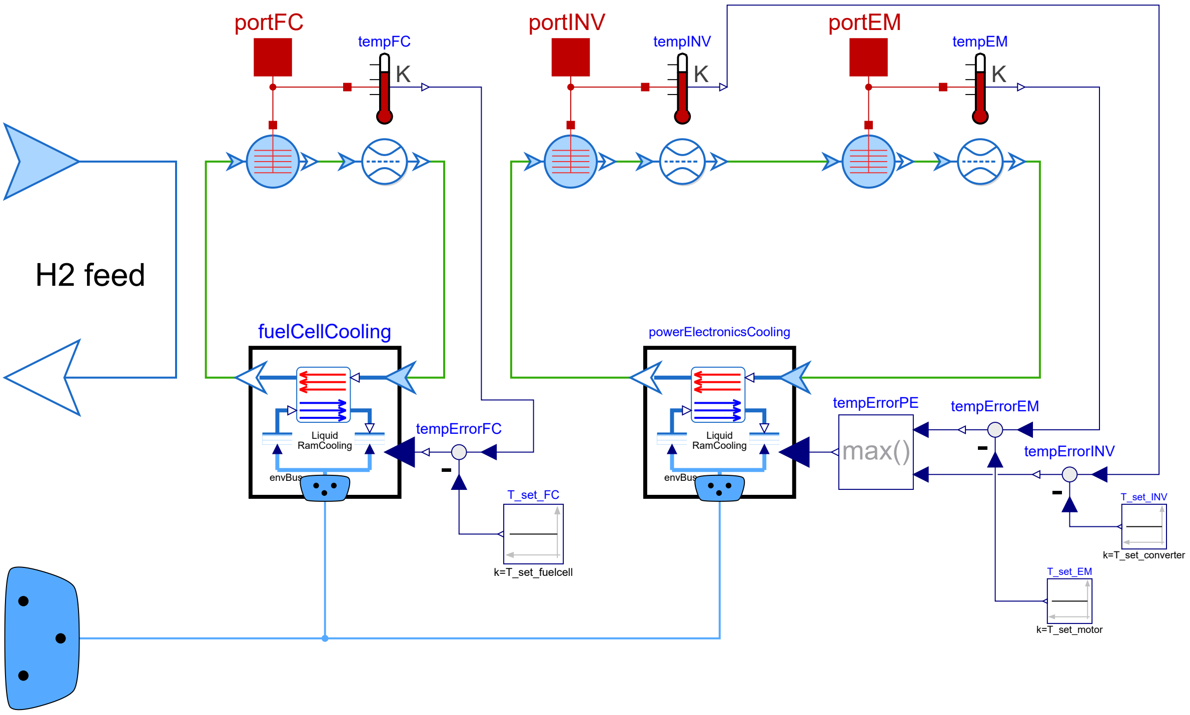


Figure 4. Modelica model of the overall cooling system. This model contains a dedicated ram air channel for the fuel cell, as well as a separate one for the remaining power electronics. No heat rejection to hydrogen is considered in this case.

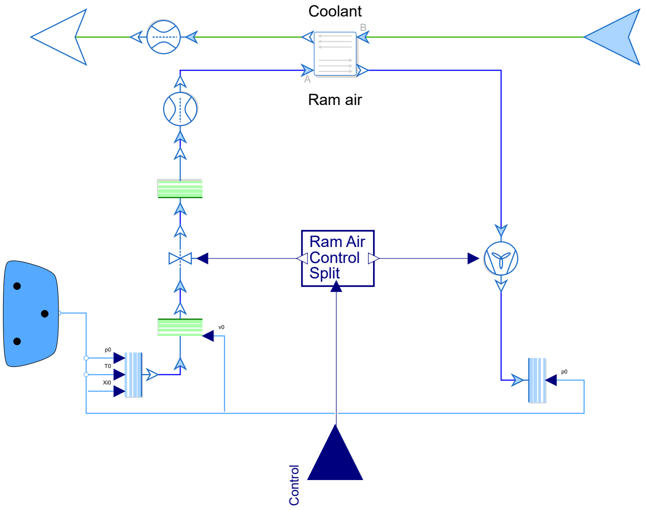


Figure 5. Modelica model of the ram air channel, consisting of a source, a sink, and a dynamic pressure flow section around the valve.

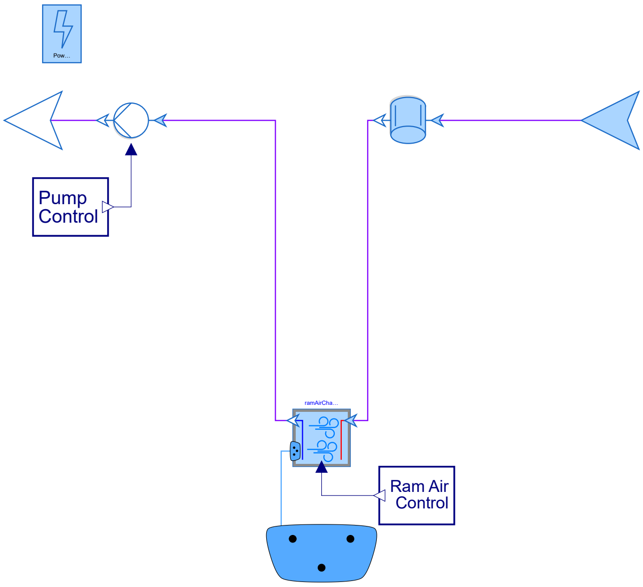


Figure 6. Modelica model of the liquid cooling system.

ing points in the generic pressure-enthalpy diagram shown in Figure 8. The vapor fraction at the outlet of the fuel cell (point 2) is controlled to a constant value by controlling the speed of the methanol pump. The limited vapor fraction prevents too much evaporation of the coolant in the fuel cell and hence avoids instabilities due to a drastic increase in volume during the evaporation process. The fuel cell temperature target is reached by controlling the ram air channel airflow. Since the evaporation at the fuel cell occurs at a constant temperature, this effectively results in regulating the pressure level p_{low} to ensure sufficient heat transfer from the fuel cell to the coolant.

An enhanced version of this system can be seen in Figure 9, where the liquid and gaseous methanol are separated to ensure that only the gaseous methanol flows through the condenser in the ram air channel. This means that not the entire coolant flow has to flow through the heat exchanger and hence it can be reduced in size and weight.

This bypass system results in the operating points 1, 2 and 3 in Figure 8. The vapor fraction at the fuel cell outlet

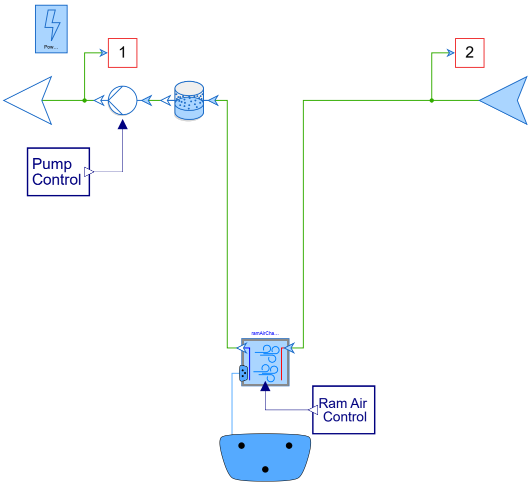


Figure 7. Modelica model of the simple methanol loop.

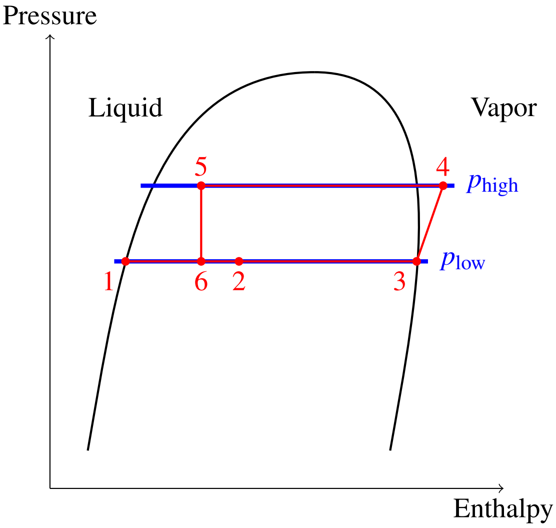


Figure 8. Generic pressure-enthalpy diagram applying to methanol cycles.

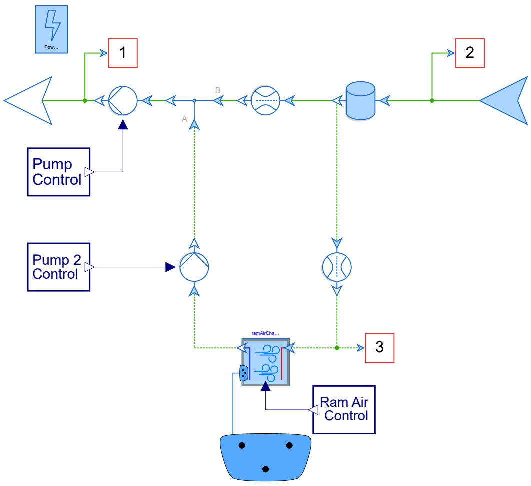


Figure 9. Methanol cooling cycle, including a separator to divide the liquid and gaseous phase.

is still controlled to a constant value at point 2, while the ram air channel controller ensures that enough airflow is supplied to fully condense the methanol, ensuring a liquid state at the inlet of the secondary pump. The second pump controls the temperature of the fuel cell by adjusting the pressure level p_{low}.

A further improvement of the cooling efficiency can potentially be achieved by increasing the condenser inlet temperature. To this end, the previous variant is expanded by a compressor that lifts the pressure and temperature before it enters the condenser, as shown in Figure 10. In take-off conditions, this can more than double the temperature difference between the coolant and ambient air. After the heat exchanger, the methanol is subsequently expanded to the desired operating pressure by an appropriate flow resistance. A direct routing to the pump would require a high level of subcooling in the condenser to ensure that the methanol still enters the pump in a liquid state after expansion. This would result in a bigger sizing of the heat exchanger, hence the methanol is recirculated to the phase separator. The corresponding operating points 4, 5 and 6 in Figure 8 showcase the process of compression and expansion in the cooling cycle. In this case, the ram air channel controller ensures that a satisfactory upper pressure level p_{high} is maintained, while the compressor ensures the load reaches its target temperature, effectively controlling the lower pressure level p_{low}. The coolant pump again ensures that a desirable vapor fraction is obtained at the fuel cell outlet.

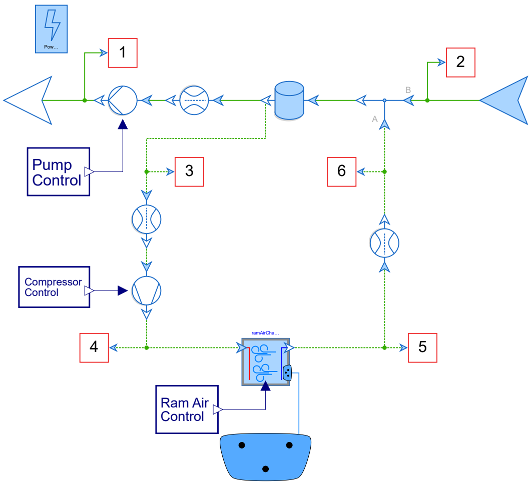


Figure 10. Methanol cooling cycle, including a compressor to increase the condenser inlet methanol temperature to improve cooling efficiency.

3.4 Vapor Compression Cooling

Besides the direct two-phase cooling in terms of evaporative cooling, an indirect two-phase cooling concept using a vapor compression system (VCS) was investigated. This system consists of two combined cycles - a conventional liquid cooling cycle and a VCS, as shown in Figure 11. The VCS consists of four main components, a compressor, condenser, expansion valve and evaporator. The compressor brings the refrigerant to higher pressure and temperature to ensure heat rejection to the ambient by condensation in the condenser. The expansion valve subsequently expands the refrigerant to a lower pressure and temperature to absorb heat during evaporation in the evaporator. On the secondary side of the evaporator, the refrigerant absorbs the heat from the liquid coolant that is used to control the fuel cell temperature.

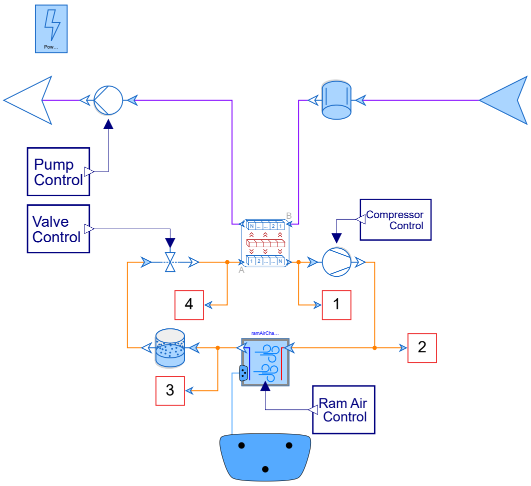


Figure 11. Vapor compression cooling system, consisting of two separate loops. The purple loop contains a purely liquid coolant, while the orange loop utilizes a phase changing refrigerant.

In the present system we make use of a high temperature refrigerant (HFO-1336mzz-Z) that enables a high condensation temperature in the condenser. We assume, that this will result in advantages with regard to the design of the heat exchanger and the dimensioning of the ram air duct in comparison to conventional refrigerants. The very low global warming potential (GWP) of the refrigerant makes it additionally interesting for applications in aviation. In principle, however, indirect two-phase cooling can also be achieved with conventional refrigerants. A detailed analysis of the advantages and disadvantages in terms of dimensioning and operation is still pending. In this system, the pump of the liquid coolant once again ensures a constant temperature difference across the fuel cell. The ram air channel controller keeps the upper pressure level of the VCS to uphold a sufficient temperature difference to the ambient air. The lower pressure level and hence the evaporation temperature is controlled by the compressor. Contrary to conventional VCS, we use the expansion valve to ensure sufficient refrigerant flow through the cycle to maintain the fuel cell target temperature instead of controlling the evaporator superheating (Michalak, Emo, and Ervin 2014).

4 Aircraft and Mission Profile

Several cooling systems were tested throughout an exemplary flight mission for a fuel cell driven electric aircraft. The reference architecture is derived within the DLR internal EXACT project (*EXACT – Conceptual study for future climate-neutral flight* n.d.) and is shown in Figure 1. It is a fully fuel cell driven aircraft for 70 passengers. The fuel cells are directly integrated in the nacelle and a total number of 10 pods is distributed across the wings. The total required shaft power of all pods combined at the corresponding altitude is shown in the mission profile in Figure 12.

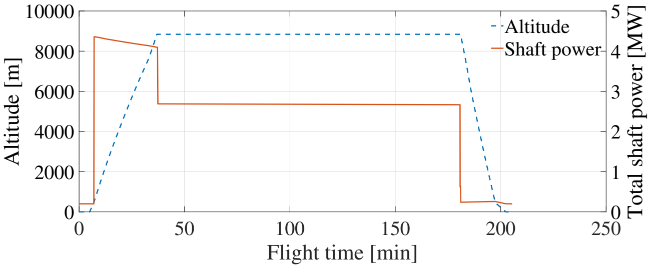


Figure 12. Mission profile with altitude and total required shaft power.

The total flight time is around 200 min and a maximum speed of Mach = 0.55 is reached during cruise at an altitude of 8840 m. The mission profile covers the main flight phases taxi out, take-off, climb, cruise, descent, approach, landing and taxi in. Ambient pressure and temperature are derived from the ISA standard conditions (15 °C on ground).

5 Simulation Results

In the following section, the results of the mission simulations will be presented. It has to be mentioned, that this paper focuses on the functionality of the TFS and the goal was to prove that a variety of different cooling systems with different coolant media can be quickly built up and tested throughout a flight mission. We are not yet in the state to make definite statements on the performance of each cooling system. This would assume not only a detailed sizing of each of the components but also an optimized control strategy. Nevertheless, we were able to simulate all of the cooling systems described in Section 3 through the fast changing operating points of a flight mission while reaching the desired fuel cell target temperature of 80 °C. As the systems differ in the number and arrangements of the components, some common variables of the cooling systems were compared. One important indicator of the performance of a cooling system is the amount of ambient air that was required to reach the fuel cell target temperature. A high demand of ram air flow has significant impact on the total drag that is induced on aircraft level. The airflow through the ram air channel for each cooling system is shown in Figure 13.

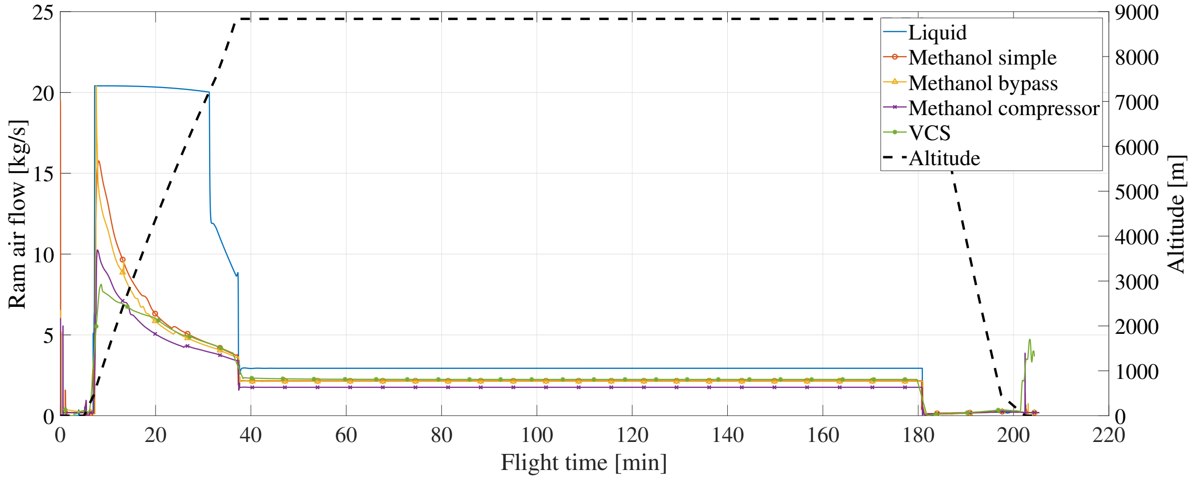


Figure 13. Required ram air flow of different cooling systems during flight mission.

It is noticeable, that the liquid cooling system requires significantly more ram air during the climb phase. This is due to the previously discussed small temperature difference of the ambient to the coolant. Especially in this flight phase, the advantages of the methanol cycles become clear, as the required ram air flow is drastically reduced. Also the benefits of the methanol cycle with the compressor to increase the condenser inlet temperature are clearly visible as it requires the least ram air flow during the climb phase. Different from our simulation, the EXACT aircraft concept includes a battery that serves as a booster in this critical flight phase and therefore could decrease the required power from the fuel cell. This would result in less heat that has to be removed and hence a smaller amount of required outside air. At the very end of the flight mission, an increase in required ram air flow can be observed for the VCS cooling system. This is mainly due to inefficient control and operation of the VCS since the set-points are not dependent on the flight phase yet.

Another indicator of the performance of the cooling system is the power that is required by the turbomachines and pumps of each system, including the air supply. In Figure 14 the accumulated energy that is consumed by a number of 10 propulsion nacelles throughout the mission is shown for each cooling system. It has to be mentioned, that the air supply system consists of a turbine that regulates the pressure before the air enters the fuel cell. The turbine is mounted on the same shaft as the air compressor to recover energy from the fluid. The power that is generated by the turbine is also taken into account in the accumulated total energy. Remembering that the liquid cooling system required the largest amount of ram air flow for cooling, it seems beneficial in terms of energy consumption. The systems including a compressor turn out to be more energy consuming while requiring less ram air for cooling.

A conventional indicator regarding energy consumption and cooling efficiency is the so-called coefficient of performance (COP) which is usually calculated for VCS cooling systems. In the following, we want to have a closer look on the VCS and its operation during the flight mission. The COP is the ratio of the cooling power during the evaporation Q_{evap} process over compressor work W_{compr}:

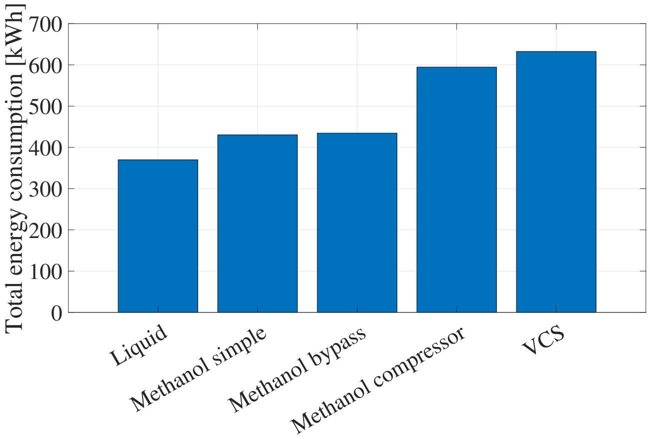


Figure 14. Accumulated total energy demand of different cooling systems during flight mission.

$$COP = \frac{Q_{evap}}{W_{compr}} \tag{1}$$

Taking a closer look on the COP of the VCS throughout the flight mission (Figure 15), the relatively high numbers need some explanation. VCS systems in aviation are usually beneficial during ground or low altitude conditions, when the ambient temperature is still relatively high. Therefore the high temperature for heat rejection into the ram air channel can be exploited.

Most of the time during the mission, especially during cruise, the VCS runs in a degenerated operating mode because of the very low ambient temperature. This results in a very low compressor work, as the condensing temperature and hence the high pressure level of the VCS can be

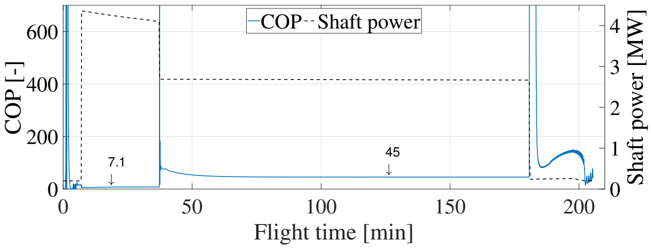


Figure 15. COP of VCS and total required shaft power during flight mission.

very low compared to normal operations. Consequently, the evaporation process takes place on a similar pressure and temperature level as the condensation process, resulting in a very high COP. Additionally, the sudden drops in the total shaft power after top of climb and in the transition from cruise to descend are leading to very high peaks in COP which goes up to values of a few thousands, which means that the compressor runs on the absolute minimum. Due to visualization reasons, the peaks are cut off in Figure 15. Those degenerated operating modes promised to be challenging to tackle in our simulations, because it results in very low refrigerant mass flow that can lead to instabilities. Strategies like including a bypass around the condenser to reduce the heat transfer rate (head pressure control) while keeping a minimum refrigerant flow are currently under investigation.

All the presented results still have to be taken with care, as a detailed sizing of the compressors, heat exchangers, ducts and also the ram air channels was not yet carried out. Anyway it still points out, that the selection of an optimal cooling system for the thermal management of the fuel cell is a complicated task with many influencing factors.

6 Conclusion

The objective of this work was to build up an aircraft fuel cell system and its subsystems with components of the TFS. The focus was on the rapid testing of different cooling systems in the environment of the overall fuel cell system throughout a whole flight mission. Common interfaces were defined that enable a quick interchangeability of the system architectures. A prototypical control scheme was developed for each cooling system to maintain a fuel cell target temperature during all operating points. This work should serve as a basis for future investigations on the benefits and drawbacks of specific cooling architectures. For more reliable statements on the performance of different systems, a more detailed sizing of the components as well as optimized control strategies will be subject to future research activities. This also refers to the air and LH2 supply systems as well as the drive train that is planned to be expanded by a battery model. In addition to that, the water separation process, that is important for the exhaust gas handling, is also intended to be improved by providing a model of a membrane humidifier. Nevertheless, the robustness of the TFS proved to be crucial during the development of thermal systems with such complexity.

Acknowledgements

This work has been supported by the LuFo 6.1 iTherNet and Lufo 6 SKAiB project. The authors would also like to thank Philip Balack and Georgi Atanasov from DLR-SL for providing detailed aircraft and mission data.

Supported by:

Federal Ministry
for Economic Affairs
and Climate Action

on the basis of a decision
by the German Bundestag

References

Chen, Qin et al. (2021). "Thermal management of polymer electrolyte membrane fuel cells: A review of cooling methods, material properties, and durability". In: *Applied Energy* 286, p. 116496. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2021.116496. URL: https://www.sciencedirect.com/science/article/pii/S030626192100057X.

EXACT – Conceptual study for future climate-neutral flight (n.d.). Accessed: 2023-05-31.

Michalak, Travis, Stephen Emo, and Jamie Ervin (2014). "Control strategy for aircraft vapor compression system operation". In: *International Journal of Refrigeration* 48, pp. 10–18. ISSN: 0140-7007. DOI: https://doi.org/10.1016/j.ijrefrig.2014.08.010. URL: https://www.sciencedirect.com/science/article/pii/S0140700714002126.

Schröder, M. et al. (2021). "Optimal operating conditions of PEM fuel cells in commercial aircraft". In: *International Journal of Hydrogen Energy* 46.66, pp. 33218–33240. ISSN: 0360-3199. DOI: https://doi.org/10.1016/j.ijhydene.2021.07.099. URL: https://www.sciencedirect.com/science/article/pii/S0360319921027634.

TILMedia Suite - Software package for calculating the properties of thermophysical substances (n.d.). Accessed: 2023-06-01.

Zimmer, Dirk (2019). "Robust Simulation of Stream-Dominated Thermo-Fluid Systems: From Unidirectional to Bidirectional Applications". In: *EUROSIM Congress 2019* (Logroño, Spain).

Zimmer, Dirk, Daniel Bender, and Alexander Pollok (2018). "Robust modeling of directed thermofluid flows in complex networks". In: *Proceedings of the 2nd Japanese Modelica Conference*. Linköping University Press, pp. 39–48.

Zimmer, Dirk, Niels Weber, and Michael Meißner (2021). "The DLR ThermoFluidStream Library". In: *Modelica Conferences*, pp. 225–234.

Supporting Infinitely Fast Processes in Continuous System Modeling

John Tinnerholm[1] Francesco Casella[2] Adrian Pop[1]

[1]Department of Computer and Information Science (IDA), Linköping University, Sweden,
{firstname.lastname}@liu.se
[2]Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, Italy,
francesco.casella@polimi.it

Abstract

This article examines the consequences of introducing a new language construct into an equation-based language to model infinitely fast processes. We do this by extending the equation-based language Modelica with a special time constant, Θ. Θ provides modelers with an additional language construct that they can utilize both to improve the performance of numerical integration for existing models as well as express and simulate models that existing tools may struggle with. In this paper we exemplify this with two examples. The first is an artificial DAE-System using a monotonic function; the second is an electrical circuit with and without a parasitic capacitance.

Based on our observations, we believe that by enabling modelers to express common idealizations using Θ we can improve both performance and maintainability. This is the case since it is possible to express the relevant idealizations can now be expressessd using Θ and are thereby explicitly encoded in the model.

Keywords: continuous system modeling, Modelica, modeling, nonlinear systems, simulation

1 Introduction

In the context of Modeling and Simulation (M&S) a commonly used modeling language is the Modelica Language which enables modelers to model complex systems using object orientation and equations to represent various physical components. By using equations, the Modelica language can model any domain that can be expressed using equations. The goal of the language is to provide modelers with the necessary abstractions to express complex cyber-physical systems.

However, modeling complex cyber-physical systems is a challenging task, and when designing such systems modelers frequently utilize idealization techniques in order to formulate models that can be simulated efficiently. Still, as of this writing, there are some techniques common in modeling practice that the Modelica language, and, according to our best knowledge and mainstream equation-based languages in general, do not support.

One such idealization technique is the method of *Artificial States* where the modeler extends the dynamics of systems by introducing additional state variables and associated equations, and in that way, reducing the complexity of the resulting nonlinear equation system, allowing efficient and reliable solving.

The use of artificial states is generally seen as malpractice in the modeling community, however, Zimmer (2013) provides recommendations concerning how M&S frameworks, and consequently equation-based modeling languages, can be extended such that modelers may express this idealization explicitly.

This idea is further expanded upon in (Zimmer 2014), where he proposes an augmentation to existing equation-based languages by introducing a new time constant, Θ.

1.1 Motivation

While the method proposed by Zimmer has been previously discussed and suggestions for implementing it have been described in (Zimmer 2013; Zimmer 2014), it has not yet been integrated into any mainstream equation-based language. In this article, we investigate the method initially proposed by Zimmer empirically. Hence, we aim to provide additional insights concerning the applicability Θ in pratice. We do this by examining and discussing the practical consequences of integrating these concepts into the equation-based language Modelica and provide details concerning how to integrate it in compilers for equation-based languages. To illustrate the concept we use a common modeling scenario with a nonlinear circuit and a DAE-System that existing Modelica environments have difficulties solving.

1.2 Organization

The remainder of this article is organized as follows: Section 2 further expands on the background provided in the introduction by discussing Θ and summarizing the mathematical background. We then provide details concerning how to extend a Modelica Simulation environment by adapting the structural transformation phases to accommodate this new operator in Section 3. Section 4 presents the results using the proposed new operator, and Section 5 presents additional related work. Finally, the conclusion is presented in Section 6.

DOI
10.3384/ecp204423

2 Handling infinitely fast processes in continuous system modeling

In an equation-based language, the system of equations of the final system that is to be simulated is derived by collecting and merging the equations resulting from the components of some model. Systems in the following form are typical:

$$\frac{dx}{dt} = f(x, u, p, t)$$

where x is the set of state variables, u is the set of algebraic variables, and p is a vector representing parameters and constants. These systems consisting of both algebraic and differential equations are called **DAE**s. Compilers for equation-based languages such as Modelica translate such systems into executable code for simulation by techniques such as index reduction and topological sorting of the dependencies between its constituent parts.

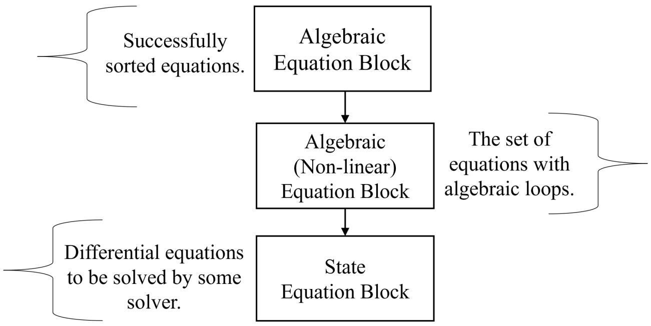

Successfully sorted equations.
Algebraic Equation Block
Algebraic (Non-linear) Equation Block
The set of equations with algebraic loops.
Differential equations to be solved by some solver.
State Equation Block

Figure 1. A sketch on how simulation code might look for a model that contains algebraic loops.

Due to the presence of algebraic loops, such systems may contain a nonlinear subsystem of equations and a system of ordinary differential equations. Figure 1 illustrates how a DAE-System might be translated by a model compiler. In this example, the part of the system that can be fully causalized is denoted the **Algebraic Equation Block**. Following this block is the **Algebraic (Nonlinear) Equation Block**. This block represents a nonlinear system of algebraic equations, caused by the existance of algebraic loops in the underlying model. Finally, in this example is the block of the differential equations, the **State Equation Block**.

Zimmer (2014) argues that a common occurrence in modeling practice is a system where parts of a system converge faster than others and provide the following example:

$$\begin{cases} \dfrac{dx}{dt} = f_x(x, y, u, t) \\ \dfrac{dy}{dt} = f_y(x, y, u, t) \end{cases} \tag{1}$$

In Equation 1, we assume that the process involves the state variable x, and converges faster than the process associated with the state variable y. Here, u denotes a set of algebraic variables. If we assume that the system in Equation 1 is stiff, and that the modeler is not interested in the dynamics of x a possible idealization is the system defined in Equation 2:

$$\begin{cases} 0 = f_x(x, y, u, t) \\ \dfrac{dy}{dt} = f_y(x, y, u, t) \end{cases} \tag{2}$$

In this case, the dynamics of the state variable x have been removed to make the system easier to solve. However, in this case we need to solve the nonlinear system:

$$0 = f_x(x, y, u, t)$$

The modeler, Zimmer (2014), argues, has to choose between two alternatives, either a stiff model represented by Equation 1 or a model with a possibly complex nonlinear system as in Equation 2. Zimmer (2014), argues that this selection is often made pragmatically; hence, once the modeler selects one alternative, the choice is not explicitly encoded in the model code. Consequently, future modelers that are to maintain such a model might not know that an idealization has been made, which arguably, makes maintenance more difficult.

Instead, Zimmer (2014) proposes the inclusion of a universal time constant, denoted Θ for equation-based languages to make this idealization explicit. Using this approach, Equation 1 can be formulated as follows:

$$\begin{cases} \dfrac{dx}{dt} \cdot \Theta = f_x(x, y, u, t) \\ \dfrac{dy}{dt} = f_y(x, y, u, t) \end{cases} \tag{3}$$

In Equation 3 the modeler explicitly expresses that $\frac{dx}{dt} \cdot \Theta = f_x(x, y, u, t)$ is an infinitely fast process. Consequently, this entails that the part of the system dependent on Θ is to be solved by means of a sub-simulation[1]. In other words, if Θ is used as in Equation 3, the system should be solved as in Equation 4. Where a sub-simulation provides the $\hat{x}$ value, by solving the differential equation $\frac{d\hat{x}}{dt} = f_x(\hat{x}, y, u, t, \hat{t})$ where the state variable y, the variables in u and the global time[2] are treated as constants. Here, $\hat{t}$ is the artificial time used by the sub-simulation to model it as a infinitely fast process.

For additional details and mathematical background concerning Θ we refer to (Zimmer 2014).

$$\begin{cases} 0 = f_x(\hat{x}, y, u, t) \\ \dfrac{dy}{dt} \cdot T_y = f_y(x, y, u, t) \end{cases} \tag{4}$$

[1] In the article **Handling infinitely fast processes in continuous system modeling** (Zimmer 2014), this is described to be solved by a continuation solver.

[2] The time t of the main simulation.

```
model DAE_Example
  Real x(start = 1.0);
  Real y;
  Real a;
function s
  input Real a;
  output Real oa;
algorithm
  if (a < -1) then
    oa := a/4 -3/4;
  elseif (a > 1) then
    oa := a/4 + 3/4;
  else
    oa := a;
  end if;
end s;
equation
  der(x) = y;
  der(y) = -0.1*a - 0.4*y;
  der(a) = (10*x - s(a));
end DAE_Example;
```

Listing 1. A first attempt of a Modelica implementation of Equation 5.

2.1 Example 1: A Differential Algebraic Equation System

We turn now to a more concrete example of the previous discussion to exemplify how Θ can be used in the context of Modelica. Later in Section 4, we provide an example on how Θ may be used to significantly speed up simulations, and arguably, make models more maintainable by allowing modelers to explicitly state their intent. In Zimmer (2013) the following system is given:

$$f(x,y,a,t) = \begin{cases} \dfrac{dx}{dt} = y \\ \dfrac{dy}{dt} = -0.1a - 0.4y \\ \dfrac{dx}{da} = 10x - s(a) \end{cases} \tag{5}$$

With the monotonic increasing function s defined as:

$$s(a) = \begin{cases} a - 4 - 3/4 & \text{if } a < -1 \\ \\ a/4 + 3/4 & \text{if } (a > 1) \wedge \neg(a < -1) \\ a & \text{otherwise} \end{cases} \tag{6}$$

If we formulate a model for the system defined by Equation 5 in the equation-based modeling language Modelica, see Listing 1, we end up with a stiff system. Still, if we attempt a similar idealization as described earlier in this section we could do so by substituting $\frac{dx}{da} = 10x - s(a)$ with $0 = 10x - s(a)$.

```
model DAE_Example2
  Real x(start = 1.0);
  Real y;
  Real a;
equation
  der(x) = y;
  der(y) = -0.1*a - 0.4*y;
  0 = (10*x - s(a));
end DAE_Example2;
```

Listing 2. An attempted idealization of the model in Listing 1, the function s has been omitted.

With this change we can formulate the Modelica model in Listing 2. If we attempt to simulate this system the state-of-the-art OpenModelica Compiler (Fritzson et al. 2020) is unable to simulate it correctly due to the resulting nonlinear system, unless a very small step size is selected. As discussed previously, this is clearly disadvantageous since it requires manual adjustments of solver settings. Furthermore, this impacts simulation performance negatively since a very small step size is needed.

(Zimmer 2013), presents a clear use case for introducing an operator called balance. The similarities to Θ means that it may be used in place of balance, much in the same way. If we apply Θ to Equation 5 we get the following system:

$$\begin{cases} \dfrac{dx}{dt} = y \\ \dfrac{dy}{dt} = -0.1a - 0.4y - \\ \dfrac{dx}{da} \cdot \Theta = 10x - s(a) \end{cases} \tag{7}$$

The resulting Modelica model and the associated code generation extensions needed for Θ will be discussed in Section 3, and the simulation resuls are presented in Section 4.

2.2 Example 2: Nonlinear Circuit

Let us now consider a less artificial example exemplified by using two configuration examples of an electrical circuit model.

The first example in question is a nonlinear circuit with a few diodes. The diodes are real diodes with an exponential voltage-current characteristic, not ideal diodes with either zero voltage or zero current. The model Circuit1Static, see Figure 2, has a series connection between the diodes and a large resistor. The result of this connection is a very strongly nonlinear system of equations. In this case the nonlinearity index (Casella and Bachmann 2021), will be >> 1. As a consequence, if simulated by the OpenModelica Compiler (Fritzson et al. 2020) the nonlinear algebraic equation solver experiences convergence issues, causing the master ODE integration

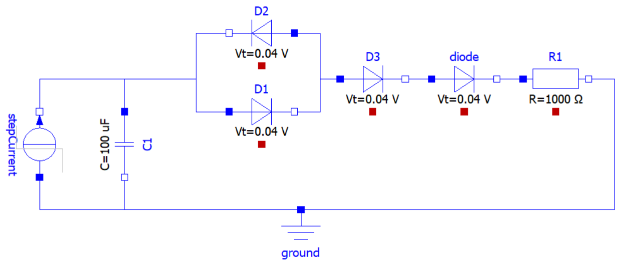


Figure 2. An electrical circuit with a nonlinear system that is difficult to solve. Model `Circuit1Static`.

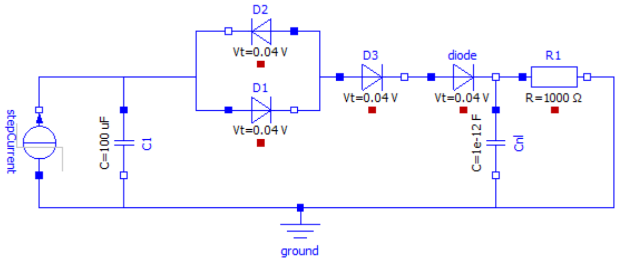


Figure 3. An electrical circuit with a less complex nonlinear system due to the parasitic capacitance `Cnl`. Model `Circuit1Dynamic`.

method to reduce the time step, and to eventually give up after 0.015 seconds.

The second circuit `Circuit1Dynamic`, see Figure 3 solves the problem by connecting a small parasitic capacitance between the diodes and the large resistor. The introduction of an additional state variable to the model makes the voltage at that node known at each time step during the simulation, hence significantly easing the solution of the system. Hence, by utilizing the method of artificial states, we ensure that the simulation can proceed without issues.

To conclude, the examples in Subsection 2.1 and Subsection 2.2 exemplify how various idealizations might be used in practice. Still, sometimes it might be difficult to get the correct simulation results as exemplified in the discussion of Subsection 2.1. In other cases we can get a system to simulate at the cost of introducing additional states. However, and as previously discussed and argued by Zimmer (2014) this might be a future detriment in terms of model maintainability. As we will see in Section 4, Θ may be used to significantly speed up simulations in this case, and arguably make the idealization more maintainable.

3 Implementation

In this section we present implementation details concerning the introduction of Θ in a Modelica compiler.

3.1 OpenModelica.jl

We implemented the ideas presented in this paper in **OpenModelica.jl**, a Julia-based Modelica Compiler (Tinnerholm, Pop, and Sjölund 2022). OpenModelica.jl is written in the programming language Julia and supports some experimental features not currently available in mainstream Modelica Compilers. This compiler integrates several Julia packages such as ModelingToolkit (MTK) (Ma et al. 2021) and DifferentialEquations.jl (Rackauckas and Nie 2017). The main feature of this compiler being its modularization and extensions that introduce support for Variable Structure System Modeling for Modelica. As a part of this work a new code generator was written to export the intermediate representation produced by MTK models to a more portable low-level representation. Furthermore, we implemented support for a significant subset of the Electrical Library of the Modelica Standard Library for this new code generator.

3.2 Extending Modelica with Θ

The typical compilation process of a compiler for an equation-based language is to transform the provided model into a suitable format for some solver. The general process is described in Figure 4.

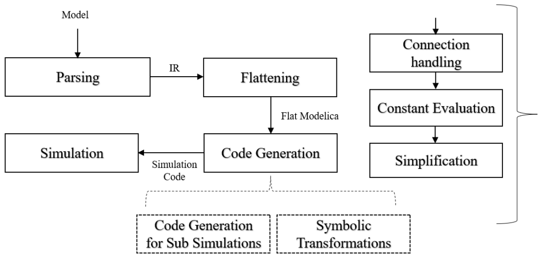


Figure 4. An illustration of a typical compiler pipeline for an equation-based language. The frontend is similar to that of an ordinary compiler; it performs parsing, syntactical, and semantical analysis of the input model. Finally, a compiler for an equation-based language typically generates code targeting a solver such as DASSL (Petzold 1982).

In principle introducing Θ involves only slight changes to key parts of this process. First of all, Θ should not only be used as a low-level operator; instead, Θ should be applicable in a non-invasive way such that the internal equations of models that it is applied to are untouched. Furthermore, Θ should be propagated and not be removed during any optimization phase. As such, Θ should be available and taken into consideration by the various structural transformation phases, such as sorting performed by the compiler backend.

3.3 Preparing Code For Simulation

Zimmer (2014) provides an initial sketch for simulation code generation when expanding an equation-based language with Θ. In summary, the steps are as follows:

1. Treat Θ as an irreducible variable.
2. Analyze that Θ has been applied correctly.

3. Generate code for simulation with respect to how Θ has been applied.

To provide initial support for Θ we implemented it as a special parameter by introducing a new reserved keyword `THETA`[3]. By reserving a name we can via static analysis follow the def-use chains in the frontend and abstain from removing the parameter during the backend optimization phases. Hence, concerning the first step, we fulfill it by omitting certain optimization phases such as not removing simple equations that have structural dependencies on `THETA`. This means that Θ remains in the system of equations, after the sorting, matching, and index-reduction phases are completed.

The second step entails adhering to several constraints, Zimmer (2014) proposes the following constraints:

1. The resulting system should be successfully balanced.

2. The resulting system should be successfully causalized.

3. Furthermore, each variable may be expressed as a factor of Θ^n where n is an integer. Moreover, Θ may not be used as a function argument for nonlinear functions such as `sin`, `cos`. Furthermore, the variables in $\frac{dx}{dt}$ should be multiplied by Θ or Θ^0, where multiplication by Θ indicates that sub-simulation code should be generated for that variable[4]

These requirements were fulfilled by augmenting the compiler backend with additional checks before proceeding with simulations, however, the check concerning invalid usage of Θ in nonlinear functions was omitted.

3.4 Generating code for state variables depending on Θ.

As previously stated, variables in $\frac{dx}{dt}$ should be multiplied by either Θ or Θ^0 where the first indicates that code representing an infinitely fast process should be generated for that part of the system. For more mathematical detail concerning the code generation for this process, we refer to (Zimmer 2014).

To exemplify the current state of our code generator let us consider the Modelica model in Listing 1. Using the aforementioned new parameter THETA we can augment our code and write a new model as in Listing 3.

The structural analysis is simple for the model depicted in Listing 3. Code for a sub-simulation is generated for

```
der(a) * THETA = (10*x - s(a))
```

```
model DAE_Example_THETA
  Real a;
  Real x(start = 1);
  Real y;
  parameter Real THETA = 1.0;
equation
  der(x) = y;
  der(y) = -0.1*a - 0.4*y;
  der(a)*THETA = (10*x - s(a));
end DAE_Example_THETA;
```

Listing 3. A Modelica implementation of Equation 5, here the function `s` is omitted.

For the main simulation, this equation is replaced with the following nonlinear equation as described in Section 2:

```
0 = (10*x - s(a))
```

During the simulation, the sub-simulation is solved using the implicit Euler integration algorithm, providing $\hat{x}$ for the main simulation. The current termination criterion for the sub-simulation is running the artificial time $\hat{t}$ from $\hat{t} = 0$ until $t_{current}$[5]. For a high-level description of the code generated for the solvers, we refer to Algorithm 1 and Figure 5.

Algorithm 1 High-level description of the code generated when translating the Modelica model in Listing 3.

```
function K($u$)
    Initialize $x$ and $y$ using $u$.
    $ox[1] \leftarrow 10y[1] - s(x[1])$
end function
function H($dy, y, u, t$)
    sub-simulation(u)
    nonlinear-solve(k)
    $dy[1] \leftarrow y[2]$
    $dy[2] \leftarrow -0.1x[1] - 0.4y[2]$
end function
function SUB-SIMULATION(u)
    extract x from u.
    solve $\frac{da}{dt} = 10 \cdot x - s(a)$
    Update $u$, provide $\hat{a}$ for the main simulation.
end function
function SIMULATION-FUNCTION
    Simulate by integrating the $H$ function.
    Report results.
end function
```

In general, however, the structural analysis needed, may be more involved. Consider, the electrical circuit in Figure 3, just as in Listing 3 the model is a suitable candidate

[3] We note that this might break existing models using parameters with the same name, however, we use it in the initial implementation to illustrate the concept.

[4] Θ^0 means that Θ has not been applied.

[5] It should be noted that a dedicated algorithm to describe the sub-simulation is presented in (Zimmer 2013). Compared to the algorithm suggested in (Zimmer 2013) we currently take more steps in the sub-simulations then necessary.

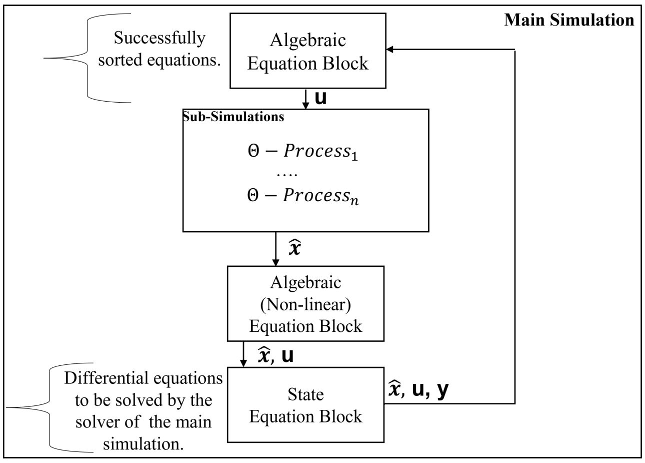


Figure 5. Graphical illustration of the simulation code generated for a model with *n* non-nested subprocesses. Showing where Θ appears in generated code, the Θ-processes supply the nonlinear solver with initial values.

```
package CircuitTest
  model ThetaCircuit2Dynamic
    parameter Real THETA = 1.0;
    extends Circuit1Static;
    Capacitor Cp(C = 1e-12 * THETA);
  equation
    connect(Cp.n, ground.p);
    connect(diode.n, Cp.p);
  end ThetaCircuit2Dynamic;
end CircuitTest;
```

Listing 4. Modelica model showcasing how the Θ is used at the top level of the component hierarchy. The components used are from the Modelica Standard Library; the package paths have been omitted.

for applying Θ. To illustrate how it can be applied to existing models without changing any equations at a lower abstraction level consider the model depicted in Listing 4 where Θ is applied at the top level.

$$\begin{cases} 0 = Cp\_v - R1\_R\_actual \cdot R1\_i \\ 0 = diode\_i - (10^{-9}(tmp53 - 1) - (10^{-8}))diode\_v \\ 0 = D2\_i + diode\_i - D1\_i \\ \dfrac{Cp\_v}{dt} = Cp\_i/(10^{-12}\Theta) \\ \dfrac{C1\_v}{dt} = 9999.99999999999 \cdot C1\_i \end{cases} \tag{8}$$

When used as in Listing 4, the compiler initially generates the equations listed in Equation 8, then, the compiler starts Θ specific code generation. During this process structural analysis is used to extract the processes that should be run as sub-simulation from the resulting equations. This is achieved by using the following steps:

1. Construct a graph based on equation-variable dependencies.
2. Extract equations were the Θ operator is used.
3. Extract the set of variables depending on Θ.
4. Return the strongly connected components of the equation-variable dependency graph.

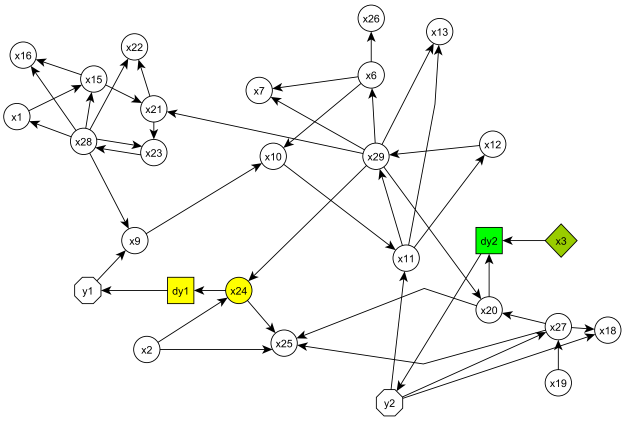


Figure 6. Excerpt of the dependency graph for `ThetaCircuit2Dynamic` in Listing 4. The variables that depend on Θ is marked in green, the other state is marked in yellow.

The last step to extract the strongly connected components uses Tarjans algorithm (Tarjan 1971). To illustrate this process graphically we refer to Figure 6. We refer to the algorithm in (Zimmer 2014) for a more formal description of the steps involved.

After the structural analysis, the compiler generates Equation 9 for the main simulation process and Equation 10 for the sub-simulation.

$$\begin{cases} 0 = Cp\_v - R1\_R\_actual * R1\_i \\ 0 = diode\_i - 10^{-9}(tmp53 - 1)10^{-8}diode\_v \\ 0 = D2\_i + diode\_i - D1\_i \\ \dfrac{C1\_v}{dt} = 9999.99999999999 \cdot C1\_i \end{cases} \tag{9}$$

$$\frac{Cp\_v}{dt} = Cp\_i/(10^{-12}\Theta) \tag{10}$$

4 Simulation Results

In Section 3 we discussed the practical integration of Θ in a Modelica compiler. In this section we will present our findings concerning concrete practical benefits of using this new construct. We do so by presenting two motivating examples, the first being a description of our results when simulating the model in Listing 3. The second example concerns simulation speedup when simulating the circuit depicted in Figure 3 compared to the circuit using Θ listed in Listing 4.

4.1 Simulating the DAE_Example

As discussed previously simulating **DAE_Example** without Θ resulted in undeseriable results, see Listing 2 both when using OpenModelica.jl and OpenModelica.

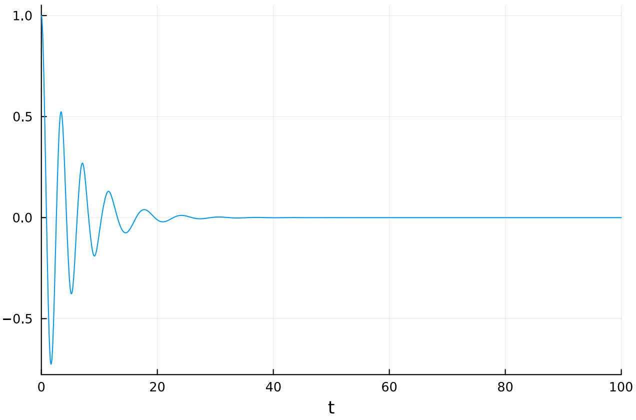


Figure 7. Simulation result showing the oscillation of x for Listing 2.

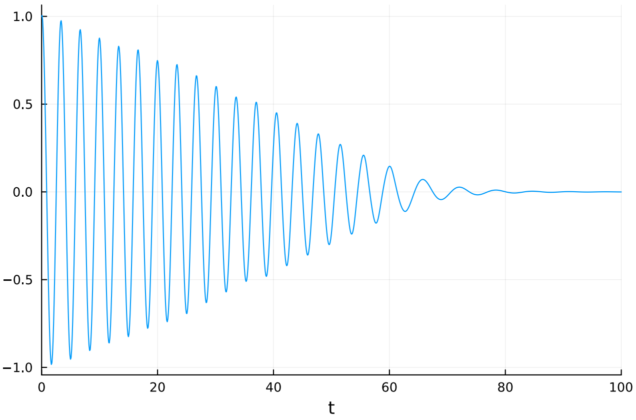


Figure 8. Simulation result showing the oscillation of x after theta has been applied.

Simulating the model using Θ as done in Listing 3 produces the correct plot; see Figure 8. As we did not have access to the original code nor to the model simulated as the small application example in (Zimmer 2013) the initial values of the system were assumed to be $x_0 = \{1, 0, 0\}$ for x, a and y respectively.

4.2 Simulating the Dynamic circuit using Θ

As previously mentioned, simulating the static circuit depicted in Figure 2 resulted in failure for the nonlinear solver.

Simulating the same system using the parasitic capacitance as depicted in Figure 3, leads to a successful simulation, however, the nonlinear system is complicated to solve leading to the solver having to take several time steps to integrate the system successfully, see Figure 9 for the plot of $C1.v$ for this circuit.

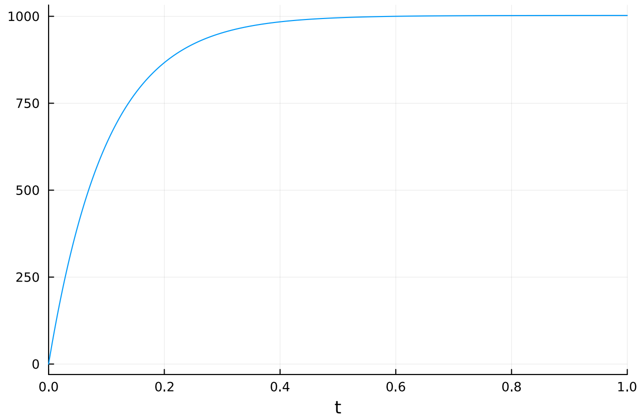


Figure 9. Simulation of $C1.v$ for the circuit in Figure 3 using Rodas5.

Using the method for code generation described in Section 3 simulation code was successfully generated for the model in Listing 4. We validated the solution of simulating the system using an infinitely fast sub-simulation by comparing the obtained results to the original results. There were no notable differences between the two simulations, see Figure 9 and Figure 10.

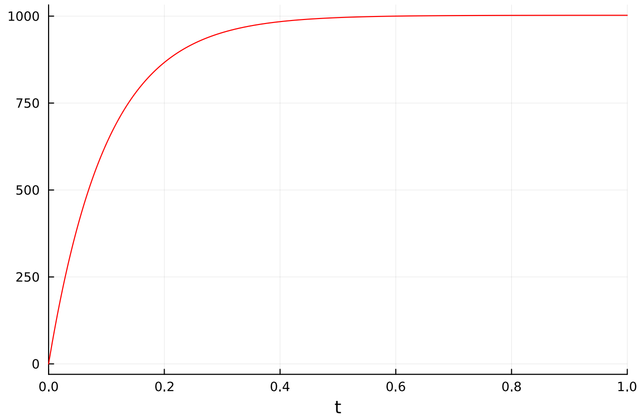


Figure 10. Simulation of $C1.v$ for the circuit model in Listing 3 using Tsit5.

Similarly, to the `DAE_Example` using Θ for the circuit model permits using solvers for non-stiff problems rather than stiff solvers such as DASSL (Petzold 1982). In this example, we compare the results of simulating `Circuit1Dynamic` using the following setup:

1. Simulating the System using the OpenModelica Compiler with the DASSL solver.

2. Simulating the System using OpenModelica.jl, the Julia-based Modelica Compiler with the Rodas5 solver[6]

3. Simulating the System using OpenModelica.jl, the Julia-based Modelica Compiler using Θ and the Rodas5 solver.

4. Simulating the System using OpenModelica.jl, the Julia-based Modelica Compiler using Θ with the explicit Tsit5 solver (Tsitouras 2011).

The simulations in the experiment had the absolute and relative tolerance levels set to $1e-6$. The experiment was run on a Laptop with an **AMD Ryzen 7 PRO 5850U with Radeon Graphics** and 32.0 GB internal memory, using Microsofts Subsystem for Linux with version **5.10.102.1-microsoft-standard-WSL2**. In terms of software, the Julia version used was **1.9-RC1** and the version of OpenModelica was **v1.21.0**.

Table 1. Solver statistics when simulating the dynamic circuit in Figure 3. C_{Rodas} refers to the simulation of the circuit without using Θ with the Rodas5 solver. $C_{\Theta Rodas}$ and $C_{\Theta TSIT5}$ refers to the result of simulating the same circuit using Θ.

Statistic	C_{Rodas}	$C_{\Theta Rodas}$	$C_{\Theta TSIT5}$
#Accepted Steps	109	36	39
#Rejected Steps	5	0	0
#Jacobians Created	109	36	0
#Linear Solves	912	288	0

The solver statistics for the models generated by the OpenModelica.jl are available in Table 1. As expected, we can see that using the Θ not only allows us to use explicit solvers such as Tsit5, it also reduces the amounts of integration steps needed to complete the simulation.

It is interesting to also compare the results with respect to the total simulation time for existing Modelica Compilers. For this purpose, we also compared the result of running the simulation using the OpenModelica Compiler. To benchmark the Julia code generated by OpenModelica.jl, we used Benchmarking software (Chen and Revels 2016) with the maximum number of samples set to 100. The simulation time for the OpenModelica Compiler was obtained by sampling the simulation statistics 10 times.

Table 2. Simulation Statistics comparing the simulation of `Circuit1Dynamic` using the OpenModelica Compiler (OMC), and the results of running the same model using Θ with the Julia-based Compiler, OpenModelica.jl (shorten to OM.jl in the table). The sample mean, median and standard deviation of the total simulation time are denoted $\hat{x}$, $\hat{M}$, and $\hat{\sigma}$ respectively.

OMC (DASSL)	$\hat{M}$	$\hat{x}$	$\hat{\sigma}$
	41.016ms	44.2496ms	11.984ms
OM.jl (Tsit5)	$\hat{M}$	$\hat{x}$	$\hat{\sigma}$
	13.709ms	14.801ms	1.892ms

The results are presented in Table 2; from these results, we can see that there is a clear speedup in the experimental compiler using this method, in this case, by about 2.9 times.

5 Related Work

The techniques discussed and implemented in this paper were proposed in Zimmer (2013) and further elaborated upon in Zimmer (2014).

A technique similar to the extension of the Modelica language presented in this paper is the `homotopy` operator. The `homotopy` operator was added to the Modelica language to provide an option for more robust initialization (Sielemann et al. 2011).

Artificial time integration in the context of Partial Differential Equations has been proposed and investigated by Ascher, Huang, and Van Den Doel (2007).

6 Conclusions and Future Work

In this article, we have demonstrated the usefulness of introducing a new construct, Θ in the equation-based language Modelica. We integrated support for Θ in OM.jl and we used two examples with an associated microbenchmark to illustrate its advantages. The example presented in Subsection 4.1 illustrates how more robust simulations can be achieved using Θ. The second example presented in Subsection 4.2, shows how Θ may speed up the total simulation time in models constructed using existing standard components.

However, several open questions remain unanswered. The first question is selecting a suitable initial step size for the sub-simulation. Currently, if the step size is too small the solver will need to take many steps. If it is too long, it corresponds to more or less to solving the algebraic equilibrium equation outright; in that case, this method will not be used.

Furthermore, the current implementation relies on the implicit solvers provided by the MTK-ecosystem. This is not optimal in this case because such solvers tend to report failures late, whereas in this case failures should be reported as early as possible. Moreover, such solvers save intermediate values resulting in unnecessary high mem-

[6] https://docs.sciml.ai/DiffEqDocs/stable/solvers/ode_solve/ Accessed 2023-04-25.

DOI
10.3384/ecp204423

ory consumption furthermore it also compute them with high precision and error control, which is not relevant in this case, only the asymptotic result is. While using the solvers from MTK worked for the example examined in this paper, a specialized embedded algorithm should be implemented instead.

An initial proposal of such an algorithm was proposed in (Zimmer 2013). Still, we believe such an algorithm could need further improvements. Improvements include utilize heuristics to select a suitable initial step size. Using an embedded subsimulation algorithm would also eliminate the need to save intermediate values, hence, reducing the memory footprint of the final simulation. As an extension to the work presented here further research should be invested to design and implement specialized algorithms and heuristics designed to be embedded for these purposes.

In this paper, we used the circuit model `ThetaCircuit2Dynamic` to illustrate how Θ could be applied to an existing model, `Circuit1Dynamic` listed in Listing 6, showing how existing models could integrate this without changing any low-level implementation. However, we have yet to investigate how well this new method scales when using nested sub-simulations. Hence, we should investigate the practical effects on larger models with more complex dependencies. This should be done both to finetune a possible heuristic for the initial step size of the sub simulations and gain even more insight concerning the robustness of the method.

To conclude we have examined the consequences when introducing a construct to an equation-based language to express infinitely fast processes; our experiments in Section 4 show clear net benefits of supporting Θ both in terms of speed and accuracy for the models that we tested.

Acknowledgements

This work has been supported by the Swedish Government in the ELLIIT project and has been partially funded by Vinnova in the context of the ITEA project EMBRACE. Furthermore, the authors gratefully acknowledge the financial support of the French Transmission System Operator RTE to the Open Source Modelica Consortium, which made this initial development possible.

References

Ascher, UM, H Huang, and K Van Den Doel (2007). “Artificial time integration”. In: *BIT Numerical Mathematics* 47, pp. 3–25. DOI: 10.1007/s10543-006-0112-x.

Casella, Francesco and Bernhard Bachmann (2021). “On the choice of initial guesses for the Newton-Raphson algorithm”. In: *Applied Mathematics and Computation* 398, p. 125991. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2021.125991. URL: https://www.sciencedirect.com/science/article/pii/S0096300321000394.

Chen, Jiahao and Jarrett Revels (2016-08). “Robust benchmarking in noisy environments”. In: *arXiv e-prints*, arXiv:1608.04295. arXiv: 1608.04295 [cs.PF].

Fritzson, Peter et al. (2020). “The OpenModelica integrated environment for modeling, simulation, and model-based development”. In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Ma, Yingbo et al. (2021). *ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling*. arXiv: 2103.05244 [cs.MS].

Petzold, Linda R (1982). *Description of DASSL: a differential/algebraic system solver*. Tech. rep. Sandia National Labs., Livermore, CA (USA).

Rackauckas, Christopher and Qing Nie (2017). “DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia”. In: *The Journal of Open Research Software* 5.1. DOI: 10.5334/jors.151. URL: https://app.dimensions.ai/details/publication/pub.1085583166%20and%20http://openresearchsoftware.metajnl.com/articles/10.5334/jors.151/galley/245/download/.

Sielemann, Michael et al. (2011). “Robust initialization of differential-algebraic equations using homotopy”. In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 063. Linköping University Electronic Press, pp. 75–85.

Tarjan, Robert (1971). “Depth-first search and linear graph algorithms”. In: *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pp. 114–121. DOI: 10.1109/SWAT.1971.10.

Tinnerholm, John, Adrian Pop, and Martin Sjölund (2022). “A Modular, Extensible, and Modelica-Standard-Compliant OpenModelica Compiler Framework in Julia Supporting Structural Variability”. In: *Electronics* 11.11, p. 1772.

Tsitouras, Ch (2011). “Runge–Kutta pairs of order 5 (4) satisfying only the first column simplifying assumption”. In: *Computers & Mathematics with Applications* 62.2, pp. 770–775.

Zimmer, Dirk (2013-03). “Using Artificial States in Modeling Dynamic Systems: Turning Malpractice into Good Practice”. In: *Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. Ed. by Henrik Nilsson. Linköping University Press, pp. 77–85. URL: https://elib.dlr.de/84089/.

Zimmer, Dirk (2014). “Handling Infinitely Fast Processes in Continuous System Modeling”. In: *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. EOOLT ’14. Berlin, Germany: Association for Computing Machinery, pp. 31–34. ISBN: 9781450329538. DOI: 10.1145/2666202.2666206. URL: https://doi.org/10.1145/2666202.2666206.

A Models

This appendix contains the Modelica models of the circuits depicted in Figure 2, Figure 3. It also contains some example models illustrating different ways the Θ operator can be used. In **ThetaCircuit1Dynamic** it is used as a part of a low-level submodel, **ThetaCapacitator** and in **ThetaCircuit2Dynamic** it is used in the topmost model to enable efficient code simulation. The components used are from the Modelica Standard Library; the package paths have been omitted. The annotations have been omitted.

```
package DAE_Examples
  function s
    input Real a;
    output Real oa;
  algorithm
    if (a < -1) then
      oa := a/4 -3/4;
    elseif (a > 1) then
      oa := a/4 + 3/4;
    else
      oa := a;
    end if;
  end s;

  model DAE_Example
    Real x(start = 1.0);
    Real y;
    Real a;
  equation
    der(x) = y;
    der(y) = -0.1*a - 0.4*y;
    der(a) = (10*x - s(a));
  end DAE_Example;

 model DAE_Example2
   Real x(start = 1.0);
   Real y;
   Real a;
 equation
   der(x) = y;
   der(y) = -0.1*a - 0.4*y;
   0 = (10*x - s(a));
 end DAE_Example2;


  model DAE_Example_THETA
    Real a;
    Real x(start = 1);
    Real y;
    parameter Real THETA = 1.0;
  equation
    der(x) = y;
    der(y) = -0.1*a - 0.4*y;
    der(a)*THETA = (10*x - s(a));
  end DAE_Example_THETA;
end DAE_Examples;
```

Listing 5. The DAE Example models.

```
package CircuitTests
  //Details and annotations are omitted
  model Circuit1Static
    extends Modelica.Icons.Example;
    Ground ground;
    StepCurrent stepCurrent(I = 1);
    Capacitor C1(C(displayUnit = "uF") =
     ↪  0.0001000000000000000001);
    Resistor R1(R = 1000);
    Diode D1(Ids = 1e-9, Maxexp = 40);
    Diode D2(Ids = 1e-9, Maxexp = 40);
    Diode D3(Ids = 1e-9, Maxexp = 40);
    Diode diode(Ids = 1e-9, Maxexp = 40);
  equation
    connect(C1.n, ground.p);
    connect(stepCurrent.p, ground.p);
    connect(R1.n, ground.p);
    connect(stepCurrent.n, C1.p);
    connect(C1.p, D1.p);
    connect(C1.p, D2.n);
    connect(D2.p, D3.p);
    connect(D1.n, D3.p);
    connect(D3.n, diode.p);
    connect(diode.n, R1.p);
  end Circuit1Static;

  model Circuit1Dynamic
    extends Circuit1Static;
    Capacitor Cnl(C (displayUnit = "F")=
     ↪  1e-12);
  equation
    connect(Cnl.n, ground.p)
    connect(diode.n, Cnl.p)
  end Circuit1Dynamic;

  model ThetaCapacitator
    extends OnePort(v(start=0));
    parameter Capacitance C(start=1)
     ↪  "Capacitance";
    parameter Real THETA;
  equation
    i = C*THETA*der(v);
  end ThetaCapacitator;

  model ThetaCircuit1Dynamic
    extends Circuit1Static;
    TestThetaMethod.ThetaCapacitator Cnl(C =
     ↪  1e-12);
  equation
    connect(Cnl.n, ground.p);
    connect(diode.n, Cnl.p);
  end ThetaCircuit1Dynamic;

  model ThetaCircuit2Dynamic
    parameter Real THETA = 1.0;
    extends Circuit1Static;
    Capacitor Cp(C = 1e-12 * THETA);
  equation
    connect(Cp.n, ground.p);
    connect(diode.n, Cp.p);
  end ThetaCircuit2Dynamic;

end CircuitTests;
```

Listing 6. The Circuit models discussing in Section 2.

A Modelica Library to Add Contact Dynamics and Terramechanics to Multi-Body Mechanics

Fabian Buse[1] Antoine Pignède[1] Stefan Barthelmes[1]

[1]Institute of System Dynamics and Control (SR), German Aerospace Center (DLR), Oberpfaffenhofen Germany,
{fabian.buse,antoine.pignede,stefan.barthelmes}@dlr.de

Abstract

The *Contact Dynamics* library extends the multi-body Modelica Standard Library with contact calculation to the environment, namely soft soil and hard obstacles. A focus is on *terramechanics*, i. e. wheels driving on soft and dry soil, and a handful of models are implemented. Additionally, a Hertz contact model for hard and elastic contact, between bodies themselves or to obstacles in the environment (e. g. rocks in the soft soil), is available as well. The capabilities of the library have been key in the development of rovers for planetary exploration such as the upcoming MMX mission to the Martian moon Phobos.

Keywords: Multi-Body mechanics, Contact dynamics, Terramechanics, Modelica library with external code

1 Introduction

When developing and analyzing off-road vehicles one of the most important factors is contact dynamics, more precisely the resulting forces and torques of a metal wheel driving on unprepared soft soil. This engineering branch known as "terramechanics" has proposed numerous of models with various levels of detail describing exactly this. Starting from agricultural and military applications it has also shifted in particular to rovers that explore celestial bodies in situ, as comes clear from section 5.

Along the growing interest in planetary exploration, usage of high-fidelity modeling and simulation has also increased for development and analysis of mobile robotic systems. Modelica already provides good material for multi-body dynamics but lacks the contact detection and reaction calculation of wheels driving in soft soil or of robot parts hitting obstacles or each other. To fill this gap, the planetary exploration group at DLR's Institute of System Dynamics and Control (SR) has developed a "Contact Dynamics" library that is the subject of this text.

The text is organized as follows: section 2 reviews theory of contact dynamics and software packages for Modelica and other simulation environments. The first main section starts with the library structure and ends with some useful additions for environment and contact object generation. Details about the models themselves (idea, equations) and how they perform in simple academic examples (verification) is subject of section 4, the second main section. Finally, some example applications are shown in section 5 and concluding remarks as well as an outlook to the future are given.

All footnote links are accessed August 7, 2023.

2 State of the Art

2.1 Contact Dynamics

The mechanics of bodies in contact with each other was first scientifically inspected by Hertz (1882). Since then, the field of rigid, elastic contact has in principle not (needed to) evolved much as comes clear from Flores and Lankarani (2016) that still builds on Hertz's work. But the advent of computers and simulation has led to a large number of models for reaction force calculation based on the penetration of the bodies in contact to correctly represent the resulting speed after contact and the energy dissipated. Another fundamental contact simulation technique based on exchange of impulse also exists and is widely applied in computer games, but is not pursued further here.

The detection whether two bodies are in contact or not, has seen a few algorithms like **G**ilbert-**J**ohnson-**K**eerthi *GJK* (Gilbert et al. 1988), **P**olygonal **C**ontact **M**odel (Hippmann 2004) (not restricted to convex shapes) or **M**inkowski **P**ortal **R**efinement *MPR*, also called XenoCollide[1]. Usually, before the expensive contact detection algorithm is run, the software checks whether the **A**xis-**A**ligned **B**ounding **B**oxes *AABB* overlap to quickly exclude object pairs definitely not in contact with each other.

The Hertz contact models, more precisely the nonlinear Hunt and Crossley model as explained in Flores and Lankarani (2016), and the MPR contact detection in the open source implementation `libccd`[2], are the source for the rigid body contact dynamics in the library.

The above-mentioned contact dynamics mainly deal with two objects colliding with each other. While this also gives usable solutions for a cylinder rolling on a cuboid (a wheel driving on flat soil), better solutions for a wheel driving in soft soil are possible. This is the "terramechanics" field whose modern analysis starts with the works of M. G. Bekker and agricultural machines engineers in the 1950s and 1960s. Chapter 2 of Wong (2008) compiles the latest knowledge at the beginning of the 21st century.

[1] `http://xenocollide.snethen.com/`

[2] `https://github.com/danfis/libccd`

DOI
10.3384/ecp204433

The highest level of detail in terramechanics is when the soil is simulated as discrete particles, each representing a small pack of grains. There is expertise also on this field at DLR SR and an in-house software `partsival` (Lichtenheldt et al. 2018). It would in theory be possible for Modelica to use `partsival` as external library, however this is not done (and won't be in the near future) for two reasons. First, discrete particles simulation requires much computer power, therefore only single wheel scenarios are possible in reasonable time. Modelica with Contact Dynamics on the other hand focus on full rover scenarios. Second, `partsival` not only simulates the particles but also the wheel dynamics themselves, thus there is no need to connect it to Modelica. A similar, divided, approach for development and analysis of the Opportunity and Spirit rovers was done with the *ARTEMIS* (Zhou et al. 2014) full rover simulation and the *COUPi* (Johnson et al. 2015) discrete particles single wheel simulations.

2.2 Contact Dynamics Simulation

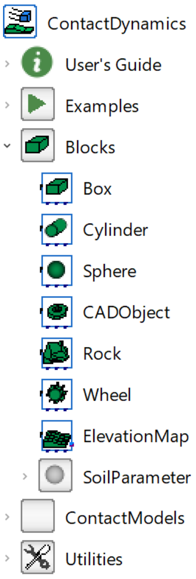
Bringing together multi-body and contact dynamics has already been done prior this work. The video game industry for example has released a dozen of *physics engines* that among other features compute contact dynamics. Emphasis there is more on visually appealing results and speed rather than on scientific accuracy and predictability of ground truth. Interestingly, the rover simulator *ROSTDyn* (Li et al. 2013) chose to code the contact dynamics themselves as a C++-library despite using the *Vortex*[3] engine that also has this capability.

Solutions for MATLAB are for example related in Tarokh (2016) and Ding et al. (2010).

Neumayr and Otter (2017) and Neumayr and Otter (2019) add collision detection and Hertz contact dynamics to *Modia3D*. The elastic, rigid body contact dynamics of the presented Contact Dynamics library share much in common with this work (MPR algorithm with AABB preprocessing, Hunt and Crossley model). However, the Contact Dynamics library goes further in that it adds a handful of contact models to Modelica. Next to the Hertz contact are specialized models for the main area of application: simulation of planetary exploration rovers, where terramechanics can be applied advantageously.

Contact dynamics for Modelica already exist as well. In fact, one can see the `ElastoGap` model of the Standard Library as a very simple contact dynamics block, as was first tried by the authors before the advent of *Contact Dynamics*. Two libraries that add contact calculation to the multi-body Standard Library, have been published so far to the knowledge of the authors. Elmqvist et al. (2015) enables Modelica for **D**iscrete **E**lement **M**ethod *DEM* using external binaries. Oestersötebier et al. (2014) uses only pure Modelica code to add punctual, linear and planar contact points to bodies, again using modern extensions of Hertz theory for the resulting forces. The free library *IdealizedContact*[4] released along this publication, is no longer maintained. Loading it into the 2022 Dymola[5] release and running the conversion script to the new Modelica Standard makes this library still usable today.

There have been a few attempts to connect Modelica to physics engines. For example, Hofmann et al. (2014) uses the collision detection capability from the *Bullet Physics*[6] as external C++-library, but Modelica for multi-body dynamics and reaction force calculations. The announced *CollisionLib* was not found by the authors. In a similar way, Bardaro et al. (2017) couples *Gazebo*[7] to Modelica. In the end, the aim of this work is more to integrate Modelica into the physics engine than to expand Modelica's capabilities with an external library.



3 Implementation

3.1 Contact Dynamics Library Overview

The base element of the Contact Dynamics library is a partial model that can be attached to any model through a multi-body frame. This base element provides a force and torque sensor, some standard parameters and small utilities such as an indicator whether a contact is present or not (this is *not* the contact detection).

Extensions from the base element come in three forms. Note that none of these blocks have mass.

- *Cuboid, cylinder, sphere, rock and CAD contact shapes:* Given parameters or path to a CAD file, the contact shape is added to the model, optionally visualized and the dynamics calculated using the BBCC or SCM contact models (or both). Primitive shapes are simulated as such while the rock asset generator subsection 3.2 is called to automatically create rocks satisfying the user's choices.

- *Wheel contact shape:* In addition to BBCC and SCM, specialized wheel-soil contact models, see subsection 4.3, can be used with this extension. The wheel asset generator subsection 3.2 is called.

- *Elevation map:* It provides two essential parts to the wheel-soil models: the geometry of the surface and the soil properties. As such it is always added as `outer` component to wheel contacts. The elevation map geometry is created using the surface asset

[3] `cm-labs.com/vortex-studio/`

[4] `github.com/oestersoetebier/IdealizedContact`

[5] `3ds.com/products-services/catia/products/dymola/`

[6] `https://pybullet.org/wordpress/`

[7] `http://gazebosim.org`

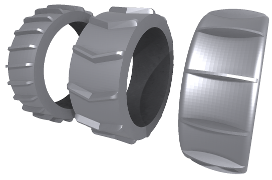


Figure 1. Examples of Wheel Generation. Left: slim with slanted grousers, center: wide with chevron grousers, right: wheel with convex curvature.

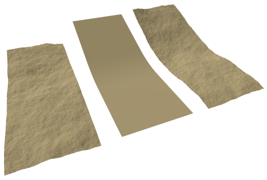


Figure 2. Examples of Surface Generation. Left: based on noise, center: based on a defined slope, right: combined.

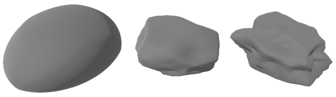


Figure 3. Examples of Rock Generation. Roughness increases from left to right.

generator subsection 3.2, the soil properties are parameters such as density and angle of repose.

The contact models are implemented independently from the contact elements. This ensures modularity and flexibility of the library. Generation of the contact shapes (cuboid, cylinder, wheel, ...) or loading from CAD is separated from the force and torque calculation too.

3.2 Assets

Part of Contact Dynamics is a procedural asset generator. This generator can generate four types of assets: wheels, surfaces, rocks and rock distributions. The procedural generation allows to create these assets based on a set of parameters and a seed. Using the same inputs will provide the same output. This feature allows the recreation of previously used assets when required.

The asset generator is a C++-library that takes the parametric definition of the various objects and provides two functions: a function to generate a unique ID for the given input and a function to generate the requested input. The Modelica interface uses these two functions to efficiently identify handles of already created objects and to create new files of not found handles. First, the ID of a requested object is generated, then the filename is defined as the combination of a type-specific prefix and the ID. Only if no file of this name already exists in the working directory, the actual wavefront `obj` file is generated and saved in the working directory with the desired name.

Wheels are generated based on an extensive parametric definition, including wheel radius, wheel width and multiple parameters defining the radial profile. This base shape can be extended by adding various features, like grousers similarly parametrized. Figure 1 shows three examples.

For surface generation, two base methods are available. The first method generates the surface based on external definitions like height maps or a profile in a single direction. The second method, procedurally generates the surface based on noise. See Buse et al. (2022) and Buse (2022) for more details on the method and validation of this noise-based terrain generation. The noise used is provided by the open source library `libnoise`[8]. These two methods can be combined: a statically defined surface based on a profile can be superimposed with a noise-based one to create more complex environments. See Figure 2 for three examples. On top of the generation of the mesh, the surface generator also allows convenient, resolution-independent access to surface information. The interface function allows access to the height and normal at given coordinates. This feature allows smooth integration with the contact models implemented in Modelica by providing the necessary information to approximate the local surface geometry into a single frame based at a given position.

Rocks are generated by deforming the surface of a sphere based on a function combining various noise types, see Buse (2022) for details. The inputs to this generation method are the average dimensions and two roughness parameters. Figure 3 shows three rocks with increasing roughness. For easier integration into a multi-body simulation, the rock generation process also computes the volume and center of gravity as well the rocks' inertia.

As rocks rarely appear alone, a generator for rock distributions combines information from the surface generator and the rock generator to create natural rock distributions. Based on a statistical distribution description, the rocks are placed on a previously generated surface. The output is either a single file including all rocks at their final position or the positions and individual files for each rock. These two options allow to either include static or dynamic rocks.

4 Contact Models in Detail

4.1 BBCC

The **B**ody to **B**ody **C**ontact model implemented in **C** *BBCC* calculates contact between two convex, rigid shapes. It is the most versatile of all models in the library and returns acceptable results in reasonable time. Special objects exist for cuboids, cylinders and spheres, a wheel is simply a cylinder with an arbitrary number of cuboids on the rim as grousers. Otherwise, a shape defined by a CAD file can be loaded, note however that the implementation of the contact detection results in effectively the convex hull being used. If wanted, collections of objects can be summarized into one "compound" object with the same properties but separate convex hulls. This is useful for scenarios with many (fixed) rocks on a surface. A surface

[8] `libnoise.sourceforge.net`

is implemented as collection of cuboids of unit height following the elevation and resolution.

The core of BBCC is an external library implemented in C that gets the position and velocity of all contact objects from Modelica and returns the resulting contact forces and torques on them. A `dll` for Windows and a shared library `so` for Linux are included as Modelica resources.

The calculation happens on a few layers. During the initialization of the simulation, a collection of objects is created in the external library containing basic information about the contact objects such as shape and size, elasticity parameters and references to other object with which collision is enabled. The Modelica BBCC shape saves this as an `ExternalObject` for later reference.

Each time instant the following is done to each object:

1. Update the position and velocity.
2. Determine whether another object is in contact:
   (a) Exclude all objects not explicitly marked as potential contact counterparts.
   (b) Exclude all objects whose AABBs do not overlap with the one of the current object.
   (c) Run the `libccd` MPR algorithm between the current and all other remaining objects.
3. Calculate the reaction forces and torques for all other objects not excluded above with the current one, see subsection 4.1 for details about the contact equations.
4. Return the sum of all forces and torques on the current object to Modelica.

As there potentially are many BBCC objects in a simulation and the order in which they're processed can't be controlled in Modelica, a special *synchronizer* is present at the top level of every model involving at least one BBCC object. It contains a connector with two variables, these are intertwined through an ordinary differential equation. This ensures that first all positions updates are sent to the external library, the contact forces are calculated only after this is done. Listing 1 shows the important code snippets for the connector, the synchronizer, the interface to the external code (`BBCC`) and the partial base class `BaseObject` from which cuboids, cylinders etc. in Modelica extend. This is similar to the synchronization idiom of Elmqvist et al. (2015). The major difference being that the external function calls are in the individual models (`BBCC`) instead of a common call in the synchronizer.

Listing 1. BBCC Synchronization

```
connector BBCC_ContactSynchronizer
  Real update;
  flow Real contact;
end BBCC_ContactSynchronizer;

model BBCC_Synchronizer
  BBCC_ContactSynchronizer sync;
initial equation
  sync.update = 0;
equation
  der(sync.update) = sync.contact;
annotation (defaultComponentName="bbccSync"
    , defaultComponentPrefixes="inner");
end BBCC_Synchronizer;

model BBCC
  BBCC_ContactSynchronizer sync;
  Real dummy;
  (...)
equation
  sync.contact = update(obj,r,v,T,w,time,
     dummy); // Update pos, vel, ...
  (force,torque) = getForce(obj,time,sync.
     update); // Calculate f, tau
  (...)
end BBCC;

partial model BaseObject
  outer BBCC_Synchronizer bbccSync;
  BBCC bbcc(dummy=bbccSync.dummy,...);
  (...)
equation
  connect(bbccSync.sync, bbcc.sync);
  (...)
end BaseObject;
```

4.1.1 Model

The normal force on objects in contact is (Flores and Lankarani 2016)

$$\|F_N\| = k\delta^{1.5}(1+d\dot{\delta}) \tag{1}$$

$$d = \frac{8}{5}\frac{1-\zeta}{\zeta\dot{\delta}^{(-)}} \tag{2}$$

$$k = \frac{4}{3}\frac{\sqrt{R}}{0.5\left(\frac{1-\nu_1^2}{E_1}+\frac{1-\nu_2^2}{E_2}\right)} \tag{3}$$

with penetration depth δ, relative penetration velocity $\dot{\delta}$ and relative impact velocity prior to contact $\dot{\delta}^{(-)}$. The resulting stiffness k and damping d are functions of the object parameters coefficient of restitution harmonic mean $\zeta = 2\frac{\zeta_1\zeta_2}{\zeta_1+\zeta_2}$, modulus of elasticity E and Poisson number ν. R is an estimate of the effective Hertz contact radius, currently estimated as half the harmonic mean of the longest edges of the objects in contact ($\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$). Note also that the intuitive meaning of the coefficient of restitution (ratio between pre- and post-impact velocity) is lost in this algorithm. In general $\zeta \neq \frac{\dot{\delta}^{(+)}}{\dot{\delta}^{(-)}}$, although this relationship is at the basis of the normal force derivation.

The default tangential contact model is

$$\|F_T\| = \mu\|F_N\|\tanh\left(\frac{\|v_T\|}{v_d}\right) \tag{4}$$

representing the Coulomb friction regularized by the velocity dead band (lower bound for velocities) parameter v_d, with $\mu = 2\frac{\mu_1\mu_2}{\mu_1+\mu_2}$ being the harmonic mean of the friction parameters of the objects in contact.

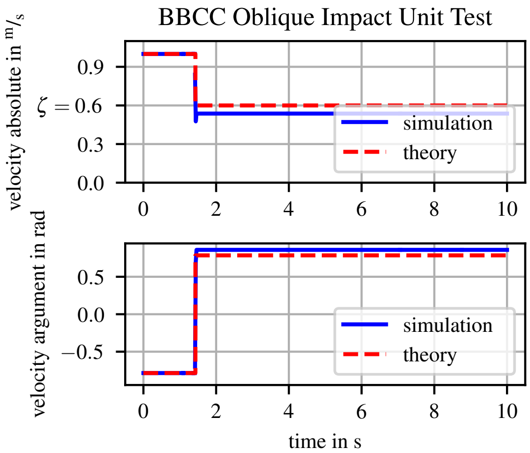


Figure 4. BBCC Impact Verification: Cylinder against cylinder in zero gravity

The user can provide two more parameters μ_s and ξ to activate a static tangential force model after Bengisu and Akay (Marques et al. 2016) which is stiction capable

$$\|F_T\| = \begin{cases} \|F_N\|\mu_s - \frac{\|F_N\|\mu_s}{v_d^2}(\|v_T\| - v_d)^2 & \forall\ \|v_T\| < v_d \\ \|F_N\|\mu + \|F_N\|(\mu_s - \mu)\mathrm{e}^{-\xi(\|v_T\| - v_d)} & \forall\ \|v_T\| \geq v_d \end{cases} \tag{5}$$

The torque in normal direction reads

$$\|\tau_N\| = \mu\|F_N\|\pi\sqrt{R\delta}^2 \tanh\left(\frac{\sqrt{R\delta}\|\omega\|}{vd}\right) \tag{6}$$

with the effective contact area $\sqrt{R\delta}$. There (currently) is no tangential torque in BBCC.

4.1.2 Verification

This text is the first publication that goes into detail about this contact model. Thus, this extensive verification subsection to prove the correctness of BBCC in academic examples and give hints about the performance in more practical applications, as those of section 5. The tests were done on a Windows 10 computer with Dymola 2022 and the `Esdirk45a` solver (tolerance 1×10^{-5}). The parameters used for all contact objects are given in Table 1.

Table 1. BBCC Verification Cylinder Parameters

Description	*Symbol*	*Value*
Radius	r	0.50 m
Height	h	1.00 m
Mass	m	1.00 kg
Inertia	I	0.25 kg m^2
Young	E	4.50×10^5 Pa
Poisson	ν	0.40
Restitution	ζ	0.60
Friction	μ	0.40

Definitions and results of the tests to verify BBCC:

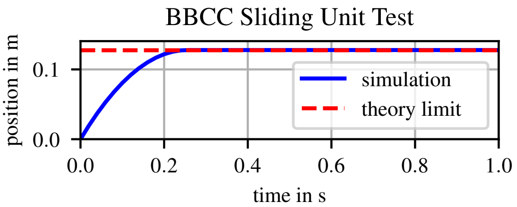


Figure 5. BBCC Sliding Verification: Non-rolling cylinder with initial velocity on flat plane

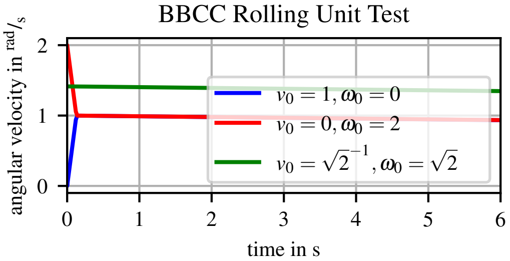


Figure 6. BBCC Rolling Verification: Free to roll cylinder with initial energy differently distributed on flat plane

- *Impact:* The wheel is initialized above the surface with an initial velocity oblique to the flat surface. Gravity is turned off and the expected outcome is that the incidence angles match.
  Figure 4 shows that the BBCC cylinder correctly applies the rules, that the incidence angle absolute value after impact is the same than before, that the velocity absolute value is reduced by the ζ, within an acceptable tolerance.
  Tests in Earth gravity with $\zeta = 1.0$ or 0.0 (full energy conservation or dissipation) verify successfully.

- *Sliding:* The wheel is initialized with an initial velocity tangential to the flat surface, the wheel can't rotate but must slide. The expected outcome is that the motion is slowed down because of friction.
  Figure 5 shows that the BBCC friction breaking of a cylinder with initial velocity on a cuboid is correct.

- *Rolling:* The wheel is initialized with an initial velocity tangential to the flat surface, the wheel can rotate. The expected outcome is that the motion is not slowed down (much) because of friction.
  The initial translational and rotational velocities of Figure 6 were chosen to have the same initial energy in all three cases. For the translational (blue) respectively rotational (red) initial energy only, conversion into matching rotational and translational velocities dissipates some energy compared to the case of equal initial energies (green). Following this, little losses due to rolling friction are seen with the same constant deceleration in the three cases.

- *Sliding on inclined plane:* The wheel is initialized with zero initial velocity on a flat, inclined surface, the wheel can't rotate but must slide. The expected

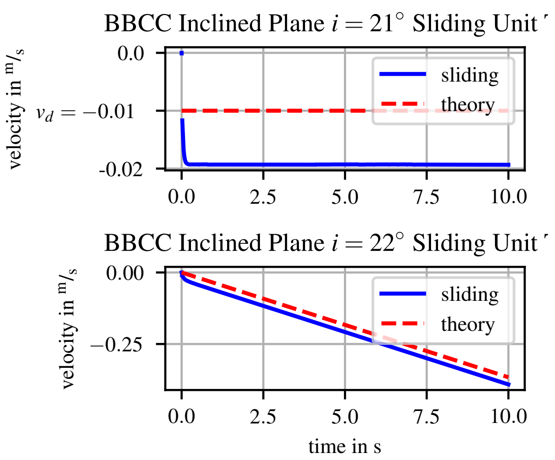


Figure 7. BBCC Sliding Verification on Inclined Plane: Non-rolling cylinder on inclined planes, $i \leq \arctan\mu$ or $i > \arctan\mu$

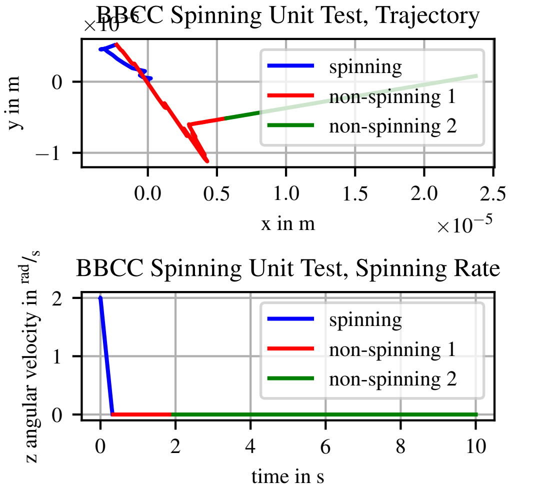


Figure 8. BBCC Spinning Verification: Cylinder with initial normal angular velocity on flat plane, the line in the plots is divided into three segments for easier understanding

outcome is that the motion is either zero (low inclination) or accelerated (high inclination).
If the inclination is *smaller* than the corresponding friction angle $\arctan\mu$ the cylinder correctly slides down the plane with constant velocity as visible in the upper plot of Figure 7. This velocity is higher than the expected v_d of Equation 4 because of the short time needed for the contact to be settled after initialization. If the inclination is *higher* than the corresponding friction angle $\arctan\mu$ the cylinder correctly slides down the plane with constant acceleration as visible in the lower plot of Figure 7. The velocity is higher than expected because of the short time needed for the contact to be settled after initialization, the acceleration though is correct.
If the cylinder can *roll down the inclined plane*, the expected constant acceleration regardless whether $i \leq \arctan\mu$ or $i > \arctan\mu$ is correctly simulated.

- *Spinning:* The wheel is initialized with an angular velocity normal to the flat surface. The expected outcome is constant deceleration because of friction.
  Figure 8 shows the correct almost complete energy dissipation. As the (x, y)-trajectory in the upper plot shows, a very little constant rest velocity remains.

The points above show the successful verification of the contact model against academic examples within an acceptable tolerance. This doesn't rule out unwanted behavior though, e. g. when stacking cylinders on top of each other on their mantle sides. The instability of the equilibrium points quickly brings the tower to fall. This effect can also be seen in stable equilibrium, e. g. stacking of cuboids. There, the contact points jump between corners of the same face and the cuboids never come completely to rest. Still, the movements remain small enough for practical time spans such that a cuboid tower doesn't fall.

4.2 SCM

The **S**oil **C**ontact **M**odel *SCM*, first presented in Krenn et al. (2008) and significantly advanced in recent years (Buse 2018; Buse 2022), aims to provide detailed terramechanical modeling in a form suitable for multi-body simulations. Explicitly modeling soil deformation SCM, or the newer version FSCM (*Flow based* Soil Contact Model), allows effects like ruts left behind by wheels or other permanent soil deformation to affect system behavior. The SCM model is a self-contained C++-library. The interface code contained in Contact Dynamics provides integration with Modelica and the DLR Visualization 2 Library.

The interface to SCM is divided into two main parts, SCM contact objects and SCM surfaces. SCM contact objects define geometries that can interface with the surface. Each object is defined by a mesh representing the geometry, position, velocity, orientation and angular velocity determining the current pose. For each of these objects, SCM provides the resulting reaction forces. The SCM surface defines the regolith surface, it is a stationary object which describes the surface geometry as well as parameters. Internally a horizontal equidistant grid is used, and the height of each node in the grid is used to represent the geometry. The Contact Dynamics blocks of type box, cylinder, sphere, CAD, rock and wheel represent the SCM contact objects. As SCM always relies on a mesh to represent the contact geometry, base meshes for a box, cylinder and sphere were manually created and placed in the library resources. These are then scaled to match the desired dimensions. For more complex shapes like rock and wheel, custom meshes can be generated, see subsection 3.2. The SCM surface is part of the elevation map block. An example of a single-wheel driving through soft regolith is shown in Figure 9.

SCM divides contact modeling into two functions: a contact dynamics function and a soil update function. The contact dynamics function is called once for each object and timestep, it computes the reaction forces based on the current object pose and the last known soil state. In

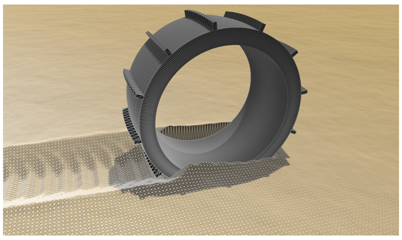


Figure 9. SCM Visualization: A wheel driving through soft soil. The grid representation of the surface is shown as wireframe.

the soil update function, the surface geometry and internal soil states are updated based on the last known object positions. The soil deformation results in a change of node height in the surface grid. These two functions are coupled to two sampled clocks in Modelica to control the rate at which these functions are executed independently.

To visualize the surface, a flexible surface from the DLR Visualization 2 Library is directly integrated with the SCM library, this allows a higher resolution compared to an integration through Modelica. This allows the visual representation of detailed rutting as shown in Figure 9.

4.2.1 Verification

Extensive validation of SCM is documented in (Buse 2022). In this campaign, the model's surface deformation and force prediction has been compared against measurements taken with the **T**erramechanics **Ro**botics **L**ocomotion **L**ab *TROLL*. This testbed allows automatic testing of various terramechanical experiments, see (Buse 2019). Figure 10 shows data from one scenario performed in the validation. SCM's predicted traction, normal and drive torque are compared with the measurements when the wheel is moved along the trajectory captured by the testbed. In the shown scenario, a wheel is placed on a flat surface and then vertically loaded to 100 N. After a short period, a movement combining a translational movement of the robot and a wheel rotation is started. A slip ratio of 60 % is enforced during this movement, thus the translational velocity is only 40 % of what the wheel's rotational velocity and radius would suggest.

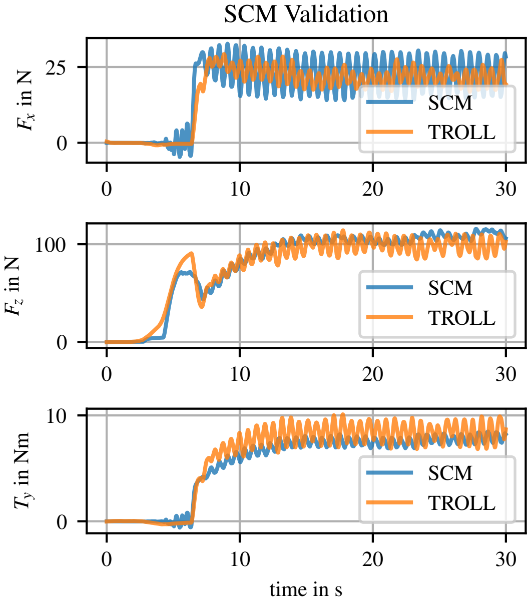


Figure 10. SCM single wheel comparison of normal force, traction force and drive torque. Terramechanics Robotics Locomotion Lab measurements in orange, simulated forces and torque when replaying the motion captured with the testbed in blue.

4.3 Terramechanical Wheel-Soil Models Directly Implemented in Modelica

These models are not general like BBCC, they are only valid for a wheel driving in soft soil.

The wheel contact shape, see subsection 3.1, contains a wider selection of models to calculate the contact dynamics than cuboids, cylinders and spheres. In contrast to the BBCC and SCM models that extend `oneFrame` multi-body interfaces, these other conditional models extend from `twoFrame`. The frame on the left-hand side is used again to get the position and orientation of the wheel and receives the reaction forces and torques. The right-hand side frame gets no forces but is connected to the elevation map and is used to detect whether the wheel is in contact, remember that contact detection is external in SCM and BBCC. Creation of elevation maps is explained in subsection 3.2. The equations governing the models are detailed in the following.

4.3.1 TerRA

TerRA is short for **Ter**ramechanics for **R**eal-time **A**pplications and is a purely empirical, fast computing terramechanics model developed by Barthelmes (2018). The scope of this model is to provide a model that captures the main effects of wheel-soil interaction while still being considerably faster than real-time to allow using it in onboard control software. The model captures dynamic slip-sinkage and its effects on the traction and resistance forces while not using any spatial discretization. TerRA consists of purely empirical relations and its parameters are not derived from any physical soil properties. They thus need to be tuned with a higher fidelity model or experimental data with the help of an optimization algorithm.

Model One very important effect in wheel-soil interaction is dynamic sinkage: A wheel sinks deeper into the soil for higher slippage and climbs out of its ditch once traction is sufficient to reduce slip. Typically, modeling these effects is either done with a spatial discretization of the ground to consider the sinkage as a position-dependent state or by altering the normal stress or normal force with the wheel slip. In TerRA a more explicit approach is developed, where the overall slip is divided into a sinkage that is effective for increasing traction and the slip-sinkage. The difference can be more easily understood when imagining a wheel with large grousers: If the wheel is pushed into the soil by normal force while not rotating, the soil below the wheel is mostly compressed, which increases the traction potential once the wheel starts rotating. If the wheel rotates but does not move forward, the grousers shovel the soil away, leading to sinkage of the wheel as well, however, the soil below the wheel is removed rather than compressed, which increases the traction potential much less.

In TerRA, the total sinkage is therefore composed of the effective and the slip-sinkage

$$z = z_{\text{eff}} + z_{\text{s}}. \tag{7}$$

The slip sinkage is calculated with a dynamic model from the inward and outward dynamic sinkage as

$$\dot{z}_{\text{s}} = \dot{z}_{\text{in}}(\omega r - v_x) + \dot{z}_{\text{out}}(z_{\text{eff}}, z_{\text{s}}, v_x, \alpha) \tag{8}$$

where $\omega r - v_x$ is the slip velocity, v_x the forward velocity and α the ground inclination. The exact relations contain several wheel/robot, soil and model parameters that result in a roughly exponential function that can be shaped for different soil and wheel types.

For the traction force, a maximum shear length is calculated by integrating the slip velocity over time. To consider changing conditions and new, unsheared, soil coming into contact, TerRA uses a special additional state that moves along the contact patch depending on the slip vs. forward velocities. While the standard Janosi-Hanamoto relations assume the shear length as a wheel state, the additional state in TerRA accounts for the fact that mainly the soil shear length results in traction potential.

Finally, a resistance force is calculated based on passive Earth pressure and therefore dependent on total sinkage.

Verification TerRA has several parameters that cannot be derived directly from physical properties of the wheel and soil. Therefore, the qualitative behavior as well as the tunability to the high-fidelity SCM model were investigated in Barthelmes (2018). The dynamic sinkage behavior of SCM can be replicated with TerRA to an error of less than 3 %, however, considerable differences remain especially in the dynamic drawbar pull force development.

Recently, TerRA is being used as one of three models of different fidelity for teaching a machine learning terramechanics model (Fediukov et al. 2022). Within this work, a better fitting between SCM and TerRA was achieved.

4.3.2 Other Terramechanics Models

Similar to TerRA, three other terramechanical models are implemented directly in Modelica, these can only be used for wheel contact shapes to a non-deformable surface. This subsection is only a short summary, because of the minor importance of these models in the Contact Dynamics library, interested readers are referred to Lichtenheldt et al. (2016) for a more elaborate discussion.

Two models attempt to implement Bekker's terramechanics equations following Chapter 2 in Wong (2008). Depending on the actual approach chosen ("pure" Bekker or Bekker-Janosi-Hanamoto) some differences in details and behavior are introduced, the parameters are also derived differently from wheel and surface (geometry and soil). But in the end, both are implementations directly in Modelica to compute the reaction forces on a wheel in a fast and easy way. Hence, precision and fidelity are low. Still, some basic effects in academic examples (sliding on an inclined plane) can be reproduced as expected.

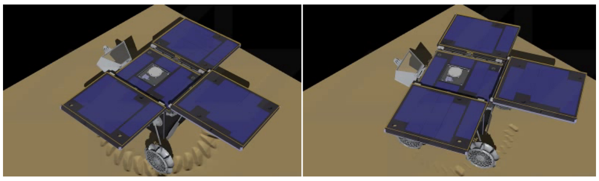


Figure 11. MMX Rover Point Turn Simulation on SCM Soil with Cohesion 20 Pa and 200 Pa

A third low fidelity, "rheological", terramechanics model for wheel to surface contact is implemented directly in Modelica as well. This one sees the ground as a spring-damper system and again derives parameters from wheel and soil properties. One interesting part of this model is the stiction capability, which is implemented using control logic elements: a PID-controller regulates the wheel frame to rest as long as the stiction force is not overcome, or if the wheel reaches a lower speed limit. The same academic tests as in the two Bekker models can be reproduced.

5 Applications

The Contact Dynamics library was originally completely integrated into the DLR **R**over **S**imulation **T**oolkit *RST* (Hellerer et al. 2017) but soon was extracted as standalone library. Planetary exploration rovers however, remain the main area of applications. RST essentially extends the contact blocks with rigid bodies from the Standard Library and adds further domains such as power and control logic. Two examples where DLR SR has applied the Contact Dynamics library are detailed in the following, with an emphasis on the contact dynamics and how the library has been key for these projects.

5.1 MMX

The **J**apan **A**erospace e**X**ploration **A**gency *JAXA* is in the preparation of a mission to the moons of Mars with sample return. This mission, known as **M**artian **M**oons e**X**ploration *MMX*, is carrying a rover jointly developed by the German DLR and French CNES to explore Phobos in situ (Ulamec et al. 2021). As detailed in Buse et al. (2022), simulations using the Contact Dynamics library were an integral part of the development process and will also be important for the operations and analysis phases.

Specifically, many mission phases were simulated with SCM for the contact of the wheels or other rover parts to the surface and BBCC for contact to rocks or between rover parts. Phobos is not completely unknown like comets and asteroids on first encounter, thus there are plausible number ranges about the topography and rock distribution. In fact, the asset generators for rocks and surface (subsection 3.2) were designed with these planetary science data in mind. Without the Contact Dynamics library, it would not have been possible to verify the sequence of movements to deploy the rover from its stowed configuration after landing on Phobos.

Contact Dynamics is also important to estimate driving performance. For example, Figure 11 shows the simulation of a point turn (90° if there would be no slip) on SCM soil with different cohesion values. Once the rover will have driven on Phobos and returned telemetry, validation of the contact model for milli-gravity will be undertaken.

The other contact models are not used for MMX.

5.2 Scout

DLR's institute of SR is currently developing a small, modular and highly agile rover for extreme terrain and cave exploration called *Scout* (Lichtenheldt et al. 2021). The team follows paradigms such as rapid control prototyping and model-based development (Pignède et al. 2022), simulation plays an important role and contact dynamics are central to the results, justifying the term "simulation driven development".

For example, the stiffness in the backbone was adjusted after an extensive simulation campaign where the rover was sent through an obstacle parcours with stairs, slopes etc. (Pignède and Lichtenheldt 2022). The BBCC model was used. A challenge in this activity is the huge number of objects. The nominal Scout rover consists of three modules (cuboid contact objects) with two wheels each. Each wheel has three spokes with an arc-like form simplified to two cuboids, and a foot at the end approximated as cylinder. This sums up to 57 contact objects for the rover to which 38 cuboids are added of the obstacle parcours, see Figure 12. Here, the BBCC capability to group objects into collections of pairs that are not tested for collision, were essential to keep the simulation time reasonable. Also, the test for overlapping axis-aligned bounding boxes filters out many pairs before calling the computationally expensive proper collision detection.

The simulation also serves to test new software before setting the prototype to risk. As low precision and fast simulation is often required, one of the simple terramechanics models of subsubsection 4.3.2 is used, with wheel contact objects of appropriate parameters as feet.

6 Conclusion and Further Work

The DLR *Contact Dynamics* library provides various types of contact dynamics to Modelica multi-body mechanics. It focuses especially on terramechanics for development and analysis of planetary exploration rovers but also includes two general models for contact between rigid, elastic bodies. Generators for environment and wheels are also included in the package. The structure and implementation of the library permits diverse applications at various levels of detail to assist engineers in all phases of projects from inception to post processing of field data. It's an integral part of the SR's toolchain for modeling, simulation and optimization of planetary exploration rovers and beyond, projects such as the MMX and Scout rovers much rely on Contact Dynamics.

This text has presented the library in general with more detailed sections about previously unpublished material. Simple models of the library are implemented directly in Modelica, more advanced ones are included as external code. These have been verified against ground truth (SCM), models of higher fidelity (TerRA) or academic examples (BBCC) to ensure validity of results generated using the library. The asset generator is a feature, unique in the Modelica world, to create random environments that meet statistical properties automatically.

Although the library is in good use already today, some tasks for further work remain. Currently only SCM has been validated against ground truth, BBCC should also go through this process. Verification of the models will be extended to milli-gravity environment using data collected by the MMX rover on Phobos. There also is potential to increase simulation speed with BBCC using multithreading as is already done with SCM.

Acknowledgements

The authors thank T. Bellman and his team, especially M. Hellerer, S. Kümper, R. Reiser and M. Neves.

References

Bardaro, Gianluca et al. (2017). "Using Modelica for advanced Multi-Body modelling in 3D graphical robotic simulators". In: *12th International Modelica Conference, Prague, Czech Republic*. Linköping University Electronic Press, pp. 887–894. DOI: 10.3384/ecp17132887.

Barthelmes, Stefan (2018). "TerRA: Terramechanics for Real-time Application". In: *5th Joint International Conference on Multibody System Dynamics, Lisbon, Portugal*. Springer.

Buse, Fabian (2018). "Using superposition of local soil flow fields to improve soil deformation in the DLR Soil Contact Model - SCM". In: *5th Joint International Conference on Multibody System Dynamics, Lisbon, Portugal*. Springer.

Buse, Fabian (2019). "Fully Automated Single Wheel Testing with the DLR Terramechanics Robotics Locomotion Lab (TROLL)". In: *15th Symposium on Advanced Space Technologies in Robotics and Automation, Noordwijk, Netherlands*. ESA.

Buse, Fabian (2022). "Development and Validation of a Deformable Soft Soil Contact Model for Dynamic Rover Simulations". Dissertation. Tohoku University.

Buse, Fabian et al. (2022). "MMX Rover Simulation - Robotic Simulations for Phobos Operations". In: *2022 IEEE Aerospace Conference, Big Sky, MT, USA*. IEEE. DOI: 10.1109/AERO53065.2022.9843391.

Ding, Liang et al. (2010). "Terramechanics-based high-fidelity dynamics simulation for wheeled mobile robot on deformable rough terrain". In: *2010 IEEE International Conference on Robotics and Automation, Anchorage, AK, USA*. IEEE, pp. 4922–4927. DOI: 10.1109/ROBOT.2010.5509217.

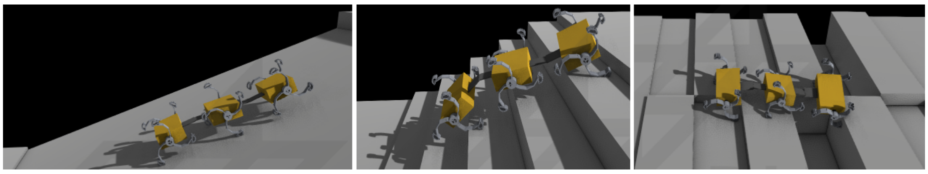


Figure 12. Scout Rover Obstacle Parcours Simulation with BBCC Contact Dynamics

Elmqvist, Hilding et al. (2015). “Generic Modelica Framework for MultiBody Contacts and Discrete Element Method”. In: *11th International Modelica Conference, Versailles, France*. Linköping University Electronic Press. DOI: 10.3384/ecp15118427.

Fediukov, Vladyslav et al. (2022). “Multi-Fidelity Machine Learning Modeling for Wheeled Locomotion on Soft Soil”. In: *11th Asia-Pacific Regional Conference of the ISTVS, Harbin, China*. ISTVS. DOI: 10.56884/WGPV6693.

Flores, Paulo and Hamid M. Lankarani (2016). *Contact Force Models for Multibody Dynamics*. Springer. ISBN: 978-3-319-30897-5.

Gilbert, Elmer G. et al. (1988). “A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space”. In: *IEEE Journal of Robotics and Automation* 4.2, pp. 193–203. DOI: 10.1109/56.2083.

Hellerer, Matthias et al. (2017). “The DLR Rover Simulation Toolkit”. In: *14th Symposium on Advanced Space Technologies in Robotics and Automation, Leiden, Netherlands*. ESA.

Hertz, Heinrich (1882). “Ueber die Berührung fester elastischer Körper”. In: *Journal für die reine und angewandte Mathematik* 1882.92, pp. 156–171. DOI: 10.1515/crll.1882.92.156.

Hippmann, Gerhard (2004). “Modellierung von Kontakten komplex geformter Korper in der Mehrkorperdynamik”. Dissertation. Technische Universität Wien.

Hofmann, Andreas et al. (2014). “Simulating Collisions within the Modelica MultiBody Library”. In: *10th International Modelica Conference, Lund, Sweden*. Linköping University Electronic Press, pp. 949–957. DOI: 10.3384/ECP14096949.

Johnson, Jerome B. et al. (2015). “Discrete element method simulations of Mars Exploration Rover wheel performance”. In: *Journal of Terramechanics* 62, pp. 31–40. DOI: 10.1016/j.jterra.2015.02.004.

Krenn, Rainer et al. (2008). “Contact Dynamics Simulation of Rover Locomotion”. In: *9th International Symposium on Artificial Intelligence, Robotics and Automation in Space Los Angeles, CA, USA*. ESA.

Li, Weihua et al. (2013). “ROSTDyn: Rover simulation based on terramechanics and dynamics”. In: *Journal of Terramechanics* 50.3, pp. 199–210. DOI: 10.1016/j.jterra.2013.04.003.

Lichtenheldt, Roy et al. (2016). “Wheel-Ground Modeling in Planetary Exploration: From Unified Simulation Frameworks Towards Heterogeneous, Multi-tier Wheel Ground Contact Simulations: Chapter 8”. In: *Multibody Dynamics*. Ed. by Josep M. Font-Llagunes. Springer, pp. 165–192. ISBN: 978-3-319-30614-8.

Lichtenheldt, Roy et al. (2018). “partsival – Collision-based Particle and many-body Simulations on GPUs for Planetary Exploration Systems”. In: *5th Joint International Conference on Multibody System Dynamics, Lisbon, Portugal*. Springer.

Lichtenheldt, Roy et al. (2021). “A Mission Concept For Lava Tube Exploration On Mars And Moon – The DLR Scout Rover”. In: *52nd Lunar and Planetary Science Conference*. LPI.

Marques, Filipe et al. (2016). “On the Frictional Contacts in Multibody System Dynamics: Chapter 4”. In: *Multibody Dynamics*. Ed. by Josep M. Font-Llagunes. Springer, pp. 67–91. ISBN: 978-3-319-30614-8.

Neumayr, Andrea and Martin Otter (2017). “Collision Handling with Variable-Step Integrators”. In: *8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, Weßling, Germany*. ACM, pp. 9–18. ISBN: 9781450363730.

Neumayr, Andrea and Martin Otter (2019). “Collision handling with elastic response calculation and zero-crossing functions”. In: *9th International Workshop on Equation-based Object-oriented Modeling Languages and Tools, Berlin, Germany*. ACM, pp. 57–65. DOI: 10.1145/3365984.3365986.

Oestersötebier, Felix et al. (2014). “A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces”. In: *10th International Modelica Conference, Lund, Sweden*. Linköping University Electronic Press, pp. 929–937. DOI: 10.3384/ECP14096929.

Pignède, Antoine et al. (2022). “Toolchain for a Mobile Robot Applied on the DLR Scout Rover”. In: *2022 IEEE Aerospace Conference, Big Sky, MT, USA*. IEEE. DOI: 10.1109/AERO53065.2022.9843816.

Pignède, Antoine and Roy Lichtenheldt (2022). “Modeling, simulation and optimization of the DLR Scout rover to enable extraterrestrial cave exploration”. In: *6th Joint International Conference on Multibody System Dynamics, 10th Asian Conference on Multibody Dynamics, New Delhi, India*. Springer.

Tarokh, Mahmoud (2016). “Kinematics-Based Simulation and Animation of Articulated Rovers Traversing Rough Terrain”. In: *2016 International Conference on Modeling, Simulation and Visualization Methods, Las Vegas, NV, USA*. WORLDCOMP, pp. 3–9.

Ulamec, Stephan et al. (2021). “The MMX Rover Mission to Phobos: Science Objectives”. In: *72nd International Astronautical Congress, Dubai, UAE*. IAF.

Wong, J. Y. (2008). *Theory of ground vehicles*. 4th. John Wiley & Sons, Inc. ISBN: 9780470170380.

Zhou, Feng et al. (2014). “Simulations of Mars Rover Traverses”. In: *Journal of Field Robotics* 31.1, pp. 141–160. DOI: 10.1002/rob.21483.

Bodylight.js 2.0 - Web components for FMU simulation, visualisation and animation in standard web browser.

Tomáš Kulhánek[1,2] Arnošt Mládek[1,2] Filip Ježek[2,3] Jiří Kofránek[1,2]

[1]Creative Connections s.r.o., Prague, Czechia
[2]Institute of Pathological Physiology, Charles University, Prague, Czechia {tomas.kulhanek, arnost.mladek, jiri.kofranek}@lf1.cuni.cz
[3]University of Michigan, Ann Arbor, USA fjezek@umich.edu

Abstract

Simulators used in teaching and education comprise a mathematical model and a user interface that allows the user to control model inputs and intuitively visualize the model states and results. This paper presents web components - that can be used to build an in-browser web simulator. The models used for the web simulators must be written in standard Modelica language and compiled as standard FMU (Functional mockup unit). The toolchain version Bodylight.js 2.0 contains tools to collect FMU into WebAssembly language, able to be executed directly by a web browser. Bodylight.js 2.0 web components can combine models, interactive animations, and charts into a rich web documents in HTML or Markdown syntax without any other programming or scripting. Samples show its usage in education, 2D and 3D graphics, virtual reality, and connected to the hardware.

Keywords: Modelica, JavaScript, WebAssembly, in-browser simulator, client-side simulator, e-learning, web components

1 Introduction

Web-based simulators can be distinguished by where the simulation computation is performed. The server-side simulators provide a user with an interface that controls simulation performed on a remote server, and the creation of such a simulator needs to employ client-server technologies. On the other hand, the client-side simulator's user interface and simulation computation are performed on a client's computer. This, however, comprises several issues that need to be addressed. First, a user may have different types of platforms; in the past, the central platform was Microsoft Windows-based system and therefore, many simulators were distributed as an installable applications on this platform. The locally installed application may need to be manually or semi-automatically updated or upgraded. Nevertheless, MS Windows-based systems are no longer significant platforms for computer or mobile devices.

One can address many different platforms, e.g. by virtualization using technologies such as VirtualBox, VMWare, or containerization such as Docker, etc. However, web standards developed into mature versions, and the vendors of contemporary web browsers cover many platforms, including mobile phones and tablets, giving standard HTML and JavaScript capabilities.

Mathematical models in biomedical engineering can be expressed in different languages or technologies. One is the Modelica language, which covers broad industry domains; therefore, commercial and open-source tools are available. Modelica is very well suited for usage in the physiology domain and biomedical teaching, as discussed elsewhere (Kofránek, Ježek, and Mateják 2019), though, it is not yet widely used in physiology modeling community.

Direct solving of Modelica models in a web browser were demonstrated, e.g., by Franke (Franke 2014). However, accurate web-based client simulation or in-browser simulation was prototyped by Short (Short 2014) and realized in the "Modelica By Example" and "Modelica University" by Tiller and Winkler (M. M. Tiller 2014; Winkler and M. Tiller 2017).

This inspired our team to create an in-browser simulator. We already published a technology called Bodylight.js (Šilar, Ježek, et al. 2019) and sample web simulators, e.g., kidney functioning model (Šilar, Polák, et al. 2019).

The present paper describes the next evolutionary stage of this set of open-source tools, titled Bodylight.js-Components version 2.0. These are distributed as framework-agnostic web components (WebComponents 2021) - i.e., custom elements enhancing the syntax of HTML or Markdown. Further sections describe a brief methodology for creating a web simulator from a model source. A demo is presented with a pulsatile heart web simulator combining buttons, sliders, interactive graphics, and charts. The main aim of the methodology is to enable creative cooperation among domain experts such as computer graphics designers, model developers, educators, and programmers (Figure 1). Their expert work results can be integrated with the Bodylight toolchain.

2 Methods

Modelica model must be exported as FMU. We have prepared scripts to compile such output into WebAssembly using the EMScripten SDK tools. Then the resulting JS with embedded WebAssembly can be controlled using

DOI
10.3384/ecp204443

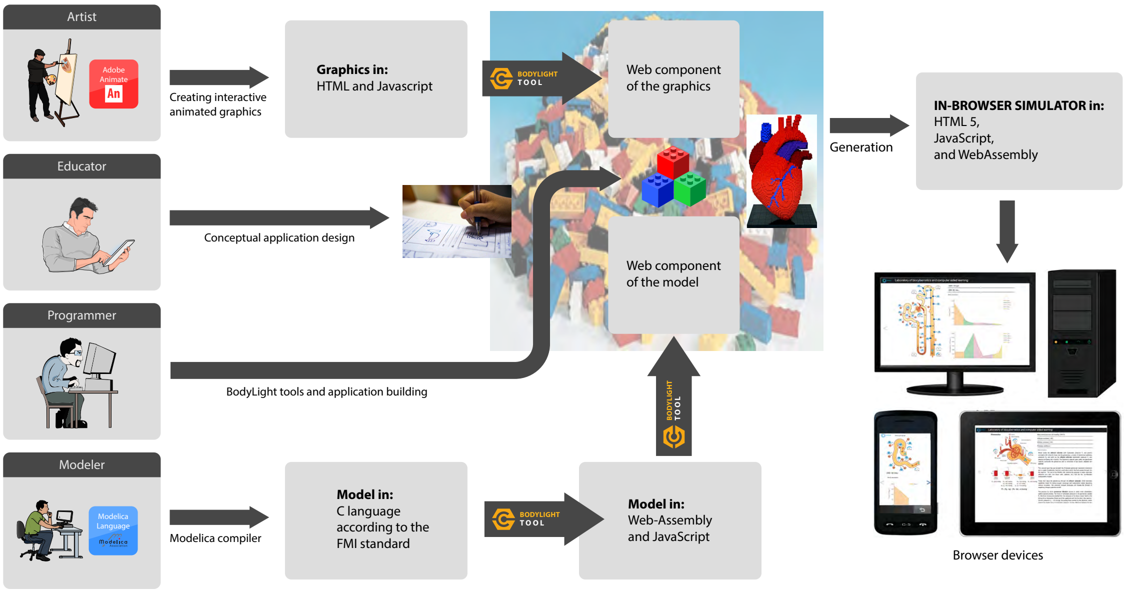


Figure 1. The main aim of the methodology is to enable creative cooperation among different domain experts such as computer graphics designers, model developers, educators, and programmers.

FMI API calls. We have prepared the web-component BDL-FMI that simplifies controlling and integrating it with other simulation related tasks like drawing charts, changing model parameters, reseting the simulation and visualising in 2D and 3D graphics. Next subsections describes the details of each particular step.

2.1 Model to WebAssembly

Modelica model must be exported as FMU v2.0 in co-simulation mode, including C source codes. This can be done either with an advanced CVODE solver in (Dymola 2023) (Dassault Systemes) or only with a more straightforward Euler solver in OpenModelica (Fritzson and et.al. 2019). Then the FMU with included source codes of solver can be compiled to JavaScript with embedded WebAssembly using Bodylight.js-FMU-Compiler[1]. It contains scripts and configuration to utilize the emscripten (EMScripten 2021) library.

In further text, the sample simulator uses exported model from Physiolibrary as seen in Figure 3.

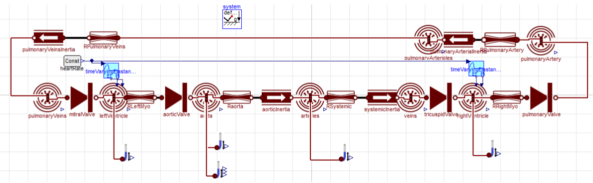
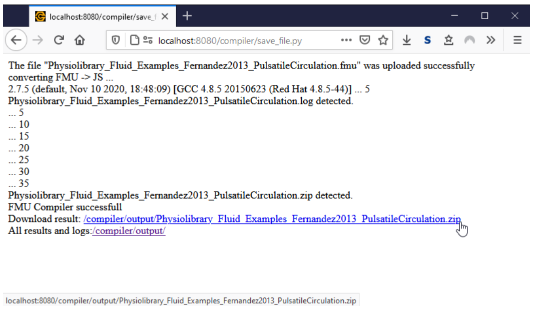


Figure 3. Model of pulsatile circulation (Fernandez de Canete et al. 2013; Kulhánek et al. 2014) in Chemical library(Matejak et al. 2015) and Physiolibrary(Mateják et al. 2014) v 3.0 using Modelica Standard Library v 4.0(Library 2021). This model is used in following sample export and web simulator.

A simple web form facilitates compilation as seen in Figure 4.


localhost:8080/compiler/save_f
localhost:8080/compiler/save_file.py

```
The file "Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.fmu" was uploaded successfully
converting FMU -> JS ...
2.7.5 (default, Nov 10 2020, 18:48:09) [GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] ... 5
Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.log detected.
... 5
... 10
... 15
... 20
... 25
... 30
... 35
Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.zip detected.
FMU Compiler successfull
Download result: /compiler/output/Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.zip
All results and logs:/compiler/output/
```

localhost:8080/compiler/output/Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.zip

Figure 4. Bodylight FMU Compiler - web form showing process of compiling FMU to JS packed as ZIP archive

[1] Bodylight.js-FMU-Compiler https://github.com/creative-connections/Bodylight.js-FMU-Compiler

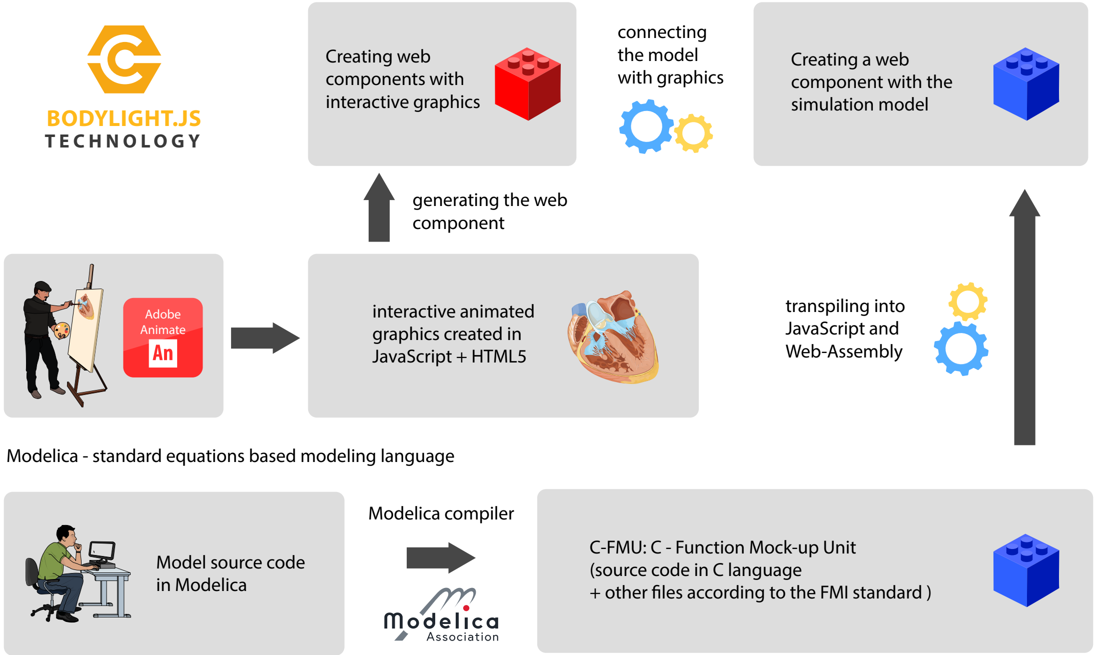

BODYLIGHT.JS
TECHNOLOGY
Creating web components with interactive graphics
connecting the model with graphics
Creating a web component with the simulation model
generating the web component
Adobe Animate
An
interactive animated graphics created in JavaScript + HTML5
transpiling into JavaScript and Web-Assembly
Modelica - standard equations based modeling language
Model source code in Modelica
Modelica compiler
Modelica Association
C-FMU: C - Function Mock-up Unit
(source code in C language
+ other files according to the FMI standard)

Figure 2. Presented web simulator creation technology is based on open web standards and available modeling standards. We create interactive animated graphics in Adobe Animate published with CreateJS library as JavaScript controlling an HTML canvas. Such an artifact is encapsulated as a web component. A model created in the Modelica language is exported into FMU with source codes, following FMI 2.0 standards. Our technology then can compile the FMU with C source codes into JavaScript with embedded WebAssembly. This artifact can then be encapsulated into another web component. Bodylight.js Components make it easier to link the graphics web component to the model's web component and create animated graphics like a model-controlled puppet.

2.2 Web components of Bodylight.js

Compiled FMU can be controlled using FMI API standard calls. However, Bodylight.js-Components[2] contains a set of components to simplify interactions among low-level FMI API, some standard HTML elements, third-party charting libraries, and 2D and 3D graphical animations.

The components are distributed as custom elements using standard WebComponent API (WebComponents 2021). It was developed using mainly Aurelia (AureliaJS 2023) framework, however, it can be used in any contemporary web application development framework or framework-agnostic way.

2.3 Changing user input, range web-component

The following sample web component defines HTML slider input and essential interaction (value change is sent as a custom HTML Event). The attributes can determine minimum, maximum, default value, and step by which the slider can change its value when moved right or left (Listing 1, Figure 5).

Listing 1. Bodylight Range Component with optional attributes (in blue), limiting user input between 40 and 180 with a step of 1 and default value 60

```
<bdl-range
  id="id1" title="Heart Rate"
  min="40" max="180" default="60" step="1">
</bdl-range>
```

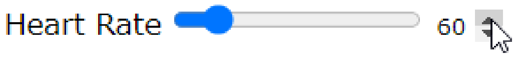


Figure 5. Range component rendered in a web browser

2.4 Control of simulation computation, FMI web-component

The following sample web component instantiates FMU from compiled JavaScript and creates standard HTML buttons to start/stop the simulation (Listing 2). When the simulation begins, a custom HTML event is sent to all potentially listening components. In every simulation step, a list of variable values is distributed as an array. The list of variables is set in `valuereferences` attribute. The components listed in `inputs` are listened to obtain interactively values the user changes during simulation.

The browser's `window.requestAnimationFrame()` method is used to call a simulation step. The browser usually calls this method up to 60 times per second to

[2] Bodylight.js-Components https://github.com/creative-connections/Bodylight.js-Components

deliver a smooth user experience to match the refresh rate of the window as well as the performance of the viewing window. This call is usually paused in most browsers when running in background tabs.

Listing 2. Declaration of Bodylight FMI Component. Instantiates model of human pulsatile circulation dynamics from Physiolibrary(Mateják et al. 2014). Setup output values to be only pressure of pulmonary veins and arteries. Input is listened from an element with id1 and changed values are set as input to heartRate parameter which is multiplied by 1 and divided by 60 (converting ’per minute’ to ’per second’ unit expected by the model).

```
<bdl-fmi id="idfmi" src="
    Physiolibrary_Fluid_Examples_Fernandez
2013_PulsatileCirculation.js" fminame="
    Physiolibrary_Fluid_Examples_Ferna
ndez2013_PulsatileCirculation" tolerance="
    0.000001" starttime="0" fstepsize="0.01
    " guid="{
    a786b906-f58b-4014-8c9b-5df08bd77f4b}"
    valuereferences="637534263,637534417"
    valuelabels="
    pulmonaryVeins.pressure,arteries.pressure
    " inputs="id1,16777329,1,60"
    inputlabels="heartRate.k">
</bdl-fmi>
```

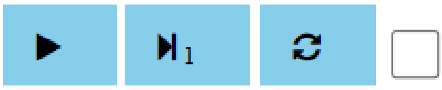


Figure 6. FMI component rendered in a web browser.

2.5 Charting web-components

Charts can make basic visualization of the data obtained from simulation. Bodylight.js library embeds open-source ChartJS (ChartJS 2021) library to support basic line charts using the component `<bdl-chartjs-time>`, see sample component listing in Listing 3. The component `<bdl-chartjs>` supports doughnuts, pie charts, and bar charts.

Listing 3. Bodylight Chart Component taking first one (indexed from 0) value of output values and converts it using expression $\frac{x}{133.322} - 760$ thus converting from Pa to mmHg and deducting ambient normal atmospheric pressure 760 mmHg

```
<bdl-chartjs-time
  id="id10" width="300" height="200" fromid
    ="idfmi"
  labels="Pressure in Aorta [mmHg]"
      initialdata="" refindex="0" refvalues
    ="1"
  convertors="x/133.322-760">
</bdl-chartjs-time>
```

Initially the chart is empty, however, it is connected to the FMI component and listens to any data obtained from it and draws it interactively as seen in Figure 7. Bodylight.js-Components externally supports time series charts made by Plotly(Plotly 2021) and Dygraphs(Dygraphs 2021) libraries too.

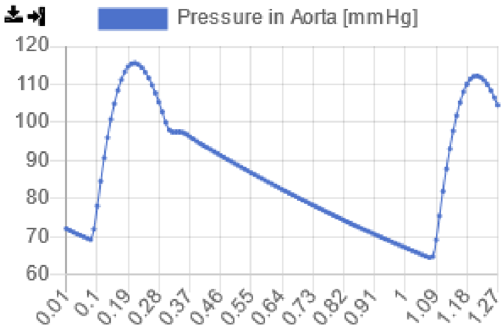


Figure 7. Chart component rendered in a web browser. This chart contains data obtained from FMI component during simulation from time 0 - 1.27s.

2.6 Interactive animation, adobe web-components

Adobe Animate is a multimedia authoring and computer animation program developed by Adobe Inc. Advanced visualization can be exported following as "standardized" open-source Javascript API (CreateJS 2023). By convention, an artist who creates interactive animation names all animatable elements with the suffix ’_anim’ and animation states between some values e.g. between 0 to 99 which visualizes the animation state. See the following listing (Listing 4).

Listing 4. Bodylight Animate component and components to bind animation element with model variable

```
<bdl-animate-adobe src="CardiaccycleStage.js" name="Faze_srdce"
      fromid="idfmi">
</bdl-animate-adobe>
<bdl-bind2a findex="1" aname="ValveMV_anim" amin="99" amax="0"
      fmin="0" fmax="1"></bdl-bind2a>
<bdl-bind2a findex="2" aname="ValveAOV_anim" amin="0" amax="99"
      fmin="0" fmax="1"></bdl-bind2a>
<bdl-bind2a findex="3" aname="ValveTV_anim" amin="99" amax="0"
      fmin="0" fmax="1"></bdl-bind2a>
<bdl-bind2a findex="4" aname="ValvePV_anim" amin="0" amax="99"
      fmin="0" fmax="1"></bdl-bind2a>
<bdl-bind2a findex="5" aname="
      ventricles.ventriclesTotal.VentricleLeft_anim" amin="100"
      amax="0" fmin="0.00015" fmax="0.00021"></bdl-bind2a>
<bdl-bind2a findex="6" aname="
      ventricles.ventriclesTotal.children.0.VentricleRight_anim"
      amin="100" amax="0" fmin="0.00012" fmax="0.00018"></
      bdl-bind2a>
```

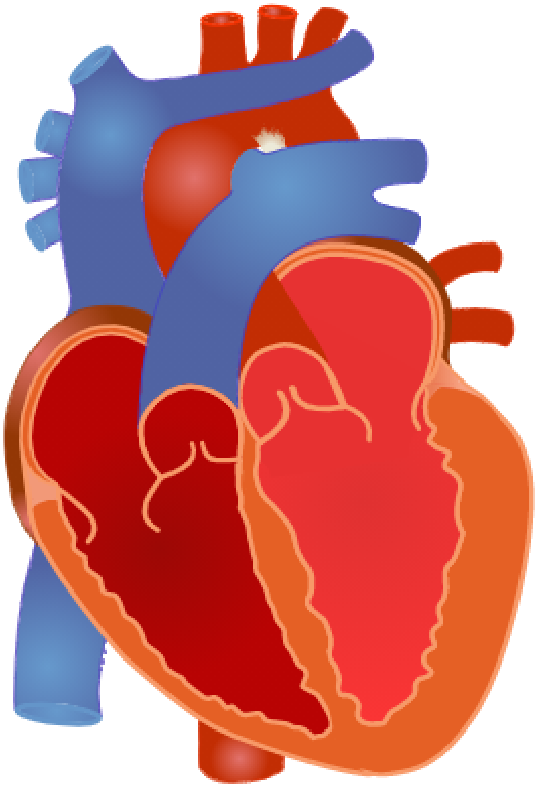


Figure 8. Animated component rendered in a web browser

2.7 Sample demo web simulator

All the previously defined component instances can be put into a single HTML page as seen in the following listing:

Listing 5. index.html containing declaration and use of web components. The aurelia.js framework was used to leverage building the web components thus the attribute 'aurelia-app' points out the DOM where web components can be located and corresponding implementation is injected there

```
 <!DOCTYPE html>
<html>
  <head>
   <script src="bodylight.bundle.js"></script>
 </head>
<body aurelia-app="webcomponents">
  <bdl-range id="id1" ...></bdl-range>
  <bdl-fmi d="idfmi" ...></bdl-fmi>
  <bdl-chartjs-time id="id10" ...></bdl-chartjs-time>
  <bdl-animate-adobe ...></bdl-animate-adobe>
  <bdl-bind2a findex="1" ...></bdl-bind2a>
  <bdl-bind2a findex="2" ...></bdl-bind2a>
  <bdl-bind2a findex="3" ...></bdl-bind2a>
  ...
 </body>
 </html>
```

The "index.html" must be published along the JavaScript file containing compiled FMU from the Modelica model: `Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.js`, JavaScript file containing published animation from Adobe Animate: `CardiaccycleStage.js` and "bodylight.js" library `bodylight.bundle.js`. However the Bodylight.js is published as NPM package and therefore can be taken from some content delivery network (CDN) caching NPM packages.

The resulting application is rendered in a web browser as seen in Figure 9.

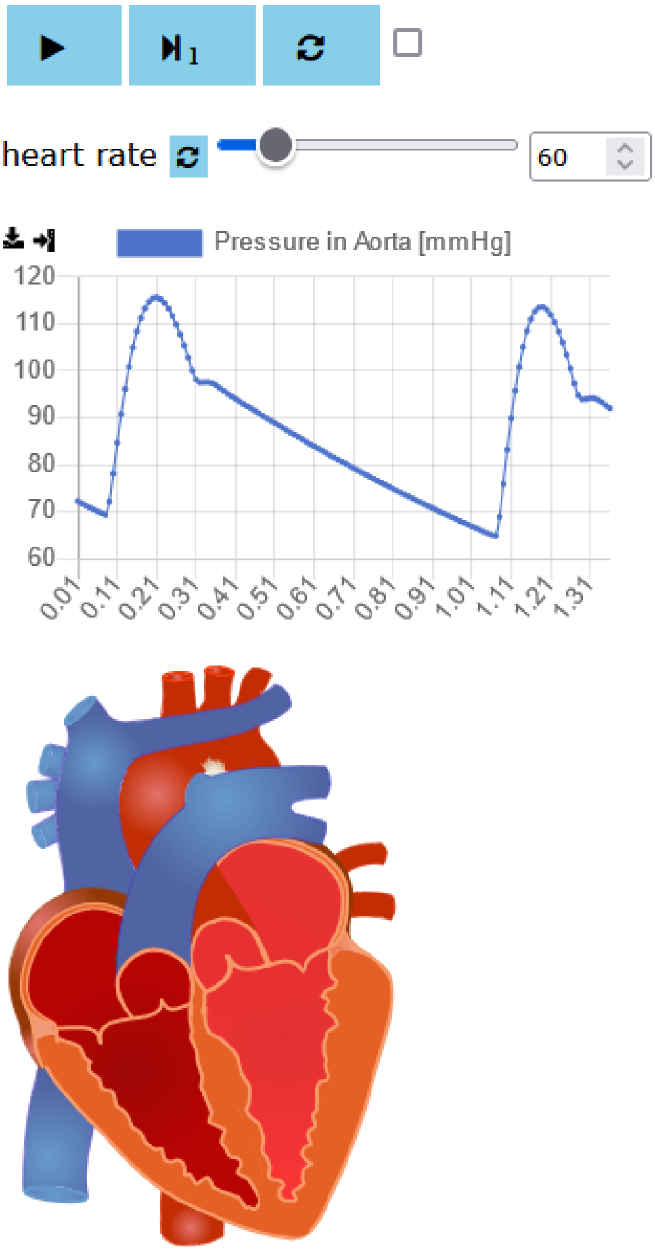


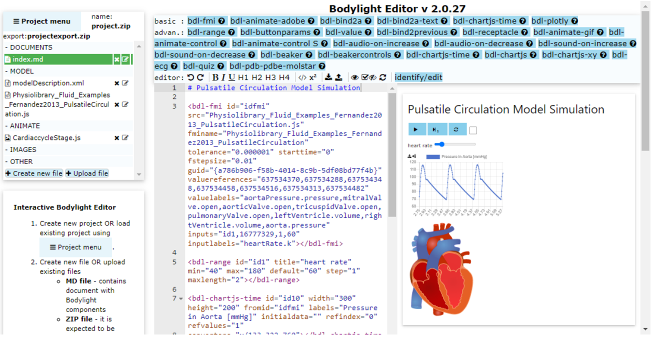
Figure 9. Web Simulator with rendered web components. The simulator can be started/restarted with buttons and the "heart rate" parameter can be changed by user interactivelly while computation of simulation is performed. Chart data is updated accordingly and animation is driven by the model variables.

2.8 Bodylight Editor

Optional tool Bodylight-Editor[3] is distributed as a static web page and allows a live preview of Markdown syntax as well as Bodylight.js-Components. Additional dialogs

[3] Bodylight-Editor https://github.com/creative-connections/Bodylight-Editor

facilitate filling the component attribute values, e.g., selecting input/output variables from the model and binding them into the appropriate component. The file management panel simplifies managing multipage documents sharing models, images, and animation, and generates multipage web simulators with shared navigation (Figure 10).



Bodylight Editor v 2.0.27

≡ Project menu
name: project.zip
export:projectexport.zip
- DOCUMENTS
index.md
- MODEL
modelDescription.xml
Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.js
- ANIMATE
CardiaccycleStage.js
- IMAGES
- OTHER
+ Create new file + Upload file

basic : bdl-fmi bdl-animate-adobe bdl-bind2a bdl-bind2a-text bdl-chartjs-time bdl-plotly
advan.: bdl-range bdl-buttonparams bdl-value bdl-bind2previous bdl-receptacle bdl-animate-gif bdl-animate-control bdl-animate-control S bdl-audio-on-increase bdl-audio-on-decrease bdl-sound-on-increase bdl-sound-on-decrease bdl-beaker bdl-beakercontrols bdl-chartjs-time bdl-chartjs bdl-chartjs-xy bdl-ecg bdl-quiz bdl-pdb-pdbe-molstar
editor: B I U H1 H2 H3 H4 x² identify/edit

```
# Pulsatile Circulation Model Simulation

<bdl-fmi id="idfmi"
src="Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.js"
fminame="Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation"
tolerance="0.000001" starttime="0"
fstepsize="0.01"
guid="{a786b906-f58b-4014-8c9b-5df08bd77f4b}"
valuereferences="637534370,637534288,637534348,637534458,637534516,637534313,637534482"
valuelabels="aortaPressure.pressure,mitralValve.open,aorticValve.open,tricuspidValve.open,pulmonaryValve.open,leftVentricle.volume,rightVentricle.volume,aorta.pressure"
inputs="id1,16777329,1,60"
inputlabels="heartRate.k"></bdl-fmi>

<bdl-range id="id1" title="heart rate"
min="40" max="180" default="60" step="1"
maxlength="2"></bdl-range>

<bdl-chartjs-time id="id10" width="300"
height="200" fromid="idfmi" labels="Pressure
in Aorta [mmHg]" initialdata="" refindex="0"
refvalues="1"
```

Interactive Bodylight Editor

1. Create new project OR load existing project using ≡ Project menu.
2. Create new file OR upload existing files
   - **MD file** - contains document with Bodylight components
   - **ZIP file** - it is expected to be

Pulsatile Circulation Model Simulation

Figure 10. Bodylight Editor with the sample components above and rendered preview.

2.9 Bodylight Virtual Machine

Bodylight.js toolchain comprises several independent tools, some of which need non-trivial configuration. Therefore, we have created an exemplar virtual machine configuration for the Vagrant tool and virtualBox, to provision a standard minimal image of CENTOS Stream 9; scripts are published as Bodylight-VirtualMachine[4].

3 RESULTS

We compared the performance of model simulation translated to FMU executed natively with the implementation of the same model translated to FMU and WebAssembly and performed in a web browser on the same machine. We used Chrome browser version 97.0.4692.71 with simulation times of native code on the same platform (win-64) and performed a simulation that took 6000 steps. Natively it took an average of 9.3 s, while the simulation in the web browser took 34.5 s (1, column 'WASM 1 step').

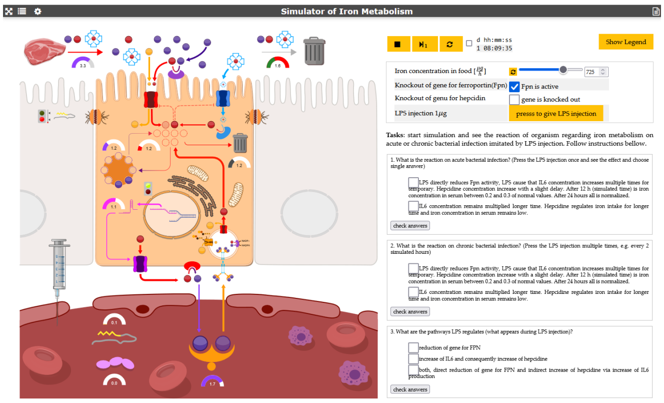
simulation	win64 bin	WASM (1 step)	WASM (3 steps)
time [s]	9.3s	34.5s	10.4s
relative [1]	1x	3.71x	1.12x

Table 1. Sample model simulation performance comparison between binary execution of FMU in win-64 and FMU translated to WASM and performed 1 or 3 FMU step() during web browser frame.

This difference might be explained by overhead due to the browser screen refresh framerate. Therefore we modified the WASM code to perform 2, 3, and 4 FMU step() calls during a frame given by the browser via requestAnimationFrame(). The browser allows max 60 frames per second when used and usually maintains a maximum—of thirty frames to support the smooth running of other apps and the operating system itself. Making more than three steps within one frame gave no better value (result not shown). Therefore in the following table, we offer times in column 'WASM 3 steps'. Thus, it can be concluded that the simulator's performance in WebAssembly (when doing multiple simulation steps during one frame) is comparable with native code (i.e., 1.12x or 12% slower than native code). This result also agrees with the more comprehensive benchmarks of WebAssembly vs. native code given by (Jangda et al. 2019).

We also measured the performance of the simulation with visualisation of charts and animation. It may significantly affect performance as the visualisation can update on each simulation step. Therefore we included configurable "throttle" property in order to do visual update only by default every 100 ms.

As all computation and rendering is done in the web browser, no interaction with a server is needed. The web simulator can be distributed as a static or server-less web page, e.g., using popular GitHub pages(GithubPages 2021). It can be utilized to distribute, e.g., digital appendices of scientific papers. It was already used by (Mazumder et al. 2023) using web simulator deployed at `https://filip-jezek.github.io/Ascites/` This way we published also first version of e-book "The Physiology of Iron metabolism" as seen in 11.



Simulator of Iron Metabolism

d hh:mm:ss
1 08:09:35
Show Legend

Iron concentration in food [$\frac{\mu g}{h}$] 725
Knockout of gene for ferroportin(Fpn) ☑ Fpn is active
Knockout of genu for hepcidin ☐ gene is knocked out
LPS injection 1μg pressss to give LPS injection

Tasks: start simulation and see the reaction of organism regarding iron metabolism on acute or chronic bacterial infection imitated by LPS injection. Follow instructions bellow.

1. What is the reaction on acute bacterial infection? (Press the LPS injection once and see the effect and choose single answer)

☐ LPS directly reduces Fpn activity, LPS cause that IL6 concentration increases multiple times for temporary. Hepcidine concentration increase with a slight delay. After 12 h (simulated time) is iron concentration in serum between 0.2 and 0.3 of normal values. After 24 hours all is normalized.

☐ IL6 concentration remains multiplied longer time. Hepcidine regulates iron intake for longer time and iron concentration in serum remains low.

check answers

2. What is the reaction on chronic bacterial infection? (Press the LPS injection multiple times, e.g. every 2 simulated hours)

☐ LPS directly reduces Fpn activity, LPS cause that IL6 concentration increases multiple times for temporary. Hepcidine concentration increase with a slight delay. After 12 h (simulated time) is iron concentration in serum between 0.2 and 0.3 of normal values. After 24 hours all is normalized.

☐ IL6 concentration remains multiplied longer time. Hepcidine regulates iron intake for longer time and iron concentration in serum remains low.

check answers

3. What are the pathways LPS regulates (what appears during LPS injection)?

☐ reduction of gene for FPN

☐ increase of IL6 and consequently increase of hepcidine

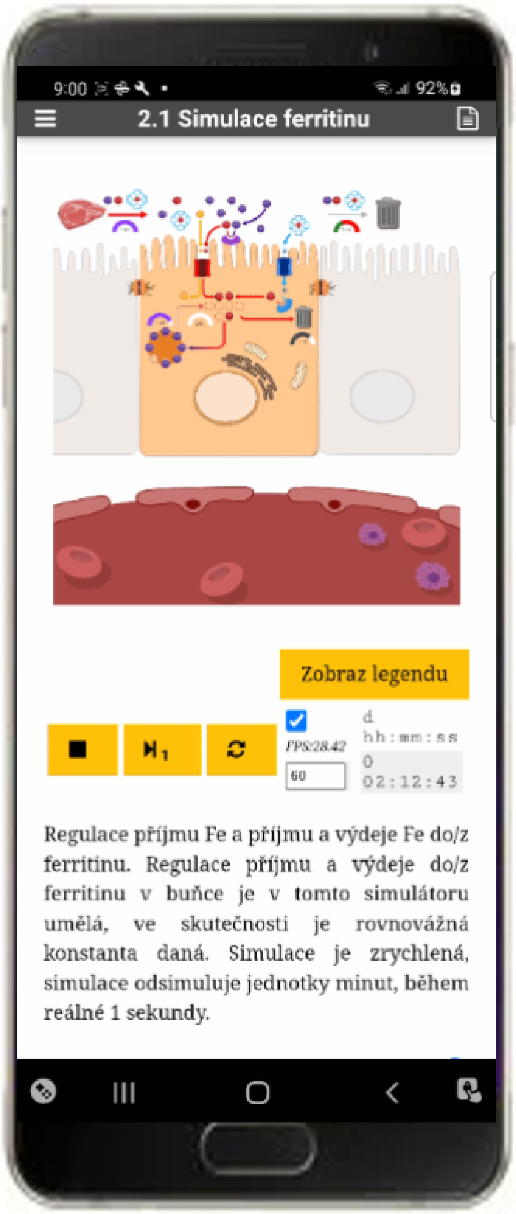
☐ both, direct reduction of gene for FPN and indirect increase of hepcidine via increase of IL6 production

check answers

Figure 11. Sample educational simulator of iron metabolism simulating gene knockout of hepcidin hormone resulting in iron overload in internal organs. It allows to enable/disable gene knockout, set a diet to affect illness and its treatment.

The same simulator can be converted also as a native mobile application e.g. by Apache Cordova (Apache Cordova 2023) tool. A sample in Figure 12 shows the digital textbook compiled as an Android application.

[4] Bodylight-VirtualMachine `https://github.com/creative-connections/Bodylight-VirtualMachine`



Regulace příjmu Fe a příjmu a výdeje Fe do/z ferritinu. Regulace příjmu a výdeje do/z ferritinu v buňce je v tomto simulátoru umělá, ve skutečnosti je rovnovážná konstanta daná. Simulace je zrychlená, simulace odsimuluje jednotky minut, během reálné 1 sekundy.

Figure 12. Educational simulator as native Android application.

A sample in Figure 13 shows a web simulator of blood gas exchange connected to the robotized virtual patient mannequin and controls his breathing. The following placed mockup of a medical device controls the extracorporeal membrane oxygenation (ECMO) process parameters. Such parameters are inputs to the model simulator connected via REST API, and the simulation shows direct feedback on user input and healthcare staff intervention to the patient state in graphs.

A sample in Figure 14 shows interactive 3D visualization of simplified human anatomy with charts of simulated hemodynamics. It leverages WebGL standard to visualize 3D objects and view the 3D scene. If this simulator is executed in a browser of a virtual reality device, then WebXR API is detected, and the simulator can be switched to an immersive view. This was tested on Oculus Quest 2 and MS Hololens 2.

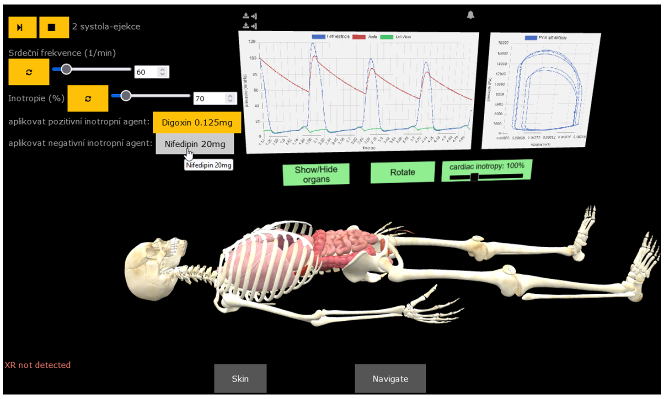


Figure 14. Sample educational simulator in 3D using WebGL and in immersive view for virtual reality using WebXR API. Virtual patient with simplified anatomy and physiology of cardiac hemodynamics and charts and controls are allowed to show the effect of drug treatment interactively.

Bodylight.js-Components is delivered using an open-source MIT License still and is still in the development stage depending on other open-source code [5]and the releases can be cited via Zenodo as (Kulhanek et al. 2023). The complete toolchain documentation and links are available `https://bodylight.physiome.cz`.

4 Discussion

Client-side simulation is appropriate for use cases where one or a few simulations must be performed. This is appropriate for interactive documents like educational materials, technical reports and digital appendices.

Client-side web-based simulation might not be appropriate for system analysis tasks like Monte-Carlo simulation.

The simulation is matched per `Window.requestAnimationFrame()` to the browser performance and is paused when the browser tab is in the background. The optimal inner FMU steps to refresh the framerate ratio have been established for a sample model; the optimal balance would vary though, depending on system performance bottlenecks and model complexity. Automatic adjustment based on the client's performance might be possible, but is currently not included in the development roadmap. The Web Workers (WebWorkers 2021) method might be more appropriate for another type of simulation (esp. for long-term models with higher memory demand etc.).

In the past, web-based simulators depended on non-standard, proprietary, but widely used plugins such as Adobe Flash player or Microsoft Silverlight. However, as technologies become obsolete (or even blocked), many older yet still scientifically relevant simulators cannot be executed on most modern computers or devices without excessive effort on virtualizing or emulating old operating systems and environments. We hope that using standard

[5] Bodylight.js-Components `https://github.com/creative-connections/Bodylight.js-Components`

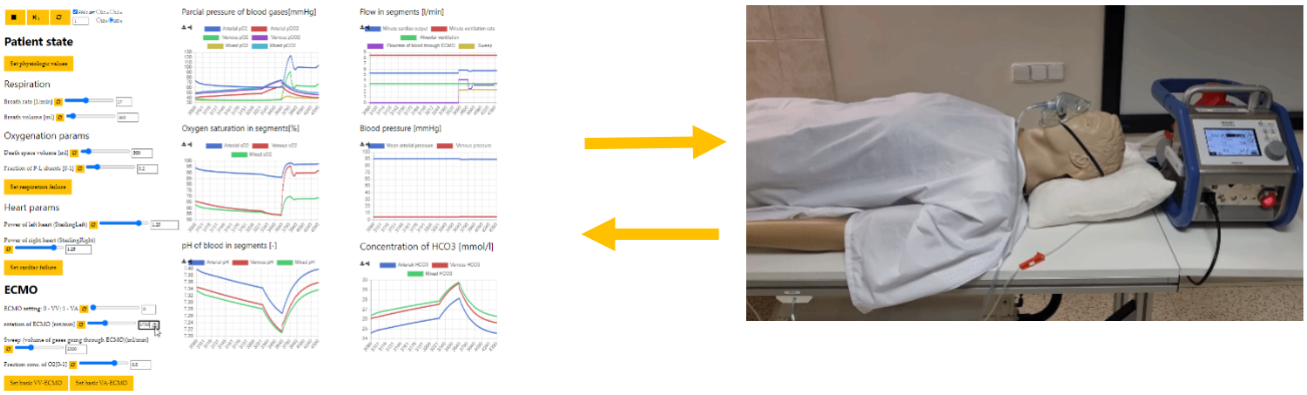


Figure 13. Web simulator connected to hardware mannequin of virtual patient and mockup of the medical device. It communicates via REST API to show breathing and mockup of medical device (extracorporeal membrane oxygenator - ECMO) controls several model parameters. User input on this hardware-in-the-loop gives direct feedback in the connected simulator and visualization of breathing.

languages like Modelica, traditional execution models like FMI, and standard web API to build components may survive over a decade. Web simulators built from now on can be run in the future seamlessly.

Additionally, thanks to the widely accepted standards, the simulators can now be executed on various devices such as mobile phones, tablets, and virtual and augmented reality devices with no or very low code intervention. Bodylight.js library brings the missing piece and tools to integrate already existing standards and technologies between web publishing and mathematical modeling in Modelica.

Acknowledgements

Commercial potential is under further investigation, however, the main tools and libraries are free and open-source thanks to previous grants, funding and support by Creative Connections company, the MPO FV20628 and MPO FV30195 grants provided by Czech Ministry of Industry and Trade, and the Prague Smart Accelerator `CZ.02.2.69/0.0/0.0/18_055/0016956` provided by the City of Prague and partly financed from EU structural funds. We would like to thank to E. Benjamin Randall, Ph.D. for proofreading and grammar correction.

References

Apache Cordova (2023). *Open-source mobile development frameworkto use standard web technologies - HTML5, CSS3, and JavaScript for cross-platform development*. URL: https://cordova.apache.org (visited on 2023-01-06).

AureliaJS (2023). *Aurelia - Aurelia is a JavaScript client framework for web, mobile and desktop*. URL: https://aurelia.io (visited on 2023-11-08).

ChartJS (2021). *Simple yet flexible JavaScript charting for designers and developers*. URL: https://www.chartjs.org (visited on 2021-05-08).

CreateJS (2023). *CreateJS - A suite of modular libraries and tools which work together or independently to enable rich interactive content on open web technologies via HTML5*. URL: https://createjs.com/ (visited on 2023-11-08).

Dygraphs (2021). *Fast, flexible open source JavaScript charting library*. URL: https://dygraphs.com/ (visited on 2021-05-08).

Dymola (2023). *Multi-Engineering Modeling and Simulation based on Modelica and FMI*. URL: https://www.3ds.com/products-services/catia/products/dymola/ (visited on 2023-01-01).

EMScripten (2021). *EMScripten - complete compiler toolchain to WebAssembly*. URL: https://emscripten.org (visited on 2021-05-08).

Fernandez de Canete, Javier et al. (2013-05). "Object-oriented Modeling and Simulation of the Closed Loop Cardiovascular System by Using SIMSCAPE." In: *Computers in Biology and Medicine* 43.4, pp. 323–33. ISSN: 1879-0534. DOI: 10.1016/j.compbiomed.2013.01.007.

Franke, Rudiger (2014). "Client-side Modelica powered by Python or JavaScript". In: *the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. DOI: 10.3384/ecp140961105.

Fritzson, Peter and et.al. (2019). "The OpenModelica Integrated Modeling, Simulation, and Optimization Environment". In: *Proceedings of The American Modelica Conference 2018, October 9-10, Somberg Conference Center, Cambridge MA, USA*. DOI: 10.3384/ecp18154206.

GithubPages (2021). *Websites for person and projects. Hosted directly from GitHub repository*. URL: https://pages.github.com/ (visited on 2021-05-08).

Jangda, Abhinav et al. (2019-07). "Not So Fast: Analyzing the Performance of WebAssembly vs. Native Code". In: *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, pp. 107–120. ISBN: 978-1-939133-03-8. URL: https://www.usenix.org/conference/atc19/presentation/jangda.

Kofránek, Jiri, Filip Ježek, and Marek Mateják (2019-02). "Modelica language - a promising tool for publishing and sharing biomedical models". In: *Proceedings of The American Modelica Conference 2018, October 9-10, Somberg Conference Center, Cambridge MA, USA*. Linköping University Electronic Press. ISBN: 9789176851487. DOI: 10.3384/ecp18154196. URL: http://dx.doi.org/10.3384/ECP18154196.

Kulhanek, Tomas et al. (2023). *creative-connections/Bodylight.js-Components:* version v2. DOI: 10.5281/zenodo.4575354.

Kulhánek, Tomáš et al. (2014). “Simple Models of the Cardiovascular System for Educational and Research Purposes”. In: *MEFANET Journal* 2.2, pp. 56–63. URL: http://mj.mefanet.cz/mj-04140914.

Library, Modelica Standard (2021). *Free (standard conforming) library from the Modelica Association to model mechanical (1D/3D), electrical (analog, digital, machines), magnetic, thermal, fluid, control systems and hierarchical state machines.* URL: https://github.com/modelica/ModelicaStandardLibrary/releases/tag/v4.0.0 (visited on 2021-05-08).

Matejak, Marek et al. (2015). “Free Modelica Library for Chemical and Electrochemical Processes”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. 118. Linköping University Electronic Press, pp. 359–366.

Mateják, Marek et al. (2014). *Physiolibrary - Modelica library for Physiology*. Lund, Sweden. URL: https://www.physiolibrary.org.

Mazumder, Nikhilesh R et al. (2023). “Portal Venous Remodeling Determines the Pattern of Cirrhosis Decompensation: A Systems Analysis.” In: *Clinical and Translational Gastroenterology*. DOI: 10.14309/ctg.0000000000000590.

Plotly (2021). *Plotly JavaScript Open Source Graphing Library*. URL: https://plotly.com/javascript/ (visited on 2021-05-08).

Short, Tom (2014). *OpenModelica models in Javascript*. URL: https://github.com/tshort/openmodelica-javascript (visited on 2021-05-08).

Šilar, Jan, Filip Ježek, et al. (2019). “Model visualization for e-learning, Kidney simulator for medical students”. In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. 157. Linköping University Electronic Press.

Šilar, Jan, David Polák, et al. (2019). “Development of In-Browser Simulators for Medical Education: Introduction of a Novel Software Toolchain”. In: *J Med Internet Res* 21.7, e14160. ISSN: 1438-8871. DOI: 10.2196/14160.

Tiller, Michael M. (2014). *Modelica By Example*. URL: https://mbe.modelica.university/.

WebComponents (2021). *Web components are a set of web platform APIs that allow to create new custom, reusable, encapsulated HTML tags to use in web pages and web apps.* URL: https://www.webcomponents.org/ (visited on 2021-05-08).

WebWorkers (2021). *Web Workers are a simple means for web content to run scripts in background threads*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers (visited on 2021-05-08).

Winkler, Dietmar and Michael Tiller (2017). “modelica. university: A platform for interactive modelica content”. In: DOI: 10.3384/ecp17132725. URL: https://ep.liu.se/ecp/132/079/ecp17132725.pdf.

Design ideas behind Bioprocess Library *for* Modelica

Jan Peter Axelsson

Vascaia AB, Sweden, `jan.peter.axelsson@vascaia.se`

Abstract

This paper discusses key design ideas behind the Bioprocess Library, BPL. The library facilitates modelling and simulation of bioprocesses mainly for the pharmaceutical industry. It borrows some structures from MSL Fluid and Media but differs in central design choices. A typical application consists of both configuration of standard components from the library and tailor-made Modelica code defining the application-dependent medium and bioprocess reactions. The guiding idea is that configuration of components works well for defining the setup of process equipment for a production line, while more flexibility is needed for modelling bioprocess reactions and therefore equations are used. Another central design idea is that components of equipment are centrally adapted to the medium used. One could say that the library is parameterised with the application media and reaction models. The focus of this paper is structural design aspects of the library rather than the content.

Keywords: Bioprocess, media, reactions, formal type parameters, packages, components, equations

1 Introduction

There is a growing interest in simulation also in the biopharmaceutical industry. Two major vendors of equipment Cytiva (*Bioreactor-scaling-tool* 2023) and Sartorius today offer services around their products based on using simulation. Two well-known companies are Siemens (with gPROMS) and Dassault (with 3DEXPERIENCE) who offer softwares and services in this market. The need is often a combination of mechanistic and more data-driven modelling. In the academic area the interest in simulation of biological systems has been there for decades and illustrated by the public repositories of models (*EMBL BioModels* 2023; *UC San Diego BiGG Models* 2023). So far, Modelica has had very little impact in this field, though. Important aspects of the question is discussed in (Wiechert, Noack, and Elsheikh 2010).

Developing bioprocesses requires a combination of knowledge from different fields like: reactor dynamics, gas-liquid-transfer, buffer-reactions, cell metabolism, recombinant protein expression, degradation processes of product proteins in the broth, impurities etc. Part of this knowledge is well established and can be re-used, while modeling of cell metabolism and product formation may be more unique, and less re-useable. Further, the same reactor may be operated in different ways: batch, fed-batch, continuous, perfusion etc. Various ideas of process control are also interesting to evaluate using simulation.

Bioprocess Library tries to meet these needs of flexibility and possibility to re-use code. It has been gradually developed over many years in-house for consultancy work and also teaching. Examples of applications are (Axelsson 2018; Axelsson 2019; Axelsson 2022). Examples of integrating black-box models together with the traditional mechanistic models has not been done yet, but can certainly be done in the framework. The architecture of the library was outlined in (Axelsson 2021). A key design aspect is to account for the different modelling needs for the process configuration and the actual reactions in the reactor. The engineered part is modelled by conveniently configure components of the library. The biological part requires more flexibility. Modelling of cell metabolism and other reactions in the broth is done by writing down the equations in a certain Modelica format.

This paper is organised as follows. An orientation of the library is given in section 2. Section 3 focuses on the reactor model and how library code is integrated with application code. In section 4 a simple example of fed-batch cultivation illustrates the structure of typical application code. In section 5 the technique of constraint-based modelling is briefly discussed and how such modelling can be handled with the library. In section 6 different aspects of the library design are discussed including the relation to MSL Fluid and Media. The availability of the library is outlined in section 7 and the paper ends in section 8 with concluding remarks.

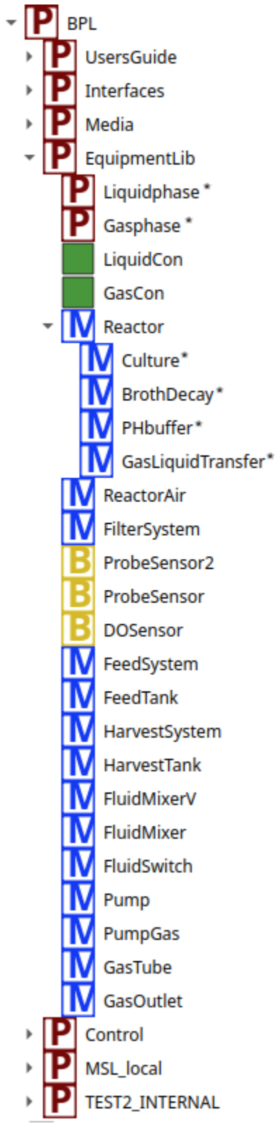
2 Library structure

The library has a flat hierarchy, limiting the use of partial packages and models. In this way it is hopefully easier to understand and later to expand. A brief orientation of the library is first given. The next section describes the EquipmentLib, followed by a section on the media connectors.

2.1 Overview

The structure of Bioprocess Library is shown in Figure 1. It includes the following packages:

- `UsersGuide` - Package of brief practical information in different records, accessible from the FMU.
- `Interfaces` - Package of templates for liquid and gas, and interfaces for the `Reactor` component and its inner application models. It is all used in `EquipmentLib`.



- BPL
  - UsersGuide
  - Interfaces
  - Media
  - EquipmentLib
    - Liquidphase*
    - Gasphase*
    - LiquidCon
    - GasCon
    - Reactor
      - Culture*
      - BrothDecay*
      - PHbuffer*
      - GasLiquidTransfer*
    - ReactorAir
    - FilterSystem
    - ProbeSensor2
    - ProbeSensor
    - DOSensor
    - FeedSystem
    - FeedTank
    - HarvestSystem
    - HarvestTank
    - FluidMixerV
    - FluidMixer
    - FluidSwitch
    - Pump
    - PumpGas
    - GasTube
    - GasOutlet
  - Control
  - MSL_local
  - TEST2_INTERNAL

Figure 1. Structure of the Bioprocess Library. The formal (type) parameters are marked with $*$.

- `Media` - Package for various standard media.
- `EquipmentLib` - Package of models of standard equipment using a generic medium, e.g. tanks, pumps, reactors, sensors, filter, mixers etc for both liquid and gas media.
- `Control` - Package of blocks for generating electrical control signals to equipment components. They complement and adjust blocks from MSL.
- `MSL_local` - Package that contains parts of MSL for PID-control and more. In this way BPL can be used more independent of the general MSL version of the compiler. Currently the MSL version 3.2.2 build 3 is used, but to be updated to MSL 4.0.0.
- `TEST2_INTERNAL` - Package with small test applications including configuration of equipment for batch, fed-batch, chemostat and perfusion cultivation.

2.2 The EquipmentLib package

The `EquipmentLib` package has a central role in the library. Components are coded in a generic way. The package is parametrised with two general formal (package) parameters for the media:

- `Liquidphase` - the package for the liquid phase
- `Gasphase` - the package for the gas phase

and four specific formal (model) parameters for just the component `Reactor` described in section 3.

To improve readability, the code sections defining base packages and partial models are lifted out to the package `Interfaces` and imported back where needed.

2.3 The media and connectors

The flow direction of both liquid and gas media are usually well-defined in bioprocesses and reflected in the present components. The media connectors are however undirected and prepared to handle back-flow.

The `LiquidCon` connector code is shown below. The `GasCon` connector is defined similarly.

Listing 1. LiquidCon connector

```
connector LiquidCon
   stream Liquidphase.Concentration c;
   flow Real F (unit="L/h");
   Real p (unit="bar");
end LiquidCon;
```

The connector uses the flow concept for the flow F and the corresponding potential is the pressure p. The concentration vector c gets its size from the actual medium. The vector c is declared as a stream variable. The density of the liquid media is calculated when needed based on the concentration c and here is a function in the liquid media base template that takes information of molar weights from the actual application medium. Similar technique as in MSL Media.

Both the inclusion of pressure p and the use of the stream-concept for c are introduced to "future-proof" the library, and facilitate local balancing of models, see (Olsson et al. 2008; Franke et al. 2009). The pressure does not play any role in the applications so far. The pumps are ideal in the sense that a given electrical signal gives the desired flow rate immediately. There has not been any obvious need to model back-flow either, which is the major motivation of the undirected connector using the stream concept (Franke et al. 2009).

The temperature has so far not been considered important in the modelling. The temperature is usually in the range from room temperature up to 37°C and well controlled. The cell culture produces heat due to metabolism and at high cell concentrations and feed rate, the cooling capacity of the reactor sets a limit. This limit is very similar to the oxygen transfer capacity. Both set about the same limit of rate of metabolism supported by mechanical reactor design. The process is usually designed with some margin to this limit, see simulations in section 4.4.

3 The reactor component

The various reactions of the process are thought to take place only in the `Reactor` component. The other equipment only transports, stores, separates media species, or mixes media from several sources. The `Reactor` combines standard and application dependent parts in a complex way. The component `ReactorAir` is an extension of the `Reactor` to also handle gas phase.

The reactions in a typical bioreactor can be divided into different parts. These parts are user-defined sub-models, i.e. `inner` components, to the `Reactor` and serve as formal (type) parameters:

- `culture` - reactions in the living cells
- `broth_decay` - degradation of substances and even of the living cells in the culture broth
- `pH_buffer` - pH-buffer reactions in the broth
- `gas_liquid_transfer` - gas-liquid-transfer between the reactor broth and gas phase

An important observation is that it is the concentration of substances that drives the rate of reactions in these different parts. The broth concentration is the "communication link" between them. Therefore, concentration is chosen as an `inner` variable in the reactor. Similarly, gas fraction is used as an `inner` variable in the extension for aerated reactor. The different parts are naturally sub-models to the reactor model. The reactor sub-models communicate back to the reactor through rate-variables. These rate-variables are normalised with respect to the biomass $m[X]$ of the reactor for reactions in the cell and with respect to liquid reactor volume V for the others.

The corresponding code-snippet for the reactor model is shown below. A key application dependent parameter is *nc*, that stands for the number of components, or we should say species, in the medium.

Listing 2. Reactor model equations

```
// Concentrations for the liquid phase:
for i in 1:Liquidphase.nc loop
   c[i] = m[i]/V;
   for j in 1:n_outlets loop
      outlet[j].c[i] = c[i];
   end for;
   for j in 1:n_ports loop
      port[j].c[i] = c[i];
   end for;
end for;

// Mass—balance for the liquid phase:
for i in 1:Liquidphase.nc loop
   der(m[i]) = culture.q[i]*m[X]
   + broth_decay.r[i]*V
   + pH_buffer.r[i]*V
   + gas_liquid_transfer.r_to_liquid[i]*V
   + sum(actualStream(inlet[j].c[i])
      *inlet[j].F for j)
   + sum(c[i]*outlet[j].F for j);
   for j in 1:n_inlets loop
      inlet[j].c[i] = c[i];
   end for;
end for;

// Liquid volume of the reactor:
der(V) = sum(inlet[i].F for i)
      + sum(outlet[i].F for i);
```

The different sub-models are all optional and Modelica provide language constructs to support use of `no_culture`, `no_broth_decay` etc. The interface standard between the reactor and the sub-models are defined in the package Interfaces in BPL.

4 Application code

A simple example will illustrate the structure of the application code, adaptation of the `EquipmentLib` to the application, and finally the configuration of the process model. The model has no gas phase or broth decay, and pH-buffer reactions are not defined either.

The mass-balance model of the example is as follows. See for instance chapter 6 in (Hu 2020).

$$\frac{d(m[S])}{dt} = -q_S(c[S]) \cdot m[X] + S_{in}F_{in}(t) \tag{1}$$

$$\frac{d(m[X])}{dt} = \mu(c[S]) \cdot m[X] \tag{2}$$

$$\frac{dV}{dt} = F_{in}(t) \tag{3}$$

where the specific cell growth rate $\mu(c[S])$ and substrate uptake $q_S(c[S])$ are directly related through the yield Y which is here a constant

$$\mu(c[S]) = Y \cdot q_S(c[S]) = Y \cdot q_S^{max} \frac{c[S]}{K_s + c[S]} \tag{4}$$

The dosage of substrate $F_{in}(t)$ is controlled from a process computer according to a pre-defined exponential scheme.

4.1 Application medium and reactions

The medium is defined by the following package.

Listing 3. Application medium

```
package Liquidphase2
   import BPL.Interfaces.LiquidphaseBase;
   extends LiquidphaseBase
      (name="Standard components X and S",
       nc=2);
   constant Integer X=1;
   constant Integer S=2;
   constant Real[nc] mw (each unit="Da") =
      {24.6, 180.0};
end Liquidphase2;
```

The reactions in the cell culture are described by the following model, see Listing 4.

Listing 4. Application culture

```
model Culture2 "Text-book culture model."
   import BPL.Interfaces.ReactorInterface;
   extends ReactorInterface(redeclare
       package Liquidphase=Liquidphase2);
   outer Liquidphase.Concentration c;
   Liquidphase.Rate q;
   constant Integer X = Liquidphase.X;
   constant Integer S = Liquidphase.S;
   parameter Real Ks (unit="g/L") = 0.1;
   parameter Real qSmax (unit="g/(g*h)")=1;
   parameter Real Y (unit="g/g") = 0.50;
   Real mu (unit="1/h");
equation
   q[X] = mu;
   q[S] = -qSmax*c[S]/(c[S]+Ks);
   mu = -Y*q[S];
end Culture2;
```

Note that the culture model relates metabolic flow rates $q[\,]$ to reactor concentrations $c[\,]$ with a static function. Even for much more complex culture models the relation is a static function, cf equation (5) in the next section. However, if we need to introduce dynamics in the culture model this can be done in this framework too.

It is a possibility to structure the culture model in more parts. This usually improves readability and can help in the dialogue with microbiologists around this part.

4.2 Adaptation of the EquipmentLib

The adaptation of the `BPL/EquipmentLib` to the application medium and culture model is done in the few lines of code below.

Listing 5. Adaptation of `EquipmentLib` to the application

```
package Equipment
   import BPL.EquipmentLib;
   extends EquipmentLib(
      redeclare package Liquidphase =
          Liquidphase2,
      Reactor(redeclare model Culture =
          Culture2));
end Equipment;
```

Note that package `Liquidphase` and model `Culture` are formal (type) parameters to `EquipmenetLib` and get their values from the application. The concept of formal parameters are discussed in section 4.4 (Fritzson 2015).

4.3 Configuration of the application process

The application process can now easily be configured using the adapted library. Note that for the component bioreactor the medium component for cell concentration must be specified and also that an inlet to the reactor is needed. The component feedtank is an integration of a feedtank with a pump. The component dosage scheme is taken from `BPL/Control` package and has an electrical signal connector and no adaptation needed.

The application code needs just the three sections shown. It is possible in the package `Equipment` to define special tailor-made models of equipment needed for the application that is not available in the library.

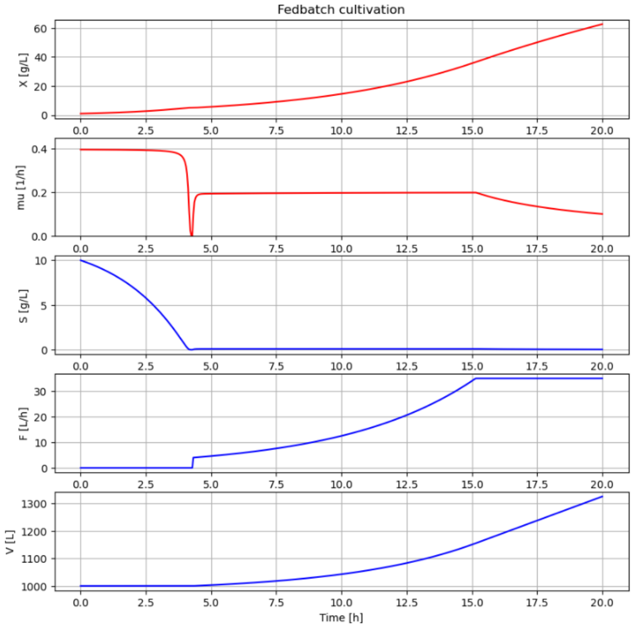


Figure 2. Simulation of fed-batch cultivation. See repository CONF_2023_10_MODELICA15 at (*BPL Applications* 2023).

Listing 6. Application configuration

```
model Fedbatch "Fedbatch cultivation"
   Liquidphase_data liquidphase;
   Equipment.Reactor bioreactor
      (X=liquidphase.X, n_inlets=1);
   Equipment.FeedSystem feedtank;
   Control.DosageSchemeExp dosagescheme;
equation
   connect(bioreactor.inlet[1], feedtank.
       outlet);
   connect(feedtank.Fsp, dosagescheme.F);
end Fedbatch;
```

4.4 Simulation results

The example above of fed-batch cultivation is simulated with quite typical parameters for yeast (*S. cerevisiae*) cultivation (by-product formation neglected), see Figure 2.

It is common to start fed-batch cultivation with a short batch phase, as we see here. When the initial substrate is consumed the substrate feeding is started at 4 h and follows a certain scheme. Here a well-designed exponential feed scheme is used. Note that the substrate level is kept low and growth rate kept constant at about half the maximal rate. From time 15 h and on, the feed rate is kept constant in order to avoid challenging the reactor capacity, but not modelled here. The capacity limit is determined by the oxygen transfer and cooling capacity. During this time with constant feed rate the culture continues to grow but at a slowly decreasing rate.

5 Note on constraint-based modelling

The culture sub-model of the reactor is in many applications a static non-linear function $f()$ relating reaction rates $q[\,]$ of cell metabolism and growth to reactor broth concentrations $c[\,]$. There are nc species in the broth and rates $q[i](t)$ defined for each of them at time t

$$q[i](t) = f_i(c[j](t)),\ j = 1...nc \tag{5}$$

The function is derived from the underlying system of equations of reactions rates. It is quite common that the system of steady state equations is under-determined and therefore complemented with constraints that determine the system. This leads to a formulation of the function $f()$ in an implicit way, as a solution of an optimisation problem. It is a convenience of modelling that has some similarity with analytical mechanics in physics.

An example showing cell growth on two substrates illustrates the idea. The example is a simplified version of yeast growing on glucose G and ethanol E. In this example the important fact that yeast can produce ethanol is dropped. Both substrates can be taken up and metabolise in parallel and the cell coordinate the uptake rates to maximize its specific growth rate μ

$$\mu = Y_{Gr} \cdot q_{Gr} + Y_{Er} \cdot q_{Er} \tag{6}$$

under the constraint of the cells limited respiratory capacity q_{O2}^{lim} see (Sonnleitner and Käppeli 1986)

$$k_{og} \cdot q_{Gr} + k_{oe} \cdot q_{Er} \leq q_{O2}^{lim} \tag{7}$$

At low substrate levels the uptakes are limited by the concentrations in the broth

$$q_{Gr} \leq \alpha G \tag{8}$$
$$q_{Er} \leq \beta E \tag{9}$$

Listing 7. Calculation of "culture" $f()$ using linear programming done with Optlang in Python

```
def f(G,E):
   qGr=Variable('qGr',lb=0)
   qEr=Variable('qEr',lb=0)

   mu_max=Objective(YGr*qGr+YEr*qEr,
      direction='max'
   qO2lim=Constraint(kog*qGr+koe*qEr,
      ub=qO2max)
   qGlim=Constraint(qGr,ub=alpha*max(0,G)
   qElim=Constraint(qEr,ub=beta*max(0,E)

   f.objective=mu_max
   f.add(qO2lim)
   f.add(qGlim)
   f.add(qElim)

   f.optimize()
   return (f,objective.value,
           f.variables.qGr.primal,
           f.variables.qEr.primal)
```

Thus, the optimization problem (6)-(9) gives the static function (5) that relates metabolic rates to the broth concentrations and describes the culture, see formulation in the high level Python package Optlang (Jensen, Cardoso, and Sonnenschein 2017) in Listing 7.

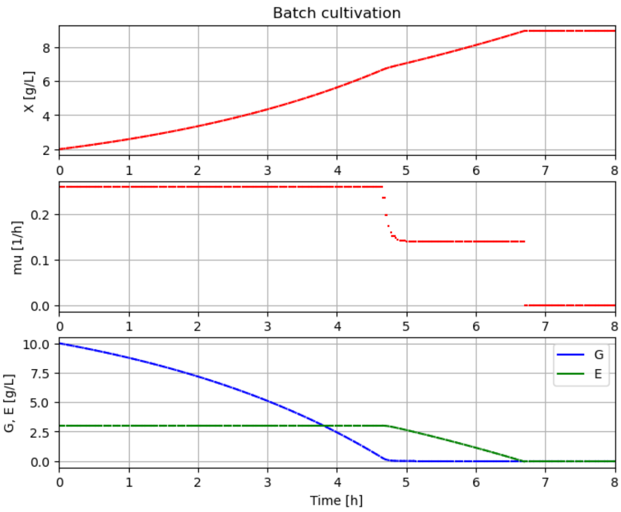


Figure 3. Simulation of batch cultivation with two substrates. CONF_2023_10_MODELICA15 at (*BPL Applications* 2023).

Listing 8. Simulation of BPL-model of bioreactor together with "culture" $f()$ done with FMU and PyFMI (or FMPy) together with Optlang in Python outlined (species index omitted)

```
n = t_final/t_delta
initialize c[0] and q[0]
c[1] = simulate(0, t_delta, c[0], q[0])
for i in 1:n:
   q[i] = f(c[i])
   c[i+1] = simulate('cont', c[i], q[i])
```

A simplified solution procedure to integrate bioreactor simulation with optimisation of culture metabolic rates at each time instant is based on the fact that here are two time scales. The concentrations $c[t]$ in the reactor broth change slowly compared to the cell culture optimisation of reaction rates $q[t]$. Thus, simulation a short step Δt of the integrated process is done with reaction rates $q[t]$ kept constant and gives concentrations $c[t + \Delta t]$. Then the culture reactions rates $q[t + \Delta t]$ are updated with an optimisation based on the concentrations $c[t + \Delta t]$ etc. The procedure is often called the "direct approach" and works well for a class of problems. Its limitations are discussed in section 2.2 and 4.4 in (Ploch, Lieres, et al. 2020). Key steps of the Python code are outlined in Listing 8.

In Figure 3 results from simulations using PyFMI (Andersson, Åkesson, and Führer 2016) are shown of batch cultivation with the two substrates G and E. For the culture to grow at maximal rate, first G is consumed and then E. Note that during a short period 4.7-4.9 hours both substrates are consumed in parallel. This occurs since levels of G is low and the constraint on oxygen allows some E to be consumed, and gradually more, and G vanishes.

In this example the model can very well instead be formulated in terms of a system of non-linear ODE as was done in the original publications (Sonnleitner and Käppeli 1986).

The advantage with constraint-based modelling is that the modelling can handle larger models and remain relatively transparent. The public model repository (*EMBL BioModels* 2023) contains today about 1000 models and 20 percent of them use constraint-based modelling and the rest use mainly ODE. The public repository (*UC San Diego BiGG Models* 2023) contains about 100 large genome scale models with thousands of reactions all modelled using a constraint-based technique. The modelling software preferred is COBRApy that is based on Optlang, see (Ebrahim et al. 2013).

Thus, the framework presented here with BPL, FMU-simulation of the bioprocess setup and Optlang for handling constraint-based modelling of the culture, can integrate the two modelling techniques, see (Axelsson 2018; *BPL Applications* 2023).

It would likely be better to handle the constraint-based modelling within Modelica. With the Modelica extension Optimica at least the basic example above can be simulated with ease (Axelsson 2018). Here is also a recently developed Modelica toolbox for Differential Algebraic Embedded Optimization (DAEO) which address constraint-based models and seems very interesting (Ploch, Zhao, et al. 2019; Ploch, Lieres, et al. 2020). The drawback with a Modelica-solution is that models from the public repositories needs to be translated to Modelica, but can perhaps be automated.

6 Reflections on the library design

Here the BPL approach is compared with techniques of MSL Fluid and Media and some other libraries. The first sub-section addresses the scope of the library more clearly. In the later half of the section more broader questions are briefly discussed.

6.1 Limitations of the library

The focus of the Bioprocess Library is after all towards bio-pharmaceutical processes, often involving recombinant protein expression. This means that processes are usually operated in the temperature range $20 - 37°C$. Further the pressure is usually just above room pressure. The viscosity of liquids is like water. The reactor volumes are often up to 1000 L but occasionally 10000 L. Thus, several properties of media accounted for in MSL Fluid and Media are not that relevant.

If we look at biotechnology processes more broadly and include antibiotics, enzyme-production, baker's yeast production, breweries, or biorefinery industry, then the reactor scale may go up to 100 m^3, and reactor media are more complex and may be viscous. In this scenario MSL Fluid and Media can be more of help. One interesting application from biorefinery industry is (Ploch, Zhao, et al. 2019) where Modelica is used together with MSL Fluid and Media in combination with tailor-made modelling of bioreactor with a microbial culture.

6.2 Central vs local definition of media

The BPL has a structure where the medium is centrally defined for all components in `EquipmentLib`. This is in contrast to MSL Fluid where each component has an individual local definition of medium. There are certainly pros and cons with the two approaches.

There has been an expressed wish to in some way automate the process of choice of medium to simplify and ensure correctness of code. In the conclusion of (Franke et al. 2009) it is stated: *"The used medium has currently to be defined for every component. It would be nicer if the medium was defined at one source and the medium definition would then be propagated through the connection structure."*

The "propagation" of medium definition is in BPL done using "adaptation" of package using formal (type) parameters, as shown in the example in section 4.2. The idea behind using application package media as well as submodels of the reactor as "type formal parameters" for the `EquipmentLib` package came from material in chapter 4.4 and an odd example in chapter 10.4 in (Fritzson 2015), combined with an urge for simplicity.

6.3 Equations vs components

The BPL defines reactions with equations rather than components. The reactions are grouped in four categories as outlined in section 3, and seen as sub-models to the Reactor component. The sub-models can be given an internal structure of interacting sub-sub-models to enhance readability and simplify modifications. Especially relevant for culture models. It is up to the user who writes the application code.

To introduce components for say the culture model is difficult. Different approaches to model biochemical networks are discussed in (Wiechert, Noack, and Elsheikh 2010). There is a public Modelica library, BioChem (Brugård et al. 2009), that brings component modelling to biochemical reaction network. The possibility to integrate BPL with BioChem has not been investigated, so far.

The focus of MSL Fluid and Media is on thermo-fluid modelling as is clearly stated in the MSL documentation. Thus, the focus of the library is not really on chemical reaction processes as pointed out earlier (Baharev and Neumaier 2012).

6.4 Code in consultant-customer relation

In my experience of consultancy work the customer is focused on results of using simulation rather than the simulation tool itself. In this context it is important to show and document the application code outlined in section 4. Provided the configuration of the process using standard components is clear and tested enough, there is little interest for the details of the library. The customer is usually satisfied to own the application code, while the consultant owns the library. If there is an interest, the consultant can deliver a compiled FMU and Jupyter notebooks that generate the results presented in the project.

DOI
10.3384/ecp204453

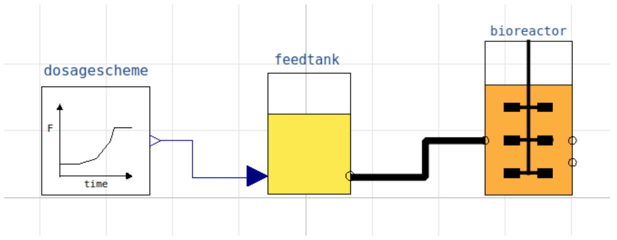


Figure 4. Graphical configuration of fed-batch cultivation in OpenModelica, see Listing 9. The formal (type) parameters Liquidphase and Culture of the library are not shown.

6.5 Challenges for the GUI

The library has been developed with little concern about a graphical user interface. One reason is that the (deprecated) software JModelica (Åkesson et al. 2010) has been used which lacks a GUI. Another reason is that there are differences between different vendors design of the GUI although the Modelica Standard Library sets an informal standard of what GUI facilities are implemented. However, the Modelica language is richer.

The challenges to configure BPL applications graphically are the following:

1. The package `BPL/EquipmentLib` has two formal (type) parameters for media: `Liquidphase` and `Gasphase`.
2. The `BPL/EquipmentLib/Reactor` has four formal (type) parameters for sub-models: `culture`, `broth_decay`, `pH_buffer`, and `gas_liquid_transfer`.
3. The `BPL/EquipmentLib/Reactor` use inner/outer implicit connection to the sub-models listed in the previous item. There is also a specific connection for each sub-model back to the `Reactor`.
4. The `BPL/EquipmentLib/Reactor` has also variable number of inlets and outlets and affect the icon of the component.

A first attempt to address GUI configuration in OpenModelica is shown in Figure 4, cf Listing 9. We see the model of fed-batch cultivation in section 4. Note that the electrical connector is represented by a thin line and the liquid phase connector with a thick filled line. For gas phase a thick open line is used, not shown.

6.6 Library modification for different GUI

Not all Modelica implementations have a GUI that supports parametrisation of packages, although the compiler does. This affects the design of the package `EquipmentLib` and the structure of the applications code. One approach to solve this dilemma is to introduce a parallel package `EquipmentLib2` where the formal (package) parameters for media as well as connector code, are moved into each component instead. Then adaptation to the application media is done at instantiation of the component instead. Thus Listing 5 and Listing 6 are then combined as shown in Listing 9, done by the GUI.

Listing 9. Application configuration when parametrisation of packages are not supported by the GUI

```
model Fedbatch "Fedbatch cultivation"
   Liquidphase_data liquidphase;
   EquipmentLib2.Reactor bioreactor(
      redeclare package Liquidphase =
         Liquidphase2,
      redeclare model Culture = Culture2,
      X = Liquidphase.X, n_inlets=1);
   EquipmentLib2.Feedsystem feedtank(
      redeclare package Liquidphase =
         Liquidphase2);
   Control.DosageSchemeExp dosagescheme;
equation
   connect(bioreactor.inlet[1], feedtank.
      outlet);
   connect(feedtank.Fsp, dosagescheme.F);
end Fedbatch;
```

The solution in Listing 9 has a structure similar to the MSL Fluid and Media. The drawback with this structure is that you need to redeclare media packages for each component. In OpenModelica this is done manually by the user. In the software Impact from Modelon there is also a possibility to automate this tedious and error-prone work in the GUI, see (*Modelon* 2023).

An alternative approach to keep redeclaration of media to one central place for components taken from package `EquipmentLib2`, is to introduce a global formal type parameter for the application code that the components media redeclaration refer to, see Listing 10 below.

Listing 10. Application configuration when parametrisation of packages are not supported by the GUI and global parameter for package `Liquidphase` introduced to centralize user interaction

```
model Fedbatch "Fedbatch cultivation"
   Liquidphase_data liquidphase;
   replaceable package Liquidphase =
      Liquidphase2;
   EquipmentLib2.Reactor bioreactor(
      redeclare package Liquidphase =
         Liquidphase,
      ...
```

The approach with a global formal type parameter can be found in some MSL Fluid examples, e.g. `ThreeTanks`, and the technique is in fact generally widely used. The drawback with this method is that it requires editing of the code manually, and can not be done with just using the (current) GUIs.

6.7 An idea for GUI improvement

The different Modelica GUI are similar and simplified to focus on components and their interconnections. The awareness of which package a certain component comes from is important at the time of configuration but later of little use. In the code view the programmer is naturally more aware of the package a component belong to.

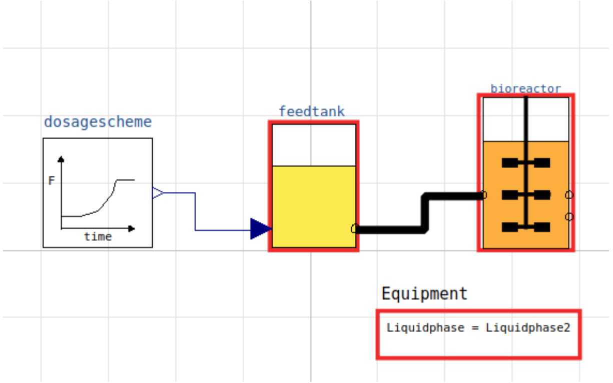


Figure 5. An idea to improve GUI by including access to package parameters for Equipment.

One way to make package information accessible in the GUI is shown in Figure 5. The package `Equipment` is represented by the box in the lower right corner. Clicking on this box shows general (type) parameters for the package and in this case the package `Liquidphase` and what that package is, and the user can change it. The code is then re-translated. Note that clicking on the box highlight the components in the configuration that belong to `Equipment` and affected by a change. A complement could be to in each component introduce a symbol for what package the component come from and when clicked the box with information of the package is shown and the other components in the configuration from the same package are high-lighted.

In MSL there are libraries where the need for expressing common properties for several components in a central way are addressed. The technique used is to define a central component that other components connect implicitly with using inner/outer variable mechanism. In the Fluid library the central component is called `system` and in the Multibody library it is called `world`. When the central component is defined on the top level it provides application wide access and can be seen as global parameters (*SystemComponent* 2023). This should be compared with the idea presented in this paper, namely package parametrisation using mechanism of inheritance. It make use of the package structure and is not global parameters. The pros and cons of the different approaches needs further analysis.

6.8 Two-level system configuration with GUI

In practice it is common to first settle the process setup and then explore various strategies for operation and control. However, in some contexts it can be fruitful to investigate the interplay of process and control design.

The process setup can effectively be coded using the procedure in section 4 and shown in Listing 11. The configuration provides an openness of how operation and control should be done. The model `Fedbatch_base` combines the feed tank with the bioreactor, but we do not include any dosage scheme. Further we equip the reactor with an on-line sensor for measurement of substrate concentration.

The operation and control of the process is then configured at a second level. Here we can let the process be operated by a fixed dosage scheme of the feed rate as before, or for example investigate the use of feedback control of the feed rate around the dosage scheme based on the on-line substrate measurement signal.

Listing 11. Application configuration fed-batch - level 1

```
model Fedbatch_base "Fedbatch cultivation"
   Liquidphase_data liquidphase;
   EquipmentLib.Reactor bioreactor(
      X=liquidphase.X, n_inlets=1,
         n_ports=1);
   EquipmentLib.ProbeSensor sensor(
      component=liquidphase.S);
   EquipmentLib.Feedsystem feedtank;
   Interfaces.RealInput Fsp;
   Interfaces.RealOutput S_measured;
equation
   connect(Fsp, feedtank.Fsp);
   connect(feedtank.outlet,
      bioreactor.inlet[1]);
   connect(bioreactor.port[1],
      sensor.probe);
   connect(sensor.probe.out, S_measured);
end Fedbatch_base;
```

This second-level configuration of the control system can be done using the GUI, as shown in Figure 6. Since we on this level only deal with electrical signals here is little difference between the GUI of different vendors, in this respect. Further, the advantage of exploratory GUI-configuration is more obvious than for the process setup.

Simulation during start-up of the process is shown in Figure 7. We see mainly that the control system is stable and keeps the substrate level at a low level close to the set-point during the exponential growth phase. To evaluate the advantage of substrate control compared to fixed dosage schemes requires a number of simulations taking into account variation in the initial cell concentration as well as variation in the culture parameters, and is outside the scope of this paper.

A two-level configuration process is natural in many situations and combine the strength of package parametrisa-

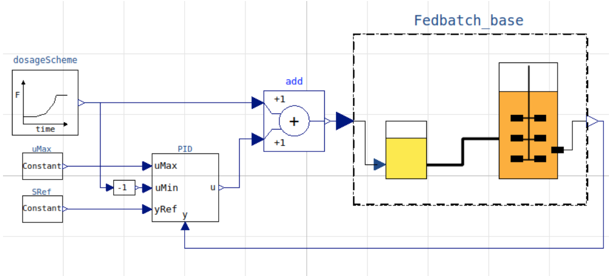


Figure 6. Application configuration fed-batch with feedback PID-control of feed rate from substrate measurement - level 2. The process setup is in the dash-lined box, see Listing 11.

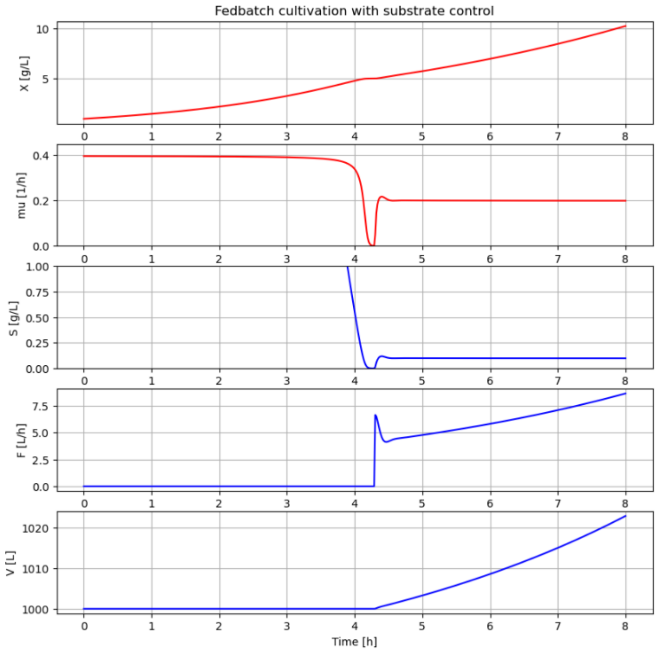


Figure 7. Simulation of fed-batch cultivation with substrate control from on-line substrate measurement and adjustments of the feed rate around the fixed dosage scheme. Focus on the start-up.

tion at the process level and the flexibility of GUI at the control system level. If you prefer, the first level configuration can also be done graphically. And if your Modelica software does not support package parametrisation you can use the procedure described in the previous section.

6.9 The importance of FMU for the library

For bioprocess simulations it is important to let it be part of the wider context of data analysis and detailed cellular modelling. Therefore compilation to FMU of high quality for further integration to Python (or Julia, Matlab) is a very important part.

One practical aspect is that information about media and culture etc in the Modelica model is good to have accessible from the FMU. This information is often declared as constants. Today vendors differ about whether constant information is available in the FMU, and OpenModelica has not implemented this facility, just yet.

7 Availability of Bioprocess Library

The library has been used in a few industrial projects over the years where I have been involved as consultant. It has also been used for teaching operators. These interactions have to some extent set the priorities made in the library development. The name Bioprocess Library is a registered trademark and owned by me.

I am now interested in that more people use the library and hope to get ideas back for further development. One possibility is to make it publicly available at GitHub. Information will be given at the GitHub-page, see (*BPL Applications* 2023).

At GitHub I already provide FMU-compiled demo examples with Jupyter notebooks that can be downloaded, or run from the web-browser using Google Colab virtual machines. The focus is here on further developing usage of simulation and post-processing using Python, by the GitHub-community. The examples include both the textbook culture model here, as well as models of microbial yeast (*S. cerevisiae*) and mammalian cell culture (CHO) for recombinant protein production. The cultures are run as batch, fed-batch, continuous and perfusion. The examples include dynamics of operation, model calibration, sensitivity analysis, design space calculation, scale-down, regulator tuning and process optimisation.

8 Concluding remarks

The presentation has focused on structural design aspects of the Bioprocess Library, and little on the content. Design aspects of more general interests are:

- Part of the user configuration is done by equations in a certain format as described in Section 3 and exemplified in Listing 4. The other part of the user configuration is done traditionally using components, as described in Section 4 and exemplified in Listing 6.
- The parametrisation of components with media are done on the package level instead of on the individual component level, as described in Section 4 and exemplified in Listing 5 and 6.
- The GUI varies between different vendors and not all supports parametrisation on the package level. Even for those vendors that allow GUI configuration as in Listing 6, it is not clear how to interact with the formal parameters on the package level using the GUI.

Hopefully, the library will lower the threshold to use Modelica for simulation of bioprocesses, and the unorthodox design appreciated. Suggestions for including important new components to facilitate usage are welcome! There is also a public place for developing and sharing Jupyter notebooks addressing bioprocess questions using simulation in combination with other tools.

Acknowledgements

I thank Stéphane Velut at Modelon (now at Emulate Energy) for encouraging my work on the library and especially for giving me an idea of what concepts to borrow from MSL Fluid and Media, as well as improvements of parts of the code. Thanks to John Batteh (Modelon) for review and thoughts around GUI. I also thank Adrian Pop at Linköping University for giving me feedback on how to simplify the adaptation of EquipmentLib to the actual application. I am also grateful for the Modelica Stackoverflow community for concrete help as well as ideas.

References

Åkesson, Johan et al. (2010). “Modelling and optimization with Optimica and JModelica.org - languages and tools for solving large-scale dynamic optimization problems”. In: *Computers & Chemical Engineering* 34.11, pp. 1737–1749.

Andersson, Christian, Johan Åkesson, and Claus Führer (2016). *PyFMI: A Python package for simulation of coupled dynamic models with Functional Mock-up Interfaces*. Tech. rep. Lund University: Centre for Mathematical Science. URL: https://lucris.lub.lu.se/ws/portalfiles/portal/7201641/pyfmi_tech.pdf.

Axelsson, Jan Peter (2018-01). “Integrating microbial genome-scale flux balance models with JModelica and the Bioprocess Library for Modelica”. In: *Proceedings of the 21st Nordic Process Control Workshop*. Åbo Akademi University, pp. 126–127.

Axelsson, Jan Peter (2019-08). “Simplified model of CHO-cultivation in Bioprocess Library for Modelica – some experience”. In: *Proceedings of the 22st Nordic Process Control Workshop*. Technical University of Denmark, Lyngby, pp. 56–63. ISBN: 978-87-93054-88-2.

Axelsson, Jan Peter (2021-02). “Design aspects of Bioprocess Library for Modelica”. In: *OpenModelica Workshop*. Linköping University.

Axelsson, Jan Peter (2022-03). “Interpretation of responses to bolus-feeding during CHO-fedbatch cultivation using an extended bottleneck model and simulation with Bioprocess Library for Modelica”. In: *Proceedings of the 23st Nordic Process Control Workshop*. Luleå University of Technology, p. 34.

Baharev, Ali and Arnold Neumaier (2012-09). “Chemical process modelling in Modelica”. In: *Proceedings of the 9th International Modelica Confererence*.

Bioreactor-scaling-tool (2023). URL: https://www.cytivalifesciences.com/en/us/solutions/bioprocessing/knowledge-center/simplify-bioreactor-scale-up-and-scale-down (visited on 2023-08-15).

BPL Applications (2023). URL: https://github.com/janpeter19 (visited on 2023-08-15).

Brugård, Jan et al. (2009-09). “Creating a bridge between Modelica and systems biology community”. In: *Proceedings of the 7th International Modelica Confererence*. DOI: 10.3384/ecp09430016.

Ebrahim, Ali et al. (2013). “COBRApy: Constraint-based reconstruction and analysis for Python”. In: *BMC Systems Biology* 7, pp. 74–79. DOI: 10.1186/1752-0509-7-74.

EMBL BioModels (2023). URL: https://www.ebi.ac.uk/biomodels/ (visited on 2023-08-15).

Franke, Rüdiger et al. (2009-09). “Stream connectors - an extension of Modelica for device-oriented modelling of convective transport phenomena”. In: *Proceedings of the 7th International Modelica Confererence*. DOI: 10.3384/ecp09430078.

Fritzson, Peter (2015). *Principles of Object Oriented Modeling and Simulation with Modelica 3.3*. 2nd ed. Wiley. ISBN: 9781118859124.

Hu, Wei-Shou (2020). *Cell Culture Bioproces Engineering*. 2nd ed. CRC Press. ISBN: 9781498762861.

Jensen, Kristian, Joao G. R. Cardoso, and Nikolaus Sonnenschein (2017). “Optlang: An algebraic modeling language for mathematical optimization”. In: *Journal of Open Source Software* 2.(9), p. 139. DOI: 10.21105/joss.00139.

Modelon (2023). URL: https://help.modelon.com/latest/guides/propagate_class/?h=propaga (visited on 2023-08-15).

Olsson, Hans et al. (2008-03). “Balanced models in Modelica 3.0 for increased model quality”. In: *Proceedings of the 6th International Modelica Confererence*., pp. 21–33.

Ploch, Tobias, Eric von Lieres, et al. (2020). “Simulation of differential-algebraic equation systems with optimization criteria embedded in Modelica”. In: *Computer & Chemical Engineering* 140. DOI: https://doi.org/10.1016/j.compchemeng.2020.106920.

Ploch, Tobias, Xiao Zhao, et al. (2019). “Multiscale dynamic modeling and simulation of a biorefinery”. In: *Biotech. Bioeng.* 116, pp. 2561–2574. DOI: 10.1002/bit.27099.

Sonnleitner, Bernhard and Othmas Käppeli (1986). “Growth of *Saccharomyces cerevisiae* is controlled by its limited respiratory capacity: formulation and verification of a hypothesis”. In: *Biotech. Bioeng.* 28, pp. 927–937.

SystemComponent (2023). URL: https://doc.modelica.org/Modelica%204.0.0/Resources/helpWSM/Modelica/Modelica.Fluid.UsersGuide.BuildingSystemModels.SystemComponent.html (visited on 2023-10-28).

UC San Diego BiGG Models (2023). URL: http://bigg.ucsd.edu (visited on 2023-08-15).

Wiechert, Wolfgang, Stephan Noack, and Atya Elsheikh (2010). “Modeling language for biochemical network simulation: reactions vs equation based approaches”. In: *Adv Biochem Engin/Biotechnol* 121, pp. 109–138. DOI: 10.1007/10_2009_64.

Creating cardiovascular and respiratory models using Physiolibrary 3.0

Marek Mateják[1,2]

[1]Institute for Clinical and Experimental Medicine, Czech,
marek.matejak@ikem.cz

[2]First medical faculty, Charles University in Prague, Czech,
marek.matejak@lf1.cuni.cz

Abstract

The free open-source Physiolibrary version 3.0 (https://github.com/MarekMatejak/Physiolibrary) has transformed components from physiological domains such as hydraulic (cardiovascular), thermal, osmotic, and chemical into the Modelica Standard Library (MSL) concept of Fluid/Media and Chemical library. Components are extended to include gas transports, acids-bases, electrolytes, nutrient delivery, and endocrines by simply selecting pre-made media. They can be connected directly (same medium) or across membranes (different media), allowing small physiological models to be integrated into more quantitative models with minimal effort.

Keywords: physiology modelling, physiological simulation, quantitative physiology, physiological model, cardiovascular, respiratory, physiolib, physiolibrary, physiomodel, systems biology, medical simulation

1 Introduction

Earlier versions of Physiolibrary (Mateják et al. 2014) mainly contain components for individual domains, e.g., hydraulic, osmotic, chemical, and thermal resistances. These components were defined to implement large integrative models such as Physiomodel (Mateják and Kofránek 2015). The hydraulic domain was not suitable for gas transport. The osmotic domain was inaccurate and difficult to connect with the chemical domain of protein distributions and electrolytes. The first version of the chemical domain was controlled by concentration gradients instead of electrochemical potentials (Mateják 2015). It was difficult for the user to implement obvious interactions between the domains. We addressed all these problems and proposed solutions. The result is that version 3.0 of Physiolibrary allows the use of standard Modelica Fluid Connectors (Casella et al. 2006) and electrochemical connectors for cross-compartment transport of substances (Mateják et al. 2015). Fluid connectors transport media such as blood, air, interstitial fluid, intracellular fluid. Drag-and-drop connections of these connectors define equations for pressures, mass flows, heat flows and mass fractions of substances between components. Electrochemical connections lead via free base substance forms. For example, the total mass fraction of carbon dioxide is represented as part of the composition of blood in fluid connector, but free dissolved carbon dioxide in blood plasma or bicarbonate in blood plasma are its electrochemical connectors proposed to model the electrochemical CO2 fluxes. Since the selected forms are precisely determined by the composition of the blood, it is not necessary to store them and pass them through the fluid connector. They are only expressed and calculated when needed.

2 Methods

2.1 SI units

In medicine, many obscure units are still in use such as mmHg (millimeters of mercury), cmH2O (centimeters of water) for pressure, calories for energy, chemical equivalents for electric charge, degrees Celsius for temperature, and so on. Modelica allows you to define these as display units. This means that it is possible to output graphs in the selected units or even set values in parametric dialogs in these selected units. However, the variables within the model in SI units explain the compatibility between all components and models. Also, the selection of zero offset is better for use within connectors and state variables, since, for example, absolute pressure is well defined, as opposed to relative pressure in circuits with heterogeneous environments that have different pressures. To see nice pressure values, it is necessary to use a pressure sensor instead of looking at the variables of the connector “p”. Even if the user wants to create a small physiological model, it is much better to achieve these interfaces because they allow others to easily reuse it without modifications or adapters.

2.2 State and connector variables

For gaseous substances, volume changes with pressure and temperature, so it is always better to use mass and mass flows instead of volume and volume flows. And, of course, volumes can always be evaluated by including pressures, temperatures, and the composition of the medium with known masses. For the same reason, it is better to use mass fractions of substances instead of volume fractions, concentrations, molalities, molarities, or even mole fractions for the compositional state of the medium. The molar quantities may change due to chemical binding (e.g., to a transporter protein), but the mass of the substance change only with external flows. If the state of the medium is known, all these quantities can be evaluated for output or parametric purposes.

2.3 Elastic vessels

One of the most important components in Physiolibrary 3.0 is Fluid.ElasticVessel. It accumulates the mass of the medium, e.g. blood, air, lymph, interstitial fluid, intracellular fluid. From the accumulated mass, heat and substances, volume, pressure, temperature, concentrations and other properties of the accumulated medium are expressed. In addition if the user use chemical substances connectors, special equations are included as Medium.ChemicalSolution model. In this case, electrochemical potentials and enthalpies are expressed to allow passive and active transport, for example, through the alveolar membrane, the capillary membrane, the cell membrane or the CSF membrane. Electrochemical processes and their calculation (Mateják 2015) are from the chemical library (Mateják et al. 2015). Here osmotic transport is the result of balancing the chemical potential of water. Similarly, Donnan's equilibrium (Donnan 1911) is the result of balancing the electrochemical potentials of electrolytes (Atkins and De Paula 2011) at a semipermeable membrane. And also, active transport or signal transduction can be modeled as electrochemical reactions involving membrane proteins.

2.4 Medium

Physiolibrary 3.0 defines examples for the following media:
Water – as pure incompressible water with constant heat capacity without any substance inside
Air – as an ideal gas model with oxygen, carbon dioxide, nitrogen and water
Blood – as an incompressible fluid containing many physiological substances such as blood gases, electrolytes, red cells, nutrients, proteins and hormones. Thanks to the shift of numerical tolerances with predefined nominal values for each substance, the calculation is numerically stable, even if the ratio between the mass fractions of substances is 10^9 (e.g. mass fraction of water / mass fraction of thyrotropin). Blood contains equations for haemoglobin oxygen saturation, acid-base balance, and carbon dioxide transfers to achieve physiological conditions in the transport of blood gases under variable conditions (Mateják, Kulhánek, and Matoušek 2015).
BodyFluid – as an incompressible fluid that simplifies other physiological fluids such as interstitial fluid, intracellular fluid, cerebrospinal fluid, or urine. In Physiolibrary 3.0 this medium represents only a homogeneous chemical solution without special transfers or binding of substances inside.

3 Results

3.1 Blood gases interface

The library has prepared a blood medium containing the dissociation model of common gasses such as oxygen, carbon dioxide, and carbon monoxide (Siggaard-Andersen 1971; Siggaard-Andersen and Siggaard-Andersen 1990; SIGGAARD-ANDERSEN and SIGGAARD-ANDERSEN 1995). With this model, we can build a gas transport between air and blood. First, we add a component ElasticVessel from the Fluid.Components package. ElasticVessel is a container for the medium in which blood accumulates oxygen and carbon dioxide. It contains dynamic calculation of blood volume, blood temperature, blood pressure, blood composition and other blood properties. In the parameter dialog we set blood from the Media package as the medium in this component. This will link the equations and properties of the medium to the model of this component. As initialization we can set normal predefined arterial blood by setting the massFractions_start parameter to predefined values Blood.ArterialDefault. Then we check the useSubstances checkbox to enable the connectors with free blood substances and the complex model of these substances in them. The bundle of substance connectors is located on the left side of the icon. Now we can handle a freely dissolved basic chemical substance defined in the medium. "Free dissolved" means that the substance connector contains chemical potential and flows only for unbound molecules. And "basic" means that this molecule is not connected in a cluster with other molecules. For example, H2O molecules are connected by hydrogen bonds, so water in these chemical compounds is represented only by the chemical potential and molar flux of the free, unbound H2O molecules. The clusters and bonds can nevertheless be calculated internally if needed (Mateják and Kofránek 2020).
The next component type we use in this model is "GasSolubility" from the "Chemical.Components" package. It represents the chemical processes of gas-liquid solubility for the selected gas molecule.
To define the air source of gas substances, we can use ExternalIdealGasSubstance from the Chemical.Sources package. Here in the listbox of the parameter dialog we should be able to set „O2(g)“ or "CO2(g)“ or “CO(g)”. Then we can define a very small partial pressure of oxygen (1 mmHg) together with a typical partial pressure of carbon dioxide (40 mmHg) and a very small partial pressure for carbon monoxide (1e-6 mmHg). This settings cause that the blood is losing the oxygen during simulation.

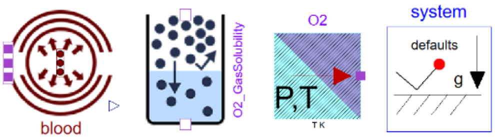


Figure 1. Selected Physiolibrary components: ElasticVesel, GasSolubility, ExternalGasSubstance, System

The standard Modelica.Fluid.System component is used to pass on ambient pressure, temperature (37°C), and gravity acceleration. The parameters of this component are accessible throughout the model.
After we have defined all the components of the model, we can connect them.

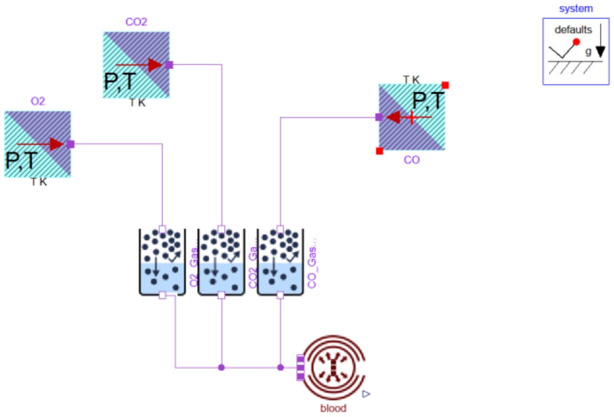
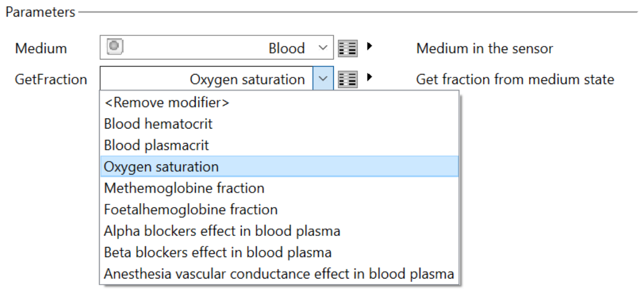


Figure 2. Model of blood gases

To see the current values of the model, it is a good practice to use sensors. These components do not affect other values of the model here, but they represent the initial values in the expected form. To measure the oxygen saturation in blood, we can use the Fraction sensor from the Fluid.Sensors package. As parameterization of this component, we need to specify the media model and the name of the predefined fraction function in the medium.



Parameters

Medium Blood Medium in the sensor

GetFraction Oxygen saturation Get fraction from medium state

<Remove modifier>
Blood hematocrit
Blood plasmacrit
Oxygen saturation
Methemoglobine fraction
Foetalhemoglobine fraction
Alpha blockers effect in blood plasma
Beta blockers effect in blood plasma
Anesthesia vascular conductance effect in blood plasma

Figure 3. Fraction sensor parameterized for measurement of oxygen saturation in blood

We can define a similar measurement for blood oxygen partial pressure by using the PartialPressure component from the Fluid.Sensors package. During parameterization we have to select the state of matter and the substance definition in the similar way.

The entire model with all source codes is accessible in one of the examples within the library as Fluid.Examples.BloodGasesEquilibrium.

If we run a simulation of this model, we can see the dynamic oxygen-haemoglobin dissociation (Severinghaus 1979) as a relationship between oxygen saturation and oxygen patrial pressure in blood.

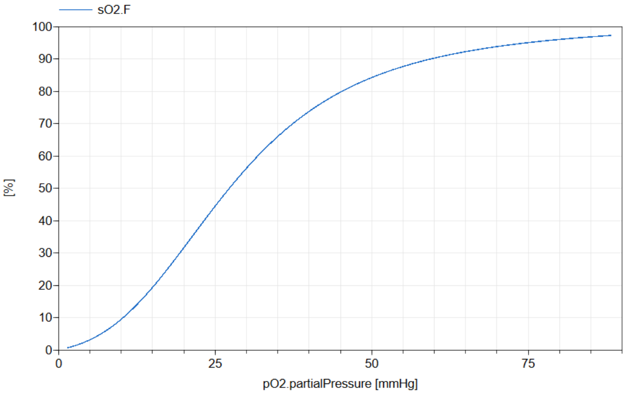


Figure 4. Result of the simulation as oxygen saturation curve

3.2 Respiratory unit

Physiolibrary has some predefined models of organs or their functional units composed of components of the base library. One of these components is RespiratoryUnit in the Organs.Lungs.Components package. For gas transport between the air and the blood, the same basic components are used as in the previous model.

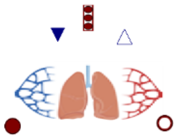
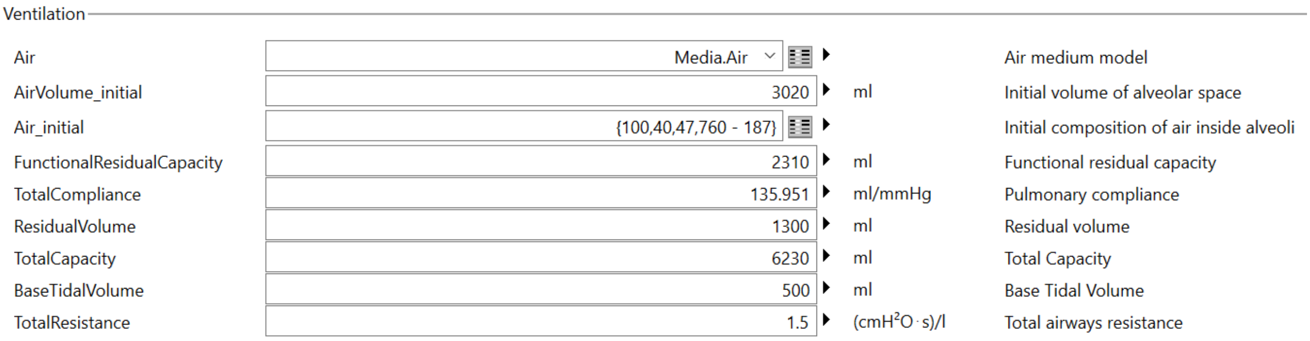


Figure 5. RespiratoryUnit component

The parameterization of the RespiratoryUnit is divided into ventilation, blood perfusion and diffusion of gases through the capillary and alveolar membrane.

Ventilation parameters are based on the physiological characteristics of the lungs, such as functional residual capacity (the volume of air remaining after a normal passive expiration), residual volume (the volume of air remaining after full expiration), total capacity (the volume of air remaining after full inspiration), base tidal volume (inhaled/exhaled volume during a normal breath), total compliance, initial air volume, and initial air composition.



Ventilation

Air	Media.Air		Air medium model
AirVolume_initial	3020	ml	Initial volume of alveolar space
Air_initial	{100,40,47,760 - 187}		Initial composition of air inside alveoli
FunctionalResidualCapacity	2310	ml	Functional residual capacity
TotalCompliance	135.951	ml/mmHg	Pulmonary compliance
ResidualVolume	1300	ml	Residual volume
TotalCapacity	6230	ml	Total Capacity
BaseTidalVolume	500	ml	Base Tidal Volume
TotalResistance	1.5	(cmH^{2}O·s)/l	Total airways resistance

Figure 6. Parameterization of ventilation in dialog of the RespiratoryUnit component

Perfusion settings are based on blood and vessel parameters such as blood model, initial blood volume, initial blood composition, ZeroPressureVolume (maximum blood volume in vessels that does not generate pressure), vessel compliance, and vessel conductance.

The RespiratoryUnit can be connected to the respiratory muscles via chest, where a negative or even a positive external pressure is generated by respiratory muscles. The medium of this compartment can be chosen as pleural fluid. This compartment should have a non-zero internal space (lung volume) that can collapse below the relaxed volume. Thus, the current volume of respiratory units can be related to the internal space of the chest, and the external pressure is transmitted from the muscles to the lungs through the pleural cavity (where it is displaced by the internal space). This pattern is illustrated by the examples SimpleRespiration and Respiration in the package Fluid.Examples.

3.3 Tissue unit

To demonstrate oxygen consumption and carbon dioxide production in body tissue metabolism, we can define a TissueUnit. This unit does not solve hypoxic situations, but it can be used for normal body conditions. Here, oxygen consumption (e.g., 15mmol/min) and carbon dioxide production (e.g., 12mmol/min) are constant parameters during simulations propagated by SubstanceOutflow and SubstanceInflow components from the Chemical.Sources package. Blood is connected from systemic arteries to systemic veins via tissue capillaries, using typical connections for modeling the cardiovascular system (Kulhánek, Kofránek, and Mateják 2014).

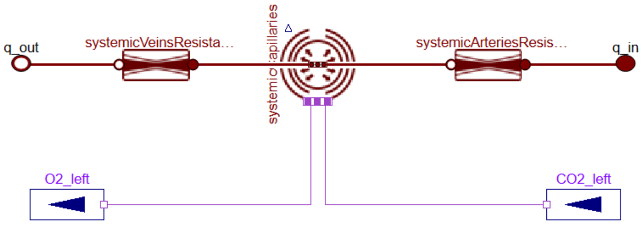


Figure 7. Diagram of simple tissue unit

3.4 Simple respiratory-cardiovascular model

If you combine all these principles, you can create a model of respiration, blood circulation and blood-gas transport. Non-medical users usually focus on oscillatory models, as presented in the examples MeursModel2011.HemodynamicsMeurs_flatNorm or Respiration in the Fluid.Examples package. However, precise non-oscillatory models can also be defined for long-term physiological simulations. Oscillation from breath to breath or even from heartbeat to heartbeat does not affect the calculated mean values that are physiologically significant (e.g. mean pressure, cardiac output, heart rate, respiratory volume, respiratory rate, etc.). Therefore, it is good practice in medical physiology to define non-oscillatory long-term cardiovascular and respiratory models (Hester et al. 2011).

A non-oscillatory respiratory model can be defined by the same RespiratoryUnit as the oscillatory one. However, the connection of the respiratory tract must be defined by a separate air inflow and outflow. And the dead space should be defined in parallel connection. Similarly, the pulmonary shunt (where the blood of the pulmonary circulation does not flow through the ventilated alveoli) should be defined in parallel connection with the blood perfusion in the RespiratoryUnits.

The non-oscillating cardiovascular model is based on non-oscillating pumps representing the right and left heart delivering cardiac output to the bloodstream.

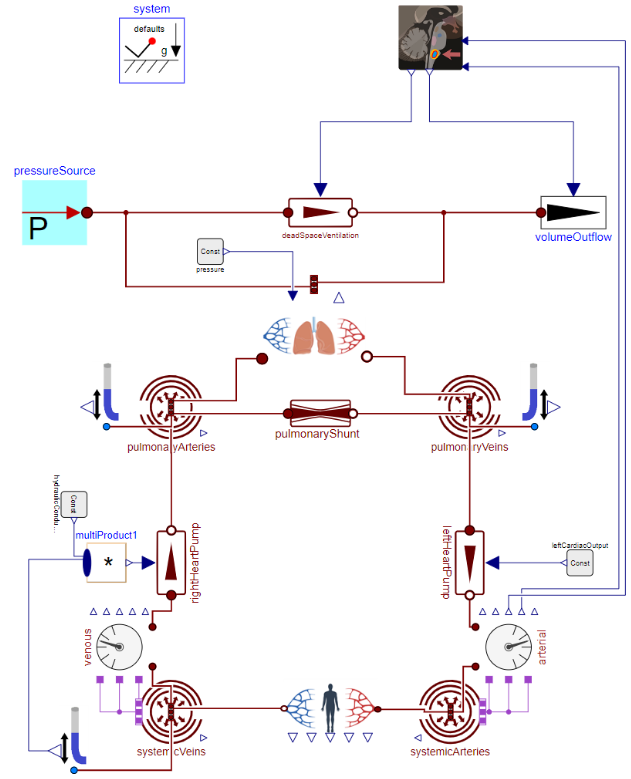


Figure 8. Respiratory-cardiovascular example

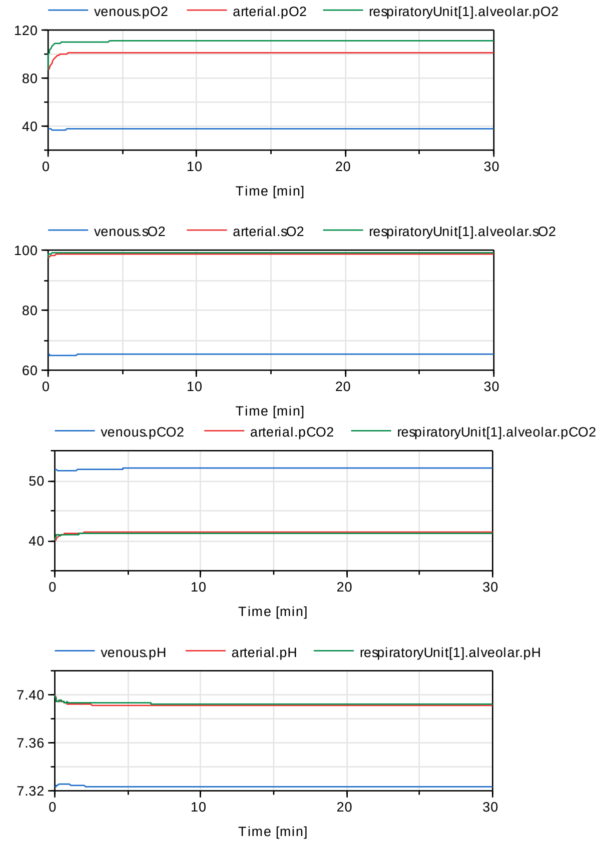


Figure 9. Results of respiratory-cardiovascular example

4 Discussion

The models presented are only examples of the use of the library. The components of the Physiolibrary are general enough to be included in more specific or/and more complex models. Today version of the blood medium is designed for the transport of blood gases, but our goal is to make it general for the transport of physiological substances. Using the nominals, even hormones and endocrines (physiologically active substances in very low concentrations) can be solved numerically in the same way. With chemical processes such as passive and active transport or signal transmission, more complex models of tissues and organs can be defined. If we connect tissues and organs, we can easily create a model of the whole body (Mateják and Kofránek 2015). With specific bodies, we can virtually transplant organs from one body to another and so on. With the non-oscillation approach, we can even simulate years of life or perhaps one day even the whole life of an organism.

This kind of mathematical modelling leads to virtual experiments that could improve experiments before using animals or humans. This minimalizes number of iteration and changes in experimental conditions, which can improve the quality of research and shorten research time. And it could also be a platform for sharing results in the form of well-defined and structured models.

-

Acknowledgements

The author would like to thank Jiří Kofránek for phone consultation and all the sponsors of https://github.com/sponsors/MarekMatejak for additional support.

References

Atkins, Peter, and Julio De Paula. 2011. *Physical Chemistry for the Life Sciences*. Oxford University Press, USA.

Donnan, F. G. 1911. “Theorie Der Membrangleichgewichte Und Membranpotentiale Bei Vorhandensein von Nicht Dialysierenden Elektrolyten. Ein Beitrag Zur Physikalisch-Chemischen Physiologie.” *Zeitschrift Für Elektrochemie Und Angewandte Physikalische Chemie* 17: 572–81. https://doi.org/10.1002/bbpc.19110171405.

Hester, Robert L., Alison J. Brown, Leland Husband, Radu Iliescu, Drew Pruett, Richard Summers, and Thomas G. Coleman. 2011. “HumMod: A Modeling Environment for the Simulation of Integrative Human Physiology.” *Frontiers in Physiology* 2.

Kulhánek, Tomáš, Jiří Kofránek, and Marek Mateják. 2014. “Modeling of Short-Term Mechanism of

Arterial Pressure Control in the Cardiovascular System: Object-Oriented and Acausal Approach." *Computers in Biology and Medicine* 54: 137–44.

Mateják, Marek. 2015. "Formalization of Integrative Physiology." Dissertation Thesis, Charles University in Prague.

Mateják, Marek, and Jiří Kofránek. 2015. "Physiomodel-an Integrative Physiology in Modelica." In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 1464–67. IEEE.

Mateják, Marek, and Jiří Kofránek. 2020. "Molar Amount of Water." *Medsoft* 32 (1): 59.

Mateják, Marek, Tomáš Kulhánek, and Stanislav Matoušek. 2015. "Adair-Based Hemoglobin Equilibrium with Oxygen, Carbon Dioxide and Hydrogen Ion Activity." *Scandinavian Journal of Clinical & Laboratory Investigation* 75 (2): 113–20. https://doi.org/10.3109/00365513.2014.984320.

Mateják, Marek, Martin Tribula, Filip Ježek, and Jiří Kofránek. 2015. "Free Modelica Library of Chemical and Electrochemical Processes." In *11th International Modelica Conference, Versailles, France*, 118:359–66. Linköping University Electronic Press, Linköpings universitet.

SIGGAARD-ANDERSEN, MADS, and Ole SIGGAARD-ANDERSEN. 1995. "Oxygen Status Algorithm, Version 3, with Some Applications." *Acta Anaesthesiologica Scandinavica* 39: 13–20.

Siggaard-Andersen, O. 1971. "Oxygen-Linked Hydrogen Ion Binding of Human Hemoglobin. Effects of Carbon Dioxide and 2, 3-Diphosphoglycerate I. Studies on Erythrolysate." *Scandinavian Journal of Clinical & Laboratory Investigation* 27 (4): 351–60.

Siggaard-Andersen, O., and M. Siggaard-Andersen. 1990. "The Oxygen Status Algorithm: A Computer Program for Calculating and Displaying PH and Blood Gas Data." *Scandinavian Journal of Clinical & Laboratory Investigation* 50 (S203): 29–45.

Design proposal of a standardized Base Modelica language

Gerd Kurzbach[1] Oliver Lenord[2] Hans Olsson[3] Martin Sjölund[4] Henrik Tidefelt[5]

[1]ESI Germany GmbH, Germany, gerd.kurzbach@esi-group.com>
[2]Robert Bosch GmbH, Germany, oliver.lenord@de.bosch.com
[3]Dassault Systèmes, Sweden, hans.olsson@3ds.com
[4]Department of Computer and Information Science (IDA), Linköping University, Sweden, martin.sjolund@liu.se
[5]Wolfram MathCore, Sweden, henrikt@wolfram.com

Abstract

This paper is presenting the design proposal of a simplified version of the Modelica language. Base Modelica is designed to serve as an intermediate representation enabling a clean separation of front-end and back-end matters when processing a Modelica model. Furthermore, it is designed to allow restructuring the Modelica Language Specification considering two parts: the basic features and the advanced language constructs.

After discussing the motivation, solution approach, and risks, the paper is highlighting a selection of design choices that have been made for the current pre-release version of the language. Code examples are given to illustrate and highlight various aspects of the language. Open issues, conclusions, and an outlook finalize the paper.

By attracting more tool vendors and researchers to work with this intermediate representation the whole Modelica community is expected to benefit from new utilities to inspect, analyze, optimize, and process equations-based models in general and Modelica models in particular.

Keywords: Modelica Language, intermediate representation, equation-based language, language design

1 Introduction

The Modelica language, published as v1.0 in September 1997, has been widely accepted as modeling language to describe the behavior of cyber-physical systems. Various tool vendors have developed and successfully marketed simulation environments including a Modelica kernel that is able to translate Modelica models describing mixed continuous-discrete differential algebraic-equation system (hybrid DAE) into highly efficient simulation code. From the very beginning, the development of tools has been accompanied by the development of model libraries covering a wide spectrum of physical domains and fields of application. The Modelica Language Specification and the Modelica Standard Library are maintained by the nonprofit Modelica Association and are licensed under the open source 3-clause BSD License for the Modelica Association. An ecosystem of researchers, tool vendors, library developers and users has evolved over the years and is continuously growing.

The LLVM compiler infrastructure (Lattner and Adve 2004) has turned out to be a great success over the last twenty years. A central part of the design is the well specified LLVM intermediate representation, which has been a source of inspiration for also introducing a well specified intermediate format for Modelica translation and tool infrastructure. Earlier, the Java Virtual Machine architecture (Lindholm et al. 2023) has also proven intermediate languages to play a central role in the development of ecosystems around a language.

1.1 State of the art

The versatile modeling language Modelica is in particular well-suited to formulate multi-physics problems starting from first principles in an easily comprehensible textbook style. The combined textual and graphical representations provide very accessible views on component, subsystem and system level. A high level of reuse is enabled through the concept of acausal connectors. Variants can be managed in a convenient fashion through inheritance and modifications to avoid code duplication, which leads to a much-improved maintainability of model libraries and performing simulation studies of entire system families. The understandability of domain libraries and larger system models hinges on designing a proper model architecture, which requires a good understanding of Modelica's object-oriented programming model. High-quality libraries are available that demonstrate how to apply powerful object-oriented language constructs in a meaningful way, along with graphics, documentation, and other usability enhancements.

Modelica modeling tools support these concepts through integrated development environments (IDE) providing multiple views on the model to navigate through the instance hierarchy and apply modifications on different levels of the system structure. In addition, the output of a "so-called" *flattened model* is supported by common Modelica compilers. This textual output gives an unobscured view of the effective set of equations as they occur after instantiation and lowering of hierarchical models, applying replacements and other modifications. Typically, the flat Modelica output looks like Modelica code, but it is not a valid Modelica model that could be further processed

DOI
10.3384/ecp204469

by other tools. The flat Modelica output is not standardized and differs between tools and can even differ between tool versions.

For tool vendors the development of a Modelica compiler is a substantial investment, strongly dependent on the availability of highly educated and knowledgeable experts in compiler construction. Especially the concept of replaceable packages – heavily used in the media and thermo-fluid libraries – are very involved to get perfectly right. The OpenModelica project estimates 40 person-years spent in the project, with 15-20 person-years for a minimal simulation environment (excluding the graphical user interface). Approximately 7 person-years are estimated for the development of a Modelica front-end capable of parsing and lowering a Modelica model into a form that can be printed as flat Modelica code. This lowered model is the starting point for further model transformations, typically carried out by the so-called back-end, towards an optimized target-specific simulation code.

As of today, the Modelica Association is listing 10 Modelica simulation environments. Not all of these have their own Modelica kernel. The test coverage of available Modelica libraries differs between the Modelica kernels.

The latest version of the Modelica Specification (Modelica Association 2023) reads 300 pages (excluding the appendix) covering basic language concepts (operators, expressions, types, classes, arrays, functions, declarations, scoping, name lookup), hybrid DAE modeling-related aspects (equations, events, synchronous language elements, state machines) and object-oriented respectively component-oriented language elements (units, interfaces, connectors, connections, stream connectors, inheritance, overloaded operators, packages).

1.2 Problem statement

While Modelica is very powerful and easy to use, it is a complex language. The high complexity leads to challenges on different levels.

End-users (modeling and simulation experts) and Modelica library developers are confronted with:

- Compatibility issues between different Modelica tools in part due to inconsistent interpretations of the Modelica specification.
- Limited expressiveness of existing flat Modelica outputs for debugging unexpected behaviors, e.g., priority of conflicting start values, or clock partitioning.
- Lack of third-party utilities for operating on the flat Modelica output due to lack of standard.
- Lock-in effects due to third-party utilities getting tied to a specific Modelica tool.

Tool vendors are facing:

- High onboarding effort for new employees working on the Modelica translator to become productive due to interdependence of front-end and back-end matters.
- High entry barrier for non-Modelica tools to participate in the Modelica ecosystem.
- Difficulty to foresee and support all possible usages of the language.
- Lack of a common format to settle questions about the interpretation of the Modelica Language Specification with other tool vendors.

The design group of the Modelica Language Specification (MAP-Lang) is challenged by:

- Need to guarantee the consistency of a large and complex specification.
- Hard to integrate changes or enhancements due to many potential side effects to be considered.
- Difficulty to specify the semantics in an unambiguous way.
- Risk of language innovations creating high implementation efforts.

This leads to the situation that the objective of Modelica as a widely accepted, free and open modeling language is threatened:

- The entry barrier for new tool vendors is very high.
- The testing effort of Modelica libraries to guarantee compatibility across different tools is so high that only a limited number of tools are fully supported.
- It is difficult to use as foundation for other standards due the high complexity.
- A risk of vendor lock-in persists despite the commitment to Modelica as an open standard.
- Poor ability to respond quickly to innovations in competing modeling technologies.

1.3 Solution approach

We propose a standardized **Base Modelica** language that could become an integral part of the Modelica Specification. The translation of a (full) Modelica model would then be described as a two-step process, where high-level language constructs are first removed by *lowering* the model to Base Modelica, after which the Base Modelica semantics define the dynamic simulation behavior.

The Modelica and Base Modelica languages have a large overlap in the syntax of expressions, functions, equations, and algorithms. These parts should be defined in a common part of the Modelica Specification, where any differences in requirements or semantics between (full)

Modelica and Base Modelica are clearly marked. In addition to the common part, the high-level language constructs of Modelica would be described in one part, while lower-level constructs specific to Base Modelica are described in another.

This is similar to the approach being taken by System Modeling Language (SysML), which is being restructured to be an extension to the Kernel Modeling Language (KerML) instead of a UML profile (The Object Management Group 2023). The difference between Modelica and SysML is that SysML will add support for domain-specific applications through language extensions whereas these are still included in the (full) Modelica language.

1.4 Benefits

From the perspective of the MAP-Lang, the separation of basic language constructs (Base Modelica) from the more high-level constructions will facilitate more efficient work and rapid development of the two aspects of the Modelica language and generally improve the readability and maintainability of the specification. Examples of how high-level constructs are lowered to Base Modelica will help to avoid misinterpretations. Changes applicable to Base Modelica can be discussed and evaluated before deciding how to integrate them in Modelica. A working group with focus on the equation model and simulation semantics could play a very important role in future developments of new language features such as varying-structure systems, or integration with PDE solvers.

From a tool vendor perspective, organizing the development work of a Modelica tool will be easier thanks to a natural separation into front-end and back-end matters, with the front-end taking care of the lowering the Modelica model to Base Modelica and the back-end transforming the Base Modelica model into an executable form, e.g., a simulator. A standardized Base Modelica output will allow a much easier identification of compatibility issues between different tools. The much simplified Base Modelica language will provide an entry point for new tools to enter the Modelica ecosystem.

These types of new tools and services could be:

- Other high-level languages or modeling tools using Base Modelica as a target language, e.g., dedicated control engineering tools, or symbolic math packages.

- Advanced model transformation techniques applied on the equation level.

- Specialized tools providing advanced analysis (e.g., occurrence of algebraic loops, model-based fault detection and isolation) and/or visualizations of equation systems (e.g., bipartite graphs).

- Extraction and injection of equations to simplify or reduce the model for simulation speed-up.

- Platform for academic research on dynamic systems, e.g., symbolic or numeric methods.

In the context of the publicly funded ITEA3 project 15016 EMPHYSIS (Sep. 2017 – Feb. 2021), an early prototype for the exchange of equation-based models between Modelica and non-Modelica tools has been developed. Based on this prototype two use cases have been evaluated and documented as demonstrators (EMPHYSIS Consortium 2021, D7.3 and D7.4) to illustrate the benefit of having an equation-based representation in a model-based development workflow for embedded control and diagnosis functions.

The end-users and Modelica library developers will benefit from a improved portability of their models and libraries due to identified and resolved inconsistencies between Modelica implementations and Modelica Specification. This will allow a more flexible usage of the available tools. Comparison of different compiler back-ends will be possible. In combination with the obfuscation of Base Modelica outputs, sharing of IP-protected models will be simplified.

Furthermore, other model exchange standards could also benefit from a standardized Base Modelica. This is in accordance with the Equation Code proposed as an additional model representation within the eFMI container architecture (Lenord et al. 2021). This additional representation has been proposed for future versions of the eFMI standard aiming to share an acausal representation of the equation system that has served as the basis for the derived algorithmic and target-specific representations. The proposed Base Modelica with some additional restrictions could be directly referenced by the eFMI standard to specify the additional Equation Code model representation that would provide more flexibility and transparency in the generation of code for embedded applications.

1.5 Potential risks

However, we also recognize that there are potential risks that may reduce some of the benefits.

In particular:

- Modelica translators rely on heuristics for symbolic transformations based on the structure of the Modelica code. Some of this structure, e.g., start-value priority, has been given a standardized representation in Base Modelica. However there also exists structure that will require the use of vendor-specific annotations, leading to portability issues of the Base Modelica code. For example the alias elimination selection may depend on whether a variable is conditional.

- Base Modelica has not been designed with intermediate stages of symbolic transformations in mind. Thus its usefulness for representing those stages is not clear.

- Despite the significant effort behind the current state of the Base Modelica design work there is a considerable remaining investment in separating the Modelica Specification along these lines. This could detract from other evolution of the Modelica language.

Understanding the risks should make it easier to avoid the bad consequences.

2 Selected design choices

From the intended usages of Base Modelica the following design goals are derived:

- Simple enough to be attractive for applications that essentially just want a simple description of variables and equations, meaning that many of the complicated high level constructs of Modelica are removed.
- Expressive enough to allow the high level constructs of Modelica to be reduced to Base Modelica without loss of semantics.
- When Base Modelica serves as an intermediate representation of the translation of a higher level language (such as Modelica), errors detected in Base Modelica code shall be traceable to the original code.
- Human readable and writable, since not all usages assume Base Modelica being produced from a higher level language by a tool.

A selection of design choices to achieve these goals is presented in the following subsections.

2.1 Variable naming scheme

Identifiers in Base Modelica fall into three namespaces:

- Space of mangled class names and component references that allow mapping back to a hierarchically structured class tree and simulation result.
- Space of reserved names for current or future use in the Base Modelica specification.
- Space reserved for tools producing Base Modelica code, without mapping to names in a simulation result.

For example, the following Base Modelica parameter would appear as `const.k` in the simulation result:

```
parameter Real 'const.k' = 1.0;
```

In general, the mangling scheme is more involved than just wrapping in single quotes, but the details are omitted for brevity.

The mangled names are always quoted identifiers (*Q-IDENT* in the grammar), and since quoted identifiers are rarely used in Modelica code, it is often easy to guess whether a code fragment is Modelica or Base Modelica by just looking at the names of classes and components.

2.2 Simplified grammar

Base Modelica has a grammar which has been simplified in many ways compared to Modelica, by removing high-level language constructs. A clear sign of this is the many Modelica keywords that are not keywords in Base Modelica, including: **block**, **class**, connect, **connector**, **constrainedby**, each, **expandable**, **extends**, final, flow, **import**, **inner**, **operator**, **outer**, **protected**, **public**, **redeclare**, and stream. However, Base Modelica also comes with syntax for lowered constructs that do not exist in Modelica.

The top-level structure of a Base Modelica program is given by the following piece of grammar:

```
base-modelica :
  VERSION-HEADER
  package IDENT
    ( decoration? class-definition ";"
    | decoration? global-constant ";"
    )*
    decoration? model
        long-class-specifier ";"
    ( annotation-comment ";" )?
  end IDENT ";"
```

A very small Base Modelica model generated from full Modelica could look like this:

Listing 1. A minimal (non-empty) Base Modelica model.

```
//! base 0.1.0
package 'M'
model 'M'
  parameter Real 'const.k' = 1.0;
end 'M';
end 'M';
```

The mandatory Base Modelica version header comment is technically necessary to tell with certainty that this is a Base Modelica listing, and not a Modelica listing. It is included here for completeness, but is generally omitted in examples where it is clear from context or content that the language is Base Modelica.

A *class-definition* in Base Modelica can only be either a record definition, a function definition, or a short class definition, and cannot contain nested class definitions. Similarly, the model defined at the end cannot contain nested class definitions, meaning that all classes are defined in a flat structure under the top-level package (which shall have the same name as the model inside it).

The *decoration* is a source location decoration for use when the Base Modelica code has been generated from another source, such as a Modelica model.

A notable example of syntax added for describing lowered constructs is the modeling of clock partitions, see section 2.13.

2.3 Restricted modification

For any Base Modelica component declaration, modifications are required to be expressed in a way that avoids the need for conflict resolution and complicated merging strategies:

- Hierarchical names are not allowed in modifiers, meaning that all modifiers must use the nested form with just a single identifier at each level.

- At each level, all identifiers must be unique, so that conflicting modifications are trivially detected.

Lookup restrictions ensure that modifications in short class definitions, record definitions, and function definitions can only make use of constant expressions. Further, these modifications are not allowed to specify different values for different elements of an array. As a result, a named type in Base Modelica can be represented very compactly compared to a named type in Modelica.

When lowering a Modelica array component with a heterogeneous modification, the modification needs to be placed inside the Base Modelica `model` part, as model component declarations is the only place where heterogeneous modification is allowed.

Base Modelica does not have the `final` keyword to indicate that further modification is not allowed. For most uses of `final` in Modelica, this just means that violations of `final` must be detected during lowering. However, the special case of a final modification of the `start`-attribute also requires preventing that the `start`-attribute can be modified at the time of simulation initialization, and will be described in subsection 2.11.

2.4 No connect equations

Connect equations in Modelica play an important role to enable reuse of components and build-up a component hierarchy, as illustrated by the very simple example in Figure 1. The Modelica semantics already describes how to transform connect equations into basic mathematical equations. Hence, there is no need in Base Modelica to keep connect equations. Furthermore it would have been difficult to preserve the concept of connect equations, as the lowering process to Base Modelica is removing the hierarchy of components in terms of which the connect equations are defined. This is a significant language simplification compared to Modelica, especially when it comes to expandable connectors.

An example of how the previously mentioned train model is translated from Modelica, see Listing 2, to Base Modelica is shown in Listing 3

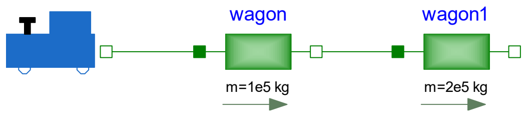


Figure 1. Diagram view of a Modelica model of a train. The `connect`-equations are represented graphically as lines between the components.

Listing 2. Shortened Modelica listing of train model.

```
model Train
  Locomotive locomotive;
  Modelica.Mechanics.Translational.
      Components.Mass wagon(L=50, m=1e5)
    annotation (Placement(transformation(
        extent={{-16,-12},{4,8}})));
  Modelica.Mechanics.Translational.
      Components.Mass wagon1(L=40, m=2e5);
equation
  connect(waggon.flange_b, wagon1.flange_a)
    annotation (Line(points
        ={{4,-2},{18,-2}}, color={0,127,0})
        );
  connect(locomotive.flange_b, wagon.
      flange_a);
end Train;
```

Listing 3. Train model lowered to Base Modelica.

```
//! base 0.1.0
package ’Train’
model ’Train’
  parameter Real ’wagon.m’ = 100000.0 "Mass
       of the sliding mass";
  parameter Real ’wagon.L’ = 50 "Length of
      component";
  parameter Real ’wagon1.m’ = 200000.0 "
      Mass of the sliding mass";
  parameter Real ’wagon1.L’ = 40 "Length of
       component";
  Real ’wagon.s’ "Absolute position of
      center of component...";
  Real ’wagon.flange_a.s’ "Absolute
      position of flange";
  Real ’wagon.flange_a.f’ "Cut force
      directed into flange";
  Real ’wagon.flange_b.s’ "Absolute
      position of flange";
  Real ’wagon.flange_b.f’ "Cut force
      directed into flange";
  Real ’wagon1.s’ "Absolute position of
      center of component...";
  parameter Real ’locomotive.mass.m’;
  parameter Real ’locomotive.m’ = 100000.0
      "Mass of the sliding mass";
initial equation
  ’locomotive.mass.m’ = ’locomotive.m’;
equation
  ’wagon.flange_a.s’ = ’wagon.s’-’wagon.L’
      /2;
  ’wagon.flange_b.s’ = ’wagon.s’+’wagon.L’
      /2;
  ...
  // From connections:
  ’locomotive.flange_b.f’+’wagon.flange_a.f
      ’ = 0.0;
  ’wagon.flange_a.s’ = ’locomotive.flange_b
      .s’;
  ’wagon.flange_b.f’+’wagon1.flange_a.f’ =
      0.0;
  ’wagon1.flange_a.s’ = ’wagon.flange_b.s’;
end ’Train’;
end ’Train’;
```

2.5 No conditional components or deselection

Conditional components in Modelica allow a limited structural variation that complements the more flexible *re-*

placeable concept. Base Modelica does not have conditional components.

When lowering to Base Modelica we must thus evaluate the condition of a conditional component, and if the condition is false remove the component and any corresponding connections. The reason for this is that conditional components go together with the connection handling in Modelica. Additionally expressions (excluding arguments to connect) should be checked to ensure that they do not use any conditional component before lowering, since that check is not possible in Base Modelica.

Component and connect deselections introduced in Modelica 3.6 are handled similarly.

2.6 No evaluation of (Base Modelica) parameters

Constant and parameter variabilities are well separated in Base Modelica. Expressions that are deemed necessary to evaluate during Base Modelica translation are required to be constant, implying that lowering a Modelica model often involves evaluating parameters in order to comply with Base Modelica variability requirements. Further, Base Modelica semantics of constant components and expressions ensure that such expressions can be evaluated during translation when needed. In particular, a *pure constant* function concept is introduced to restrict the functions that can be used in constant expressions.

As an example of not having semantics relying on the ability to evaluate parameters during translation, a natural simplification compared to full Modelica is that a Base Modelica **`if`**-equation is required to have the same equation count in every branch. That is, whenever a Modelica **`if`**-equation has unbalanced branches, lowering of the equation must cause the **`if`**-equation conditions to be evaluated, so that branches breaking the balance can be eliminated.

2.7 Types are constant

Types created in Base Modelica can only hold constant properties. Since a constant property can always be evaluated in Base Modelica, this ensures that the internal representation of a type in a Base Modelica tool does not require the complexity of expressions and component references. This makes Base Modelica types much more similar to types found in other popular programming languages, compared to (full) Modelica types. It also significantly reduces the implementation effort for a pure Base Modelica tool compared to a full Modelica tool.

As an example of Base Modelica types being constant, each dimension of a Base Modelica array type has a size that is either *constant* or *flexible*, where the latter only indicates the absence of a constant expression for the size of an `Integer` dimension. Outside functions, component declarations may only specify constant array sizes. In an array equation, the array type must have constant sizes.

The `constsize`-expression allows expressing constant assertions on array dimensions. In the listing below, the OK assignment in `’h’` shows a way to assign the result of `’f’()` to `’z’`. The assignment below it is an error because `size(’z’)` has non-constant variability due to the flexible size of the first dimension.

Listing 4. Using the `constsize`-expression.

```
function ’f’
  output Real[:, ’MyEnumType’, :] ’y’;
  ...
end ’f’;

function ’h’
protected
  Real[:, ’MyEnumType’, 3] ’z’;
algorithm
  ’z’ := constsize(’f’(), :, size(’z’, 2),
      3); /* OK. */
  ’z’ := constsize(’f’(), size(’z’)); /*
      Error. */
end ’h’;
```

2.8 Variability-constrained types

Similar to Modelica, a record definition may have variability prefixes `parameter` or `constant` on the component declarations of the record members. In Base Modelica, such a type is denoted a *variability-constrained* type, and needs to obey additional rules that ensure more clarity in the semantics compared to Modelica. For example, a function component may not be of variability-constrained type, but is allowed to receive an argument of variability-constrained type. A model component of variability-constrained type is also allowed to be in solved position of an equation or assignment, in which case the variability-constrained members shall be disregarded as needed to match the type of the other side of the equation or assignment.

2.9 Short class definitions

Short class definitions in Base Modelica may not include array dimensions. Hence, all named types in Base Modelica are scalar types, and when a component has array dimensions, all array dimensions will be present at the component declaration.

It should be noted, however, that a record type may contain members of array type (where the sizes must be constant, as the component declarations are not inside a function):

```
record ’R’
  Real ’x’[3];
end ’R’;
```

2.10 Array subscripting of general expressions

While Modelica only allows array subscripts as part of the component reference syntax, Base Modelica allows applying array subscripts to general expressions. The only requirement is that the array subscripts are applied to a parenthesized expression, for instance, `(x.y)[1, 2]` . The ambition is to introduce the new syntax for Modelica, thereby avoiding the need to introduce new expression syntax in Base Modelica, while also making the generally useful feature available on both language levels.

2.11 More explicit initialization

Model initialization is very explicit in Base Modelica compared to Modelica. A notable difference compared to Modelica is that both `fixed` and `final` in Modelica are modeled using more elementary mechanisms in Base Modelica.

A parameter with a declaration equation in Base Modelica corresponds to a non-final fixed parameter in Modelica, and it is required that the declaration equation is solved with respect to the parameter so that it is possible to override the equation during initialization. For variables of higher variability, fixed initialization in Modelica is lowered to an explicit initialization equation, and non-fixed initialization is lowered to an explicit representation of guess-values. For example, consider the following Modelica parameter:

```
  final parameter Real p = 4.2;
```

In Base Modelica, this can be represented as:

```
  parameter Real 'p';
initial equation
  'p' = 4.2;
```

To make handling of guess values explicit, there is an implicitly declared *guess value parameter* `guess('x')` to represent the guess value for `'x'`. The guess value parameter may be defined by an initial equation in case it should not be possible to override during initialization, or using a Base Modelica *parameter equation*. A parameter equation is a special construct that is only allowed for guess value parameters, and in addition to expressing that the guess value may be overridden during initialization, it allows the guess value to be conveniently located next to the declaration of the variable to which it belongs (without leaving the variable declaration section of the model):

```
Real 'x';
parameter equation guess('x') = 1.5;
Real 'y';
parameter equation guess('y') = 2.5;
```

Guess value prioritization is made explicit in the form of a special kind of initial equation:

```
initial equation
  prioritize('x', 2);
```

Further, **`when`**-equations impose no constraints on the initialization problem. Lowering a Modelica **`when`**-equation may therefore result in explicit equations in the **`initial equation`** section of the Base Modelica model.

One notable thing which is not explicit in Base Modelica is the selection of which guess values should come into play, or how. This requires an analysis of the initialization problem equation structure that goes beyond what the lowering of Modelica is expected to deliver.

2.12 Records and function default arguments

Function input components are allowed to have declaration equations, but these are ignored. Hence, Base Modelica functions cannot have function default arguments. In the same spirit, declaration equations in record types do not define defaults of an implicit record constructor function (as they do in Modelica). Instead, the declaration equations (which can only have constant expressions by design) only define default modifications when the type is used in component declarations, and then get meaning depending on what modifications mean for different kinds of component declarations. For example, a modification on a model component declaration is equivalent to an equation in the model, whereas a modification on a function local or output component declaration equation is used to give initial values for the evaluation of the function body. A modification on a function input component is ignored similar to declaration equations, as argument values must always be passed for all function inputs.

2.13 Explicit clock partitioning

The implicit clock partitioning carried out by tools for full Modelica is made explicit in Base Modelica. The equations solved in a clocked sub-partition are placed in a dedicated `subpartition` construct, and the variables being determined by the sub-partition can be determined by a simple inspection of the equations. Listing 5 shows an example of a Base Modelica model with clock partitions.

Listing 5. Explicit representation of clock partitions.

```
package 'M'
model 'M'
  Real 'x';
  Real 'baseVar', 'cVar1', 'cVar2', 'cVar3'
     ;
  Real 'mixedVar1';

equation
  der('x') = 1;

partition
  Clock 'myClock' = Clock(1);
  Clock _subClock0 =
    subSample('myClock', 2);
  Clock _subClock1 =
    superSample(subSample('myClock', 2), 8)
      ;

  subpartition (clock = 'myClock')
  equation
    'baseVar' = sample('x');

  subpartition (clock = _subClock0,
      solverMethod = "ImplicitEuler")
  equation
    der('cVar1') = noClock('baseVar');

  subpartition (clock = _subClock1)
  equation
    'cVar2' = noClock('baseVar');
    'cVar3' = noClock('cVar1');
  algorithm
    'mixedVar1' := 'cVar2' + 'cVar3';

partition
  Clock _baseClock0 = Clock(1.1);
  ...
end 'M';
end 'M';
```

2.14 Source locations

The Base Modelica grammar allows code to be decorated with source location information to enable reporting errors pointing back to another source (typically, a Modelica model) from which the Base Modelica was produced. Each decoration consists of the @ sign followed by an integer that references some external, tool-specific, table of source location details. As illustrated by the listing below, decorations can be attached to expressions as well as many other constructs.

Listing 6. Source location decorations.

```
package 'Decorations'
@101 model 'Decorations'
  @202 Real x(@203 min = 0.0 @204);
equation
  @301 if x > 0.5 then
    @302 w = 1;
  else
    6 + w @304 = atan2(1 @305, 1);
  end if;
algorithm
  @306 w := 1 + (2 @303) @304;
end 'Decorations';
end 'Decorations';
```

3 Base Modelica open issues

The design of the proposed Base Modelica language is an ongoing effort documented and discussed on GitHub under the Modelica Change Proposal (MCP) 0031. This paper is presenting the results after reaching the first milestone of a design proposal version 0.1 including resolutions of all collected issues considered crucial for a first complete Base Modelica language. The design proposal has been specified as a modified Modelica grammar file along with a separate textual description of the semantic differences between Base Modelica against Modelica. This design proposal has been developed by representatives from four different Modelica tool vendors and is considered mature enough for being implemented and tested by Modelica tools.

Based on forthcoming test implementations it will be possible to reveal and collect issues needing further attention. Some potential issues are in need of gaining experience with test implementations to pinpoint the problems in order to decide if there really are any.

The following issues are already known:

- Reject or add support for non-constant nominal-attribute.
- Handling of `ModelicaServices`.
- Handling of external functions.
- Reuse of common components.

4 Conclusions & Outlook

The selection of designs proposed in section 2 illustrate how the complexity of the Modelica language can be significantly reduced by removing keywords from the grammar related to higher level constructs, enforcing implicit declarations to be expressed more explicitly, and being generally more restrictive. All this leads to a language that is still expressive enough to capture the semantics of a Modelica model being lowered to Base Modelica, but much more accessible for non-Modelica tool vendors and researchers seeking a standardized form of equation-based mathematical models. This will enable new parties to participate and enrich the Modelica ecosystem with new applications and methods producing or consuming Base Modelica.

It is clearly outlined how Base Modelica is derived by lowering Modelica. This indicates that Base Modelica is consistent with existing Modelica tools and applicable as a standardized intermediate format. Only limited development effort is expected to enhance existing Modelica front-ends to produce a Base Modelica output and Modelica back-ends to consume it. This will improve the abilities of tool vendors and library developers to understand and eliminate incompatibilities between tools to the benefit of the entire Modelica community.

This paper is aiming to lay the foundation for future discussions within the Modelica community and with the MAP-Lang in order to collect feedback to further improve this design proposal.

In future work, we are aiming to work closely together with tool vendors to develop prototypes to generate as well as consume Base Modelica representations. These prototypes will be an important proof of concept to reveal shortcomings of the current design and to give realistic estimates of the development efforts to be expected.

If this evaluation phase can be concluded with positive feedback a revised definition of the Base Modelica language shall then be defined. Based on this version a change proposal to refactor the Modelica Language Specification, considering Base Modelica as an integral part, shall be worked out and submitted to the MAP-Lang.

In the long run, we are aiming to convince Modelica and non-Modelica tool vendors to embrace Base Modelica as a widely used standardized language for equation-based models for many more tools and other standards to build upon.

Acknowledgements

This work has been funded in the context of the ITEA3 project 15016 EMPHYSIS by the Swedish Vinnova under grant number 2017-05121 and the German Federal

Ministry of Education and Research (BMBF) under the grant number FKZ 01|S17023. It has also been funded in the context of the German PHyMoS project by the Federal Ministry for Economic Affairs and Climate Action (BMWK) under grant number FKZ 19|20022A. The authors would also like to thank Kai Werther, Michael Tiller, Francesco Casella, and others for their inputs to the many discussions.

References

EMPHYSIS Consortium (2021). *EMPHYSIS – D7.9 eFMI for physics-based ECU controllers*. Tech. rep. ITEA3. URL: https://itea4.org/project/workpackage/document/download/7675/D7.9_Public_DemonstratorSummary.pdf.

Lattner, Chris and Vikram Adve (2004-03). "LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation". In: *CGO*. San Jose, CA, USA, pp. 75–88. DOI: 10.1109/CGO.2004.1281665.

Lenord, Oliver et al. (2021). "eFMI: An open standard for physical models in embedded software". In: *Proceedings of the 14th International Modelica Conference 2021*. Linköping, Sweden. DOI: 10.3384/ecp2118157.

Lindholm, Tim et al. (2023-03). *The Java Virtual Machine Specification, Java SE 20 Edition*. Tech. rep. ORACLE. URL: https://docs.oracle.com/javase/specs/jvms/se20/html/index.html.

Modelica Association (2023-03). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.6*. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.6/MLS.html.

The Object Management Group (2023). *OMG System Modeling Language version 2.0 Beta 1*. URL: https://www.omg.org/spec/SysML/2.0/Beta1.

A preCICE-FMI Runner to Couple FMUs to PDE-Based Simulations

Leonard Willeke[1] David Schneider[1] Benjamin Uekermann[1]

[1]Institute for Parallel and Distributed Systems IPVS, University of Stuttgart, Germany,
benjamin.uekermann@ipvs.uni-stuttgart.de

Abstract

Partitioned simulation or co-simulation allows simulating complex systems by breaking them up into smaller subsystems. The Functional Mock-Up Interface (FMI) enables co-simulation for models based on ODEs and DAEs, but typically not PDEs. However, only PDE-based models are able to accurately simulate physical aspects requiring spatial resolution, such as heat transfer or fluid-structure interaction.

We present a preCICE-FMI runner software to integrate FMUs with the open-source coupling library preCICE. preCICE couples PDE-based simulation programs, such as OpenFOAM or FEniCS, in a black-box fashion to achieve partitioned multi-physics simulations. The runner serves as an importer to execute any FMU and to steer the simulation. Additionally, it calls preCICE to communicate and coordinate with other programs. The software is written in Python and relies on the Python package FMPy. We showcase two example cases for the coupling of FMUs to ODE- and PDE-based models.

Keywords: Functional Mock-Up Interface (FMI), multiphysics, preCICE, coupling, co-simulation, FMPy, OpenFOAM

1 Introduction

The simulation of complex, dynamic systems is an important task in science and engineering. It includes multiphysics simulations and the simulation of cyber-physical systems. There are two paths to achieve such simulations, the monolithic and the partitioned approach. In the monolithic setup, one software includes all the necessary computations to model the different phenomena. Contrary to that, the partitioned approach relies on multiple independent pieces of software. Each of these programs covers a specific aspect of the simulation. The programs are then coupled or co-simulated to achieve the correct outcome. This approach allows splitting complicated systems in smaller, simpler subsystems and re-using them in different scenarios. Examples include climate modeling (Gross et al. 2018), but also engineering applications such as the modeling of wind turbines (Sprague, J. M. Jonkman, and B. J. Jonkman 2015).

The *Functional Mock-Up Interface FMI* (Blochwitz et al. 2011) follows a so-called *framework approach* for co-simulation. The simulation models are implemented as standardized Functional Mock-Up Units (FMUs). FMUs are zip-archives with a pre-defined structure and content. They contain the simulation model as library (*.dll, *.so) and meta data about the model such as documentation or reference results. The coupling is done by an additional program, a so-called importer (see Figure 1). The importer loads and executes the FMUs and implements a co-simulation algorithm to ensure communication and data exchange between the models. This approach works well for simple models composed of ODEs and DAEs, but reaches its limits for more complex PDE models. The computation of PDEs often requires legacy software packages and high-performance computing, which are in general not compatible with the regulations for FMUs.

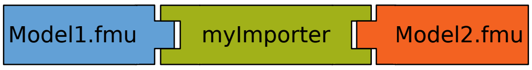


Figure 1. Co-simulation approach of the FMI standard (a framework approach): The standardized FMU models are called and coupled by an importer program. We denote the caller (executable) with a female connector and the callee (library) with a male connector.

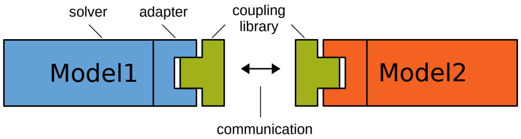


Figure 2. Co-simulation approach of preCICE (a library approach): The simulation programs call preCICE as a library to perform the coupling.

preCICE (Chourdakis et al. 2022), on the other hand, is an open-source coupling library for partitioned multiphysics simulations. It couples PDE-based simulation programs, such as OpenFOAM or FEniCS, in a black-box fashion. preCICE follows a *library approach* for co-simulation: The simulation programs call the coupling (see Figure 2). The programs itself ideally remain untouched and connect to preCICE through an additional software layer called *adapter*. Each simulation program requires a specific adapter. Ready-to-use adapters for many popular simulation programs exist. This setup ensures the re-usability of all components and a decent time-to-solution for new applications.

The main idea of this work is to combine both worlds: to couple FMU models to other simulation programs via preCICE. To this end, a new software component, the preCICE-FMI runner is developed (see Figure 3). It acts as an importer towards the FMU, loading and executing the model. Additionally, the runner calls preCICE to couple the simulation to other programs and steer the simulation. It allows coupling FMU models to any simulation program in the preCICE ecosystem.

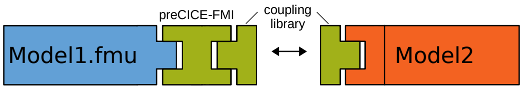


Figure 3. Concept of the preCICE-FMI runner: The new software executes the FMU and calls preCICE to couple the simulation to other programs.

In order to understand how preCICE and the FMI standard can be coupled, Chapter 2 introduces the two existing software components. The chapter continues to explain the concept of the newly developed software, the preCICE-FMI runner, its configuration and functionalities. To showcase the software, Chapter 3 then describes two example cases. First, a partitioned mass-spring oscillator system is used to test the runner against an analytical solution and an alternative Python-based implementation. Second, a FMU model is coupled to a fluid-structure interaction scenario using OpenFOAM as fluid solver and a simple Python script as solid solver. For this scenario, we compare against results from the literature. The final Chapter 4 summarizes the presented ideas and reflects on the impact of the developed software. This paper summarizes and extends the master's thesis of Willeke (2023).

The idea of coupling FMUs to PDE-based programs has already attracted further attention. A recent tool to execute this task is *FMU4FOAM*[1]. It couples FMUs exclusively to the CFD program OpenFOAM. The coupling is realized with a function object, a library that is loaded by OpenFOAM at runtime. The function object embeds a Python interpreter to handle the FMU. FMU4FOAM is configurable on runtime and can be further adapted to specific needs due to its object-oriented structure.

We see the main advantage of the preCICE-FMI runner in its ability to leverage the advanced coupling functionalities of preCICE. The coupling of FMUs is not limited to OpenFOAM, but can be performed with any program in the preCICE ecosystem. Different coupling schemes and topologies are possible. Furthermore, the coupling library implements a toolbox of coupling algorithms to ensure robust and accurate simulations.

2 Software description

Before detailing the concept, functionalities, and limitations of the new runner software, we start this section with describing the existing software components: preCICE and FMI.

2.1 Existing software components

preCICE (Chourdakis et al. 2022) is a widely-used and open-source coupling library for multi-physics simulations. In preCICE terminology, the coupled simulation programs are called *solvers* or *participants*. preCICE implements data mapping and communication between the solvers. The user can choose between explicit and implicit as well as serial and parallel coupling schemes (Gatzhammer 2014). The coupling is not limited to two solvers, but can be easily extended to perform multi-coupling (Bungartz et al. 2015). Acceleration algorithms, such as quasi-Newton methods (Uekermann 2016) are available to stabilize and speed up implicit coupling schemes. Finally, time interpolation allows individual solvers to use different time step sizes. The exact numerical implementation is out of scope here, but can be found in the cited literature. All these coupling configurations are defined in the *precice-config.xml* file, a global file accessed by all participants.

Each solver is connected to preCICE with a specific *adapter*. The adapter allows the solver to access the preCICE API and call the coupling. It can take many forms, depending on the solver itself. For example, the preCICE-OpenFOAM adapter is an OpenFOAM function object, an indepent library, while the preCICE-FEniCS adapter is a Python module. Adapter development is facilitated by the language bindings of preCICE, which are available for C/C++, Python, Fortran, Julia, and Matlab. A more extensive introduction to preCICE is given in Chourdakis et al. (2022).

The *FMI standard* (Blochwitz et al. 2011) defines different types of FMU models. They share a similar interface, but implement different functionalities. *Model Exchange* FMUs hold the model equations and present them to an external solver algorithm. *Co-Simulation* FMUs hold the model equations and the solver algorithms. As such, they can compute the next time step on their own. *Scheduled Execution* FMUs were introduced with FMI v3 (Junghanns et al. 2021) and hold different model partitions, which are accessed by an external scheduler. We only consider the coupling of Co-Simulation FMUs in this work.

The Python library FMPy[2] can be used to load, execute and steer FMU models with Python. We choose to use FMPy over other libraries such as *PyFMI* (Andersson, Akesson, and Führer 2016) and *fmipp* due to its easy installation and usability. An overview of many further FMI tools can be found on the FMI website[3].

2.2 Concept of the preCICE-FMI runner

The *preCICE-FMI runner* is a new piece of software, which connects FMI and preCICE. It uses FMPy to load

[1] `https://dlr-ry.github.io/FMU4FOAM/`

[2] `https://fmpy.readthedocs.io/en/latest/`

[3] `https://fmi-standard.org/tools/`

and execute any FMU and preCICE to couple the simulation. The concept is explained in the simplified code in Figure 4. First, the runner script imports and initializes both FMPy and preCICE (lines 1-9). This includes loading the FMU, setting initial simulation parameters and preparing the communication to other participants. The main loop (lines 11-29) executes the coupled simulation. The program reads data from preCICE (line 16) and writes it to the FMU model (line 18). The model can now compute the next time step (line 20). Afterwards, the program reads the new results (line 22) and writes them to preCICE (line 24). Finally, preCICE advances the simulation as a whole and communicates data with other participant (line 26). Here, preCICE also needs to know how much the FMU model advanced in time to synchronize all participants. This is the main simulation mechanism used for both explicit and implicit coupling. Implicit coupling, however, requires one more feature: the option to repeat a time step. The solver state can be stored (line 14) and reset (line 29) to iterate over a time step. preCICE indicates whether this is necessary or not (lines 12, 27). Finally, preCICE and FMPy are terminated (lines 31-32) to close the communication channels and release the FMU. The software is developed on GitHub[4] and documented in the preCICE user documentation[5].

```
1  import fmpy
2  import precice
3  # FMU Setup
4  fmu = fmpy.fmi3.FMU3Slave(...)
5  fmu.instantiate()
6  # preCICE Setup
7  interface = precice.Interface(...)
8  ...
9  dt = interface.initialize()
10 # main time loop
11 while interface.coupling_ongoing():
12   if interface.action_required(...):
13     # Save state (implicit coupling)
14     state_cp = fmu.getFMUstate()
15   # Get read data from preCICE
16   interface.read_vector_data(...)
17   # Set read_data in FMU
18   fmu.setFloat64(...)
19   # Compute next time step
20   fmu.doStep(t,dt)
21   # Get write_data from FMU
22   write_data = fmu.getFloat64(...)
23   # Send write_data to preCICE
24   interface.write_vector_data(...)
25   # Advance preCICE in time
26   dt = interface.advance(dt)
27   if interface.action_required(...):
28     # Load state (implicit coupling)
29     fmu.setFMUstate(state_cp)
30
31 interface.finalize()
32 fmu.terminate()
```

Figure 4. Concept of the FMI runner: The script utilizes the library FMPy to execute the FMU and calls the preCICE API to couple the simulation. For conciseness, API calls are simplified.

2.3 Functionalities and limitations of the preCICE-FMI runner

The ideal coupling tool should support the full functionality of both preCICE and FMPy. It should work with any co-simulation FMU and should be configurable at runtime. The software has to be well-documented and include tests to guardrail further development. This section gives insight into the abilities, limitations, and configuration of v0.1 of the preCICE-FMI runner.

The software is available for preCICE v2 and compatible with FMI 1, 2, and 3. It supports explicit and implicit coupling schemes, including acceleration. It is configurable at runtime via two .json files and can be adapted to different FMU models. Time-dependent input signals can be set, output signals are stored as timeseries. The tool is easy to install and comes with a regression test.

A major difference in the coupling of two PDE solvers compared to the coupling of a PDE solver to an FMU is the role of the mesh. For PDE-PDE coupling, the exchanged data is spatial for both solvers and mapped accordingly. However, many FMUs can not deal with spatially resolved data but are designed to receive signal data. To enable the coupling of PDE solvers to such FMUs, we use the mapping capabilities of preCICE: spatial data from one mesh is mapped to a single vertex on another mesh to create signal data and vice versa.

As a result, data exchange is limited to one vertex. Moreover, the exchange of multiple data points and gradient information is not yet supported. Also, no internal errors of the FMU model are logged. Finally, the runner software can currently only be executed in serial.

To explain the configuration of the runner, we assume a FMU model `Suspension.fmu`, which contains the model equations for a spring-damper system. It should be coupled via preCICE to receive the variable `force` from another participant, calculate the displacement internally, and send the variable `position` back. The configuration file *fmi-settings.json*, shown in Figure 5, holds the settings for FMPy. The dictionary `simulation_params` (lines 2-9) is used to choose the FMU model and set the read and write variables. A CSV file can be set to store results. The dictionaries `model_params` and `initial_conditions` (lines 10-16) allow setting model parameters before the simulation start. Time-dependent input signals are set in `input_signals` (line 17-23). In this case, the variable `damping_coeff` is initialized with value 0.0 and

[4] https://github.com/precice/fmi-runner

[5] https://precice.org/tooling-fmi-runner.html

increases to value 5.0 at $t = 2.0$.

To execute the coupling, the runner needs some information on how to interact with preCICE. The file *precice-settings.json* (Figure 6) points the runner to the global configuration file of preCICE shared with all participants. The following entries define which participant the runner is, which mesh it owns, and which preCICE variables are accessed.

```
{
"simulation_params": {
  "fmu_file": "../Suspension.fmu",
  "fmu_read_data": ["force"],
  "fmu_write_data":["position"],
  "fmu_instance": "suspension_1",
  "output_file": "./output.csv",
  "output": ["position"]
  },
"model_params": {
  "apply_filter":         true,
  "spring_coeff":         65.0
},
"initial_conditions": {
  ...
},
"input_signals": {
  "names":["time", "damping_coef"],
  "data": [
          [0.0, 0.0],
          [2.0, 5.0]
  ]
  }
}
```

Figure 5. Example for fmi-settings.json

```
{
"coupling_params": {
  "config_file": "../config.xml",
  "participant": "Suspension",
  "mesh_name": "Suspension-Mesh",
  "read_data": {"name": "Force"},
  "write_data": {"name": "Position"}
  }
}
```

Figure 6. Example for precice-settings.json

3 Example cases

Two example cases show the functionality of the preCICE-FMI runner. First, the simulation of a partitioned mass-spring oscillator system demonstrates the coupling of two ODE systems. The obtained results are compared to a numerical simulation in Python and an analytical solution. Second, the fluid-structure interaction of a moving cylinder is adapted to include a FMU model for controlling the movement and compared to reference results from literature. The case files are available for reproduction.[6]

3.1 Partitioned mass-spring oscillator system

We assume an ideal mass-spring system (Schüller et al. 2022) as shown in Figure 8. Three springs k_1, k_{12}, and k_2 connect two masses m_1 and m_2 with each other and two fixed walls. The variables u_1 and u_2 denote the positions of the masses. To create a partitioned system, spring k_{12} is cut in the middle. The resulting subsystems exchange interface forces F_1 and F_2. The system is adaptable to perform different kinds of oscillations. We follow Schüller et al. (2022) and set the spring stiffness to $k_1 = k_2 = 4\pi^2 N/m$ and $k_{12} = 16\pi^2 N/m$, while the two masses are set to $m_1 = m_2 = 1kg$. Additionally, the initial conditions are chosen as

$$\begin{aligned} u_1(0) &= 1 \quad & \dot{u}_1(0) &= 0 \\ u_2(0) &= 0 \quad & \dot{u}_2(0) &= 0 \end{aligned}$$

This setup has the analytical solution depicted in Figure 7. The two masses oscillate with a period of $T_P = 1s$.

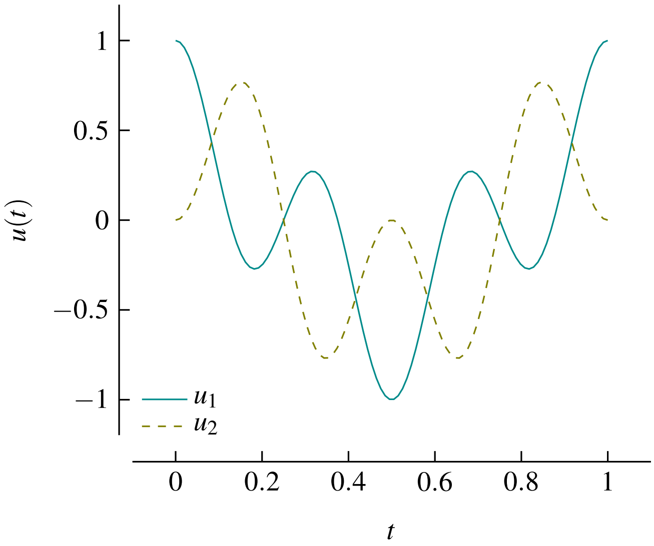


Figure 7. Analytical solution of the mass-spring system (Schüller et al. 2022). The plot shows the displacements u_1 and u_2 for the initial conditions $u_1(0) = 1, \dot{u}_1 = 0$ and $u_2(0) = 0, \dot{u}_2 = 0$.

We use this testcase to compare an FMU-based implementation executed with the preCICE-FMI runner and an existing implementation in Python (Schüller et al. 2022). Both implementations can be coupled to themselves or to one another. We end up with four possible combinations: runner-runner, Python-Python, runner-Python, and Python-runner. Moreover, we compare against the analytical solution.

[6] `https://doi.org/10.18419/darus-3549`

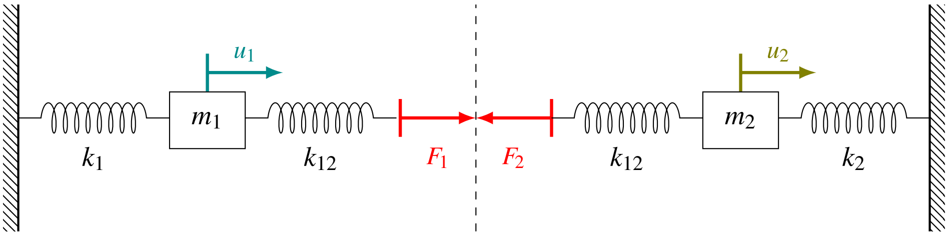


Figure 8. Mass-spring system: The springs k_1, k_{12} and k_2 connect the two masses m_1 and m_2 which have the positions u_1 and u_2. The middle spring k_{12} is cut in half to create a partitioned setup with two subsystems that exchange interface forces F_1 and F_2.

Both implementations use the Newmark-β method (Newmark 1959) with $\beta = \frac{1}{4}$ and $\gamma = \frac{1}{2}$ for time integration. A serial-implicit coupling with Aitken acceleration is used. The simulation time is set to $T = 5s$ with a time step of $\Delta t = 0.005s$. A more extended description is given in Willeke (2023).

The results in Figure 9 show the trajectory of m_1. No differences are visible between all four possible combinations. Furthermore, all combinations match the analytical solution well. For a more accurate comparison, the maximum norm $\|e\|_\infty$ between analytical solution and numeric result for u_1 is calculated over all time steps. The maximum serves as a worst-case approximation. The calculated error $\|e(u_1)\|_\infty \approx 3.48 \times 10^{-2}$ is identical for the FMU and the Python implementation up to a precision of 10^{-5}.

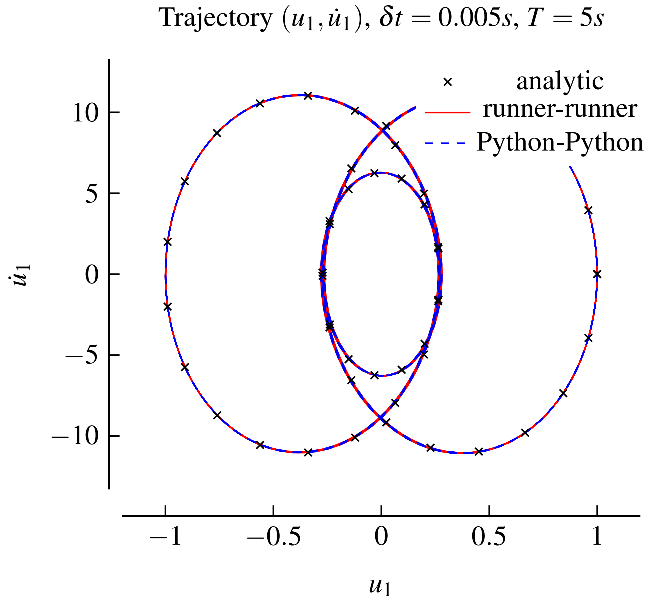


Figure 9. Comparison of different implementations for the mass-spring oscillator system. Velocity $\dot{u}_1$ is plotted over position u_1 to show the trajectory of m_1. The results from a coupling of both the runner and the Python solver to itself have no visible differences and track the analytical solution well. The results for the cross-combinations runner-Python and Python-runner show no visual difference and are omitted for simplicity.

3.2 Flow around a moving cylinder

As a second example, consider the flow in a channel around a cylinder as shown in Figure 11. The cylinder with diameter D and mass m is not fixed in its position y. Instead, the cylinder is mounted upon a spring-damper system with spring stiffness k and damper coefficient d. The flow with velocity v_0 induces vortex shedding behind the cylinder. This leads to varying lift forces and results in an oscillation of the cylinder position y. This setup has been used as a test case for numerical simulations (Placzek, Sigrist, and Hamdouni 2009) and is especially interesting because experimental reference data is available (Anagnostopoulus and Bearman 1992). The experimentalists report lock-in effects for the cylinder oscillation for Reynolds numbers between $104 < Re < 126$ with the highest excitation at the lower end. For our simulation, we chose a Reynolds number of $Re = 108.83$. The remaining system parameters are set to $m = 0.03575kg$, $d = 0.0043N/s$ and $k = 69.48N/m$. All are in accordance with the referenced literature. We further adapt the case to be able to move the root point of the spring u (Sicklinger 2014). Now, the spring force acting on the cylinder can be actively controlled by adjusting u.

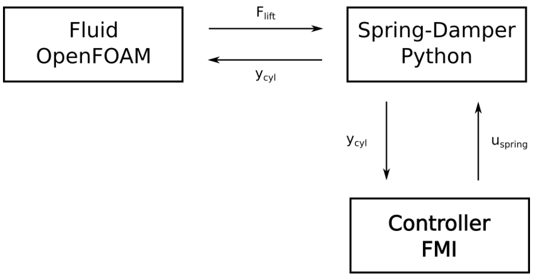


Figure 10. Coupling topology for the flow around a moving cylinder example: Three participants are coupled in two bi-coupling schemes. The fluid participant and the spring-damper participant exchange lift force F and cylinder displacement y. The spring-damper participant and the controller exchange displacement y and spring root displacement u.

The goal of this setup is to couple a controller FMU to a PDE-based simulation. The FMU holds the equations of a Proportional-Integrative-Derivative (PID) controller (Ang, Chong, and Li 2005). It reads the displacement of the cylinder y and tries to minimize it by setting the root point of the spring u.

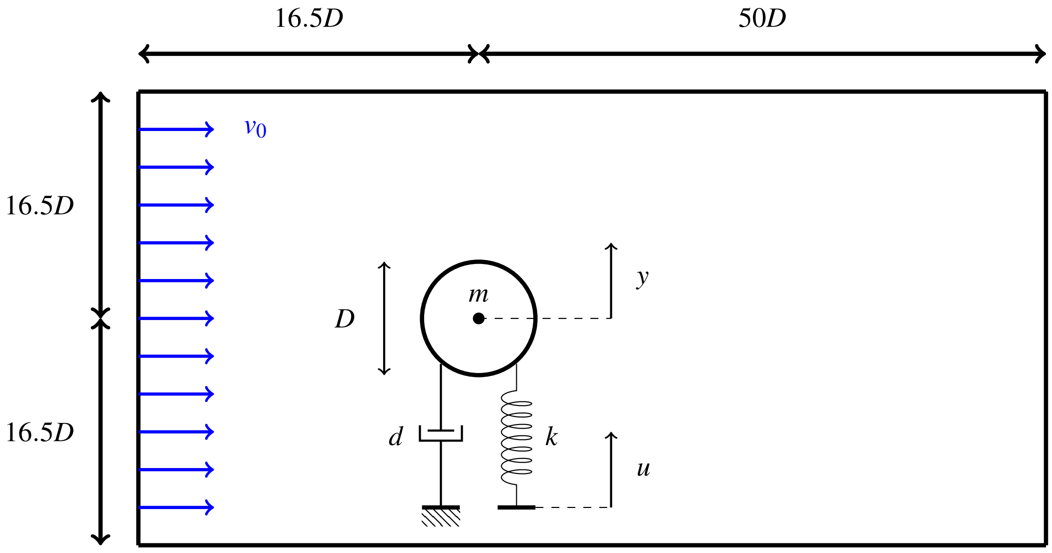


Figure 11. Case setup for flow around a moving cylinder: The object is mounted upon a spring-damper system, which allows it to move in y-direction. The root point of the spring u can be moved to vary the force acting on the cylinder.

The spring-damper system itself is calculated in a separate Python program, while the fluid flow is computed with the CFD program OpenFOAM. The resulting coupling topology is shown in Figure 10. Two explicit bi-coupling schemes are combined. A fitting mapping configuration ensures the transition from the spatial domain in OpenFOAM to the signal domain in the FMU.

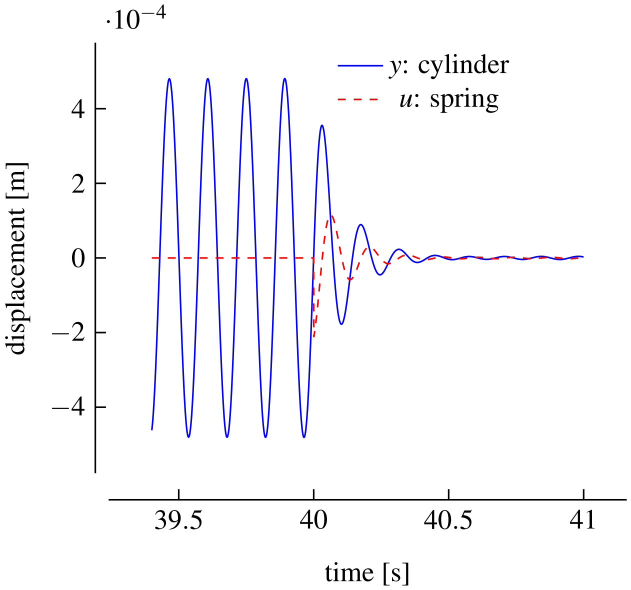


Figure 12. Cylinder displacement with activation of the PID controller: The cylinder oscillates in a stable state until the controller is activated at $T = 40s$. This reduces the displacement y by orders of magnitude.

Figure 12 shows the simulation results obtained with a time step of $\Delta t = 0.0001s$. First, the controller is deactivated until the cylinder has reached a state of stable oscillation. The displacement has an amplitude of $\hat{y} = 0.48mm$, which is close to the reference amplitude of $\hat{y}_{ref} = 0.6mm$. The differences may be attributed to the explicit solver in the spring-damper system and the explicit coupling scheme, both of which are implicit in the reference simulation. The PID controller is activated at $T = 40s$. The control gains are set to $K_P = 0.02$, $K_I = 0.02$ and $K_D = 0.01$ to ensure a robust transient behaviour (Sicklinger 2014). The cylinder displacement is reduced by orders of magnitude with a fast transition phase.

4 Conclusions

We presented the preCICE-FMI runner, a new software to couple FMU models to PDEs. The software loads and executes FMU models and calls preCICE to execute the coupling. It is written in Python to leverage the Python package FMPy and the preCICE Python bindings. The runner software is configured with two settings files, enabling different simulation scenarios. An easy, standard installation process lowers the entry barrier for new users. The preCICE-FMI runner is compatible with co-simulation FMUs of FMI versions 1, 2, and 3 and preCICE version 2. It supports explicit and implicit coupling via preCICE, as well as the use of acceleration methods. Time-dependent input signals can be set for the FMU model during simulation. Output signals from the FMU are stored for post-processing.

Two example cases introduced the functionalities of the runner software. The partitioned simulation of a mass-spring oscillator system showed good agreement with another numerical solver and an analytical solution. The simulation of the flow around a moving cylinder was coupled to a FMU-based control algorithm and showed qualitatively meaningful results.

The new software is focused on providing a general coupling of co-simulation FMUs to PDE-based solvers. It enables plug-and-play coupling to many popular solvers

DOI
10.3384/ecp204479

thanks to the preCICE library approach. The software is documented on *precice.org* and the presented test cases are available for reproduction. Some limitations, such as the lack of error logging or full mesh exchange remain.

FMI is a widely used industry standard for the co-simulation of cyber-physical systems. preCICE, on the other hand, has a growing user base in academia and industry focused on high-fidelity multi-physics applications. The preCICE-FMI runner connects these two communities. With our work, we hope to spark a discussion about the specific needs of both communities to guide further developments.

Acknowledgements

Many thanks to Mosayeb Shams from Heriot-Watt University, UK, who shared his expertise on OpenFOAM with us. Great thanks to Ishaan Deshai, University of Stuttgart, for his manifold support throughout the project.
Funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2075 – 390740016. We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech).

References

Anagnostopoulus, P. and P.W. Bearman (1992). "Response Characteristics of a vortex-excited cylinder at low Reynolds numbers". In: *Journal of Fluids and Structures* 6.1, pp. 39–50. DOI: 10.1016/0889-9746(92)90054-7.

Andersson, Christian, Johan Akesson, and Claus Führer (2016). "PyFMI: A Python Package for Simulation of Coupled Dynamic Models with the Functional Mock-up Interface". In: Technical Report in Mathematical Sciences LUTFNA-5008-2016.2.

Ang, Kiam Heong, G. Chong, and Yun Li (2005). "PID control system analysis, design, and technology". In: *IEEE Transactions on Control Systems Technology* 13.4, pp. 559–563. DOI: 10.1109/TCST.2005.847331.

Blochwitz, T. et al. (2011). "The functional mockup interface for tool independent exchange of simulation models". In: Technical University; Dresden; Germany.

Bungartz, Hans-Joachim et al. (2015). "A plug-and-play coupling approach for parallel multi-field simulations". In: *Computational Mechanics* 55, pp. 1119–1129. DOI: 10.1007/s00466-014-1113-2.

Chourdakis, G et al. (2022). "preCICE v2: A sustainable and user-friendly coupling library [version 2; peer review: 2 approved]". In: *Open Research Europe* 2.51. DOI: 10.12688/openreseurope.14445.2.

Gatzhammer, B. (2014). "Efficient and Flexible Partitioned Simulation of Fluid-Structure Interactions". PhD thesis. Technical University of Munich, pp. 1–183.

Gross, Markus et al. (2018). "Physics-dynamics coupling in weather, climate, and Earth system models: Challenges and recent progress". English (US). In: *Monthly Weather Review* 146.11, pp. 3505–3544. DOI: 10.1175/MWR-D-17-0345.1.

Junghanns, A. et al. (2021). "The Functional Mock-up Interface 3.0 - New Features Enabling New Applications". In: DOI: 10.3384/ecp2118117.

Newmark, N. M. (1959). "A method of computation for structural dynamics". In: *J. Eng. Mech. Div.-ASCE* 85.3, pp. 67–94. DOI: 10.1061/JMCEA3.0000098.

Placzek, A., J.F. Sigrist, and A. Hamdouni (2009). "Numerical Simulation of an oscillating cylinder in a cross-flow at low Reynolds number: Forced and free oscillations". In: *Computers and Fluids* 38.1, pp. 80–100. DOI: 10.1016/j.compfluid.2008.01.007.

Schüller, Valentina et al. (2022-07). "A Simple Test Case for Convergence Order in Time and Energy Conservation of Black-Box Coupling Schemes". In: DOI: 10.23967/wccm-apcom.2022.038.

Sicklinger, S.A. (2014). "Stabilized Co-Simulation of Coupled Problems including Fields and Signals". PhD thesis. Technical University of Munich, pp. 126–135. DOI: 10.13140/2.1.1103.7762.

Sprague, M. A., J. M. Jonkman, and B. J. Jonkman (2015). "FAST Modular Framework for Wind Turbine Simulation: New Algorithms and Numerical Examples". In: *SciTech 2015: 33rd Wind Energy Symposium*.

Uekermann, B. (2016). "Partitioned Fluid-Structure Interaction on Massively Parallel Systems". PhD thesis. Technical University of Munich, pp. 62–70. DOI: 10.14459/2016md1320661.

Willeke, Leonard (2023). *A preCICE-FMI Runner to Couple Controller Models to PDEs*. Master thesis. DOI: 10.18419/opus-13130.

Secure Exchange of Black-Box Simulation Models using Functional Mockup Interface in the Industrial Context

Christian Wolf[1] Miriam Schleipen[1] Georg Frey [2,3]

[1]EKS InTec GmbH, Germany, {christian.wolf,miriam.schleipen}@eks-intec.de
[2]Chair of Automation and Energy Systems, Saarland University, Germany, georg.frey@aut.uni-saarland.de
[3]Department of Industrial Security, Center for Mechatronics and Automation Technologies (ZeMA), Germany, georg.frey@zema.de

Abstract

Functional Mockup Interface (FMI) is a standard for exchanging simulation models described as Funcional Mockup Units (FMUs) in a platform-agnostic way. FMUs can be implemented as white-box or black-box models. In the industrial context, it is common to exchange black-box models between partners to hide intellectual property. Using and running such models, though, is a security issue as there is no way to verify and validate the content of the models. This security issue must be addressed especially in the industrial context where security is considered high priority in general. Based on an exemplary model exchange, possible attacks and possible countermeasures are analyzed in this work. By using cryptography, three different approaches to pack the additional metadata are presented that aim at providing end-to-end integrity checks to a black-box simulation model. Together with administrative measures, this allows to define those FMUs to be trusted and executed. For the sake of completeness, a prototype was implemented to help with the cryptographic processes and show the effectiveness of the provided solution.

Keywords: Simulation, Security, FMI/FMU, AutomationML, Black-Box Model, Certificate, PKI

1 Introduction

In the current design and development process of products and goods, simulation provides a way to verify functionality and optimize in an early stage of development. For the sake of reducing the involved manpower to build such simulation models, these models are typically shared between stakeholders. Current developments like the FMI standard show the need to provide a standardized way to simplify model exchange. However, not all models can and should be provided as a white-box model, e.g. there might be restrictions on who should be able to read and understand the model in depth.

Current practice is that FMUs are transmitted via email or download from the vendor website. Although there might be SSL-based security added on the point-to-point transports, like HTTPS, these processes must be considered insecure as there is no end-to-end guarantee of the security. The FMU stored on the web server could be changed intentionally or by accident without notice.

One way to tackle the problem would be to use secure data spaces where the platform itself is designed for security, see Christoph Schlueter Langdon and Karsten Schweichhart (2022). On such a platform, access is coupled with the trust in the corresponding entity, so within this data space everyone can be trusted. One project to create a safe space is Catena-X (Hedda Massoth 2022).

By closing down the models in form of black-box simulations, it is (ideally) impossible to check the model for correctness, feasibility, and security. There might be arbitrary code included in the model that will get executed once the simulation is run. There are approaches to reduce the risk by e.g. running the model in an secured, immutable environment ("sandboxing" the model) but this is prone to bugs and should only serve as a last resort. Instead, this paper presents a method that allows to check statically the validity of a simulation model before even running it.

In this work, some basic tools and concepts from various domains are presented in Section 2. The underlying problem of this paper is presented in Section 3 and some risk analysis on possible attacks in the state of the art carried out in Section 4. To tackle the issues, some abstract considerations in Section 5 as well as some rejected considerations in Section 6 are used to derive multiple different propositions in Section 7. Finally, a proof-of-concept prototype is presented in Section 8 and an outlooks as well as a summary of the results is given.

2 Preliminaries

Throughout this paper, concepts from different domains need to be combined. In this section, the basics of different fields are covered for later connection in Sections 5 through 7. By integrating cryptogrphical methods with simulation models/FMUs, new benefits can be generated.

2.1 Functional Mockup Units

Functional Mockup Interface (FMI) is a standardized way to exchange simulation models for various purposes. The current version 3 is described in *FMI V3* (2023). A single model is called a Functional Mockup Unit (FMU).

DOI 10.3384/ecp204487

An FMU is technically speaking a compressed folder with some files inside. There is for one the manifest, an XML file describing the model. In the manifest the inputs and outputs are specified, as well as parameters and other options.

The actual implementation of the model logic can be provided by means of embedded source code or a precompiled library (DLL or shared object depending on the operating system). It is also possible to have both or a mixture of these approaches.

There are in fact three types of FMUs available: model exchange, co-simulation, and scheduled execution. Each type has its dedicated use case. While the co-simulation type brings its own mathematical kernel and is run as a dedicated process during execution, the other two types run within the process context of the host's simulation environment.

There exists the *FMU Trust Center* presented by Johannes Mezger et al. (2011). This allows to encapsulate the FMU under consideration into its own trusted environment. By using a dedicated and secured infrastructure, the FMUs can be encrypted during transport. Only during the time of simulation, a decrypted version is existing. That way, one can provide encrypted FMUs without the risk of sharing internal knowledge.

2.2 Digital Twin and Meta Models

In typical industrial applications, there is the need for a common description of arbitrary data in a structured way. This allows to exchange data in generic ways and is commonly referenced as a *digital twin* (DT).

Currently, two major meta models to describe such digital twins are established: AutomationML (AML, Rainer Drath (2021)) and the Asset Adminstration Shell (AAS, Deutsches Institut für Normung (2019) and *IEC 63278-1* (2022)). Both models have a similar goal and are interchangeable in terms of this paper. Thus, throughout this publication only AML is going to be used as an example. Similar results can be obtained with AAS with slight modifications.

Coming from the DT context, these meta models provide a method to describe things in a portable way. By defining common file formats the problem is reduced to understanding the semantic of the various parts. Instead of defining their own vocabulary, these models rely on existing descriptions. For example, simulation models can be integrated as FMUs.

AML uses XML files and all linked files just are connected using so-called *external interfaces*. There is however the option to embed all relevant files into a bundle called an *AutomationML container* (Rainer Drath 2021). Such an AMLX file is a zipped folder with all attachments plus additional files to satisfy formal requirements. According to Rainer Drath, Markus Rentschler, and Michael Hoffmeister (2019), both AMLX and AASX (from AAS) files comply with the Open Packaging Conventions (OPC) in accordance to *IEC 29500* (2012).

2.3 Security

For computing systems, there exist various requirements that a typical user assumes but that need careful planning and engineering. In Avizienis et al. (2004) these are categorized into dependable and secure aspects.

Dependable in this context means that a system is behaving as expected and not causing any major risks during operation. This is mostly the same definition that we would call a physical product *functional*: a freezer must keep the goods frozen even during e.g. an outage of a half an hour while not emitting any toxic gases to the environment.

There however is the concept of security that mainly became common with digitalization. Security breaks down to three main aspects: confidentiality (Is my data safe and nobody can access it?), integrity (Can I trust my data or has anyone tampered with it?), and availability (Can I access my data anytime I want?). Typically, not all of these goals can be achieved 100 % at the same time.

Some methods to establish these aspects of security are hashing, asymmetric cryptography, and certificates. The following sections will give a brief introduction for the reader.

2.3.1 Hashing

A hash function is a cryptographic algorithm that can be applied to data and files and that results in a deterministic value, aka there must be no randomization involved (Bart Preneel 1994). There are different hashing functions commonly used also with different goals and motivations involved like SHA-1, SHA-256, BLAKE-256, etc (Jean-Philippe Aumasson et al. 2008; Alex Biryukov, Dumitru-Daniel Dinu, and Dmitry Khovratovich 2015; Rajeev Sobti and G Geetha 2012; Stefan Tillich et al. 2009).

The output of such a hasing function is called a hash and of fixed size. For example, the SHA-256 hash function uses 256 bit to represent a hash. This is true for arbitrary long input data sequences, so the function cannot be injective: multiple input streams can result in the same hash.

This property of non-injectiveness has multiple effects. First, hashing cannot be undone. Once a data steam has been hashed it is no longer possible to conclude the original data. Hashing is thus a one-way function.

In order to compare the content of files/data streams, comparing the hashes is one typical method. One has to keep in mind that comparing function values and deduce the equality of the function arguments only works for injectve function by definition. For non-injective functions, like the hashing functions, there is a (small) probability that two distinct data streams result in the same hash, called a *hash collision*. Therefore, hash functions are constructed such that small changes in the input data (even as small as a single bit) lead to significant changes in the resulting hash value. This should reduce the probability of small defects to get unnoticed.

DOI
10.3384/ecp204487

One typical use case for hashing is to establish integrity of data. Unless there is a collision, one can identify and reject a compromised data stream if the hash is known.

2.3.2 Asymmetric Cryptography and Keys

The RSA algorithm named after Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman (1978) is a well-known method for asymmetric encryption and signatures. In the meantime there were different variants developed to overcome some shortcomings in the original version. These modified versions serve as a backbone of the current internet encryption in e.g. HTTPS, IMAPS, or other encrypted protocols.

All cryptographical methods base on pure numbers, thus all data is interpreted as such (big) numbers. There are now two different operators that are typically used in cryptography: encrypt/decrypt and sign/validate.

Assume a (big) number M that contains a message. By using a publicly known mapping $E(\cdot)$, anyone can *encrypt* M and will obtain $M_{\text{enc}} = E(M)$. The mapping however is designed to not be reversible in a simple manner. By only knowing M_{enc} one cannot deduce the value of M. The intended recipient of the message knows another mapping $D(\cdot)$ that allows to *decrypt* the message again $D(M_{\text{enc}}) = D(E(M)) = M$. Of course, the mapping $D(\cdot)$ must be kept secret.

The RSA algorithm provides an implementation of this scheme by using big prime numbers and some properties of integer arithmetic. The important part is that the message M_{enc} can be transmitted over insecure lines and its confidentiality is guaranteed by cryptography.

When the mappings $E(\cdot)$ and $D(\cdot)$ are constructed accordingly, it is possible that they are mutual inverses

$$D(E(M)) = M = E(D(M)).$$

That means that applying both operations once will cancel each other independently of the order of application. If the decryption is applied on M, this is called *signing* and the later encryption process *verification*.

The signing can only be carried out using the private mapping $D(\cdot)$. This allows to create a proof that a certain message was written by a certain person. Everyone can verify using $E(\cdot)$ if the message was changed. Thus, this method allows to ensure integrity using cryptography.

In the RSA algorithm, the mappings E and D use specially crafted numbers. These numbers are called public (in case of E) and private (in case of D) keys. These keys can be seen as parameters to general functions to obtain the individual implementations of the two mappings E and D mentioned above.

If the private mapping $D(\cdot)$ gets compromised (leakage of the private key) or the encryption method itself gets broken, the methods do no longer work. In case of encryption, the unencrypted message might be leaked. in case of signatures, an attacker might fake valid signatures.

For the sake of speed and memory consumption, these methods are typically combined with other cryoptographic methods that use symmetric cryptography. These details are more implementation-related and do not play a significant role for the whole process when implemented accordingly. They will therefore be neglected here.

2.3.3 Certificates and PKI

In order to simplify the management of public keys to be trusted, a new infrastructure needed to be built. A first step is to encapsulate the public keys into another structure to attach some meta data with them. Such a structure is called a X.509 certificate (Sharon Boeyen et al. 2008). The infrastructure to manage and use these certificates is called a *Public Key Infrastructure* (PKI).

Each certificate contains (apart from the public key) some additional information: there are some pieces of technical data (like the key size) stored. Also, there are a few human-readable strings describing the person or organization the certificate is associated with. These attributes are defined in Sharon Boeyen et al. (2008) and contain e.g. the *country*, the *common name*, and others. There are standardized abbreviation (`CN` for common name, etc). The attributes are rather restricted in terms of size and type: only 64 byte of ASCII text are possible per attribute for simple fragments of information like mail addresses or host names.

Additionally, certificates are issued by some authority. Each certificate has exactly one issuer. Issuing a new certificate is twofold: First, all relevant data like public key, meta data, etc is combined in a so-called *certificate signing request* (CSR). Next, the issuer uses his private key to sign the CSR after checking it. The new certificate is built from the CSR, the signature, and some unique reference to the issuer.

This principle of issuing certificates is its strength: it allows to delegate trust from a single source (so called *trust anchor*) recursively. Any user can trust some authorities, typically done by the operating system. Every service just needs to provide a chain of certificates to such a trust anchor, a so called trust chain. In that way the user can trust a public key without manually verifying the authenticity of said public key.

One last link is missing in order to establish trust in a thing, a connection, a name, or anything outside the cryptographic environment. The trust so far is *in a certificate*. For example, to obtain a certificate for a dedicated host name, the CN of the underlying CSR must have been set to the host name. This connects the certificate with the host name: a HTTPS connection requires that the `CN` of the signed certificate is equal to the host name. There are extensions for more complex scenarios that allow multiple host names per certificate (see `subjectAltName` in Sharon Boeyen et al. (2008)) but this is out of scope for this paper.

2.4 Intellectual Property

In Keith Eugene Maskus (2000), intellectual property is defined as following: “Human thought is astonishingly creative in finding solutions to applied technical and scientific problems [. . .]. These intellectual efforts create new technologies [. . .], develop new products and services [. . .]. They result in intellectual assets [. . .], that may have economic value if put into use in the marketplace. Such assets are called intellectual property to the extent they bear recognized ownership."

All pieces of software, tools and knowledge obtained by a company in its productive effort is considered to be owned by said company. The company will use these pieces for further benefit and protect them against access by third parties. This is intellectual property (IP).

3 Problem Formulation

Let's assume there are two (industry) partners A and B. One of them (A) is a vendor that sells some products. The other one (B) is a big player that buys these and integrates them together with dozens other products into complex systems. In order to simplify and optimize the engineering process, A provides simulation models for the relevant products to B. In the rest of the paper, we are focusing on one such model.

3.1 Concerns of Model Provider

These models are generated by the staff of A and contain parts of the internal knowledge of A. There could e.g. be some fancy algorithm implemented in the controller firmware of these products. The simulation model will eventually mirror these controller features to provide a good match of the model with the real world. Thus, part of the simulation model would probably be the proprietary firmware which A considers IP. As a result, partner A might have concerns to publish a white-box model to the partner B. Speaking in terms of security from Section 2.3, the partner A has a high requirement for confidentiality.

There might be additional wishes by the model provider A. For example, a model should have a lifetime after that it should be updated to the newest version. There might also be companies whose business model is to sell model hours.

3.2 Concerns of Model User

After transmitting the model to partner B over a to-be-defined transport, there are also some other aspects to be taken into account. At first, the model must be running in general and according to the specification. That is mostly covered by the dependability. However, there are also security considerations: The model might, in general, contain malicious code that is executed on the infrastructure of partner B. These threads range from attaching spamware by accident to active attacks (e.g. distribution of ransomware by a former employee).

So, from the perspective of B there are other requirements.

- Integrity: The model in question should be exactly the model of the product and not be tampered with.

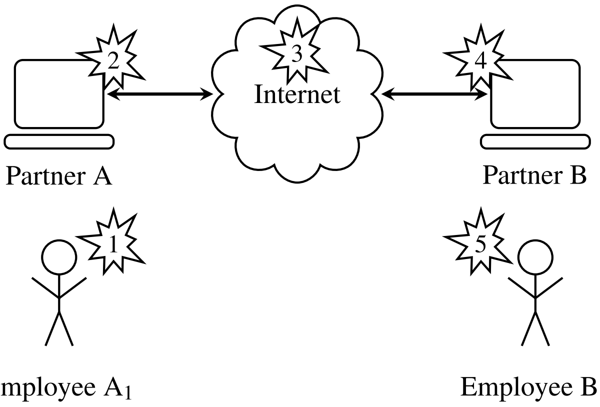


Figure 1. Data flow of the simulation model with possible attack vectors.

- Availability: The model should run anytime the partner B needs to run simulations without interfering with or blocking B's infrastructure.
- Confidentiality: The complete simulation model, where the model of A is integrated to, might be IP-related from the perspective of B. No data should be extracted using the simulation model.

3.3 General considerations

Having seen the positions of both parties one has to realize there are multiple levels of concerns. On the administrative level the partners use legal contracts to fix their mutual responsibility and accountability. This results in (internal) policies about what model from which partner might be run in which context. The technical level is located below that to ensure the correct realization of the administrative decisions and policies. This paper focuses on the technical level on how the parties can establish trust in the exchanged model. Therefore, it is assumed that both parties can be trusted and an appropriate policy is in place.

In that sense the exchange is very similar to sending potentially malicious data like documents with macros enabled, DLLs/SOs, etc via email. In contrast to best practices (e.g. only opening documents with macros globally disabled), exchanging FMUs in this way will enforce the user to actively open such untrusted documents.

4 Attack vectors

There are multiple parties involved, thus the analysis of possible attack vectors need to be carried out for all of them.

Starting with the process description in Section 3, one might follow the path of the simulation model. In Figure 1 the data exchange between the participants is depicted as well as the possible points of attacks. The possible attacks are numbered to simplify referencing them later.

1. The first opportunity for an attack is the user A_1 who generates the FMU on the side of A. He might will-

ingly or accidentially add malicious code to the FMU to be shipped.

2. The next opportunity is the infrastructure of company A. Although A_1 does his best in order to ship the FMU in a valid state, e.g. a virus on his machine might leak into the process and thus compromise the generated FMUs.

3. Knowing the dangers of the current time, the internet cannot be considered a safe place. Traffic might be transmitted over insecure channels that are not end-to-end encrypted. This also covers simpler methods like spoofing messages or just crawling and collecting data.

4. Analogously to attack 2, the infrastructure of B could be affected.

5. Finally, the user B_1 might be responsible for the potential attack similar to vector 1.

These attack vectors are not exhaustive, in fact there are attacks possible that combine different approaches. An example of such a combined attack would be the following: via social engineering the attacker could gain knowledge on A_1 (attack vector 1) to get known about a common project. Using a forged mail (attack vector 3) he tricks B_1 into opening and starting the model (attack vector 5).

For almost all possible attack vectors, there are concerns for both parties A and B involved. Some cases do not make sense, though: exposing some IP of A by A_1 can be done in various forms and does not fit in the context of this paper. Similarly, the IT department of A will do its best to prevent attack vector 2 to be used against A. The same is true for vectors 4 and 5 with respect to protection of B.

5 Abstract Solution Approaches

There are some general approaches that should be considered before the implementation can be done.

First, in order to simplify implementation effort on both sides A and B, the FMI standard is used as a basis. This directly addresses the requirements of the partner A with respect to confidentiality: the FMU can contain precompiled versions of the models leading almost to blackbox models. In theory, one could reverse-engineer the binary codes and obtain knowledge. There however are options to obfuscate and encrypt binary code until used (Michael Klooß, Anja Lehmann, and Andy Rupp 2019; Thomas Agrikola 2021). In order to use such methods, it is required for the FMI standard to support such enhancements officially: the official standard (*FMI V3* 2023) allows to augment functionality by using so called layered standards, the resulting unit must, however, still be compliant to the basic standard. If the encapsulated binary code was encrypted, this is no longer covered by the standard. The simulation environment needs to be able to decrypt accordingly on the fly in order to make any use of the FMU.

The primary goal for B is to establish trust on technical level between the partners A and B. This should be done using cryptography: a method is to be derived that can crypographically prove the integrity of the model and its origin (partner A). No model should be run that was tampered with or that was broken during transport.

The most prominent issues come from attack vector 3, potentially in combination with other ones. Apart from that, it is typically sufficient to require trust between A and B: by contracts and legal bindings these parties typically consent on mutual trust. The trust of B in A_1 is not needed as A will trust A_1 and hold him liable and responsible for his actions. Also, from B's perspective, still A is liable for any issues. The same holds true for the trust of A in B_1. Therefore, in this work the attack vector 3 is to be considered the primary one.

In order to prevent the model from accitential changes, a hash of the model is generated directly after forging the FMU by A. This hash can be delivered with the model and checked just before the real execution of the model. The model verification will detect changes to the model with a very high probability (i.e. if there are no hash collisions) and report that. It might be up to said user how to cope with such a situation but this decision is part of the administrative policies excluded from this paper. In fact, the tests are just an addon to FMI and can always be overridden by the simulation environment.

So, in general the hash allows to detect transmission and storage errors between A and B. In case of a malicious actor, the hash will only provide little help: the attacker could simply calculate the hash of the modified FMU and replace the original hash as well. The user will check the (modified) FMU and compare with the faked hash. Thus, no warning will be issued and the user might be running the tampered FMU without further notice.

To prevent such an attack, it is necessary to establish trust in the hash. This can be achieved by crypographical means in the form of certificates. The vendor A creates a valid certificate (plus its private key) before deploying the FMU. The private key is used to sign the hash. This certificate with the complete trust chain including the signature of the hash is delivered with the FMU.

On the user's side (B), first, the hash is validated. To do that, the hash of the FMU is calculated by B. If the signature matches with the calculated hash, B has proven that the person who created the signature had access to the corresponding private key.

It is not required to provide a hash of the FMU in the metadata as long as there is a croptographic signature available. However, deploying the hash as well has the benefit that the test of the validity can provide more detailed error messages on why the validation failed. With the hash it is possible to distinguish an accidential transmission error from an attack that might need further investigation.

This process of authentication is however not sufficient to provide authorization to run the FMU. Here, the administrative policies as depicted in the problem formulation need to be addressed. The question is: which entities are to be allowed to provide trusted FMUs and how to prevent other entities from pretending and faking their identity? Typically, this is done by enforcing certain rules on the certificate chain. For example, there could be a requirement that the certificate is (indirectly) signed by a certificate with a well-known CN. If the systems are online, one can use authentication using other systems, but this is not the major point in this paper (see e.g. challenges `HTTP-01` and `DNS-01` in Daniel McCarney (2017)).

In any case, the simulation environment on the infrastructure of B must run these tests just before the FMU gets loaded and started. This is similar to the signing process of system libraries in the Microsoft Windows operating system that refuses to install unsigned libraries. These checks are in fact a requirement for the software providers that handle the FMUs. These providers need to add support for appropriate steps in order to ensure security of the models, potentially establishing an accompanying standard to the plain FMI standard.

It is vital to understand that once the simulation environment separates the simulation model from the security data by checking the signatures (and only passing the simulation model on), the protection ends. For example, assume that party B checked some FMU by whatever means, decides the FMU to be secure, and stores the FMU in a local simulation library without the security data. The attack vector 1 through 3 has been ruled out so far. However, the attack vector 4 is still open: changing the FMU in the library could easily get unnoticed. Thus, the separation should be done as late as possible to cover most of the possible attack surface. Ideally, the check is done right before the simulation itself is run and the first call to a function in the FMU is carried out.

Note, that the suggested approach does only secure the transport of the models. The execution is not affected, which can be seen that the existing simulation cores do not need to be altered. So, there is no protection against online-changes to the FMU while it is running, e.g. due to defective RAM.

The approach of using certificates also allows for the additional feature of lifetimes of the models. All certificates have a lifetime which ends at some time. This can be used to invalidate a model after a certain point in the future. The checker in the software will refuse the validity of the certificate due to an expired certificate if the lifetime of the model has expired. This is however not cryptographically enforced.

There is no guarantee about confidentiality so far from the perspective of A. Anyone who has access to the FMU will be able to run the system. To prevent that, one could use encryption on top of the provided solution. The encryption is not as simple as signatures as each legitimate user of the FMU needs a way to decrypt it with their individual private keys. One possible solution was that vendor A provides a public API where any potential customer (like e.g. B) can request an individually encrypted simulation model. To do so, the potential buyer needs to authenticate. Before the model is encrypted and provided, the vendor can check if the request is authorized.

The certificate chain can be adopted to the needs of the use case. If the vendor A provides individual certificates for their employees, A_1 would use his personal certificate (and key) to sign the FMU certificate. This means an additional benefit of accountability: whenever a problem arises it can be tracked down to the individual user A_1 who has signed the FMU in question. This simplifies incident analysis and provides some internal mutual protection between A and A_1.

By defining an extension to the FMI standard, one could introduce a formal certification scheme: FMUs could be labeled as “certified secure according to the standard”. This could also be applied for the simulation tools to certificate that these abide some security guidelines about when a model is considered harmful and not to be run.

6 Alternatives considered

There are a few alternatives to the abstract solutions presented above. These can be ruled out for different reasons and the argumentation should be mentioned here, shortly.

6.1 Classical transport with cryptography

The most direct approach to this problem would be to use state of the art mechanisms provided by the IT to establish cryptographal security. One could use S/MIME (Blake C. Ramsdell 1999) or GPG/PGP (Simson Garfinkel 1995) to add end-to-end encryption to e.g. mail delivery. This has however the drawback that the signatures are typically bound to a single person instead of the corresponding company. Volatility in personnel will make handling hard and error-prone.

Additionally, these approaches need manual work by the users. While this seems only a minor burden, it adds the risk of wrong application and user errors. In case of problems, people might tend to avoid the system altogether. A fully automatic solution is preferable here.

6.2 Callback in FMU

Looking at the problem from the FMI context, one straight forward approach would be to put the security information into the code. The user’s simulation environment could use `fmi3GetBinary` to extract and check it. This will however not serve well as the function call would already execute code from the FMU that is (not yet) to be trusted. So, in order to check the validity, the simulation environment must only use statically available information.

After the security has been established by other means, there are still use cases to call a custom callback in the FMU. This would allow to realize the already mentioned business models that require the validation of a license of the customer. So, after the initial security check, the FMU itself could issue a check if the user (B) is to be allowed to run the simulation and abort if not. There could be some license fees to be paid or simply the model must only be run on behalf of real customers. The checks could be arbitrary to match with the business model of A. Currently, different other callbacks are (mis-)used for such functionality, having a dedicated callback would be preferable.

6.3 Storage of hash in CN

One has to note that the attributes of a certificate are generally not potent enough to store the complete FMU. This is true for the proposed solution as well.

Complying with the de-facto standard of storing in the CN a unique identifier, one could try to calculate an immutable identifier for an FMU using hashing. Then, this hash could be used in the CSR to generate a certificate with the CN set to the hash of the underlying FMU. That way, the certificate would not just be a generic certificate but the certificate of said FMU. In that sense, it would avoid to use external signatures and keep all cryptography in the PKI.

By looking at the recent hash functions SHA-512 or BLAKE-512, these use 512 bit or 64 byte in binary form to represent such a hash. As the attributes are only ASCII text, the binary must be mapped into ASCII which enlarges it further. For example, to represent the complete hash one could use base64 encoding but will then need 88 characters (86 as the length can be considered known). Considering that the attributes are capped at 64 characters, the hash will not fit. Consequently, one could only use hashes with 384 bit or less.

Larger hashes provide better security in theory and make (exploitable) hash collisions less likely. The size of the hashes was therefore growing in the past and one has to assume that this trend will continue. So, one must at least consider larger hashes and cannot reject them in the architecture of this approach.

7 Proposed Solutions

The so far described approaches are of rather abstract nature. There are different ways thinkable how this could be implemented and especially where the hash and the signature could be stored.

7.1 Adaption in a layered FMI standard

The most simple way would be to implement the hashing and signing inside an extended layered FMI standard. That way, the FMU would still be an encapsulated and complete model that has the security features included. The FMI standard allows to provide additional data like static (XML) files in an appropriate folder under the top level folder called `meta`. For example, one could define a folder `meta/de.eks-intec.fmi-sec` that contains further files with the required security-related data.

The trivial approach of hashing the complete FMU has one major drawback, though. In order to zip the FMU file, one needs the hash, leading to an chicken-and-egg problem.

Consequently, one needs a more fine-tuned approach. Just before packing of the FMU, all included files are listed. The algorithm will filter out the files in the folder `meta/de.eks-intec.fmi-sec`. For each file, a hash is calculated and is stored individually with the corresponding (relative) filename in the `hashes.xml` file in said folder. Having finalized this file, one can calculate a hash and a signature of `hashes.xml` and write these into a sibling file `security.xml`. The latter is augmented by the complete certificate chain to help validating.

To check the validity of such a composed FMU, the simulation environment has to carry out multiple steps: first, the validity of the certificates need to be ensured (like checking for expiry, trust, and pairwise signatures). Then, the certificate can be used to check the validity of the `hashes.xml` file. Once this is confirmed, each file in the FMU has to be checked against the stored hashes. There should be no additional files found, so, if that happened, the algorithm would issue a warning or even fail validation of the complete FMU. Once all files in the FMU have been validated, it can be considered harmless and processed further. In contrast to the approach in Section 7.3, this one does not need additional tools (e.g. to read AML files).

7.2 Externally in the Network

Another option to handle the hashes, certificates, and signatures would be to define one URL per FMU that will provide all relevant information. So, the software running the model in question would need to download the security information from the site and check the FMU against that.

This is possible because the URL can be pre-defined and thus statically embedded in the FMU somehow. This might work similar to the approach in Section 7.1. In the open source world it is common to have for each downloadable file an additional file with some hash or signature to check for download errors. In a similar fashion, one could manage the deployment of the security-related data of FMUs.

As the security information is generally available online, this process allows also to update the certificates (renew it in case its lifetime should be extended past the original end). Regular updates of short-lived certificate enhance the overall security (Emin Topalovic et al. 2012; Ronald L. Rivest 1998): the longer the certificates are valid, the more time is to break the keys. Short-living certificates and keys reset these time windows and make breaking the keys by pure chance very unlikely.

As the vendor A has control over the certificates, it is also possible to think of new business models. One could pay per issued certificate, per hour of model usage, per simulation run, or other metrics. The need to fetch a certificate makes it rather simple to control these type of usage-based billing. The online approach works similar to a physical token but without the overhead of physically transferring it.

The drawback of this method is that the simulation environment needs to have continuous access to the certification service. This makes it impossible to use air-gapped systems. There could be caches included that provide a way to store the certificate until it expires, still, the renewal process needs to be triggered sometime. Also the certification provider (typically A) could anytime stop issuing new certificates (e.g. also due to technical issues or bankruptcy) and the certificates as well as the simulation models will cease to work.

7.3 Embedding in a Digital Twin

By combining technologies of different fields, one might achieve a matching solution for most of the problems presented. Instead of embedding the data in the FMU as described in Section 7.1, there might be already a location present to store these additional information. According to Roberto Minerva, Gyu Myoung Lee, and Noel Crespi (2020) many products are in the meantime supported by their individual DT. Using the example of AML as a description language for DTs, one has to realize that for many use cases (not only in the context of this paper but for general problems), on might fall back to pre-existing solutions.

In AML, e.g. a library to import FMUs into the AML file exists already (Olaf Graeser et al. 2011). If the vendor A already provides a DT for his products, the FMU can easily be added into said DT. If no DTs are yet existing, the overhead to create such a twin is minimal.

With the AML in place, one has a clear and well-defined outer shell. The security elements in AML need to be modeled separately, the basic elements are certificate chain links. One example modelling of such a DT can be seen in Christian Wolf, Miriam Schleipen, and Georg Frey (2023).

The benefit to use a dedicated format over extending the FMU in 7.1 is that other modelling standards (like e.g. SSP (*SSP* 2022)) can be used without change. This makes this approach a very generic one.

For the AAS, Andre Bröring et al. (2022) present an approach that allows to prove integrity of the data. However, the basic idea is to mimic the GIT version control system in terms of a DT meta model. As with GIT, it is possible to rewrite the history to insert a tampered FMU unnoticed. The integrity of the history and trust in the FMU can thus only be guaranteed for read-only access. Having said that, by augmenting the suggestions with cryptography (and migrating to AML), one would have a very similar result to the one presented in this paper.

7.4 FMI as Open Document

The current FMI standard is very similar to the storage format described by the open document standard *IEC 29500* (2012). As the open documents standard has some security features embedded, this would allow to secure the FMUs directly. It might thus be considerable to make the FMI fully conforming with *IEC 29500* (2012), probably a rather small change. As this would require a change in the core FMI standard, it is neglected for this paper, though.

8 Prototype

In order to show feasibility of the proposed approach in Section 7.3 a prototype as a proof of concept has been implemented. This approach is a good tradeoff: it allows arbitrary extensions, is attached with products more and more, works offline, and is intentionally not backward compatible (which might cause security risks by false trust). The prototype does not carry out any real simulations but provides a way to execute and validate the complete process as described in this paper in a minimal environment.

The prototype has three major functions: first, in a bootstrapping process a self-signed PKI can be generated. This allows for local testing and understanding the concepts involved. Additionally, there is the option to sign an FMU and pack it into an AMLX file. This allows also to generate AMLX files with broken or spoofed FMUs to test the detection. Finally, one can extract the FMU again from the AMLX while checking the security measures.

The bootstrapping function allows to create a complete PKI from scratch with a self-signed root certificate as CA. Obviously, this should not be used for production but only serves as a demonstrator of the process. For more details, please have a look at the corresponding code and documentation (Christian Wolf 2023). This step will as well generate multiple AMLX files in accordance to the description in Section 7.3:

nominal A fully functional FMU, correctly hashed and signed

broken To simulate a defective file/transport, the FMU is modified but the other metadata are copied from the nominal case

tampered The FMU and its corresponding hash is modified in the AMLX by a targeted attack, all other metadata is copied over

The second step is to sign a custom FMU with the certificates as provided in the test instance. This process is rather straight forward, as no tests are carried out. Only the required files are read and interpreted and a valid signature is created. The process is depicted in Figure 2 as a flow diagram.

As final step, the prototype provides a way to check any AMLX file against a given root certificate. It allows to export the embedded FMU into a stand-alone file that can be run in legacy FMI-3-based simulation tools. Alternatively, it can serve as a boilerplate to implement an import feature for productive third party simulation tools.

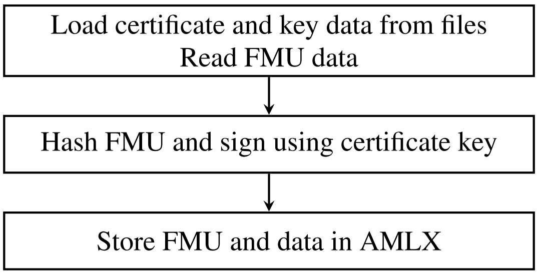

Load certificate and key data from files
Read FMU data
Hash FMU and sign using certificate key
Store FMU and data in AMLX

Figure 2. Flow diagram of the signing of FMUs.

The verification process is more complex than the signing as various checks need to be carried out. The ordering is in general of lower priority as all test must necessarily pass.

- (optional) The hash stored in the AMLX file must match the hash of the stored FMU.
- The certificate chain must be anchored on a user-provided trust anchor
- The complete certificate chain must be a chain, no branching is allowed and the chain must be in the correct ordering. Each certificate must be signed by the next certificate in the chain.
- Each certificate in the chain must not be expired.
- The signature must be valid

The overall process is sketched in Figure 3

Please note that the extraction process checks the certificates but after the splitting, the FMU is just a common FMU. There are no security features attached anymore. Changing the FMU after the export will not be detectable anymore in a secure manner. As a minimal measure to prevent accidential changes, a hash file is generated automatically during export.

By testing the various auto-generated examples in the first step through the checker, it is possible to see that the demonstrator can detect the changes in the FMU and metadata. Only the nominal version is accepted by the prototype.

9 Further Work

A survey with various industry partners is carried out at the time of writing. The goal is to identify detailed requirements and wishes from both FMU providers and consumers. Especially the position of the producers and their need to protect/encrypt the models is not yet analyzed strictly. Using this as a baseline, further improvements should be investigated as well as the other implementations in Section 7 addressed.

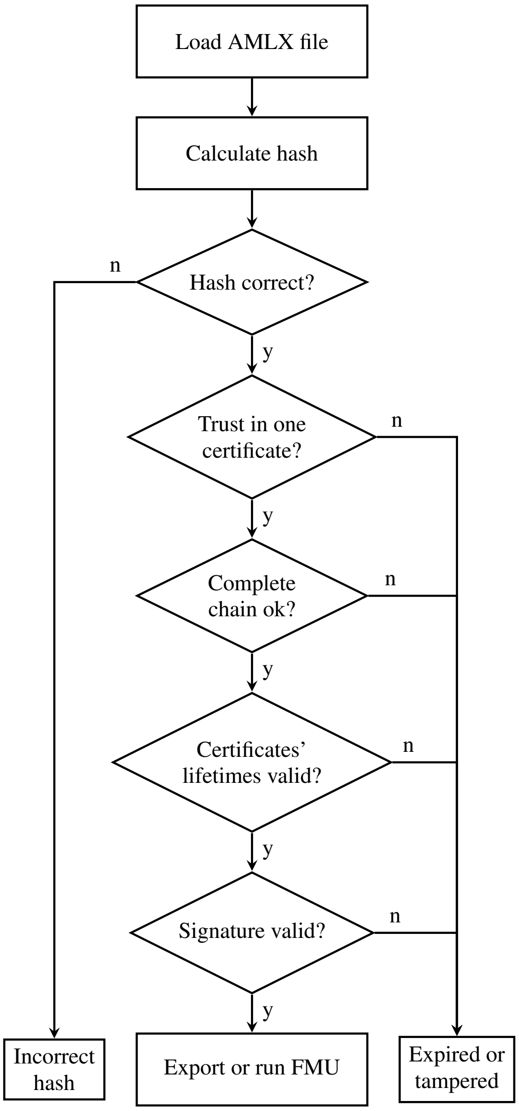


Figure 3. Flow diagram of verifying and exporting an FMU.

10 Summary

In this paper the general security problem especially for untested, unverified, or black-box simulation models is focused. There are different attack vectors presented that could be used to compromise the usage of shared models. Each attack vector has its individual attack surface and risk involved.

Due to the impossibility to prevent all attacks in general, an abstract analysis of the situation and possible general solutions are given. There are four possible implementations shown to realize the abstract considerations. For one of the four options a prototype has been implemented to show the effectiveness of the approach as a proof of concept.

The same approach as presented in this paper can be used to augment combinations of FMUs: one can use the System Structure and Parametrization standard (*SSP* 2022) instead of plain FMUs to be embedded.

Acknowledgements

The EKS contribution to this paper is part of the AIToC project. AIToC is a project of the European ITEA Initiative, funded by the German Federal Ministry of Education and Research under the funding code 01IS20073C.

References

Alex Biryukov, Dumitru-Daniel Dinu, and Dmitry Khovratovich (2015). *Argon and Argon2*. Password Hashing Competition (PHC).

Andre Bröring et al. (2022-11-03). “An Asset Administration Shell Version Control to Enforce Integrity Protection”. In: *Kommunikation in Der Automation : Beiträge Des Jahreskolloquiums KommA 2022*. Vol. 13, pp. 192–203.

Avizienis, A. et al. (2004). “Basic Concepts and Taxonomy of Dependable and Secure Computing”. In: *IEEE Transactions on Dependable and Secure Computing*. DOI: 10.1109/TDSC.2004.2.

Bart Preneel (1994). “Cryptographic Hash Functions”. In: *European Transactions on Telecommunications* 5.4, pp. 431–448.

Blake C. Ramsdell (1999-06). *S/MIME Version 3 Message Specification*. DOI: 10.17487/RFC2633.

Christian Wolf (2023). *Christianlupus-Phd/Prototype-AMLX-checker*. URL: https://github.com/christianlupus-phd/Prototype-AMLX-checker.

Christian Wolf, Miriam Schleipen, and Georg Frey (2023-05-11). “Dynamische Systemmodelle austauschen mit FMI – aber sicher!” In: *atp magazin* 2023.5, p. 26.

Christoph Schlueter Langdon and Karsten Schweichhart (2022). “Data Spaces: First Applications in Mobility and Industry”. In: *Designing Data Spaces: The Ecosystem Approach to Competitive Advantage*. Ed. by Boris Otto, Michael ten Hompel, and Stefan Wrobel, pp. 493–511. DOI: 10.1007/978-3-030-93975-5_30.

Daniel McCarney (2017-06). “A Tour of the Automatic Certificate Management Environment (ACME)”. In: *The Internet Protocol Journal* 20.2, pp. 2–14.

Deutsches Institut für Normung, ed. (2019). *Asset Adminstration Shell Reading Guide*. 1st edition. ISBN: 978-3-8007-4990-4 978-3-410-28919-7.

IEC 63278-1 (2022-06-17). *DIN EN IEC 63278-1: Asset Administration Shell for Industrial Applications - Part 1: Administartion Shell Structure*.

Emin Topalovic et al. (2012). *Towards Short-Lived Certificates*.

FMI V3 (2023). *Functional Mock-up Interface Specification v3.0*. URL: https://fmi-standard.org/docs/3.0/ (visited on 2023-01-06).

Hedda Massoth (2022-11-21). *Catena-X Operating Model Whitepaper, Release V2*. URL: https://catena-x.net/fileadmin/user_upload/Publikationen_und_WhitePaper_des_Vereins/CX_Operating_Model_Whitepaper_02_12_22.pdf.

IEC 29500 (2012-09-01). *ISO/IEC 29500-2: Open Packaging Conventions*.

Jean-Philippe Aumasson et al. (2008). “Sha-3 Proposal Blake”. In: *Submission to NIST* 92.

Johannes Mezger et al. (2011-07). “Protecting Know-How in Cross-Organisational Functional Mock-up by a Service-Oriented Approach with Trust Centres”. In: *2011 9th IEEE International Conference on Industrial Informatics*, pp. 628–633. DOI: 10.1109/INDIN.2011.6034951.

Keith Eugene Maskus (2000). *Intellectual Property Rights in the Global Economy*. Peterson Institute.

Michael Klooß, Anja Lehmann, and Andy Rupp (2019). “(R)CCA Secure Updatable Encryption with Integrity Protection”. In: *Advances in Cryptology – EUROCRYPT 2019*. Ed. by Yuval Ishai and Vincent Rijmen. DOI: 10.1007/978-3-030-17653-2_3.

Olaf Graeser et al. (2011). “AutomationML as a Basis for Offline and Realtime Simulation - Planning, Simulation and Diagnosis of Automation Systems:” in: *Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics*, pp. 359–368. DOI: 10.5220/0003537403590368.

Rainer Drath (2021-07-19). *AutomationML: A Practical Guide*. Walter de Gruyter GmbH & Co KG. 290 pp. ISBN: 978-3-11-074623-5.

Rainer Drath, Markus Rentschler, and Michael Hoffmeister (2019-09). “The AutomationML Component Description in the Context of the Asset Administration Shell”. In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1278–1281. DOI: 10.1109/ETFA.2019.8869214.

Rajeev Sobti and G Geetha (2012). “Cryptographic Hash Functions: A Review”. In: *International Journal of Computer Science Issues (IJCSI)* 9.2, p. 461.

Roberto Minerva, Gyu Myoung Lee, and Noel Crespi (2020-10). “Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models”. In: *Proceedings of the IEEE* 108.10, pp. 1785–1824. DOI: 10.1109/JPROC.2020.2998530.

Ronald L. Rivest (1998). “Can We Eliminate Certificate Revocation Lists?” In: *Financial Cryptography*. Ed. by Rafael Hirchfeld. Red. by Gerhard Goos, Juris Hartmanis, and Jan Van Leeuwen. Vol. 1465, pp. 178–183. DOI: 10.1007/BFb0055482.

Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman (1978-02-01). “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Communications of the ACM* 21.2, pp. 120–126. DOI: 10.1145/359340.359342.

Sharon Boeyen et al. (2008-05). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. DOI: 10.17487/RFC5280.

Simson Garfinkel (1995). *PGP: Pretty Good Privacy*. "O’Reilly Media, Inc." 442 pp. ISBN: 978-1-56592-098-9.

Stefan Tillich et al. (2009). “Compact Hardware Implementations of the SHA-3 Candidates ARIRANG, BLAKE, Grøstl, and Skein”. In: *Cryptology ePrint Archive*.

SSP (2022-07-25). *System Structure and Parametrization V 1.0.1*.

Thomas Agrikola (2021). “On Foundations of Protecting Computations”. PhD thesis. Karlsruher Institut für Technologie (KIT). 281 pp. DOI: 10.5445/IR/1000133798.

The Common Requirement Modeling Language

Daniel Bouskela[1] Lena Buffoni[2] Audrey Jardin[1] Vince Molnár[3] Adrian Pop[2] Ármin Zavada[4]

[1]Electricité de France, France, {daniel.bouskela,audrey.jardin}@edf.fr
[2]Linköping University, Sweden, {lena.buffoni,adrian.pop}@liu.se
[3]Budapest University of Technology and Economics, Hungary molnarv@mit.bme.hu
[3]IncQuery Labs cPlc., Hungary armin.zavada@incquerylabs.com

Abstract

CRML (the Common Requirement Modeling Language) is a new language for the formal expression of requirements. The goal is to release the language as an open standard integrated into the open source modeling and simulation tool OpenModelica and interoperable with the open systems engineering standard SysMLv2. CRML allows for the expression of requirements as multidisciplinary spatiotemporal constraints that can be verified against system design by co-simulating requirements models with behavioral models. The requirements models must be easily legible and sharable between disciplines and stakeholders and must capture realistic constraints on the system, including time-dependent constraints with probabilistic criteria, in recognition of the fact that no constraint can be fulfilled at any time at any cost. The theoretical foundation of the language lies on 4-valued Boolean algebra, set theory and function theory. The coupling of the requirements models to the behavioral models is obtained through the specification of bindings, the automatic generation of Modelica code from the CRML model and use of the FMI and SSP standards. CRML and the proposed methodology is compatible with SysMLv2, forming a comprehensive workflow and tool-chain encompassing requirement analysis, system design and Validation and Verification (V&V). The final objective is to facilitate the demonstration of correctness of system behavior against assumptions and requirements by building a workflow around Model-Driven Engineering and Open Standards for automating the creation of verification simulators.

Keywords: cyber-physical systems, systems engineering, requirement modelling, systems verification, Modelica, FMI, SSP, SysML

1 Motivations and Challenges

Large numbers of stakeholders are involved in the design and operation of complex cyber-physical systems (CPS), especially but not exclusively in the energy sector. When working on a common system, stakeholders tend to express requirements from their own perspective, resulting in a global set of constraints on the system that can be conflicting, even contradictory. Also, to avoid questioning the motivations of poorly-documented past design decisions, new requirements are often added without questioning the soundness of existing ones. This results in over-specifications, delays, and cost overruns. The search for a common agreement between stakeholders that preserves degrees of freedom for optimal design is always difficult and lengthy (Azzouzi et al. 2022).

CRML (Common Requirement Modeling Language) is a new language for the formal expression of requirements collaboratively developed by different industrial and academic stakeholders. **The goal is to release CRML as an open standard to offer stakeholders from different domains and disciplines a common language to express, organize, negotiate, and simulate requirements in order to find the best compromise that suits their needs while complying with their mutual commitments**.

This goal raises the question of expressing CPS requirements that are realistic, understandable, and verifiable by and between stakeholders. More precisely, it means that the underlying formal language and method associated have to tackle the following challenges:

- The language should provide **comprehensive descriptions of all spatiotemporal assumptions and constraints** that bear on the system under study. Constraints can be of all kinds and may vary depending on the system operating mode: physical, performance (reliability, availability, economical…), and regulatory (safety, security, environmental, reserve capacity for grid balancing, grid access, and priority dispatch…).

- The **requirements models must be easily legible and unambiguous**. It is expected that a requirement language common to all stakeholders regardless of their expertise and business domain will improve the productivity of studies. To that end, the syntax *must be close to natural language*.

- CPSs exhibit strong physical aspects. Therefore, **particular attention must be paid to physical aspects**: physical units, real-time, events, synchronism and asynchronism, components and objects, failures, and uncertainties. Time-dependent continuous and discrete variables must be dealt with in a hybrid synchronous and asynchronous framework. This goes

DOI
10.3384/ecp204497

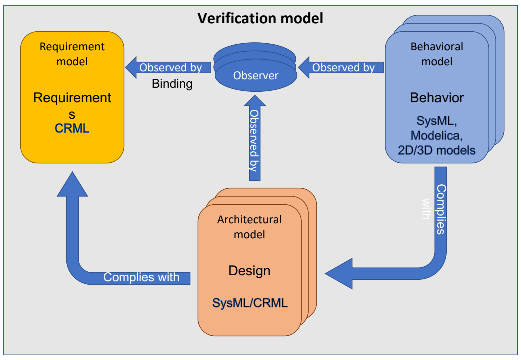


Figure 1. Architecture of the verification model

well beyond finite automata that are the reference in model checking (Baier and Katoen 2008).

- **Verification should be automated as much as possible all along the system lifecycle by co-simulating requirements models with solution models of any kind and growing complexity** ranging from *min* and *max* limits (that represent authorized operation domains), to finite state automata (that represent logical system operation), and to multidomain physical 0D/1D/2D/3D models (that represent detailed system physical behavior).

  The verification model (Fig. 1), which tells whether requirements are satisfied or not, consists of the requirement models, the solution models (behavioral and architectural), the observers of the solution models, and the links between observers and requirements, which are called "bindings". Requirements models with observers act as virtual sensors to detect possible requirement violations of the solution models. Behavioral models capture the dynamic behavior of the system in various forms (state automata, algebraic or differential equations), whereas architectural models carry static information about the system.

- **Probabilistic criteria must be added to requirements** because no realistic requirement can be satisfied with absolute certainty: in general, it is not enough to specify what should happen in nominal mode, one should also define what should happen in the event of specific hazards and with which probability the real system will not enter in one of these cases (i.e. how reliable the system should be). These stochastic requirements have to be verified against stochastic solution models. Monte Carlo techniques could hence be used to simulate the verification models.

2 State of the Art

The current state of the art regarding the above challenges tends to consider them separately. Consequently, there is no integrated tool able to deal with all of them in a consistent way. The main gaps concern the links between logical design, physical design, and dependability analysis as they currently involve completely different methods and tools: logical design uses methods such as UML (OMG 2017) and SysML (OMG 2023) based on first-order logic that originates from the software industry. Physical design uses tools such as Modelica tools, Matlab, Simcenter Amesim, etc. that deal with physical laws in the form of DAEs (Differential-Algebraic Equations) (or block diagrams) and dependability analysis uses probabilistic methods.

2.1 Limitations of Requirement Modeling Tools for CPS

Regarding the modelling of requirements no convincing solution exists to express CPS requirements independently from design solutions in a formal way. Current state-of-the-art tools hence focus on expressing requirements in natural language such as in Rational DOORS, Polarion, etc. This comes from the fact that the existing formal requirements modelling methods such as LTL (Linear Temporal Logic) and CTL (Computation Tree Logic) (Baier and Katoen 2008), timed (Alur 1999) and hybrid (Henzinger 2000) automata or UML/SysML state behavioral diagrams (OMG 2019) tend to bear on abstractions of the system in the form of state machines, which *already express a solution* and hence **is not appropriate to correctly deal with CPS physical aspects**. For instance, with LTL, one can prove that a system will always or eventually pass through a given state. Timed automata can handle real-time, but only when the states are known in advance. This corresponds to an idealistic view of the system that is not for instance subject to wear or external aggression. Hence they do not consider situations where existing states are subject to gradual drift due to wear, or new states appear due to unexpected events. In other words, CPS contain finite-state machines, but they cannot as a whole be considered as finite-state machines. Other limitations could be quoted such as:

- **Lack of object-orientation**: temporal constraints cannot be (easily) associated to the system architecture (i.e. its decomposition into subsystems and components).

- **Difficult mathematical syntax**: although mathematical syntax (and semantics) is necessary to perform formal proofs (model checking) or even model simulation, it is difficult to use on a day-to-day basis for the whole system.

Various attempts have been made to alleviate these limitations, for example by extending OCL (Object Constraint Language) with temporal constraints, but none of them are used convincingly in practice (Kanso and Taha 2013). Therefore, formal requirements languages are usually only used for small (sub)systems with critical safety concerns, and they cover only the very early phases of system design when (only) the logic of the system is investigated.

The relatively new SysPhS (SysML Extension for Physical Interaction and Signal Flow Simulation) profile for SysML (OMG 2021) provides extensions to model physical interaction and flows independently of the simulation platform. It includes a textual syntax for mathematical expressions as well as reusable simulation elements. It also defines translations to Modelica and Simulink. Along with the previous approaches, SysPhS also focuses on the precise modeling of specific designs and not the specification of a *design envelope*.

The new SysMLv2 language (OMG 2023) will feature a native expression language and better modeling of requirements. It provides a mechanism to bind requirements to design artifacts to formalize them in terms of the chosen subject. Furthermore, analysis and verification cases provide a way to model evaluation and verification steps, and the language supports modeling spatiotemporal aspects as well. While this is much closer to our requirements, native constructs in SysMLv2 are not designed to fully cover aspects like the modeling of time-dependent hybrid systems, probabilistic criteria, or automated verification. The extensibility of the language, however, provides a way for us to make the two languages interoperable.

2.2 Why Modelica is Natively Suited for CPS Modeling but Not for Requirements?

Modelica (Mattsson, Elmqvist, and Otter 1998) is a language that comes with a convenient graphical interface fit for the description of the physical real-time behavior of CPS. However, Modelica does not allow one to express constraints on a system when its architecture is partially unknown (for example express a constraint on a valve, when it is unknown how many valves will be in the final design), and expresses the behavior of the system in the form of DAEs (Differential-Algebraic equations) rather than the constraints on the behavior of the system. As a consequence Modelica is insufficient to express all that is needed at the early design phases, especially when one wants to specify only the acceptable envelopes without going into realization details, so that the solution space is refined progressively, rather than committing to a single design decision that fits the criteria.

Graph-based design languages with their capability to explicitly modify product topology and parametrics are on the one hand partially able to fill this gap, but need to be extended on the other hand by more powerful formal methods for requirements processing, tracing and consistency checking (as illustrated in Section 2.1 with SysML).

Therefore, connecting formal requirement modeling languages such as the ones mentioned above directly to Modelica does not solve the general problem of having a model-based methodology that covers the whole engineering lifecycle for CPS, as such kind of solution is only valid if the system is considered as a state-machine and for the engineering phases past the detailed design phase (which is somewhat contradictory).

2.3 CRML Origin and History

As previously mentioned, a Modelica model expresses the behavior of the system but does not say for what purpose the model is made. For instance, the model of a cooling system features heat exchangers, but does not say anything about the properties of the system that we want to verify, e.g. whether the flow velocity inside the heat exchanger stays below a given threshold. To alleviate the problem, one of the ITEA EUROSYSLIB project (2007 – 2010) objectives was to investigate the possibility of associating constraints that represent requirements to a Modelica model. For the above example, that meant associating the constraint that the flow velocity must not exceed a given threshold to the model of a heat exchanger. The first idea was to express this kind of constraint directly in the Modelica model, for instance in a dedicated ‘constraint’ Modelica section (similarly to the existing ‘equation’ and ‘algorithm’ sections). However, this solution had the drawback to modify model components in order to handle specific constraints, which is not consistent with the generic nature of model components. Besides, with this solution there was no way to express something like *‘No pump in the system must cavitate’* or *‘At least one pump in the system must be started’*, for two reasons: (1) The notion of quantifier does not exist in Modelica, so the constraint must be written taking into account the current topology of the circuit (i.e. the number of pumps in the system), and modified when the model topology is changed (i.e. when the number of pumps changes), even if the meaning stays the same. (2) The notion of a pump being started or not is usually not present in the Modelica model, because it expresses the *physical state* of the pump, not its *operational state*. It becomes clear then that the requirement model must be separated from the behavioral model. The rest of the EUROSYSLIB project was then mainly devoted to look at different languages for expressing the properties of systems.

As no interesting requirement modelling language emerged, the idea to create a new language came within the ITEA MODRIO project (2012 – 2016) with the following aspects in mind: (1) the syntax must be close to natural language, (2) the language must handle time periods and probabilistic aspects, (3) the language must handle quantifiers (i.e., sets) and be object-oriented, and (4) it should be possible to automatically generate test sequences from the constraints that represent assumptions on the system. It resulted in the specification of a new language called FORM-L (for Formal Requirement Modeling Language), written by EDF (Nguyen 2014). In parallel other works emerged around the same period with rather close similarities: TOCL (a temporal extension of OCL) and Stimulus (Dassault Systèmes). Regarding TOCL, there is no known implementation and it seems that there is no ongoing effort. For Stimulus, it appears that the tool is more focused on debugging requirements for a particular design and express them as temporal stochastic state machines, which is less general than FORM-L which aims at expressing requirements that are generic and that could be refined throughout the whole engineering cycle to evaluate multiple design solutions.

In 2013, FORM-L was proposed to the Modelica community (most of the Modelica community was participating in the MODRIO project, except OpenModelica), which offered to extend Modelica with the notion of ‘blocks as functions’ and develop a new Modelica library for the modelling of requirements (called Modelica Requirements) (Otter et al. 2015). This library proved to be unsatisfactory because the time period and the condition to be verified were mixed within the same block. Therefore, it was impossible to have a stable library because of the combinatorial explosion of possibilities of associating conditions with time periods (a new block would have to be developed each time a new case occurs, which would be almost as frequent as when a new requirement is written). Therefore, EDF decided to develop a new requirement Modelica library called ReqSysPro that would clearly separate time periods from conditions, so that a requirement would be obtained by associating a block representing a time period to a block representing a condition (and not having the two aspects in the same block). The first idea was to use state machines to evaluate requirements: a requirement would pass through different states, starting from ‘Untested’ until it becomes either ‘Satisfied’ or ‘Violated’. The problem with this solution is (1) that it was not possible to find a single state machine that would handle all possible requirements, (2) that it was not possible to combine requirements together to form more complex requirements such as ‘Requirement1 and Requirement2’. Then the idea came to use Boolean logic instead and, more precisely, a 4-value Boolean logic called ETL (Extended Temporal Language) (Bouskela and Jardin 2018). A new version of ReqSysPro was developed successfully (the only difficulty was the handling of time periods that needs dynamic allocation of memory) that was able to handle the temporal and condition aspects of FORM-L and to evaluate requirements to one of the four values {undefined, undecided, false, true}. To handle the other aspects (except the probabilistic aspects), it became necessary to have a FORM-L compiler. A prototype of a FORM-L compiler was developed by Inria and Sciworks on an EDF contract with a first operational version released in 2021. It demonstrated the feasibility of having a FORM-L to Modelica compiler. However, the compiler suffers from the drawback that it must be modified each time a new function is added to FORM-L. To alleviate this problem, the specification of a new language CRML was released at the end of 2021 within the ITEA project EMBRACE (2019 – 2022). The idea was to add the notion of functions (called operators in CRML) to be able to build complex functions from a limited number of elementary native functions as described in the following sections.

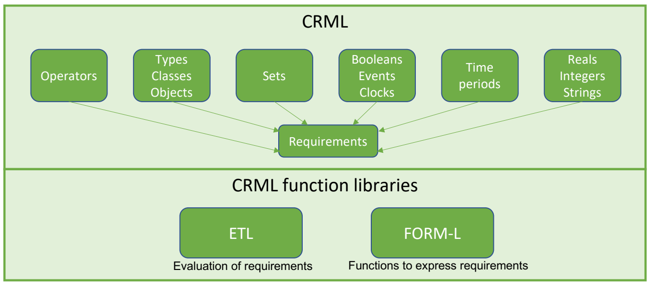


Figure 2. Architecture of the CRML language

3 The CRML Language

The language uses the concept of requirement made of four parts as introduced in the FORM-L language (Bouskela, Nguyen, and Jardin 2017; Nguyen 2019):

Spatial locator (WHERE): it defines the objects that are subject to the requirement. “Spatial” means that the objects are selected by some criteria on their properties that can be time-dependent.

Time locator (WHEN): it defines the time intervals when the requirements should be satisfied. A time interval is initiated when an event, called the opening event, occurs, and terminated when an event, called the closing event, occurs. An event occurs when a condition becomes true. A time locator can be composed of multiple time intervals that can overlap if several opening events occur before the closing event. In the following, the term ‘time period’ will be used as a synonym for ‘time locator’.

Condition to be fulfilled (WHAT): it is the condition to be verified by the objects selected with the spatial locator within the bounds of the time periods selected by the time locator.

Probabilistic constraint (HOW_WELL): it defines a probabilistic constraint on the condition to be fulfilled.

The general architecture of the language is given in Fig. 2. Time periods and probabilistic constraints constitute the novelty of the approach. They are required to handle realistic requirements, because realistic requirements cannot be satisfied anytime at any cost. Time periods define when requirements are in effect and the time delay to satisfy them. Probabilistic constraints define some tolerance for the system to fail complying with the requirements. These two aspects have profound technical and economic impact on the design and operation of the system.

Classes, objects, sets and operators allow one to define the system structure and the objects properties, and enable to express generic requirements on sets of objects selected by their properties that can depend on time. All definitions can be stored in libraries for further reuse. There are two operator libraries provided with the language: the FORM-L library that implements the aspects of the FORM-L language that are related to requirement modelling (Nguyen 2019), and the ETL library that implements the low-level functions for requirement evaluation (Bouskela and Jardin 2018).

In the following, we present the salient aspects of CRML through the following example of a requirement: "During operation, the system should always stay within its operating domain. However, if the system fails to stay within its operating domain, then it should not stay outside of its operating domain for more than ten minutes more than three times per year, with a probability of success of 99.99%." First, we use a *mathematical notation* to explain the semantics of requirements. Then, starting from Section 3.3, we introduce the *CRML syntax*. The full CRML specification can be found at `http://crml-standard.org`.

3.1 Clocks and Time Periods

To express formally the example, we first define two Boolean variables. The first one called $b1$ is true when the system is in operation and false otherwise. The second called $b2$ is true when the system operates within its authorized operational domain and false otherwise. They are both external variables, which means that their values are given by an external behavioral model.

```
external Boolean b1;
external Boolean b2;
```

We then define the time period while the system is in operation. The time period consists of possibly multiple time intervals that start when $b1$ becomes true and end when $b1$ becomes false. The event $b1$ becoming true is denoted by $b1\uparrow$. The event of $b1$ becoming false is denoted by $b1\downarrow$. Thus $b1\downarrow = (\neg b1)\uparrow$. There are actually two time periods of interest to express when the system is in operation:

$P1 = [b1\uparrow, b1\downarrow]$

$P2 = [b1\uparrow, b1\downarrow[$

$P1$ includes the opening and closing events, whereas $P2$ includes the opening event, but excludes the closing event. Note that there are in general several occurrences of $b1\uparrow$ so that $P1$ and $P2$ are composed of multiple time intervals. The set of occurrences of an event is called a clock. Therefore, $b1\uparrow$ and $b1\downarrow$ are clocks.

In general, time periods P are sets of time intervals Δ_i that can overlap. This is denoted by $P = \{\Delta_i\}_{1\leq i\leq n}$. In the sequel, the opening and closing events of a time interval Δ_i are resp. denoted $\Delta_i\uparrow$ and $\Delta_i\downarrow$, and the clocks of opening and closing events of a time period P are resp. denoted $P\uparrow$ and $P\downarrow$.

3.2 Requirements and 4-valued Boolean Logic

The following expression combines condition $b2$ with time period $P1$ to form requirement $R1$ which states that $b2$ should be true at any time instant along $P1$.

$R1 = \mathit{ensure}(b2 \otimes P1)$

$R1$ is a Boolean that is true when $R1$ is satisfied and false when it is not satisfied. The sign $\otimes$ denotes that condition $b2$ is combined with time period $P1$. The precise meaning of $\otimes$ and *ensure* will be given in the sequel.

$R2$ ensures that the number of failures of $R1$ should not exceed 3 over a sliding time period $P3$ of one year, that continuously shifts over the time period while the system is in operation. The failures of $R1$ corresponds to the events $R1\downarrow$. $P3$ is modelled as a time period composed of time intervals of one year that start at each occurrence of $R1\downarrow$. Note that $R1\downarrow$ cannot occur when the system is not in operation. The formal expression of $R2$ states that if a failure occurs at time t, there should not be more than two additional failures before $t+1$ year, and that this condition should be ensured at any time instant t while the system is in operation. Thus, time intervals are not continuously created, but only when they are needed to verify the requirement (i.e., when $R1$ fails).

$P3 = [R1\downarrow, R1\downarrow + 1\,\mathit{year}[$

$R2 = \mathit{ensure}(((\mathit{count}(R1\downarrow, P3) <= 3) \otimes P3) \otimes P2)$

$R3$ states that if $R1$ is not satisfied at time instant t, then $R1$ should be satisfied at $t+10$ mn, unless the end of $P1$ occurs before $t+10$ mn.

$R3 = \mathit{ensure}(((\neg R1 \wedge b1) \implies R1 \otimes [R1\downarrow, R1\downarrow + 10mn]) \otimes P1)$

Expressions are evaluated at each time instant t. $R3$ is active within $P1$. A new time interval is created at each occurrence of $R1\downarrow$ while $R3$ is active. The purpose of adding $\wedge b1$ to the precondition $\neg R1 \wedge b1$ is to avoid $R3$ being undecided if $P1$ ends before 10 mn after a failure of $R1$ (the meaning of undecided is given in the sequel).

$R4$ is the non-probabilistic version of the final requirement.

$R4 = R1 \wedge R2 \wedge R3$

$R5$ is the final requirement that corresponds to the probability of success of $R4$. The probability is evaluated at $b1\downarrow$ (when the system is stopped). The *checkAtEnd* function evaluates the probabilistic condition at the end of $P1$ (thus at $b1\downarrow$).

$R5 = \mathit{checkAtEnd}((\mathit{prob}(R4, b1\downarrow) > 99.99\%) \otimes P1)$

The reason why the satisfaction of $R5$ is evaluated at $b1\downarrow$, and not before, is because the probability of success of $R4$ is not defined before $b1\downarrow$.

We have so far only used logical functions with some elementary arithmetic to formally express the requirement. The question now is the mathematical type of a requirement and the meaning of $\otimes$. To clarify this, let us temporarily take a simpler example of a requirement R: "The project report must be completed before the end of the project". Before the start of the project, the requirement is *undefined*, which means that the requirement is not applicable. After the start of the project and before the end of the project, the requirement is *undecided* which means that the requirement is applicable, but its outcome is uncertain until either the report is completed before or at the end of the time period (thus before the deadline or just in time), in such case the requirement is *satisfied*, or not completed until the end of the time period, in such case the requirement is *not satisfied* (the report is late or canceled). Therefore, the requirement can take any value in the set $\mathbb{B} = \{true, false, undecided, undefined\}$.

The question now is whether variables of $\mathbb{B}$ comply with the usual Boolean algebra. If the answer to this question is positive, then requirements can be combined using the usual Boolean operators. It is very natural to assign true to the satisfaction of a requirement, and therefore false to the non-satisfaction of a requirement. To find the mathematical meaning of undecided and undefined, let us assume that R is a logical combination of two requirements $R1$ and $R2$: $R = R1\, op\, R2$, where op is a binary logical operation. $R1$ is undefined means that it is not applicable, and that its value should not affect the outcome of the logical operation. Therefore, undefined is a neutral element for any logical operation: $R = R1\, op\, R2 = R2$ for any $R2$ if $R1$ is undefined.

$R1$ is undecided means it is not known whether it is true or false. Therefore, if $R1$ is undecided then $R = R1 \wedge R2$ is false if $R2$ is false and undecided if $R2$ is true or undecided, and $R = R1 \vee R2$ is true if $R2$ is true and undecided if $R2$ is false or undecided. The same argument applies to find the value of $\neg R$: if R is undefined or undecided, then $R = \neg R$. Of course, the standard Boolean algebra applies if $R1$ and $R2$ are true or false.

It is then easy to verify that this algebra verifies the De Morgan's laws (such as $\neg R1 \wedge \neg R2 = \neg(R1 \vee R2)$) and all other Boolean laws (including $\neg(\neg R) = R$), except the complements law (because $R \wedge \neg R = false$ is not verified if R is undecided or undefined). Therefore, this 4-valued algebra can be considered as a Boolean algebra that accommodates all standard logical operators with the usual meaning (with some precaution regarding the complements law). Then the logical implication operator is defined as $R1 \implies R2 \equiv \neg R1 \vee R2$.

The main difference between the 4-valued and 2-valued Boolean algebras is that time is taken into account with the former: the requirement is undefined as long as the time period it is associated with has not begun, and undecided while inside the time period and before the decision can be made whether it is satisfied or not. This event is called the decision event. After the decision event, the requirement is true (satisfied) or false (not satisfied). Let us now assume that the completion of the project report is modeled by a Boolean C that tells whether the report has been signed or not. Then C is a 2-valued Boolean that takes the values true (the report is signed) or false (the report is not signed). Obviously, at a given time instant t, the value of the requirement R can be different from C, as R can take the additional values undefined and undecided. As explained before, the value of R results from a combination of C with the time period P that corresponds to the delay granted for the satisfaction of C. This is denoted $R = C \otimes P$. Therefore, mathematically speaking, a requirement is a function that associates a couple (C,P) to R, where C is the condition of the requirement, P is the time period of the requirement, and R is the value of the requirement. The definition domain of this function can be extended to conditions C that are 4-valued Booleans without any difficulty. It is then possible to formally express requirements on requirements, such as if requirement $R1$ fails, then requirement $R2$ should be satisfied within a given time delay $P : R = R1 \wedge (\neg R1 \implies R2 \otimes P)$. Fig. 3 summarizes how requirements are built.

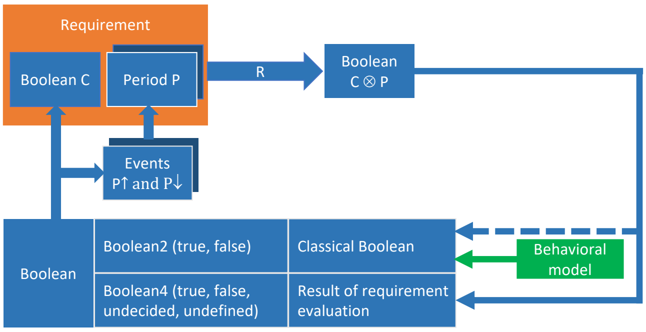


Figure 3. The requirement factory

We are now interested in giving a formal definition for the value of $C \otimes P$. Let us consider first a time period $P = \{\Delta_1\}$ composed of a single time interval Δ_1. The meaning of $R = C \otimes P$ is that the decision whether C is satisfied or not is made as soon as possible within Δ_1, i.e., at the decision event. The decision event for $C \otimes P$ is denoted $\delta(C,\Delta_1)$. From the decision event, if C is true or false, then $C \otimes P$ is true or false (it cannot be undefined or undecided) and keeps its value until the next decision event if any. For instance, if the condition C is that the number p of events $e\uparrow$ within Δ_1 should be less than a fixed integer n, i.e. if $C \equiv (p = count(e\uparrow,\Delta_1) \leq n)$, then $\delta(C,\Delta_1) = (p > n)\uparrow \vee \Delta_1\downarrow$ where $a\uparrow \vee b\uparrow$ denotes the clock containing the occurrences of events $a\uparrow$ and $b\uparrow$: $C \otimes P$ is false as soon as p is larger than n within Δ_1, otherwise $C \otimes P$ is true at the end of Δ_1. For the condition $C \equiv (p > n)$, the same decision event applies but the outcome is different: $C \otimes P$ is true as soon as p is larger than n within Δ_1, otherwise $C \otimes P$ is false at the end of Δ_1. Note that because the counter starts from $p = 0$, while $p \leq n$, C is false but $C \otimes P$ is undecided, thus in general $C \otimes P \neq C$. Note that when writing $(C,\Delta_1) = a\uparrow \vee b\uparrow$, there are in fact two decision events, $a\uparrow$ and $b\uparrow$. Unless it is something desirable, one must be careful that if $b\uparrow$ occurs after $a\uparrow$, the decision over $C \otimes P$ made at $a\uparrow$ is not reversed when $b\uparrow$ occurs. If the decision over C can be made at any time instant within Δ_1, then $\delta(C,\Delta_1) = (C \vee \neg C)\uparrow \vee \Delta_1\downarrow$. This is the case if C is the satisfaction of another requirement R' (i.e., if $C \equiv R'$): $C \otimes P$ is true or false as soon as C becomes true or false (if the decision over the satisfaction of a requirement is not reversed). If the decision over C cannot be made before the end of Δ_1, then $\delta(C,\Delta_1) = \Delta_1\downarrow$.

The function $checkAtEnd(C \otimes P)$ used in the example means that $\delta(C,\Delta_1) = \Delta_1\downarrow$. $ensure(C \otimes P)$ means that C must be true all along Δ_1, thus that it should never be false within Δ_1. This can be expressed as $ensure(C \otimes P) = (count(C\downarrow,\Delta_1) <= 0) \otimes P \wedge C \otimes [\Delta_1\uparrow,\Delta_1\uparrow]$. Within P, the value of the requirement is undecided until the condition is not verified, in such case it turns to false and stays false, or until the end of P if the condition is always satisfied, in such case it turns to true.

To evaluate $C \otimes P$ at time instant t, we consider two temporal Boolean operators:

- A filter $\times$ with the following properties: $a \times b = b$ if a is true, $a \times b = undecided$ otherwise.

- An accumulator $+$ with the following properties: $a + b = b + a$; $a + a = a$; $undefined + a = a$; $undecided + a = a$ if a is true or false; $true + false = false$;

The filter filters out all events that are not decision events. The accumulator computes the value of $C \otimes P$ at instant t taking into account the history of $C \otimes P$. It ensures that if C is false after the decision event, $C \otimes P$ will stay false whatever the future values of C. For a fixed integer i and $P = \Delta_i$, the value of $C \otimes P$ at t is

$$C \otimes \{\Delta_i\}(t) = undefined + \int_{\tau=\Delta_i\uparrow}^{min\{\Delta_i\downarrow, t\}} \delta(C, \Delta_i)(\tau) \times C(\tau) d\tau$$

The integral operator accumulates all values of the Boolean integrand for all time steps $d\tau$ within Δ_i until t. For t occurring before $\Delta_i \uparrow$, the value of $C \otimes \{\Delta_i\}$ is undefined. For t occurring after $\Delta_i \uparrow$, the value of $C \otimes \{\Delta_i\}$ is undecided until the integrand is true or false. This equation is generalized to a multiple time period $P = \{\Delta_i\}_{1 \leq i \leq n}$ by taking the logical conjunction of all $C \otimes \{\Delta_i\}$:

$$C \otimes P(t) = \bigwedge_{i=1}^{n} C \otimes \{\Delta_i\}(t)$$

This equation states that the condition must be satisfied for all time intervals of the time period. Then, provided that for any condition C and time period P, we know how to express $\delta(C, \Delta_i)$, we can evaluate $C \otimes P$, and consequently we can evaluate any temporal CRML expression. The operator $\otimes$ is expressed using elementary CRML temporal operators on 4-valued Booleans that are implemented in the ETL library using truth tables. This is why there is no built-in operator in CRML for $\otimes$, and also no built-in type for requirements. The operator $\otimes$ is used in high-level operators such as 'while' 'check count', 'while' 'ensure' or 'while' 'check at end' that express different kinds of decision events and constitute the FORM-L library. Such high-level operators are used in Section 3.5. It is possible to have user-defined types for requirements derived from the built-in type Boolean. Types are introduced in Section 3.3, and operators are introduced in Section 3.4. The following sections detail these constructs with the CRML syntax.

3.3 Types

User-defined types can be created from built-in types. For instance, the types Requirement and Assumption can be created from the type Boolean.

```
type Requirement is Boolean;
type Assumption is Boolean;
```

User-defined types can be used to define physical or monetary units, the syntax of which is not detailed here. They enable users to write expressions involving units such as

```
Pressure P is 1 bar + 1 mbar;
```

The value of P is computed in the unit system defined by the type Pressure. An error is raised if the unit is wrong or omitted.

3.4 Operators and Sets

Anything in CRML is a set or an element of a set. An element of a set has a fixed value, or a value returned by an operator. Therefore, the CRML syntax is entirely based on the notions of sets and functions. A CRML operator is a standard mathematical function. Two syntaxes are possible: the traditional mathematical syntax and the natural language syntax. In the natural language syntax, the names of user-defined operators are divided into snippets enclosed between singe quotes, and the input arguments are placed in front or after each snippet. Names of built-in operators or user-defined operators in the mathematical syntax are not enclosed within quotes. As an example, we define two operators using the natural language syntax: one that generates the set of occurrences of a Boolean becoming true (this set is a clock), and one that defines the logical implication operator (the keyword *Template* can be used when all input and output arguments are Booleans).

```
Operator [ Clock ] Boolean b 'becomes true'
     = Clock b;
Template b1 'implies' b2 = not b1 or b2;
```

The above operators are invoked as follows:

```
Clock c is b 'becomes true';
Boolean b3 is b1 'implies' b2;
```

A CRML set is a standard mathematical set. It is possible to apply unary operators to a set. This amounts to applying these operators to all elements of the set as follows:

```
Boolean {} b is { n1, n2, n3, n4 } < p;
```

is equivalent to

```
Boolean b {} is { n1 < p, n2 < p, n3 < p,
    n4 < p };
```

In this example, the unary operator is $x \mapsto x <$ p. It is also possible to apply a binary operator to a set:

```
Boolean b is and { n1 < p, n2 < p, n3 < p,
    n4 < p };
```

is equivalent to

```
Boolean b is n1 < p and n2 < p and n3 < p
    and n4 < p;
```

It is possible to construct subsets by filtering set elements depending on their properties, as defined for instance in classes of objects.

3.5 Classes and Objects

CRML is equipped with the notion of class in order to express requirements on complex objects, which can be physical subsystems like cooling or heating systems, physical components like vessels, pumps or heat exchangers, or abstract objects that capture generic notions. For the sake of the example, we define the abstract class Equipment that carries requirement R5, which is rewritten using the CRML syntax.

```
partial class Equipment is {
  external Boolean b1;
  external Boolean b2;
  Integer n is 3;                 // Default
      value
  Time dt is 10 mn;       // Default value
  Periods P1 is [b1 ’becomes true’, b1 ’
      becomes false’];
  Periods P2 is [b1 ’becomes true’, b1 ’
      becomes false’[;
  Periods P3 is [b1 ’becomes true’ or R1 ’
      becomes false’, 1 year[;
  Boolean R1 is ’while’ P1 ’ensure’ b2;
  Boolean R2 is ’while’ P2 ’ensure’
    (’while’ P3 ’check count’ (R1 ’becomes
        false’) ’<=’ n);
  Boolean R3 is ’while’ P1 ’ensure’
    ((not R1 and b1) ’implies’ (’while’ [R1
         ’becomes false’,
    R1 ’becomes false’ + dt] ’check’ R1));
  Boolean R4 is R1 and R2 and R3;
  Requirement R5 is ’while’ P1 ’check at
      end’
    ((’probability’ (R4) ’at’ b1 ’becomes
        false’) > 99.99%);
};
```

A partial class cannot be instantiated because it carries partial information that is not sufficient to instantiate objects. Class Equipment is partial because it is not possible to provide values for b1 and b2 without having some knowledge about the type of the equipment. However, requirement R5 will be automatically applied to all instances of the classes derived from Equipment that are not partial. Let us now create a new class Pump that derives from class Equipment. Class Pump will inherit all attributes of Equipment. It is however possible to redefine (or redeclare) within class Pump the attributes of Equipment that are not suitable to pumps. The operational domain for a pump corresponds to the non-cavitation of the pump: pumps should never cavitate while they are in operation. This can be enforced by redeclaring attribute n to be a constant integer equal to zero in the class Pump definition.

```
class Pump is {
  redeclare n constant Integer n is 0;
  redeclare R5 Requirement no_cavitation;
} extends Equipment;
```

In the above statement, requirement R5 is renamed to no_cavitation for better legibility. Then pumps can be instantiated with the statement

```
Pump pump is Pump ();
```

The following statement expresses that all pumps in the system should never cavitate, no matter the number or types of the pumps. The fact that class System extends Equipment means that requirement R5 is applicable to the system as a whole (provided that the values of b1 and b2 for the whole system can be obtained from a behavioral model).

```
class System is {
  Pump {} pumps;
  Requirement no_cavitation is and pumps.
      no_cavitation;
} extends Equipment;
```

In the above statement, the global no-cavitation requirement for all pumps in the system is obtained by taking the logical conjunction of the no-cavitation requirement for all individual pumps, that can be of different nature, and do not need to be known when this statement is written. Class extension and attribute redefinition can be used in combination to add and refine requirements in the course of system design. For instance, if during detailed design the chosen type for pumps is centrifugal, and if centrifugal pumps come with their own set of requirements, then a new class DetailedSystem can be derived from the class System that redefines the set of objects from class Pump to be a set of objects from a new class CentrifugalPump that extends class Pump and that carries the additional requirements for centrifugal pumps.

4 Implementation of the CRML toolchain

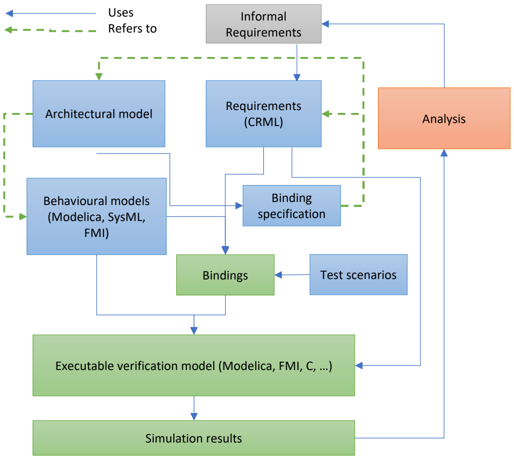


Figure 4. Architecture of the verification model

As illustrated in Fig. 4, the requirements are connected to the architectural and behavioural models through bindings. The compiler developed in the project translates CRML to Modelica. This also enables the use of Functional Mockup Itnerfaces (FMI) - a standard for exporting models for co-simulation supported by many tools. This means that any behavioural model that can be exported as a Functional Mockup Unit (FMU) can be connected to the requirements model.

4.1 CRML to Modelica Compiler

The compiler consists of two parts, first a grammar is specified using ANTLR and this is used to generate a parser for the compiler in Java. This grammar can also be used to generate parsers for compilers in other languages and can serve as a more formal specification of the language syntax.

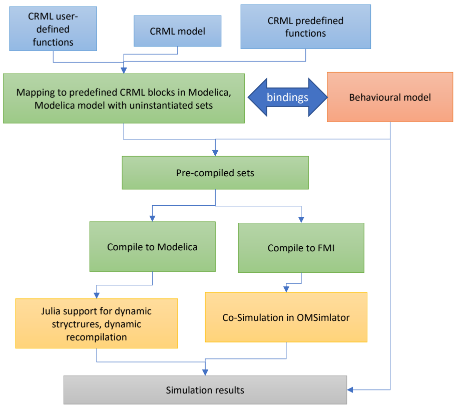

CRML user-defined functions
CRML model
CRML predefined functions
Mapping to predefined CRML blocks in Modelica, Modelica model with uninstantiated sets
bindings
Behavioural model
Pre-compiled sets
Compile to Modelica
Compile to FMI
Julia support for dynamic stryctrures, dynamic recompilation
Co-Simulation in OMSimlator
Simulation results

Figure 5. Architecture of the CRML to Modelica compiler

In a second step, the generated abstract syntax tree (AST) is then traversed to generate Modelica. Many elements of the language can be translated in a straightforward way, but in this section we will focus on the main CRML elements that require special consideration (Fig. 5).

4.1.1 4-valued Booleans

In Modelica booleans are two valued, therefore all references to booleans and boolean operators need to be converted to a special Modelica type Boolean4. To simplify, the definitions of this type and of the fundamental logical operators are given in a library and then calls to this library are directly generated (Fig. 5).

```
import CRML.ETL.Types.Boolean4;
model Bool1
  Boolean4 b0 = CRML.ETL.Types.Boolean4.
      true4;
  Boolean4 b1 = CRML.ETL.Types.Boolean4.
      false4;
  Boolean4 b2 = CRML.ETL.Types.Boolean4.
      undecided;
  Boolean4 b4 = CRML.Blocks.Logical4.and4(
      b1,b2);
end Bool1;
```

4.1.2 Time dependent functions, templates and operators

Another big difference in CRML and Modelica is that in CRML functions can be dependent on time while in Modelica they cannot. Therefore CRML functions need to be mapped to Modelica blocks. Built-in functions are mapped to the predefined blocks in the CRML library and user-defined Operators and Templates generate new Modelica blocks.

For example the user defined

```
Template R1 'xor' R2 = (R1 'or' R2) and not
     (R1 and R2);
```

is translated to the following Modelica snippet:

```
import CRML.ETL.Types.Boolean4;
model "xor"
  input Boolean4 R1, R2;
  output Boolean4 out;
  "or" "or0"(R1 = R1,R2 = R2);
equation
  out = CRML.Blocks.Logical4.and4(_or0.out,
        CRML.Blocks.Logical4.not4
          (CRML.Blocks.Logical4.and4(R1,R2)
              ));
end "xor";
```

And the call to the function is translated to an instantiation of the corresponding block and corresponding connectors. Quotation marks are used around operator names to distinguish between user defined keywords in CRML and Modelica keywords.

4.1.3 Sets

CRML is built around the concept of sets which are not present in Modelica. We distinguish between event sets that change in size throughout the execution of the program, and object sets that are either statically specified or calculated during the binding process.

Object sets are translated to arrays that are instantiated during the binding. This can either be done in a semi-automated manner or manually depending on whether the behavioural model is in Modelica or FMI.

Since the number of events can potentially be unlimited, event sets need to be mapped to dynamic structures, also known as Variable Structured Systems that increase in size as needed.

We have developed OpenModelica.jl (Tinnerholm et al. 2021), a modular and extensible Modelica compiler framework in Julia targeting ModelingToolkit.jl and supporting Variable Structured Systems. We extend the Modelica language with several new operators to support continuous time mode-switching and reconfiguration via recompilation at runtime. Our compiler supports the Modelica language as well as these aforementioned extensions.

A special type, a dynamic event array is defined that relies on the recompilation primitive to grow (or shrink the array size at runtime). This model is extended to store specific events.

```
partial model EventArray
  Event events[N];
  parameter Integer N=10; // size
  Integer i (start = 1); // index
equation
  when (i = N) then
    recompilation(N, N + 100);
  end when;
end EventArray;
```

4.2 Integrating CRML with Architectural and Behavioral Models

4.2.1 The Binding Mechanism

The most obvious way to bind variables is to fill in their full pathnames in a connect statement. However, this requires a large amount work from the user if a large number of external variables are involved. Also, if the models that provide the values are restructured, the pathnames must be changed accordingly, inducing work overhead. To simplify the work of binding, the idea is to prepare as much as possible binding at the class level. Manual binding is then done at the instance level by connecting together objects instead of variables. The final binding of individual variables is performed automatically from the information provided at the class and instance levels. As information provided at the class level can be reused for any model, the amount of manual work to be done for a given model is on average divided by the number of objects carrying the external variables. It is also easier to manually bind objects than variables because objects are much easier to spot than variables. Therefore, it always requires less effort to bind objects than variables, even in the worst case of having no more than one external variable per object.

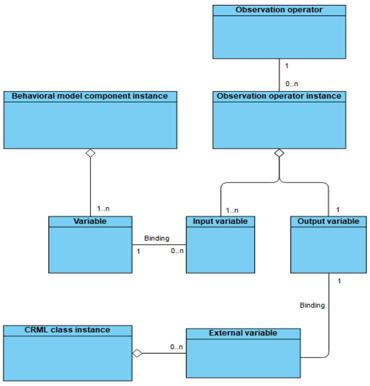


Figure 6. Binding output information

The output information to be produced to bind external variables of the requirement model to variables of the behavioral models is given in Fig. 6. There are two kinds of variable bindings: (1) the connections of the input variables of the observation operators instances to the variables of the behavioral models, and (2) the connections of the output variables of the observation operators instances to the external variables of the requirement model.

The information to be provided by the user in order to automate binding as much as possible is given in Fig. 7. It shows that the only information to be provided at the instance level is the correspondence between the objects in the requirement models and the objects in the behavioral or architectural models. The only assumption that makes this procedure possible is that the behavioral and architectural models are expressed using an object-oriented methodology, where objects are instances of classes. No assumption is made on the way connections between variables are expressed. Binding input and output information can be provided in a relational database. Connections between variables can be expressed using Modelica statements for white-box Modelica behavioral models, or FMI statements for black-box behavioral models equipped with an FMI interface.

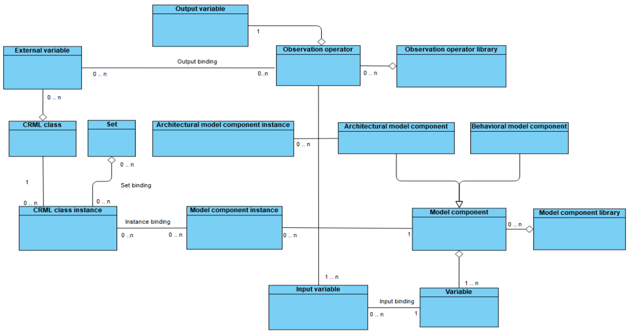


Figure 7. Automatic binding input information

4.2.2 Bindings to Behavioral Models

For Modelica behavioral models, the CRML model is automatically translated to a Modelica model by the CRML compiler. If the behavioral model is a Modelica model, then it is possible to have bindings using Modelica statements. Then the verification model (cf. Fig. 1) can be automatically generated as a Modelica model from the binding output information (cf. Fig. 6) (which itself can be automatically generated from the binding input information).

For black box models we can use the System Structure and Parametrization standard (SSP) to connect to models defined in other formalisms provided they can also be exported as FMUs as illustrated in Fig. 4. SSP connections can be generated automatically or manually based on binding specifications.

One difference with when generating bindings for FMUs is that there is no notion of classes in FMI, only simple types. Therefore it is impossible to automatically calculate sets of object of a certain type and they need to be specified manually.

4.2.3 Bindings to Architectural Models

If the architecture model is given in SysMLv2, bindings can also be defined in SysMLv2. The idea (shown below on a simplified example) is to use CRML as the constraint language to formalize requirements over the *attributes of the requirements element*. These attributes can be bound to attributes of the subject in specialized requirements, essentially defining the binding between variables of the architecture and of the requirement. A compiler can then generate observation operators and requirements models based on this information whenever the SysML model is translated to a computable model for verification (e.g. Modelica).

4.2.4 Execution of verification models

Having the requirements defined separately and bound through a technology-agnostic syntax, means that different approaches of verifying the requirements agains Modelica or FMU models are possible. For instance probabilistic requirements can be verified through Monte-Carlo simulation, ensuring for example that failure rates are below a certain threshold. Currently, test scenarios are generated by hand but in the future, generating test values based on the behavioural envelopes specified by the requirement models will also be investigated.

```
requirement def SpeedLimiterReq {
  attribute speed : SpeedValue;
  attribute limit : SpeedValue;
  attribute on : Boolean;
  require constraint { language "CRML" /*
      during on ensure speed < limit; */ }
}
requirement <r1> citySpeedLimiter :
    SpeedLimiterReq {
  subject vehicle : Vehicle;
  attribute :>> speed = vehicle.
      currentSpeed;
  attribute :>> limit = 50[km/h];
}
```

5 The Intermediate Cooling System Use Case

As explained in Section 1, the goal of the CRML initiative is to support the engineering of complex CPS that are very often over-constrained by a set of numerous (even conflicting) requirements. The idea is to rely on a shared evaluation toolset that helps stakeholders find the best trade-offs and foster innovative solutions. The CRML language and its underlying methodology make the traditional V-model fully executable and should act as an enabler for taking appropriate decisions at each step of engineering projects. Inspired from (Azzouzi et al. 2022), the methodology should be seen as an iterative approach whose main steps are summarized in Fig. 8.

Nuclear power plants are divided into approx. 200 subsystems. One of them is called the Intermediate Cooling System (ICS). The goal of this section is to illustrate how CRML can help justifying that the ICS properly fulfills its missions.

Step 0: Identification of System's Missions The ICS missions sum up to:

- **Evacuate the heat produced by a served systems** when they are in operation (served systems are auxiliary equipment such as the alternator or pumps);
- With the **use of demineralized water** (because water flowing directly from the cold source -sea or river-could damage equipment);
- At an **acceptable availability** rate (the plant must be shut down if the ICS is unavailable).

Step 1: Formalize the System's Environment and Its Interfaces To fulfill its missions, the ICS should physically interact with the different served systems to be cooled but also with (Fig. 4):

- A source of demineralized water (SED) to provide "clean" water to the ICS;
- A source of cold water (SEN) to cool the ICS itself (here a sea or a river);
- A drain sewer (SEK) in case of ICS leaks;
- A means to communicate with the plant operator (OP).

Although the ICS is not a large system, it involves 9 stakeholders coming with their own data and requirements.

For the engineer in charge of the ICS design, this makes his/her work even more challenging as it multiplies the information channels, the sources of potential changes during the project and the types of verification studies he/she needs to produce (i.e. one stakeholder being interested only by the achievements of its own goals). Formalizing the different constraints using CRML appears as a means to gather all these sources of data in a more rigorous and reproductible way than relying on a one-person expertise. It also enables the automation of verification tasks and hence provides more flexibility when input data changes and design studies must be rerun to assess the impact on the current solutions.

This formalization step should be performed for each system (or stakeholder) in interface with the ICS. For the sake of conciseness, let us focus on two different interfaces: the one between the ICS and the served system and the one between the ICS and the cold-water source.

Contract between the ICS and the served systems Each served system should have a physical interface with the ICS to be cooled by its refreshed demineralized water. In practice, there should be some physical means such as a valve at the inlet of a heat exchanger to "activate" cooling when necessary. In order to leave as many design options open as possible, we set the expectations on the served systems as follows: The cooling service should be provided as soon as the minimum temperature is reached (because too cold water could damage equipment). Requirements being common to all served systems, they are modelled in a generic class "Served_system" as follows:

```
class Served_system is {
  [...]
  // Mission: When the served system is
     operating, all the heat produced
     should instantly be evacuated by the
     ics
```

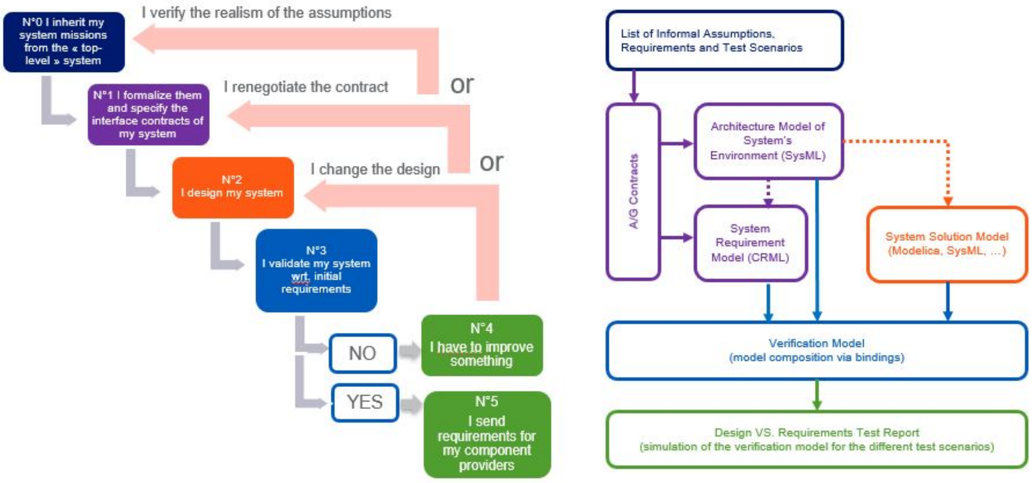

I verify the realism of the assumptions
N°0 I inherit my system missions from the « top-level » system
I renegotiate the contract
or
N°1 I formalize them and specify the interface contracts of my system
I change the design
or
N°2
I design my system
N°3
I validate my system wrt. initial requirements
NO
N°4
I have to improve something
YES
N°5
I send requirements for my component providers
List of Informal Assumptions, Requirements and Test Scenarios
A/G Contracts
Architecture Model of System's Environment (SysML)
System Requirement Model (CRML)
System Solution Model (Modelica, SysML, ...)
Verification Model (model composition via bindings)
Design VS. Requirements Test Report (simulation of the verification model for the different test scenarios)

Figure 8. The main steps of the CRML methodology (left) leading to a structured set of models (right)

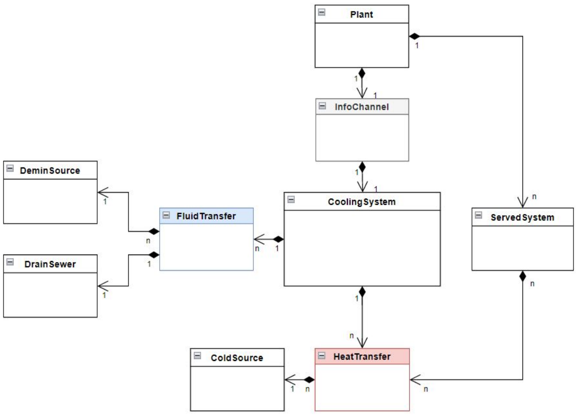


Figure 9. Architecture model of the ICS environment

```
constant Real epsilon is 0.001; //
    tolerance
Boolean cool_served_system_0 is
  during inOperation ensure 0 <= ics.W -
     W <= epsilon
// Requirement: The served system should
    not be supercooled by the ics (to
    avoid thermal stresses).
Boolean cool_served_system_1 is
  during ics.tRW <= (tRWMin - 0.1 Celsius
     ) ensure not open; // 0.1 Celsius
     is a design margin (tolerance)
// Requirement: The served system should
    be cooled by the ics as soon as the
    ics water temperature is above the
    minimum acceptable.
Boolean cool_served_system_2 is
  during ics.tRW >= tRWMin ensure open;};
```

"inOperation" is a Boolean indicating whether the served system is operating, "W" is the power of the served system, "tRW" is the ICS temperature, "tRWMin" is the minimum temperature acceptable by a served system, "open" is a Boolean stating whether the cooling service should be provided. These requirements are then automatically instantiated when the different served systems are themselves defined and parameterized.

Contract between the ICS and the cold-water source The ICS should have a physical interface with the cold source such as sea or river. To avoid environmental damages on the fauna and flora, regulations are enacted by a local authority regarding thermal effluents. Failure to comply with the regulations forces the plant to be shut down. In CRML, these requirements are modelled as follows:

```
class Cooling_system is {
 [...]
 // Requirement: When the ICS is operating,
     the temperature of the cold water
    source should
 // be below its acceptable maximum (no
    overheat of the sea or river).
 Boolean coldW_ics_1 is
   during not (state_stopped or
      state_stopping)
   ensure sen.tCW <= sen.tCWMax;
 // Requirement: When the ICS is operating,
     the temperature increase of the cold
    water source should
 // be below its acceptable maximum.
 Boolean coldW_ics_2 is
   during not (state_stopped or
      state_stopping)
   ensure tWW < (sen.tCW + sen.deltaTMax);
      };
```

"state_stopped" and "state_stopping" are Booleans stating whether the ICS is respectively stopped or in a stopping state, "tCW" is the temperature of the cold-water source, "tCWMax" is the maximum acceptable temperature of the cold-water source, "deltaTMax" is the maximum increase of temperature of the cold-water source.

Step 2: System Design After having identified and formalized the different requirements and constraints, the ICS engineer imagines some design alternatives. Fig. 10 represents one possible solution modeled in Modelica with the use of the open source ThermoSysPro[1] library.

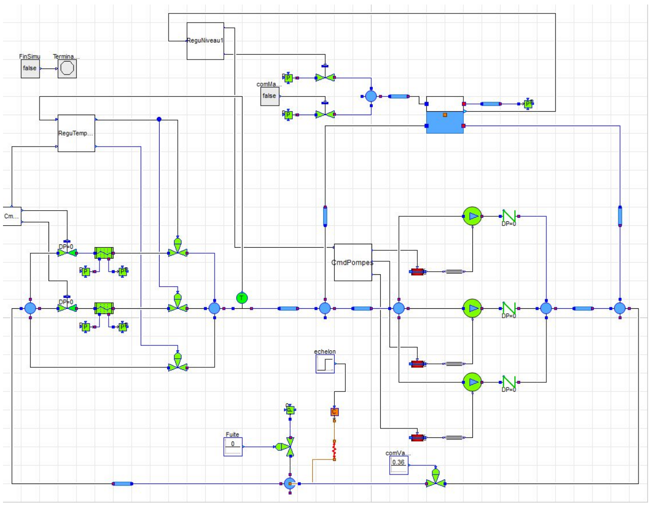


Figure 10. Behavioral model of the ICS

During this step, the ICS engineer could add some specific performance goals such as maintaining the circuit temperature around a setpoint (here 17 ° C) to optimize thermal transfer in heat exchangers. In CRML, this is expressed by adding the following requirement in the "CoolingSystem" class:

```
class Cooling_system is {
   [...]
   // Kpi: When operating, ics temperature
        should be around 17 Celsius
   constant Temperature tolerance is 0.5
        Celsius;
   Boolean kpi_1 is
      during inOperation
      ensure tRW >= (17 Celsius - tolerance)
           and tRW <= (17 Celsius + tolerance)
           ;
};
```

Step 3: System Verification Using the concept of bindings, the ICS engineer is then able to compose the different models to build a verification model. The purpose is to check by simulation whether the requirements are satisfied.

Fig.11 shows the simulation results obtained to monitor the 4-valued Boolean variable "kpi_1". The green curve is the evolution of "tRW" given by the behavioral model. "kpi_1" is computed by the CRML model. One can easily see that the "kpi_1" is not achieved (is false) as soon as "tRW" goes beyond its tolerance interval.

Although this seems like quite a simple performance target, it involves the dynamics of the system, so that possible violations of constraints are difficult to spot and interpret by visual inspection of continuous curves such as "tRW". The final verdict given by "kpi_1" alleviates this difficulty.

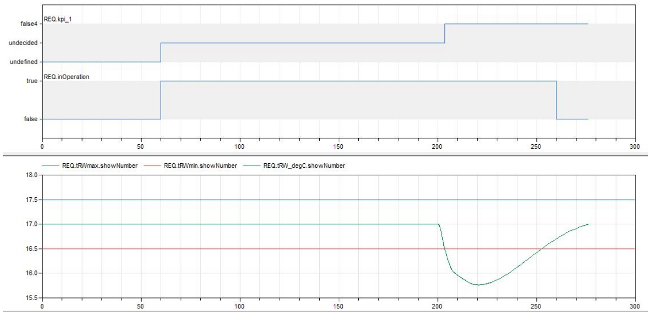


Figure 11. Simulation results of the verification model to monitor "kpi_1"

Step 4: Design Report Test reports can be customized. Below is an example that filters the requirements associated to the served systems only.

```
model GlobalReports is {
   // Conjunction of all requirements on
        Served_system
   Boolean globalReport_ForServedSystems is
        and flatten filter
     (flatten filter sriRequirements
        (type element == Served_system)) (
            type element == Boolean);};
```

6 Conclusion and Future Work

A new formal language called CRML (Common Requirement Modeling Language) for the modeling and simulation of requirements has been presented. The goal is to release CRML as an international standard interoperable with other standards such as SysML, Modelica, FMI and SSP. The purpose of CRML is to enable different stakeholders in different disciplines in charge of the design and operation of complex cyber-physical systems to reach a formal common agreement in terms of contracts made of formal constraints, so that they can successfully build, modify and operate such systems.

The salient innovative features of the language are its ability to capture all possible constraints on the real system, including real-time dependent constraints complemented with probabilities for failure, because no requirement can be fulfilled at any time at any cost (the lower the probability for failure, the higher the cost of the system). The language includes the possibility to structure the system using objects as instances of classes and build libraries of standard requirements that can be customized when used to express constraints on particular systems. A formal definition for the satisfaction of requirements has been given that uses a 4-valued Boolean algebra. This algebra provides the framework to combine requirements together to form high-level generic constraints that can be stored in libraries for further reuse.

Design can be verified against requirements by coupling, via so-called bindings, requirements models with behavioral models that capture the dynamic behavior of the physical system under study. To that end, a CRML to

[1] URL: https://www.thermosyspro.com/

Modelica compiler is being developed in the OpenModelica framework that automatically produces the executable verification models from the CRML models and the binding specifications (i.e., the way to associate the variables to be monitored in requirements to the variables that represent the states of the physical system).

Ways to use CRML within SysML v2 are being explored. It is believed that CRML will efficiently bridge the semantic gap between SysML and physical modeling and simulation, so that digital twins will be more efficiently used for the engineering and operation of cyber-physical systems. The reason is that CRML provides a formal way to translate functional concepts embedded in requirements models to the various concepts (such as physical concepts) embedded in behavioral models.

The way to use CRML for the design of a subsystem of a nuclear power plant has been presented. Future work will essentially bear on providing a graphical design methodology that will enable designers to specify systems without explicitly writing CRML code. Such graphical work will also improve the understandability of CRML by non-experts and empower the spread of CRML across different engineering teams.

Acknowledgements

This work has been supported by the ITEA3 EMBrACE project, as well as National Research, Development and Innovation Fund of Hungary, financed under the [2019-2.1.1-EUREKA-2019-00001] funding scheme.

References

Alur, Rajeev (1999). “Timed automata”. In: *International Conference on Computer Aided Verification*. Springer, pp. 8–22.

Azzouzi, Elmehdi et al. (2022). “A Model-Based Engineering Methodology for Stakeholders Coordination of Multienergy Cyber-Physical Systems”. In: *IEEE Systems Journal* 16.1, pp. 219–230. DOI: 10.1109/JSYST.2021.3071231.

Baier, Christel and Joost-Pieter Katoen (2008). *Principles of model checking*. MIT press.

Bouskela, Daniel and Audrey Jardin (2018). “ETL: A new temporal language for the verification of cyber-physical systems”. In: *2018 Annual IEEE International Systems Conference (SysCon)*, pp. 1–8. DOI: 10.1109/SYSCON.2018.8369502.

Bouskela, Daniel, Thuy Nguyen, and Audrey Jardin (2017). “Toward a rigorous approach for verifying cyber-physical systems against requirements”. In: *Canadian Journal of Electrical and Computer Engineering* 40.2, pp. 66–73.

Henzinger, Thomas A (2000). “The theory of hybrid automata”. In: *Verification of digital and hybrid systems*. Springer, pp. 265–292.

Kanso, Bilal and Safouan Taha (2013). “Temporal Constraint Support for OCL”. In: *Czarnecki K., Hedin G. (eds) Software Language Engineering. SLE 2012. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg.* 7745.

Mattsson, Sven Erik, Hilding Elmqvist, and Martin Otter (1998). “Physical system modeling with Modelica”. In: *Control Engineering Practice* 6.4, pp. 501–510.

Nguyen, Thuy (2014-03). “FORM-L: A MODELICA Extension for Properties Modelling Illustrated on a Practical Example”. In: *Proceedings of the 10th International Modelica Conference*. Linköping Electronic Conference Proceedings. Lund, Sweden: Modelica Association and Linköping University Electronic Press, pp. 1227–1236. DOI: 10.3384/ecp140961227.

Nguyen, Thuy (2019). “Formal Requirements and Constraints Modelling in FORM-L for the Engineering of Complex Socio-Technical Systems”. In: *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pp. 123–132. DOI: 10.1109/REW.2019.00027.

OMG (2017). *Unified Modeling Language (UML) version 2.5.1*. URL: https://www.omg.org/spec/UML/2.5.1/PDF (visited on 2023-05-13).

OMG (2019). *Precise Semantics of UML State Machines (PSSM)*. formal/19-05-01.

OMG (2021). *SysML Extension for Physical Interaction and Signal Flow Simulation (SysPhS)*. formal/21-05-03.

OMG (2023). *Systems Modeling Language (SysML) version 2*. URL: https://github.com/Systems-Modeling/SysML-v2-Release/blob/master/doc/2-OMG_Systems_Modeling_Language.pdf (visited on 2023-06-08).

Otter, Martin et al. (2015-09). “Formal Requirements Modeling for Simulation-Based Verification”. In: *Proceedings of the 11th International Modelica Conference*. Linköping Electronic Conference Proceedings. Versailles, France: Modelica Association and Linköping University Electronic Press, pp. 625–635. DOI: 10.3384/ecp15118625.

Tinnerholm, John et al. (2021-09). “OpenModelica.jl: A modular and extensible Modelica compiler framework in Julia targeting ModelingToolkit.jl”. In: *Proceedings of the 14th International Modelica Conference*. Linköping Electronic Conference Proceedings 181. Linköping, Sweden: Modelica Association and Linköping University Electronic Press, pp. 109–117. ISBN: 978-91-7929-027-6. DOI: 10.3384/ecp21181109.

Variable Structure System Simulation via Predefined Acausal Components

Andrea Neumayr[1] Martin Otter[1]

[1]German Aerospace Center (DLR), Institute of System Dynamics and Control (SR), Germany,
{andrea.neumayr,martin.otter}@dlr.de

Abstract

This article outlines a new approach of the experimental open-source modeling and simulation system Modia to simulate systems where the number of variables and equations can be changed after compilation and also during simulation, without having to re-generate and re-compile the code. Details are given for heat transfer in an insulated rod, where the discretisation of the rod is completely hidden from the symbolic engine. It is discussed how this approach could also be used in a future version of Modelica and/or FMI. Furthermore, this feature is also used in various variants to speed up collision handling in 3D mechanical systems. For example, by rigidly fixing an object after it has been gripped, with or without calculating the elastic response, and thereby dynamically changing the number of degrees of freedom.

Keywords: Modia, Julia, multibody, segmented simulation, heat transfer, collision handling

1 Introduction

Modia (Elmqvist et al. 2021) is an experimental, open source modeling and simulation system to develop new approaches to overcome the limitations of declarative, equation-based modeling languages such as Modelica (Modelica Association 2023). Modia is implemented with the powerful Julia programming language (Bezanson et al. 2017). It consists of a set of Julia packages, in particular of Modia.jl[1] for equation-based modeling à la Modelica and of Modia3D.jl[2] for modeling of multibody systems.

Neumayr and Otter (2023) extend Modia to process so called *predefined acausal components*[3]. These model components consist of a (usually small) set of Modia equations in which Julia functions are called that contain the core variables and equations of the components. These variables and equations can appear and disappear during simulation, without re-generation and re-compilation of code and without knowing in advance which model equations are utilized during such a simulation.

In contrast to this new approach, all previous proposals for systems with variable structure must either know in advance the entire models for all modes and switch between these models during simulation, (e.g., Mehlhase 2014; Mattsson, Otter, and Elmqvist 2015; Tinnerholm, Pop, and Sjölund 2022). Or the entire model is newly processed and code is re-generated and re-compiled (or interpreted) whenever the equation structure is changed[4], (e.g., Zimmer 2010; Tinnerholm, Pop, and Sjölund 2022).

In this article, the novel approach in Modia for modeling predefined acausal components is demonstrated with 1D heat transfer in a rod, where the number of discretization nodes can be changed before simulation start without re-compilation. It would also be possible to change the number of discretization nodes during simulation.

Modia3D is a more complex predefined acausal component. It was recently extended to cope with variable structure systems where the number of degrees of freedom can change during simulation, without re-compilation. The core part of this article discusses how this new feature is used to improve collision handling as an extension to the elastic response calculation introduced in (Neumayr and Otter 2019).

Elmqvist et al. (2021, Section 2) describe the Modia Language, and Neumayr and Otter (2023, Appendix A) provide a short overview of it. A more detailed explanation is available in the Modia Tutorial[5].

2 Predefined Acausal Components

Neumayr and Otter (2023) introduce predefined acausal components which are based on a proposal of Elmqvist (2022): The equations of an acausal component are split into causal and acausal partitions. The intuition is that the causal partition is always evalu-

[1] https://github.com/ModiaSim/Modia.jl, v0.12.0, visited on 2023-06-13

[2] https://github.com/ModiaSim/Modia3D.jl, v0.12.0, visited on 2023-06-13

[3] Neumayr and Otter (2023) refers to these components as acausal built-in components. We decided to rename them to predefined acausal components to be more descriptive.

[4] Generated compiled code maybe cached.

[5] https://modiasim.github.io/Modia.jl/stable/, visited on 2023-06-13

DOI
10.3384/ecp204511

ated in the same order, regardless of how the component is connected with other components. This partition is sorted, explicitly solved for the unknowns, and implemented with one or more functions. In contrast, sorting and solving of the acausal partition depends on the actual connection of the component. This partition is kept as a set of equations. Note that, the figures and some text fragments used below in this section are from Neumayr and Otter (2023)

In Neumayr and Otter (2023), various variants of this basic approach are discussed. In particular, (a) the variables computed in the causal partitions can either be still visible in the equation part as proposed by Elmqvist (2022) or (b) a large part of these variables is hidden in the functions and do no longer appear in the equation part. Variant (a) has the advantage, that index reduction is still possible by differentiating the functions of the causal partitions. Variant (b) has the advantage that the causal partition, in particular the number of its variables and equations, can be changed after compilation and during simulation. Variant (b) has the drawback that index reduction is no longer possible for the causal partitions. Index reduction in the acausal partitions is still possible and is sufficient in many practical cases. However, it is not possible to use a predefined, acausal component as an inverse model if implemented with variant (b). In Modia and Modia3D variant (b) is used.

In Figure 1 the communication structure between the solver, the sorted and solved equations and the functions[6] of the causal partitions are shown: The variables of the solver (state vector $\boldsymbol{x}$ and the vector of event indicators $\boldsymbol{z}$) are split into an invariant and a variant part: $\boldsymbol{x} = (\boldsymbol{x}^{\text{inv}}, \boldsymbol{x}^{\text{var}})$ and $\boldsymbol{z} = (\boldsymbol{z}^{\text{inv}}, \boldsymbol{z}^{\text{var}})$. The dimensions of the invariant parts are fixed before the simulation starts. The dimensions of the variant parts, which are contained in the functions of the causal partitions, can change at events during simulation. Since $\boldsymbol{x}^{\text{var}}, \boldsymbol{z}^{\text{var}}$ are communicated directly between the functions and the solver, the symbolic processing of the equation part of a model is not affected by these variables. Therefore, these variables can in principle be changed at event times - variables can be added or can disappear.

This basic approach is demonstrated using the predefined acausal component of Figure 2, which models heat transfer in a rod with an insulated surface. On the left and right sides of the rod, thermal connectors a, b are present (called `port_a, port_b` in Listing 1) with potential variables a_T, b_T (temperatures) and flow variables $a_{Q_{\text{flow}}}, b_{Q_{\text{flow}}}$ (heat flow rates). The partial differential equation, which mathematically describes the heat transfer in one dimension is discretized in space by volumes $V_i = \Delta x \cdot A$ of equal lengths Δx and identical areas A. In the center of volume i, a temperature T_i is defined, leading to a temperature vector $\boldsymbol{T} = [T_1, T_2, \dots, T_n]$.

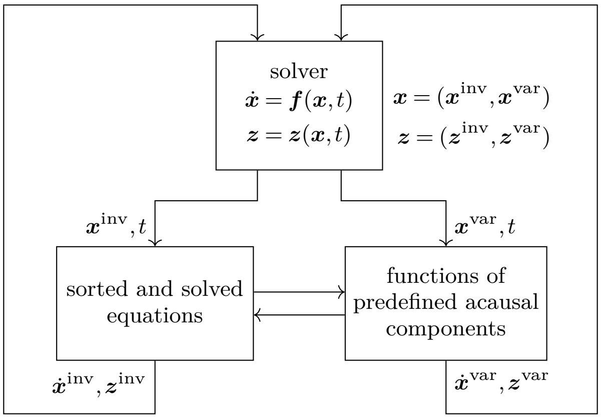
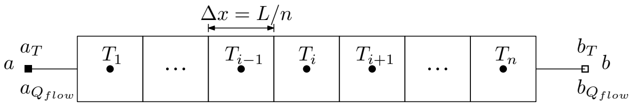

solver
$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, t)$
$\boldsymbol{z} = \boldsymbol{z}(\boldsymbol{x}, t)$
$\boldsymbol{x} = (\boldsymbol{x}^{\text{inv}}, \boldsymbol{x}^{\text{var}})$
$\boldsymbol{z} = (\boldsymbol{z}^{\text{inv}}, \boldsymbol{z}^{\text{var}})$
$\boldsymbol{x}^{\text{inv}}, t$
$\boldsymbol{x}^{\text{var}}, t$
sorted and solved equations
functions of predefined acausal components
$\dot{\boldsymbol{x}}^{\text{inv}}, \boldsymbol{z}^{\text{inv}}$
$\dot{\boldsymbol{x}}^{\text{var}}, \boldsymbol{z}^{\text{var}}$

Figure 1. Communication between the solver, the sorted and solved equations, and the functions of the predefined acausal components. The state vector $\boldsymbol{x}$ and the event indicators $\boldsymbol{z}$ are split into an invariant and a variant part: $\boldsymbol{x} = (\boldsymbol{x}^{\text{inv}}, \boldsymbol{x}^{\text{var}})$, $\boldsymbol{z} = (\boldsymbol{z}^{\text{inv}}, \boldsymbol{z}^{\text{var}})$. The variant parts consist of the states defined and used in the causal partitions of all predefined acausal components. The dimensions of the invariant parts are fixed before simulation starts. The dimensions of the variant parts can change at events during simulation.



$$
Q_{\text{flow},i} = \lambda \frac{A}{\Delta x} \begin{cases} 2(a_T - T_1) & i = 0 \\ T_i - T_{i+1} & i = 1, \dots, n-1 \\ 2(T_i - b_T) & i = n \end{cases}
$$

$$
\begin{aligned}
a_{Q_{\text{flow}}} &= Q_{\text{flow},0} \\
b_{Q_{\text{flow}}} &= -Q_{\text{flow},n} \\
\varrho c A \Delta x \dot{T}_i &= Q_{\text{flow},i-1} - Q_{\text{flow},i} \quad i = 1, \dots, n \\
T_i(t = t_0) &= T_0
\end{aligned}
$$

Figure 2. Space discretized partial differential equation of one-dimensional heat transfer in a rod with an insulated surface. It is defined with parameters L (length of rod), n (number of volumes), A (area), ϱ (density), c (specific heat capacity), λ (thermal conductivity), T_0 (initial value in each volume), states T_i (temperatures in the center of each volume), thermal connectors a, b with potential variables a_T, b_T (temperatures), and flow variables $a_{Q_{\text{flow}}}, b_{Q_{\text{flow}}}$ (heat flow rates).

Listing 1. Simple usage of insulated rod `InsulatedRod2` with one-dimensional heat-transfer. On the left side it is connected with a fixed temperature source `FixedHeatFlow` with $T = 220\,°\text{C} = 493.15\,\text{K}$, and on the right side with a fixed heat flow source `FixedHeatFlow` with $Q_{\text{flow}} = 0$.

```
using Modia
include(
```

[6] These functions have a memory and are therefore no mathematical functions.

```
  "$(Modia.path)/models/HeatTransfer.jl")

# Temp. source - rod - heat flow source
HeatedRod = Model(
  # temperature source
  fixedT = FixedTemperature |
    Map(T=493.15),

  # heat flow source (Q_flow=0)
  fixedQflow = FixedHeatFlow,

  # insulated rod with 5 volumes
  rod = InsulatedRod2 |
    Map(L=1.0, T0=273.15, nT=5),

  # connecting the components
  equations = :[
    connect(fixedT.port, rod.port_a),
    connect(rod.port_b, fixedQflow.port)]
)

# generate and compile Julia code
heatedRod = @instantiateModel(HeatedRod)

# change to 8 volumes and simulate model
simulate!(heatedRod, stopTime = 1e5,
  merge=Map(rod = Map(n=8))

# plot temperatures
plot(heatedRod,
  ("fixedT.port.T", "rod.T"))
```

In Listing 1, a Modia model is shown with a predefined acausal component `InsulatedRod2` of the rod[7]. Its left thermal connector `port_a` has a fixed temperature source `FixedTemperature`. Its right thermal connector `port_b` has a fixed heat-flow source `FixedHeatFlow` with the default zero heat-flow rate. This means that, the rod is completely insulated on the right side and has a fixed temperature on the left side. Note that, `A|B` merges model or parameters `B` with model `A`. Command `@instantiateModel(HeatedRod)` symbolically processes this model and generates Julia code that is translated to executable code. The `simulate!` statement changes the discretization, and thus the dimension of the temperature vector $\boldsymbol{T}$, from 5 to 8 volumes before simulation starts without a new translation. The plot of Figure 3 is generated with `plot(heatedRod, ...)`, displaying the temperatures at the temperature source and in the rod volumes.

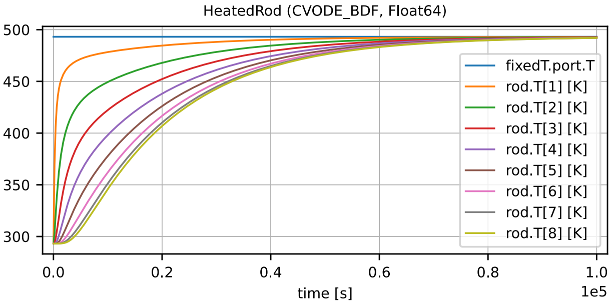


Figure 3. Plot of temperatures of heated rod model.

Listing 2. Modia definition of `InsulatedRod2` model.

```
include("HeatTransfer/InsulatedRod2.jl")

InsulatedRod2 = Model(;
  # Called once before symb. processing
  _buildFunction= Par(functionName =
    :(buildInsulatedRod2!)),

  # Called once before new sim. segment
  _initSegmentFunction=Par(functionName=
    :(initSegmentInsulatedRod2!)),

  # Parameters
  L      = 1.0,
  A      = 0.0004,
  rho    = 7500.0,
  lambda = 74.0,
  c      = 450.0,
  T0     = 293.15,
  nT     = 1,

  # Connectors
  port_a   = HeatPort,
  port_b   = HeatPort
)
```

The implementation of the `InsulatedRod2` model is shown in Listing 2. To start with, file `InsulatedRod2.jl` is included containing the definition of a Julia struct holding the data and the local variables of the component, as well as some Julia functions. More details are given below. A standard Modia definition of the model is then given defining the parameters and connectors of the component. Contrary, to a standard Modia component, no equations are present. For simplicity, no units are used in this model and its associated functions. However, the actual implementation of the component in Modia supports units. The component is a predefined acausal component model, since the following special parameters are provided at the beginning of the model, defining functions to be used for symbolic processing and during simulation:

- `_buildFunction`: Before symbolic processing begins, the hierarchical dictionary of the root model to be compiled is traversed and function `buildInsulatedRod2`, defined from `_buildFunction`, is executed for each sub-model it contains. This function (a) defines additional model variables and equations that are merged with the corresponding model and (b) returns an instance of the Julia structure, which acts as the internal memory of the component.
- `_initSegmentFunction`: This function is called by the simulation engine before the root model is initialized and at each `FullRestart` event before

[7] This model can be found in Modia, v0.12.0, models/HeatTransfer.jl.

the root model is re-initialized at a new simulation segment. In both cases, all local variables of the predefined acausal component model (including states and zero-crossing functions) must be redefined, as well as initial values for newly defined states.

Listing 3. `_buildFunction` function definition.

```
# Called once before symb. processing
function buildInsulatedRod2!(model,..,ID)
  model = model | Model(
    # Instance of an internal struct
    insRod  = Var(hideResult=true),

    # Dummy return argument
    success = Var(hideResult=true),

    equations = :[
      # copy states into insRod
      insRod = openInsulatedRod!(
        instantiatedModel, $ID)

      # equations at the boundaries
      port_a.Q_flow = getGe2(insRod)*
        (port_a.T - getT1(insRod))
      port_b.Q_flow = getGe2(insRod)*
        (port_b.T - getTend(insRod))

      # compute der(T)
      success =
        computeInsulatedRodDerivatives!(
        instantiatedModel, insRod,
        port_a.T, port_b.T)
    ]
  )
  return (model, InsulatedRodStruct())
end
```

In Listing 3, the implementation of function `buildInsulatedRod2` is shown. In this function, the model instance (of the actual `InsulatedRod2` component) is merged with additional model definitions consisting of two variables and several equations. It returns the merged model. Additionally, the internal memory of the component is instantiated with `InsulatedRodStruct()` and is also returned by `buildInsulatedRod2!`. This internal memory is later identified by the unique identifier `ID`, which is specified in the function call. Function call `openInsulatedRod!` in the equation section copies the rod temperatures $\boldsymbol{T}$ from the state vector of the simulation engine into the `InsulatedRodStruct` memory and returns a reference to it as `insRod`. The argument list of this function call includes the unique identification `ID` of the predefined acausal component. It is provided when `buildInsulatedRod2!` is called. `$ID` is inside an Abstract Syntax Tree, due to `:[...]` and `$` inserts the actual (literal) value at this place. In Julia terminology this is called "interpolation". The `insRod` reference is then used in subsequent function calls, for example, to retrieve the value of the first temperature node with `getT1(insRod)`. This value is used in an equation to calculate the heat flow from `port_a` to the first internal node. Finally, `computeInsulatedRodDerivatives!` computes the derivatives of the temperatures and copies them into the state derivative vector of the simulation engine. As can be seen, the equation section is independent from the number of discretization elements `nT`. Therefore, the number of these discretization elements can be changed without re-generation and re-compilation.

Listing 4. `_initSegmentFunction` definition.

```
# Called once before new sim. segment
function initSegmentInsulatedRod2!(
        m, path, ID, parameters)
  insRod=get_instantiatedSubmodel(m,ID)

  if isFirstInitialOfAllSegments(m)
    initFirstSegmentInsulatedRod2!(
      insRod; parameters...)
  end

  # Define new states and state derivat.
  insRod.T_startIndex =
    new_x_segmented_variable!(m,
      path*".T", path*".der(T)",
      insRod.T, "K")
  return nothing
end
```

The implementation of the `_initSegmentFunction` is shown in Listing 4. This function is called before a new simulation segment is initialized. The first statement inquires the reference `insRod` of the internal memory of the component. Before the first simulation segment, the (evaluated) `parameters` are stored in `insRod`. Furthermore, some dependent parameters are computed and also stored in this memory. Finally, `new_x_segmented_variable` is called to define the name of a new state, its derivative and its unit together with the initial value of `insRod.T` which is the current value of vector `T`. It is initialized with parameter `T_init` in `initFirstSegmentInsulatedRod2!`. Note that, even if the `InsulatedRod2` component always uses the same definition, the states must be newly defined for each new simulation segment.

Listing 5. Function to compute state derivatives

```
# Inquire values from InsulatedRodStruct
getT1(  insRod) = insRod.T[1]
getTend(insRod) = insRod.T[end]
getGe2( insRod) = insRod.Ge2
                  # = 2*lambda*A/dx

# Compute and copy state derivatives
function computeInsulatedRodDerivatives!(
    m, insRod, Ta, Tb)
  T = insRod.T
  k = insRod.k   # = lambda/(c*rho*dx*dx)
  for i in 1:length(T)
    insRod.der_T[i] =
      k*(T_grad1(T,Ta,i) -
      T_grad2(T,Tb,i))
  end
  copy_der_x_segmented_value_to_state(m,
    insRod.T_startIndex, insRod.der_T)
  return true
end
```

In Listing 5, the most important remaining functions are shown. The state derivatives are computed and copied to the state derivative vector of the simulation engine with `computeInsulatedRodDerivatives!`.

Listing 6. Sketch to implement InsulatedRod model as Modelica ExternalObject.

```
class InsulatedRodObject
  extends ExternalObject;

  function constructor
    input Real L, A, rho, lambda, d, T0;
    input Integer nT;
    output InsulatedRodObject insRod;
    external "C" insRod =
      openInsulatedRod(L,A,rho,lambda,
        d,T0,nT);
  end constructor;

  function destructor
    input InsulatedRodObject insRod;
    external "C"
      closeInsulatedRod(insRod);
    end destructor ;
end InsulatedRodObject;

model InsulatedRod
  import H=Modelica.Thermal.HeatTransfer;
  parameter Real    L;
  parameter Real    A;
  parameter Real    rho;
  parameter Real    lambda;
  parameter Real    T0;
  parameter Integer nT=1;

  H.Interfaces.HeatPort_a port_a;
  H.Interfaces.HeatPort_b port_b;
protected
  InsulatedRodObject insRod =
    InsulatedRodObject(
       L,A,rho,lmbda,T0,nT);
  Boolean success;
equation
  // equations at the boundaries
  port_a.Q_flow = getGe2(insRod)*
    (port_a.T-getT1(insRod));
  port_b.Q_flow = getGe2(insRod)*
    (port_b.T-getTend(insRod));

  // compute der(T)
  success =
    computeInsulatedRodDerivatives(
      insRod,port_a.T,port_b.T)
end InsulatedRod;
```

Note that, a similar approach could be implemented in Modelica with reasonable effort: The simplest implementation would be to use External Objects and add additional utility functions (Modelica Association 2023, Section 12.9.6–12.9.7). These are equivalent to the utility functions of Modia, e.g., to add variables at events. These utility functions would directly communicate with the underlying simulation engine. If they are available, the Modia example of the insulated rod could be implemented as outlined in Listing 6. The main benefit would be that the number of temperature nodes can be changed after translation and that the Modelica model consists essentially of three scalar equations. These equations are independent from the number of temperature nodes. The drawback is that functions `openInsulatedRod`, `closeInsulatedRod`, `getGe2`, `getT1`, `getTend`, `computeInsulatedRodDerivatives` need to be implemented in C. Note that these would be simple C-functions. For example, `computeInsulatedRodDerivatives` could be implemented as shown in Listing 7.

Listing 7. C-function to compute state derivatives

```
int computeInsulatedRodDerivatives(
    struct M *m, struct InsRod *insRod,
    double Ta, double Tb) {
  double* T   = insRod->T;
  double* der_T = insRod->der_T;
  double  k   = insRod->k;
  int     nT = insRod->nT;
  double  k1 = insRod->k1;

  der_T[1] =
    k1*(2*(Ta-T[1])-(T[1]-T[2]));
  der_T[nT] =
    k1*(T[nT-1]-T[nT]-2*(T[nT]-Tb));
  for (i=2; i < nT-1; ++i) {
    der_T[i] =
      k1*(T[i+1]-T[i]-(T[i]-T[i-1]));
  }
  copy_der_x_segmented_value_to_state(m,
    insRod->T_startIndex, der_T);
  return 0;
}
```

The eFMI standard (Functional Mockup Interface for embedded systems, (Lenord et al. 2021)) defines an intermediate language GALEC to transform acausal models to production code. GALEC is basically a very small subset of the Modelica language with some extensions as needed for embedded systems. The extension also includes a simple form of member functions. If such member functions were supported in Modelica, the implementation of predefined acausal components could be done completely in the Modelica language and without External Objects or C-code.

Modelica models can be exported in FMI format (Modelica Association 2022). This includes Modelica models with External Objects. The FMI standard communicates the values of variables explicitly with setter and getter function calls. In principle, it would be possible to add another variable type to the FMI standard, where internal variables (including states) are communicated directly to the solver and no longer via the setter/getter function calls. If such an enhancement were available, the causal partition part of the Modelica model of Listing 6 could be transformed to an FMU, where the node temperatures would no longer be communicated via the FMI setter/getter functions and would no longer be visible in the environment in which the FMU is used. This would have the great advantage that the number of variables and equations can be changed during the simulation.

3 Segmented Simulation and Collision Handling

Modia3D is a multibody tool for 3D mechanical systems implemented as a predefined acausal component of Modia according to section 2. Modia3D is targeted for solvers with adaptive step size control to compute results close to real physics including collision handling using the Minkowski Portal Refinement (MPR) algorithm (Snethen 2008; Neumayr and Otter 2017) and collision response for elastic contacts (Hertz 1896; Flores et al. 2011; Neumayr and Otter 2019). Modia3D has a very flexible and modular design pattern. It is extended (since v0.12.0) to cope with variable structure systems where the number of degrees of freedom (DoF) can change during simulation, without re-compilation.

Modia3D offers two kinds of joints: The first kind of joints contains Modia equation sections with invariant variables, including invariant states. These invariant elements are visible for Modia and cannot be removed or added during simulation. The interface to the Modia3D functionality is designed to define differential equations only on the Modia side in Modia equation sections, so that state constraints can be defined and index reduction can be performed on invariant states. The joints of the second kind define variant variables, including variant states, which are visible only in the Modia3D predefined acausal component. These joints can be added or removed during simulation. For example, an Object3D has an optional keyword `fixedToParent` with a default value of true. In this case, the Object3D is rigidly connected to its parent Object3D. This means it has zero degrees of freedom. If the value is set to false, the Object3D is allowed to move freely with respect to its parent, meaning it has 6 degrees of freedom and 12 variant states. At events, keyword `fixedToParent` can be changed from false to true and vice versa. Neumayr and Otter (2023, Table 2) define Modia3D actions which modify the second (variant) kind of joints and trigger structural changes during simulation, e.g., `actionAttach`, `actionReleaseAndAttach`, `actionRelease`, `actionDelete`. The new states (joints) added during simulation with e.g., `actionRelease` are initialized based on the last known position, velocity, acceleration and rotation. All remaining states are re-initialized with their last known values. Based on that, the internal 3D structure is rebuilt and executed until another action for a structural change is triggered. This restructuring is performed with dynamic data structures and is extremely fast (< 1 ms).

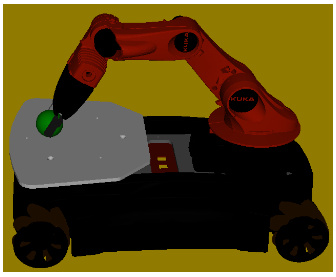


Figure 4. YouBot gripping or releasing a sphere on a plate.

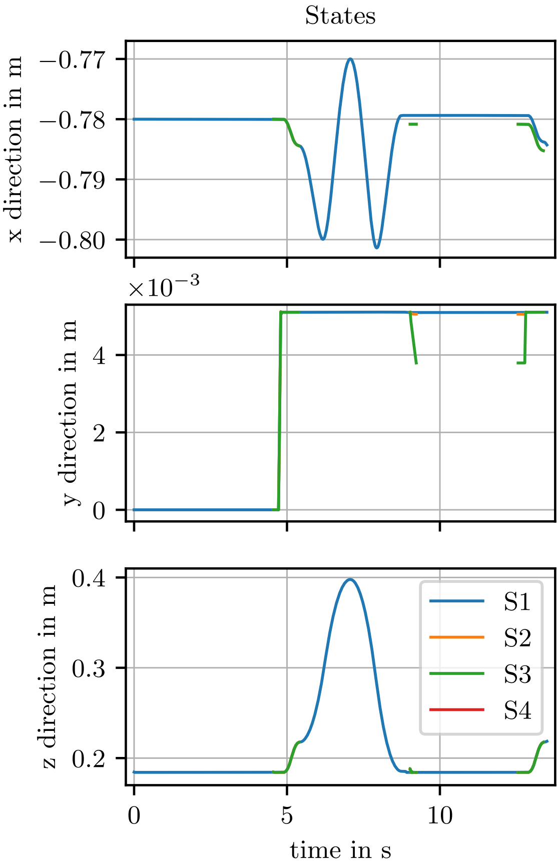


Figure 5. States of the sphere. They are equivalent to the translation of the sphere center in x, y, z direction with respect to its parent. If the sphere is freely moving, world is its parent. States can only be displayed if they are present. If the sphere is rigidly attached to the plate or gripper, there are no states, and nothing is displayed. The sphere in Scenario 4 (S4) has no states. Therefore, nothing is displayed. The states for scenarios 2 and 3 (S2, S3) are available and are displayed if the sphere is freely moving. For the absolute position of the sphere center see Figure 6.

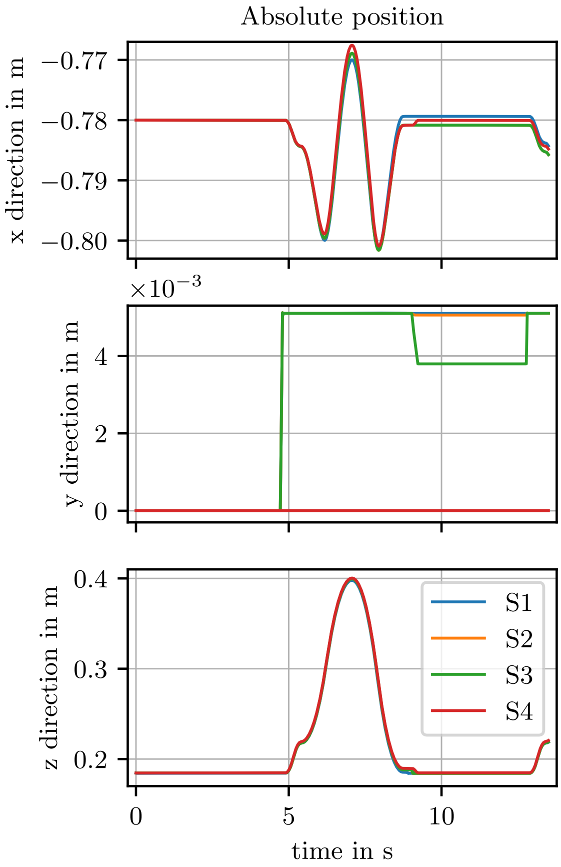


Figure 6. Absolute position of sphere center for the 4 scenarios.

Table 1. Mean $\bar{x}$ and standard deviation s of the simulation time of all four scenarios (S1–S4) each for $n = 12$ runs on a standard notebook[8].

	$\bar{x}$	s
S1	7.816 s	0.123 s
S2	7.255 s	0.075 s
S3	6.863 s	0.388 s
S4	0.397 s	0.016 s

In this section, several combinations of segmented simulation and collision handling are discussed using a KUKA YouBot robot gripping and transporting a sphere, see Figure 4. This robot has a 5 DoF arm and was manufactured in the years 2010–2016. Elmqvist et al. (2021) model the drive trains and controllers of the robot in Modia, and the 3D mechanics with Modia3D.

Four variants of the following transportation scenario are simulated. In all these scenarios, the robot follows the same trajectory. Initially, the cargo, e.g., a sphere, rests on a plate. It is gripped by the robot's gripper and transported upwards until it is placed down again, where it rests on the plate until it is gripped again. The states of the freely moving sphere (see Figure 5), if available, and the absolute position of the sphere center (see Figure 6) are displayed. The simulation times of all four scenarios are compared in Table 1.

Scenario 1 (S1)[9]: The transportation scenario is modeled with collision handling, compare (Neumayr and Otter 2023, Scenario 2(b)). This means, the sphere collides with the plate, as well as with the fingers of the gripper.

Scenario 2 (S2)[10]: The transportation scenario is modeled with segmented simulation and collision handling, see Listing 8. DoFs are added or removed during simulation: At the beginning, the sphere is rigidly attached to the plate. Shortly before the gripper reaches the sphere, the sphere is released (+6 DoF) and collides with the plate. Shortly afterwards it collides with the gripper. After approximately one second, the sphere is rigidly attached to the gripper (-6 DoF). Until the gripper is again close to the plate to release the sphere (+6 DoF), which collides with the plate. Collision handling remains on even if the sphere is rigidly connected to the gripper or plate, as collisions with other bodies can still occur.

Listing 8. Robot program of Scenario 2. Collision handling is enabled by default. It can be turned off or on again in all action commands with `enableContactDetection`. Scenario 3 is defined, by setting this flag to false, as indicated in the comment lines.

```
function robotProgram(actions)
  addReferencePath(actions, ...)

  # 1. attach sphere to plate, -6 DoF
  ActionAttach(actions, "sphereLock",
    "robot.base.plateLock",
    # enableContactDetection = false)

  # 2. some movement of robot
  ptpJointSpace(actions, [
    # open gripper + move to top
    # open gripper + move to plate  ])

  # 3. release sphere off plate, +6 DoF
  # it collides with plate and gripper
  ActionRelease(actions, "sphereLock")

  # 4. gripping via collision handling
  ptpJointSpace(actions, [
    # grip
    # grip + transport a bit        ])

  # 5. attach sphere to gripper, -6 DoF
  ActionAttach(actions, "sphereLock",
    "robot.gripper.gripperLock",
    # enableContactDetection = false)

  # 6. some movement of robot with sphere
```

[8] Intel(R) Core(TM) i7-9850H CPU @ 2.6 GHz, RAM 32 GB

[9] This model can be found in Modia3D, v0.12.2, test/Robot/-ScenarioCollisionOnly.jl.

[10] This model can be found in Modia3D, v0.12.2, test/Segmented/ScenarioSegmentedCollisionOn.jl.

```
  ptpJointSpace(actions, [
    # grip + move to top
    # grip + transport
    # grip + move near to plate
    # open gripper                  ])

  # 7. release sphere off gripper, +6 DoF
  # it collides with plate
  ActionRelease(actions, "sphereLock")

  # 8. some movement of robot
  ptpJointSpace(actions, [
    # open gripper + move to plate  ])

  # repeat step 1. - 8.
  ...
end
```

Scenario 3 (S3)[11]: The transportation scenario is modeled with segmented simulation and collision handling. Scenario 3 is very similar to Scenario 2, except that collision handling is disabled when the sphere is rigidly connected to the gripper or plate, since in this scenario it is already known that no further collisions will occur. This is deactivated with `enableContactDetection = false` in Listing 8. Basically, this means that the distance calculations between each collision pair is switched off in these phases.

Scenario 4 (S4)[12]: The transportation scenario is modeled with segmented simulation only, compare (Neumayr and Otter 2023, Scenario 2(a)) and Listing 9. Collision handling is switched off for this scenario. This means, the sphere is rigidly attached to the plate, when resting, and rigidly attached to the gripper during transportation. Each time the sphere is rigidly connected to the plate or gripper, the segment is re-initialized. Since the relative velocity and angular velocity between the sphere and the gripper is zero, when the sphere is attached to the gripper or attached to the plate, the physics is correctly modeled under the idealized assumption that gripping time is infinitely small. Basically, this means that gripping effects are neglected.

Listing 9. Robot program of Scenario 4.

```
function robotProgram(actions)
  addReferencePath(actions, ...)

  # 1. attach sphere to plate
  ActionAttach(actions, "sphereLock",
    "robot.base.plateLock")

  # 2. some movement of robot
  ptpJointSpace(actions, [
    # open gripper + move to top
    # open gripper + move to plate
    # grip                          ])

  # 3. attach sphere to gripper
  ActionAttach(actions, "sphereLock",
    "robot.gripper.gripperLock")

  # 4. some movement of robot
  ptpJointSpace(actions, [
    # grip + transport a bit
    # grip + move to top
    # grip + transport
    # grip + move near to plate
    # open gripper                  ])

  # 5. release sphere off gripper
  # attach it to plate
  ActionReleaseAndAttach(actions,
    "sphereLock", "robot.base.plateLock")

  # repeat step 2. - 5.
  ...
end
```

The simulation time of Scenario 4 is about 19 times less than that of Scenario 1. This is because Scenario 4 (segmented simulation only) is basically a non-stiff system where the solver can use large step sizes. In addition, the time for reconfiguration of the multibody system, for gripping and releasing, is negligible. Fine-tuning of collision handling during transportation of the gripped freight is no longer required. Furthermore, any type of cargo can be transported, regardless of its shape. The disadvantage is that the details of the gripping are not modeled, but this can be important.

Scenario 1 (collision handling only) is a stiff system because the gripper holds the sphere by elastic contact and friction forces, which change during transport. Therefore, the solver must use much smaller step sizes. One limitation of collision handling with the MPR algorithm is that it only supports point contacts. If the cargo would be a box, see (Neumayr and Otter 2023, Scenario 3(b)), it would not be possible to calculate a unique point contact that is continuous over time, for example, because one box and one gripper face or one box and one plate face are parallel to each other during contact. All these considerations lead to a compromise in modeling the gripping and releasing of the cargo with collision handling, and otherwise rigidly attaching the sphere to the plate or gripper, resulting in Scenario 2 and Scenario 3.

There is not such a big difference in simulation time for Scenarios 1,2,3, see Table 1. In all three cases, the calculation of the elastic contact response is the limiting factor. This effect is modeled in all these cases. In more realistic scenarios, the approach of Scenario 2 or 3 may pay off, if the number of collision phases is small relative to the remaining actions.

4 Conclusion

The novel approach of variable structure systems with Modia/Modia3D seems to be very promising. De-

[11] This model can be found in Modia3D, v0.12.2, test/Segmented/ScenarioSegmentedCollisionOff.jl.

[12] This model can be found in Modia3D, v0.12.2, test/Segmented/ScenarioSegmentedOnly.jl.

pending of the predefined acausal component and the application one can design (extend) specific actions to trigger new segments and re-initialize the model. In this paper, existing Modia3D actions are extended by enabling or disabling collision handling during simulation, which speeds up the simulation and allows to model form locked fixing of cargos. Furthermore, an example was sketched of how the Modelica language and the FMI standard could be enhanced to allow the number of variables and equations to be changed during simulation.

References

Bezanson, Jeff et al. (2017). "Julia: A fresh approach to numerical computing". In: *SIAM review* 59.1, pp. 65–98. DOI: 10.1137/141000671.

Elmqvist, Hilding (2022). *Slides 7-10 of Modia – A Prototyping Platform for Next Generation Modeling And Simulation Based on Julia. Jubilee Symposium 2019: Future Directions of System Modeling and Simulation.* URL: https://modelica.github.io/Symposium2019/slides/jubilee-symposium-2019-slides-elmqvist.pdf (visited on 2022-12-04).

Elmqvist, Hilding et al. (2021). "Modia - Equation Based Modeling and Domain Specific Algorithms". In: *14th International Modelica Conference*, pp. 73–86. DOI: 10.3384/ecp2118173.

Flores, Paulo et al. (2011). "On the continuous contact force models for soft materials in multibody dynamics". In: *Multibody system dynamics* 25.3, pp. 357–375. DOI: 10.1007/s11044-010-9237-4.

Hertz, Heinrich (1896). *On the contact of solids - On the contact of rigid elastic solids and on hardness. In Miscellaneous papers, MacMillan, 1896, pp. 146–183.* https://archive.org/details/cu31924012500306, accessed on 2023-01-13.

Lenord, Oliver et al. (2021). "eFMI: An open standard for physical models in embedded software". In: *14th International Modelica Conference.* DOI: 10.3384/ecp2118157.

Mattsson, Sven Erik, Martin Otter, and Hilding Elmqvist (2015). "Multi-mode DAE systems with varying index". In: *11th International Modelica Conference*, pp. 89–98. DOI: 10.3384/ecp1511889.

Mehlhase, Alexandra (2014). "A Python framework to create and simulate models with variable structure in common simulation environments". In: *Mathematical and Computer Modelling of Dynamical Systems* 20.6, pp. 566–583. DOI: 10.1080/13873954.2013.861854.

Modelica Association (2022). *Functional Mock-up Interface Specification - Version 3.0.* https://fmi-standard.org/docs/3.0/.

Modelica Association (2023). *Modelica – A Unified Object-Oriented Language for Systems Modeling, Language Specification, Version 3.6.* https://specification.modelica.org/maint/3.6/MLS.pdf, accessed on 2023-06-10.

Neumayr, Andrea and Martin Otter (2017). "Collision Handling with Variable-step Integrators". In: *8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools.* EOOLT'17. ACM, pp. 9–18. DOI: 10.1145/3158191.3158193.

Neumayr, Andrea and Martin Otter (2019). "Collision Handling with Elastic Response Calculation and Zero-Crossing Functions". In: *9th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools.* EOOLT'19. ACM, pp. 57–65. DOI: 10.1145/3365984.3365986.

Neumayr, Andrea and Martin Otter (2023). "Modelling and Simulation of Physical Systems with Dynamically Changing Degrees of Freedom". In: *Electronics* 12.3. ISSN: 2079-9292. DOI: 10.3390/electronics12030500.

Snethen, Gary (2008). "Xenocollide: Complex collision made simple". In: *Game Programming Gems 7.* Course Technology. Charles River Media, pp. 165–178. ISBN: 978-1-58450-527-3.

Tinnerholm, John, Adrian Pop, and Martin Sjölund (2022). "A Modular, Extensible, and Modelica-Standard-Compliant OpenModelica Compiler Framework in Julia Supporting Structural Variability". In: *Electronics* 11.11, p. 1772. ISSN: 2079-9292. DOI: 10.3390/electronics11111772.

Zimmer, Dirk (2010). "Equation-based modeling of variable-structure systems". PhD thesis. ETH Zurich. DOI: 10.3929/ethz-a-006053740.

MoCITempGen: Modelica Continuous Integration Template Generator

David Jansen Fabian Wüllhorst Sven Hinrichs Dirk Müller

Institute for Energy Efficient Buildings and Indoor Climate, E.ON Energy Research Center, RWTH Aachen University, Germany, {david.jansen, fabian.wuellhorst, sven.hinrichs, dmueller}@eonerc.rwth-aachen.de

Abstract

Modelica enables an object-oriented approach to model complex systems in product development and research, and, thus, the development of various model libraries. Library development requires collaborative development in a team of multiple developers. A typical challenge in collaborative development, especially in the area of open source, is to create models of uniform quality despite different levels of knowledge among developers. Techniques, such as Continuous Integration (CI) from the field of software development, can help to solve these challenges. However, the adaptation of CI for the area of Modelica model development currently requires the manual creation of complex templates and a high degree of manual configuration. In this paper we present *MoCITempGen*, an open source tool for automated generation of CI structures for Modelica. The tool is succesfully applied on two Modelica libraries to demonstrate its functionality.

Keywords: Continuous Integration, Modelica testing

1 Introduction

With the progressive use of modeling and simulation, both in research and in product development, ensuring the model quality is becoming increasingly important. As the complexity of systems continues to increase, so does the number of model developers contributing to libraries and individual models. The concept of open source can partially address this problem, as the increased reach enables cross-institutional and cross-company collaboration. However, this creates communities of sometimes very different levels of knowledge with regard to modeling, which further complicates compliance with uniform quality criteria. In the university context, where both research assistants and students work together on models, this effect is partially amplified. Continuous Integration (CI) is a technique from software development, more precisely from *DevOps*, that was first described by Grady Booch (Booch 1991) and later defined as one of the 12 principles of extreme programming (Beck 2000). CI processes in the context of modeling aim to assure the quality of models. These processes include testing of the produced code/models in a designated environment. Vasilescu et al. reviewed 246 GitHub projects that use CI, finding that the use of CI increases the quality of repositories in terms of a higher number of reviewed, merged and rejected pull requests (Vasilescu et al. 2015).

As Modelica models and packages are stored as ASCII-files, the usage of *git* platforms like *GitHub* or *GitLab* is strongly recommended when developing Modelica libraries anyway (Gall et al. 2021). Thus, the application of CI to Modelica code is not new. For instance, the well-known modelica-buildings (Wetter et al. 2014) library has an extensive CI structure based on the *BuildingsPy* Python library (Wetter 2019). Over the past few years, we too have built a comprehensive CI structure for the *AixLib* library that partially reuses functions from *BuildingsPy* and combines them with self developed functionality (Maier et al. 2023). However, these approaches are tailored to the corresponding libraries and their application to other Modelica libraries is not straightforward. To overcome this issues, we developed *MoCITempGen*, an open source tool that allows to generate a complete Modelica CI structure based on a few inputs with various testing stages and functions. With the release of the tool, we want to lower the hurdle of applying CI structures to Modelica libraries and thus increase the quality of open source Modelica modeling projects in the long run.

Before describing *MoCITempGen* in Section 3, we review existing CI solutions in Section 2. To demonstrate the usage of *MoCITempGen*, we apply it in Section 4 to the open-source libraries *AixLib* and *BESMod*. Subsequently, we provide a critical discussion about the limitations of the created CI-structure in section 5.

2 State of the Art

Following, we give an overview of common CI hosts and infrastructures (2.1), and show which approaches and solutions for applying CI to Modelica already exist (2.2).

2.1 Common CI hosts

In order to take advantage of CI, an appropriate infrastructure must be used. In the following, we provide a brief overview about three often used systems, how they can be deployed, and what costs they generate.

Travis-CI[1] offers a standalone CI/CD service, that can be

[1] `https://www.travis-ci.com/`

DOI
10.3384/ecp204521

connected to multiple platforms like GitHub. In the end of 2020, Travis-CI stopped offering free CI-minutes and is now only available via priced plans. Even if Travis-CI is completely open source, there is currently no option to self-host the service, so using it will always generate costs.

GitHub Actions[2] is a service that was launched in 2018. It is directly integrated into GitHub and offers 2000 free minutes of usage per month. There is also the option to add a self-hosted runner to a repository.

GitLab-CI/CD[3] was first released in 2012 and is the inbuilt CI/CD feature of GitLab. GitLab itself can be self-hosted without costs, and the GitLab runner can be self-hosted as well. If self-hosting is not an option for the application, multiple paid plans for GitLab and its runners exist.

With all self-hosted variants, the costs for providing the respective hardware must, of course, be taken into account.

2.2 Available CI for Modelica

Following, we want to give a short overview about uses of CI in context of Modelica and the tools that are developed around it.

Rabuzin et al. introduced a CI workflow for testing Modelica models via OpenModelica and Travis-CI for their power system library OpenIPSL (Rabuzin, Baudette, and Vanfretti 2017). Their workflow includes a checking stage that checks the compliance with the Modelica syntax and a model validation stage that runs the most current model implementation against existing simulation results of the model to verify that results have not changed.

Schoelzel et al. implemented a fine-grained unit and regression test setup to solve the problem of reproducibility in the context of Modelica models for biology using a CI pipeline based on GitHub Actions(Schölzel et al. 2021).

Hugues et al. perform not only testing but also integrate Continuous Deployment (CD) in their pipelines to create digital twins in form of Modelica functional mock ups (FMU) for cyber-physical systems in their project *TwinOps* (Hugues et al. 2022).

The Buildings[4] library uses an extensive approach with a combination of *Travis* CI and *GitHub Actions* (Wetter 2019). GitHub Actions is used to run different scripts which check HTML syntax, model order, experiment setup, existing documentation and much more. As the job's runtime is short, developers get direct feedback on their contributions. *Travis* CI is used to run regression tests for different software (Dymola, OpenModelica, and Optimica). Furthermore, library specific developments such as the control description language or spawn of energy plus are tested. Some of these scripts are used in the IBPSA[5] and IDEAS as well. The regression results have to be generated manually.

The Open Source Modelica Consortium (OSMC) built an open source pipeline to test all Modelica libraries included in their package manager[6]. For this purpose, all models that have an experiment stop time are simulated with OpenModelica and the different process steps are documented and published in an HTML report.

Furthermore, the Modelica Standard Library also uses CI processes to ensure the quality of the models.

Additionally, there are several tools that can be used in a Modelica CI environment. These include, among others, the BuildingsPy python package[7] library, the modelica formating tool modelica-fmt[8], the ModelicaPy python package[9], and the regression test package MoPyRegtest[10]. Although these tools and their applications already demonstrate a certain potential of CI in the context of model development, their use is often limited to the respective libraries. Even if some approaches can be reused, this often means a manual adaptation to the own library. From our point of view, this creates a gap that can be filled with the development of tools that generate CI structures adapted to one's own library on the basis of a few input parameters. Therefore, we present this new tool, which we call *MoCITempGen*, in the following section.

3 Methodology

In the following sections, we first provide a nomenclature about common terms in the context of CI and Modelica for better understanding (section 3.1). After that, we provide a brief explanation about the general structure of CI setup (section 3.2) and then describe the template creation process (section 3.3). Finally, in section 3.4 we describe how the developed CI processes work in detail.

3.1 CI-Nomenclature

Before we describe the template generation and the functionality of the templates, we want to give a short definition of the common terms for CI in the context of Modelica for better understanding. Since *MoCITempGen* is currently based on GitLab-CI, we use the names and terms based on the definition of GitLab[11]. However, most functionalities and terms of CI pipelines are similar across the different services. Table 1 provides an overview of the most important terms for applying CI to Modelica.

Figure 1 shows the CI-setup based on *MoCITempGen*[12] that can be adapted to any Modelica library. The *MoCITempGen* repository holds the template generation

[2] https://github.com/actions

[3] https://docs.gitlab.com/ee/ci/

[4] https://github.com/lbl-srg/modelica-buildings

[5] https://github.com/ibpsa/modelica-ibpsa

[6] https://github.com/OpenModelica/OpenModelicaLibraryTesting

[7] https://github.com/lbl-srg/BuildingsPy

[8] https://github.com/modelica-tools/modelica-fmt

[9] https://github.com/ORNL-Modelica/ModelicaPy/

[10] https://github.com/modelica-tools/MoPyRegtest

[11] https://docs.gitlab.com/ee/ci/pipelines/

[12] https://github.com/RWTH-EBC/MoCITempGen

Table 1. Nomenclature for CI and Modelica terms.

Term	Explanation
`job`	Smallest part that holds definition of what to do
`script`	Section in a `job` definition with actual commands to execute in a `job`
`stage`	Bundles `jobs` for specific use case (e.g. testing) and defines the order to execute them
`pipeline`	Top-level definition of a CI workflow
`runner`	Receives the jobs `jobs` from the CI structure and executes them
`variables`	Central definition of values to use throughout `stages` and `jobs`
`artifacts`	Store results of `jobs` throughout a `pipeline` and after the `pipeline` finished
`rules`	Define if a job should be executed or not
`extends`	Reuse a `job` definition and only overwrite certain parts of it
`include`	Command that allows to reuse the definition of a `job` in different `pipelines`
`library`	Collection of reusable modeling components, such as models, classes and functions
`package`	Hierarchical grouping of related components within a Modelica `library`
`whitelist`	List of models that should be excluded from specific `stages`

tool that needs to be cloned locally and copied to the directory of the users Modelica library once for setup. Subsequently, the template generation will be performed. The generation process itself will be explained in more detail later.

3.2 Setup description

The resulting library related CI-structure of templates can then either be placed directly inside the target Modelica library (dashed lines) or placed outside in a separate repository using the `include` command of GitLab-CI. The way to store the templates mostly influences how easy updates of the templates can be deployed. Storing the templates in a separate repository is recommended for repositories with intensive development, multiple developers and therefore multiple branches. In such repositories, a deployment or update of the CI templates would otherwise require a merge of the updated templates back into every branch, because every branch has its own version of the templates. For smaller repositories with a small amount of developers, storing the templates directly with the Modelica code is acceptable as well.
The GitLab runner executes the stages and jobs defined in the library related CI-structure using the *ModelicaPyCI* package we released. This package holds the functionalities which will be described in the following sections.

If the target Modelica Library repository is hosted in GitLab, the GitLab runner directly interacts with the main repository and the process is completed. However, any other Git provider, like GitHub, BitBucket or AWS CodeCommit can be used by taking advantage of the GitLab mirroring feature. This way, the target Modelica library is mirrored into a separate GitLab repository (dotted objects). For GitHub and BitBucket it is also possible to display the pipeline status in the target Modelica library repository.

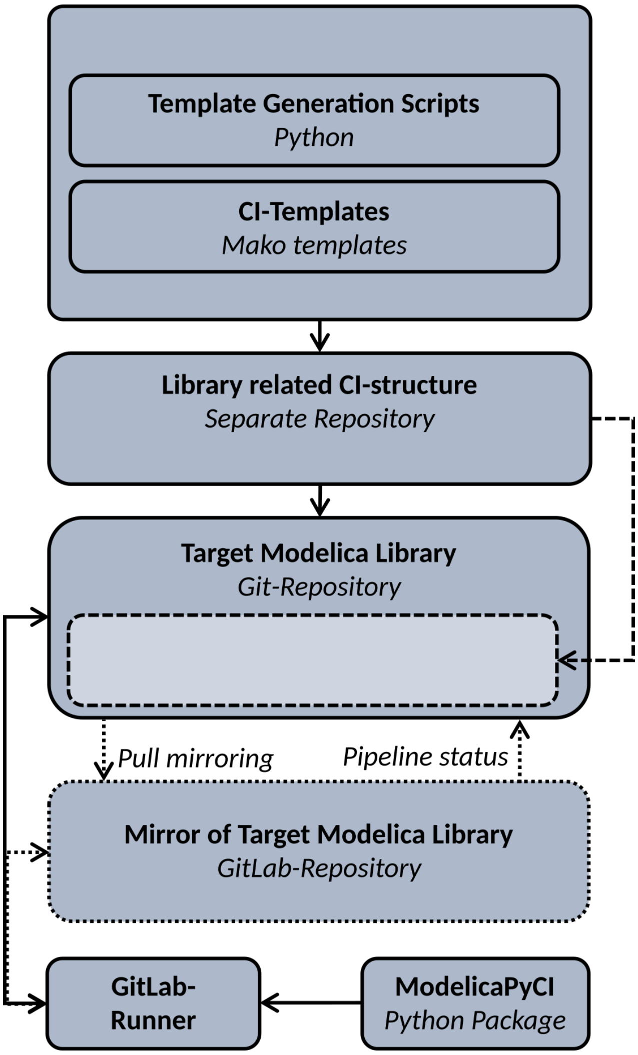


Figure 1. CI setup, scattered objects are optional, dotted objects are only needed if not using GitLab as main repository.

3.3 Templates Generation Process

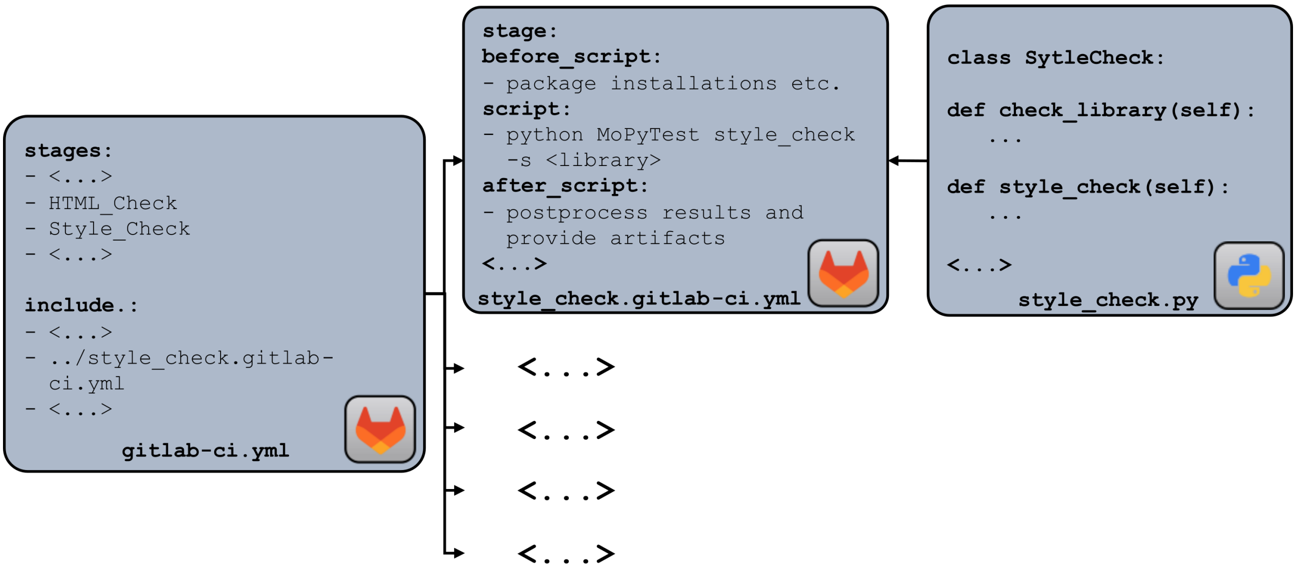
The template generation is performed via Python using the Python templating package Mako[13]. This allows the dynamic creation of templates based on the respective Modelica library. The generation process needs information about the repository, whether a mirroring process is required, whether certain models should be excluded from the CI process, what steps and tasks should be performed, and whether and how to include manual interaction with the CI.
To get this information, the setup process is possible in two ways. The first option is an interactive way, where the setup process uses command line interaction and checks the file structure and existing Modelica packages based on the library structure first and creates the setup subsequently based on user inputs. The second option is a configuration based way, where the user fills out a `.toml` configuration file in advance. The latter is more suited to apply adjustments to an already generated CI-structure as the interactive process creates the `.toml` configuration file for later adjustments.

3.4 Template Structure and Functionality

Figure 2 shows a simplified structure of the templates generated by *MoCITempGen*. The created template structure consists of single template file for each stage, which are combined in a top level `gitlab-ci.yml` file. Some jobs are performed separately for every Modelica package of the library, which can lead to redundant and duplicated jobs and statements in the CI-structure. These redundancies can be reduced by using a combination of Python to create the GitLab templates and the `extends` command in GitLab-CI/CD. Basic jobs, such as the Modelica check of a package, only have to be defined once as a base job in the Mako templates. Using for-loops, variables and the extends command, when exporting the GitLab templates, the individual jobs are then created for each package and adjusted by variables for the respective package.
Using the artifact functionality, the results and outputs of the various tasks and phases can be made available for download or later publication to provide a more structured view.
The complete CI-structure, if all stages are selected, is shown in Figure 3. The script section of each job is comparably short, as the functionality itself is outsourced into an additional Python library that currently comes with the template generation repository.
Since not all phases should be executed in all scenarios, we use the implementation of `rules` in GitLab-CI/CD to set different triggers for each job. In the current CI-structure, we use `rules` in three ways.

1. Trigger specific jobs based on specific commit messages. E.g. to allow the creation of reference results through a specific commit.
2. Identify commits on special branches like development and main. This allows to handle main and development branches different than feature branches.
3. Identify commits on branches with existing pull request. Same as for special branches.

To exclude specific models from certain stages, we integrated a whitelist functionality. This is useful, if the library author is aware that some already existing models are failing, but this should not impede the development or integration of new models. In addition, the whitelist functionality is necessary if the existing library contains models from other libraries, but does not manage these itself. In this case, errors in the source library models should be detected and fixed within this source library and not lead to failed tests in the extended library. If



```
stages:
- <...>
- HTML_Check
- Style_Check
- <...>

include.:
- <...>
- ../style_check.gitlab-
  ci.yml
- <...>
```

```
stage:
before_script:
- package installations etc.
script:
- python MoPyTest style_check
  -s <library>
after_script:
- postprocess results and
  provide artifacts
<...>
```

```
class SytleCheck:

def check_library(self):
    ...

def style_check(self):
    ...

<...>
```

Figure 2. Excerpt of the exported templates (simplified).

[13] https://www.makotemplates.org/

these models are not tested, the CI runtime will also be reduced. This is particularly relevant since the IBPSA library represents such a library, on which four libraries are based.

Additionally, an option exists, to allow certain stages to fail. This may be useful if, for example, the library's current status with respect to style checking does not meet the requirements, but insights into style quality are still of interest.

3.4.1 Description of Stages

Figure 3 presents the stages of the created CI-structure. The scattered stages hold jobs which are only executed under special conditions, while the other jobs are executed every time. As some jobs currently require a Dymola installation, we added information about the compatibility with Dymola and OpenModelica to the figure. Following, we give a detailed description of each job and stage, its functionality and its output.

The **Regression Result** stage is conditional and only executed if a pull request is opened for this branch. The executed jobs in this stage will create missingreference results. This stage is beneficial because manually creating reference results requires a Python installation on Linux with BuildingsPy, which raises the hurdle for creating examples with reference results, especially for inexperienced developers. To create the reference results, the developer only needs to add the `.mos` script for the simulation model that holds information about the model to simulate, the experiment settings, and the variables of the models that should be taken into account for regression testing. By comparing the existing `.mos` scripts with existing regression results, the CI identifies not existing regression results and creates them. The resulting reference results are directly pushed to the branch and a new CI pipeline is triggered that is based on the updated reference results.

This automated process only creates reference results that do not yet exist in order to avoid unwanted changes to reference results. However, the process can also be used to update existing reference results. To do this, the developer can delete existing reference results. This way, new updated reference results are generated throughout the explained process. This semi-automatic procedure leaves the sovereignty over the reference results with the developers and yet simplifies the process.

The **Create Whitelists** stage creates whitelists if the used library extends models of another library (e.g. AixLib extending IBPSA). The stage is only executed, if one of the listed commit messages in 3 is used.

Modelica uses HTML code for model documentation. The **HTML-Check** stage checks the HTML code in all Modelica files against valid HTML syntax using the python package PyTidyLib[14] and is always executed. If the check is not successful, due to invalid HTML syntax, a new branch will be created using an API and an automatic repair process is performed on the existing HTML code to fix common errors in the HTML section. Subsequently, the check is performed again and if the check succeeds, a merge request is created on the target Modelica repository with the fixed HTML code. This way there is still a manual review process, but the Modelica

[14] `http://countergram.github.io/pytidylib/`

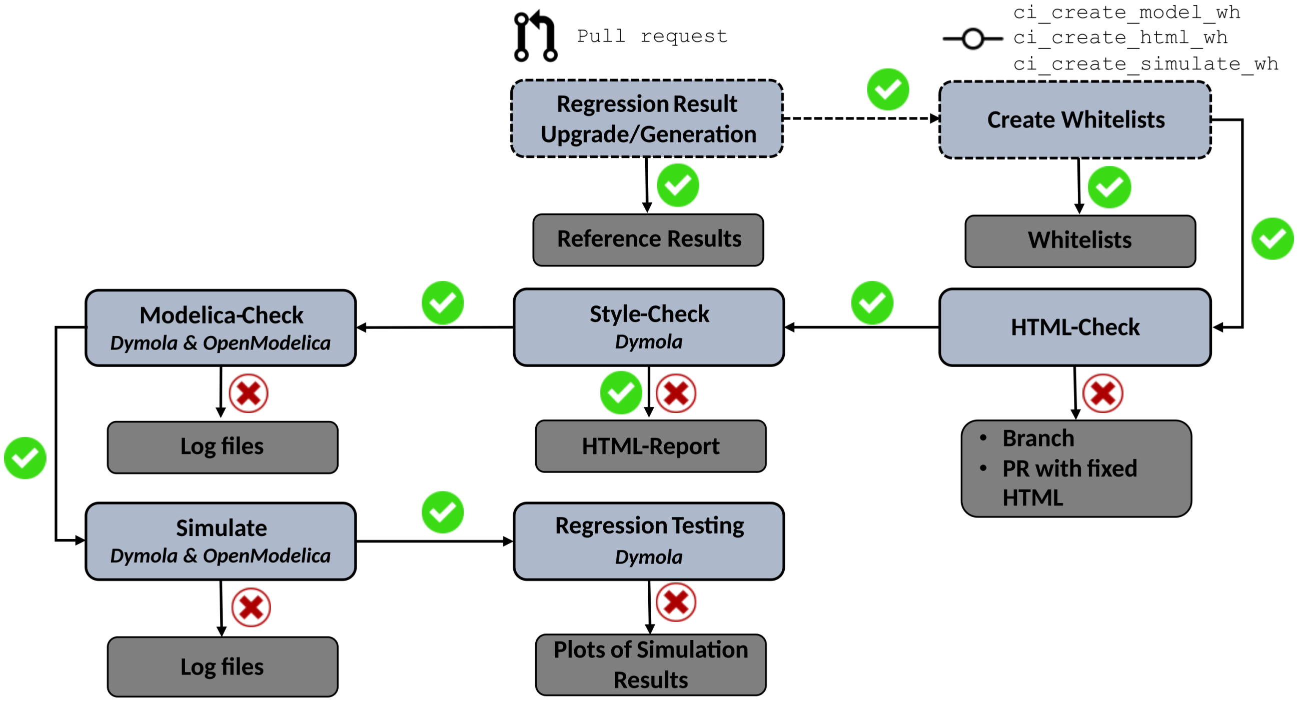


Figure 3. Stages of the invented CI-structure, scattered stages are conditional.

user itself does not have to deal with common issues like unclosed HTML tags.

Next, a **Style-Check** stage is performed using the *Model Management* library of Dymola and runs its inbuilt style check against the whole library. The checking process includes three types of checks: class checks, component checks and general checks. These checks among other things evaluate if the existing model code holds correct documentation, descriptions and class names, but does not perform any checks regarding model functionality. As the *Model Management* can only be used by Dymola this stage is currently limited to the usage of Dymola. The result of the stage is a HTML-report about the quality of the checked library that gives detailed insights about the quality. The report is stored as an `artifact` which is available for download.

Subsequently, the **Modelica-Check** gives detailed insights about the syntactic and logical correctness of the model code by using the check functionality of Dymola or OpenModelica. If errors arise, these will be saved to a log file, which is available via artifacts. The stage is implemented for both, OpenModelica and Dymola.

The **Simulate** stage runs all models inside the library which extend the *Modelica.Icons.Example* model. This is useful, because a model might pass the previous checks, but won't simulate successful. The stage is available for OpenModelica and Dymola.

Regression testing stage runs simulation for all models against their existing regression results using the BuildingsPy.

In case of a failure in the regression tests, plots of the simulation results are created. This stage should allow a fast identification of which model failed and why by creating plots of the expected and the new result trajectory using Google Charts[15]. The plots will be deployed directly via an GitLab page and be posted to the GitHub pull request, see Section 3.4.3

As some jobs, like simulation of the models or regression testing, are computationally and time intensive, we implemented the option to only run these jobs for models, where the source code of the model changed throughout a commit. By using the `git diff` feature, we can check the differences between the target branch and the current branch and identify the changed models. This function currently does not consider inheritance and integration of submodels in other models. This means that if a single component that is part of multiple models changes, only the component itself is checked, but not all models in which it is integrated. For certain events, like the assignment of a reviewer in a pull request, this option is disabled and all models, even if not changed, are checked.

3.4.2 *IBPSA* Library Specific

In addition to the stages shown and described, there is also the **IBPSA-Merge**. This is very tailored to the existing setup of IBPSA library and extending libraries and therefore not shown in the general process schema. This stage allows performing an automatic merge of the source library *IBPSA* into the extending libraries like *AixLib*. Therefore, the merge script delivered by *BuildingsPy* is used. After the automatic merge, a conversion script is created based on the existing conversion script of *IBPSA* and the latest conversion script of the extending library.

Additionally, we add the annotation `__Dymola_LockedEditing` to all IBPSA models, which allows displaying these models as locked inside Dymola as shown in Figure 4. This is very useful to prevent changes to Modelica models that are not part of the extending library, which would lead to merge conflicts when performing the next *IBPSA* merge.

3.4.3 Interfacing and Communication

Automating tasks like the *IBPSA* merge, or the creation of reference results, can save a lot of time when maintaining a Modelica library. But not all tasks can be completely automated and even if a complete automation is possible, the results need to be communicated with the library users. To fulfill the need for communication, we use the GitHub REST API. If using GitHub as the main repository, the GitHub REST API allows writing messages via a bot account which give feedback and instructions. E.g. in case of failed regression tests, the GitLab page with plots of the simulation results will be linked to the corresponding pull request, so that the user can directly see which models are failing and might also be able to identify why the simulation results differ from the regression results.

4 Exemplary Application on Two Libraries: *AixLib* and *BESMod*

In the following sections, we provide insights into how we implemented a working CI infrastructure at our institute (4.1) and show the application of the developed CI infrastructure to two libraries: *AixLib* (4.2) and *BESMod* (4.3).

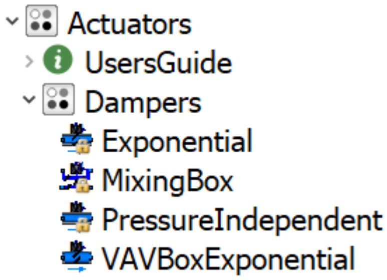


Figure 4. Locked *IBPSA* components in *AixLib* library.

[15] `https://developers.google.com/chart`

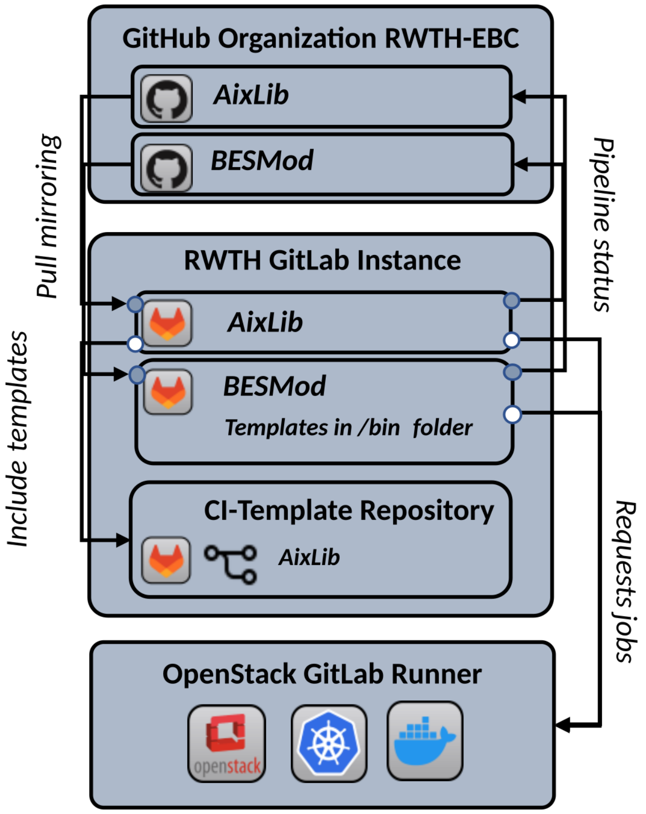


Figure 5. CI-infrastructure at RWTH in Aachen.

4.1 Setup at RWTH in Aachen

The used setup at our institute in Aachen is shown in Figure 5. We have a public available GitHub organization that hosts both later described libraries *AixLib* and *BESMod*. Both libraries are mirrored to our university GitLab instance, where we have a specific subgroup for mirrored projects. The jobs are executed by a scalable GitLab runner, provided through the GitLab runner docker image and Kubernetes. The Runner setup is hosted on our internal cloud service OpenStack. Both *AixLib* and *BESMod* hold a `gitlab.ci-yml` file. In case of *AixLib* this links to another GitLab repository, where the template structure, created by *MoCITempGen* is stored in a separated branch. This way, we can easily implement changes to the CI, by simply updating the external repository branch of the templates. For *BESMod* the created templates are stored directly in the repository in the `bin` folder.

4.2 *AixLib*

The development of *MoCITempGen* took place on the basis of *AixLib* (Maier et al. 2023). A brief explanation of the application of *MoCITempGen* at *AixLib* has already been given in this previous paper. The template generator was developed based on the existing CI of *AixLib* and generalized so that it can be applied to other Modelica libraries.

To show that the concept of *MoCITempGen* works, we opened a demonstration pull request[16] in the *AixLib* and provoked the CI to perform to give two examples of how the CI works and interacts with the modeler.

The second example is a failing regression test, that was provoked by changing the scaling factor for the heat pump model in the example `AixLib.Fluid.HeatPumps.Examples.-HeatPump`. Due to the changes to the model, the regression test stage fails and the *ebc-aixlib-bot* posts a comment into the pull request and links the GitLab page with the plots showing the differences between existing regression results and new results. A screenshot is shown in Figure 6a. The second example is the creation of new reference results. Therefore, we deleted the existing results and pushed them to the branch. The CI notices a missing reference result for an existing simulate and plot `.mos` script and creates new results, pushes them to the branch, and informs about the new created results inside the pull request with a link to the GitLab page with plots of the new reference results (see Figure 6b). Further information about the usage of *AixLibs* CI can be found in the *AixLib*-Wiki[17].

4.3 *BESMod*

Contrary to the *AixLib*, the library *BESMod* is not an extension to the library *IBPSA*. Rather, it uses currently existing libraries, such as the *IBPSA*, *Buildings* or *AixLib*. Thus, to load the *BESMod* in the CI, installation of these additional requirements is necessary. GitLab-CI offers *before-scripts* to execute specific commands prior to the actual script. Before generating the CI configurations using *MoCITempGen*, adding an additional line for requirement installation to the *before-script* section in the *.txt* template files was required. Afterwards, all features of the CI were directly accessible and useable, even for a more complicated setup, as is the case in *BESMod*. In summary, while smaller adjustments may be necessary to a specific library, *MoCITempGen* decreases the CI setup time drastically.

5 Discussion

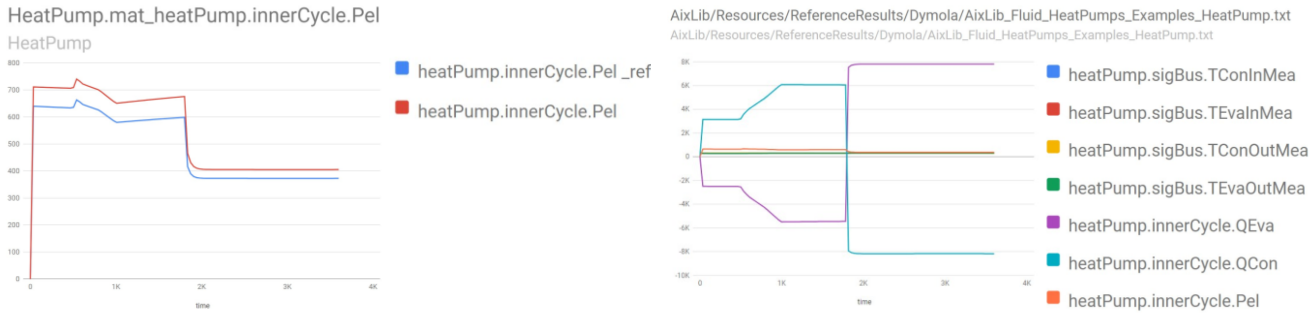
The presented methodology was successfully applied to two Modelica libraries. Even though the template creation tool was developed with the goal of high flexibility, there are currently still some requirements. These requirements are:

1. the target Modelica library must be hosted in or mirrored to a GitLab repository,
2. a GitLab runner must be configured (via SaaS or by hosting an own),
3. for jobs that need a simulation environment, either an OpenModelica or Dymola image must be provided
4. in case of using Dymola, a Dymola license is required

The limitation to GitLab repositories can be circumvented by using the GitLab mirroring function in section 3 which

[16] `https://github.com/RWTH-EBC/AixLib/pull/1389`

[17] `https://github.com/RWTH-EBC/AixLib/wiki/GitLab-CI`



(a) New and existing results for failed regression test.

(b) New regression results created by CI process.

Figure 6. Example of AixLib CI for regression testing.

allows applying the presented approach to GitHub, BitBucket or AWS CodeCommit. The required GitLab runner can be self-hosted without requiring to pay any service as described in section 2. However, the presented version of MoCITempGen currently enforces the use of GitLab-CI/CD in the background, which requires familiarization with the GitLab-CI/CD syntax and runner infrastructure. Furthermore, as described in section 3, some stages need Dymola, or at least OpenModelica. Dymola requires a paid license. OpenModelica on the other hand is open source and already offers public available images on DockerHub[18]. But currently, not all stages are compatible with OpenModelica. Therefore, we are aiming to make all stages available via OpenModelica in the future. Additionally, there are further possibilities for improvement. E.g., the outputs of the different stages are not uniform. Some stages output log files, others HTML-reports, others files to download. This could be unified with a central report, which holds all relevant information.

6 Conclusion and Outlook

This paper gives an overview of CI applications in the context of Modelica and presents the tool *MoCITempGen* that aims to facilitate the use of CI for authors of Modelica libraries. The tool and the underlying methodology are explained, and the application of the tool on two Modelica libraries is shown. Even though we have applied *MoCITempGen* to two libraries in the context of the building energy efficiency sector, it is also possible to apply it to other libraries from other domains.

In order to increase the application possibilities and to support different repository architectures, we want to extend the tool in the future. This concerns on the one hand the support of OpenModelica in all stages, in which a simulation environment is used. On the other hand, the export format of the templates will be extended so that templates for GitHub Actions can also be exported in the future. This is especially useful since GitHub Actions also supports the possibility of self-hosted runners.

Furthermore, we want to increase the flexibility and maintainability by separating the Python library from the template generation tool.

Eventually, the goal is to unify the various existing approaches to CI in the context of Modelica and make them available across use cases.

Acknowledgements

This work has been supported by the German Federal Ministry for Economic Affairs and Climate Action as part of the project BIM2Praxis (grant numbers 3EN1050A, 3EN1050B).

References

Beck, Kent (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional. ISBN: 978-0-201-61641-5.

Booch, Grady (1991). *Object Oriented Design with Applications*. Benjamin/Cummings Publishing Company. ISBN: 978-0-8053-0091-8.

Gall, Leo et al. (2021-09-27). “Continuous Development and Management of Credible Modelica Models”. In: *Modelica Conferences*, pp. 359–372. ISSN: 1650-3740. DOI: 10.3384/ecp21181359.

Hugues, Jerome et al. (2022-03-24). *TwinOps: Digital Twins Meets DevOps*. report. Carnegie Mellon University. DOI: 10.1184/R1/19184915.v2.

Maier, Laura et al. (2023). “AixLib: an open-source Modelica library for compound building energy systems from component to district level with automated quality management”. In: *Journal of Building Performance Simulation* 0.0, pp. 1–24. DOI: 10.1080/19401493.2023.2250521. eprint: https://doi.org/10.1080/19401493.2023.2250521. URL: https://doi.org/10.1080/19401493.2023.2250521.

Rabuzin, Tin, Maxime Baudette, and Luigi Vanfretti (2017-07-01). *Implementation of a continuous integration workflow for a power system Modelica library*. Pages: 5. 1 p. DOI: 10.1109/PESGM.2017.8274618.

Schölzel, Christopher et al. (2021-07-19). “Countering reproducibility issues in mathematical models with software engineering techniques: A case study using a one-dimensional mathematical model of the atrioventricular node”. In: *PLoS ONE* 16.7, e0254749. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0254749.

[18] `https://hub.docker.com/r/openmodelica/openmodelica/tags`

Vasilescu, Bogdan et al. (2015-08-30). “Quality and productivity outcomes relating to continuous integration in GitHub”. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, pp. 805–816. ISBN: 978-1-4503-3675-8. DOI: 10.1145/2786805.2786850.

Wetter, Michael (2019). *BuildingsPy*. Language: en. DOI: 10.11578/DC.20190430.2.

Wetter, Michael et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Steady State and Dynamic Simulation of a Small-Scale Hollow Fiber Membrane Humidifier

Markus Pollak[1] Manuel Kutz[2] Christian Schulze[2] Wilhelm Tegethoff[1,2] Jürgen Köhler[1]

[1]Institut für Thermodynamik, Technische Universität Braunschweig, Germany, {m.pollak, w.tegethoff, juergen.koehler}@tu-braunschweig.de
[2]TLK-Thermo GmbH, Germany, {m.kutz, c.schulze}@tlk-thermo.com

Abstract

Membrane humidifiers are commonly used in mobile proton exchange membrane (PEM) fuel cell systems to humidify the fuel cell supply air with the purpose of preventing the fuel cell membrane from drying out. In this paper, a humidifier model based on the number of transfer units (NTU) approach is set-up in Modelica, calibrated and validated using measurements of a test rig. The mass transfer model of our humidifier model is extended with a first order transfer function to capture dynamic operation. In a first step, the model is evaluated for steady state operating conditions. Second, the developed membrane humidifier model is simulated with dynamically changing operating conditions that are typical for mobile applications. Those simulation results are then compared to measurements. The aim of our study is to evaluate the accuracy of the humidifier model under various operating scenarios. Our results indicate that the NTU model is suitable to predict the water transfer under steady and dynamically changing operating conditions with low deviations to measurements.

Keywords: membrane humidifier, dynamic simulation, NTU, PEM fuel cell

1 Introduction

To ensure a high efficiency and a long lifetime of a Proton Exchange Membrane (PEM) fuel cell it is necessary to keep the membrane hydrated (Brandau, Heinke, and Koehler 2016; Ozen, Timurkutluk, and Altinisik 2016; Wu et al. 2020). This goal can be achieved by humidifying the supply air of the fuel cell. For this purpose, a membrane humidifier that transfers water along a concentration gradient from the wet fuel cell exhaust gas to the supply air can be used (Brandau, Heinke, and Koehler 2016). Alternative humidification methods are discussed in the literature but membrane humidifiers are considered a well-suited solution for the humidification of PEM fuel cells (Chen, Li, and Peng 2008). In automotive applications the operating conditions of such membrane humidifiers vary dynamically. Most of the studies in the literature focus on steady state operating conditions when assessing membrane humidifiers, e.g. (Cahalan et al. 2017; Nguyen, Vu, and Yu 2021; Pollak et al. 2023). A previous study (Pollak et al. 2023) investigated the same type of humidifier as discussed in our work, but focuses on detailed CFD model that is not suitable for system simulations with transient operation due to its long calculation times. Only few studies discuss transient operation of membrane humidifiers (Chen, Li, and Peng 2008; Park, Choe, and Choi 2008; Yun et al. 2018; Vu, Nguyen, and Yu 2022). Three of them do not compare their simulation results to experimental data (Park, Choe, and Choi 2008; Yun et al. 2018; Vu, Nguyen, and Yu 2022) and the fourth uses liquid water instead of a wet air flow as humidity source (Chen, Li, and Peng 2008).

The first aim of our study is to fill the existing gap of measurement data for the validation of transient operation. Moreover, we use our data to analyze if an NTU model is suitable to represent the dynamic operation accurately. Therefore, we first set up a model of a hollow fiber humidifier in Modelica based on the NTU method for mass exchangers as proposed by Brandau et al. (Brandau, Heinke, and Koehler 2016) and extend this approach with a first order transfer function in the mass transfer model. The purpose of the introduced first order transfer function is on the one hand to describe the dynamics of mass transfer and on the other hand to break potential non-linear systems in the mass transfer model. We use measurement data from a test rig to calibrate and validate the model for steady state and transient operating conditions. Furthermore, the calibration results of the model are compared to the results of a computational fluid dynamics (CFD) model of the same humidifier that was developed in a previous study (Pollak et al. 2023). The motivation for using Modelica to develop the humidifier model is the possibility to describe the humidifier model in an object-oriented way. This allows essential aspects such as the calculation of the mass transfer coefficient or the NTU characteristic to be described and modified as a replaceable submodel. This results in a high flexibility of the model, while the model code remains lean and understandable. A further advantage is that the governing physical laws can be written as equations in Modelica with variables whose physical units can be easily defined and checked. Moreover, the developed humidifier model integrates well into system simulation models with various operating conditions like PEM fuel cell systems.

DOI
10.3384/ecp204531

The structure of our paper is as follows. In the following section, the working principle and the main features of the hollow fiber humidifier are introduced. Next, the test rig used for the measurements is presented. The third section discusses the modeling of the humidifier. It is followed by a section focusing on the steady state and dynamic simulation results.

2 Measurement of Hollow Fiber Humidifier

In this section, the general working principle of a mass exchanger is described. Moreover, the geometry of the investigated humidifier is introduced and its main features are presented. Second, the configuration of the test rig and the measured quantities are explained briefly.

2.1 Working principle

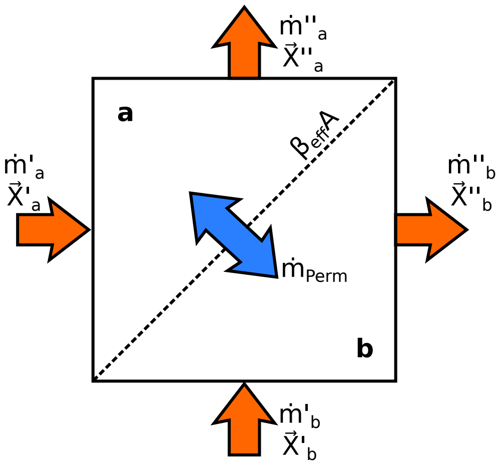


Figure 1. Sketch of a general mass exchanger that transfers mass from one stream to another. The vector $\vec{X}$ represents the state of the fluid at the depicted locations.

In this study, the hollow fiber membrane humidifier is investigated as special type of a mass exchanger. A sketch showing the most important quantities of such a mass exchanger is depicted in Fig. 1. All inlet quantities are denoted by $\square'$ whereas outlet values are marked as $\square''$. Typically, two mass flows enter a membrane humidifier. Both mass flow rates are altered due to the vapor transfer taking place inside the humidifier. Finally, two mass flows leave the mass exchanger. The mass transfer inside the mass exchanger is driven by a concentration difference according to Fick's Law. An effective mass transfer coefficient $\beta_{\mathrm{eff}}A$ describes the ability of a device to transfer certain species. The mass transfer coefficient depends on the geometry, the used materials and the state of the depicted fluid flows (cf. Fig. 1). In a membrane humidifier a semipermeable membrane is used that poses a low resistance to water transfer but a high resistance to the transfer of other species. Depending on the inlet concentrations, the mass transfer can either occur from side A to B or from B to A. To achieve high water transfer rates membrane humidifiers are typically operated in counter- or crossflow arrangement.

2.2 Description of the Hollow Fiber Membrane Humidifier

In Fig. 2 a sketch of a hollow fiber humidifier geometry is shown. Only 12 fibers are depicted in Fig. 2 to highlight the geometric features and flow situation. As shown in Fig. 2, we investigate a counterflow arrangement of wet and dry air flows. As depicted in Figure 2, the wet air stream flows through the fibers, whereas the dry stream is fed to the shell. As a result of the manufacturing process, the fibers are placed randomly inside the shell. The investigated humidifier is available from the company Fumatech (FUMATECH BWT GmbH 2019) and was investigated using a CFD model and simulation in a previous study (Pollak et al. 2023). The material of the hollow fiber membranes is undisclosed by the manufacturer (Pollak et al. 2023). The relevant geometrical data of the fibers and the housing are given in Table 1.

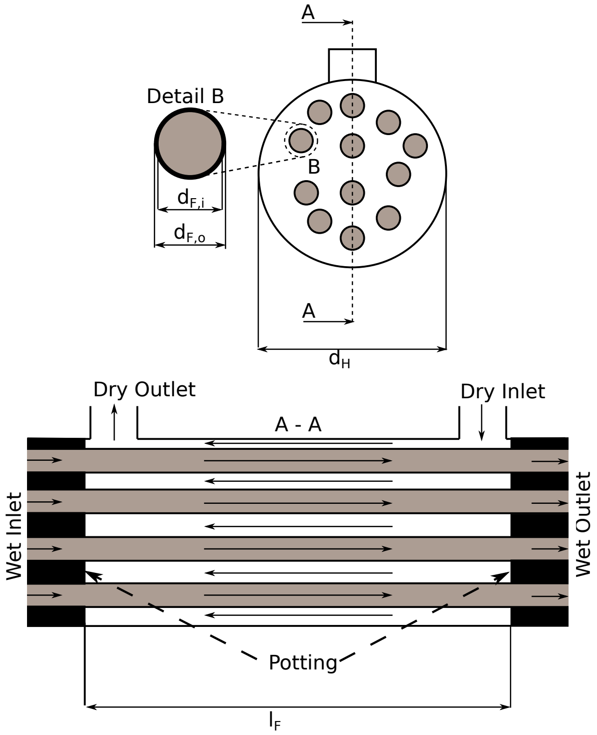


Figure 2. Geometric features of the modeled hollow fiber humidifier. The fibers are shown in light gray and the shell side with white background. The humidifier operates in counterflow, as seen in cut view A-A. A detailed view of a fiber is given in detail view B. Adopted from (Pollak et al. 2023)

2.3 Description of the Test Rig

The test rig is used to investigate the mass transfer of membrane humidifiers at various operating conditions. On the test rig, the same boundary conditions as in the simulation can be varied in their respective limits as described for the model (see Section 2.4).

Table 1. Parameters of the investigated hollow fiber membrane humidifier geometry.

Quantity	Symbol	Unit	Value
Number of fibers	n_F	1	488
Fiber outer diameter	$d_{F,o}$	mm	1
Fiber inner diameter	$d_{F,i}$	mm	0.9
Fiber length	l_F	mm	150.8
Housing inner diameter	d_H	mm	39.2

The water transfer is calculated using either sensor values of the dry side:

$$\dot{m}_{\mathrm{H2O,dry,perm}} = \dot{m}''_{\mathrm{dry}}\xi''_{\mathrm{H2O,dry}} - \dot{m}'_{\mathrm{dry}}\xi'_{\mathrm{H2O,dry}} \tag{1}$$

or using the sensors used on the wet side:

$$\dot{m}_{\mathrm{H2O,wet,perm}} = \dot{m}'_{\mathrm{wet}}\xi'_{\mathrm{H2O,wet}} - \dot{m}''_{\mathrm{wet}}\xi''_{\mathrm{H2O,wet}} \tag{2}$$

A steady-state operating point is only considered when the water transfer rates of wet and dry side match within a tolerance of 5 %. For calibration and validation of the models, both results are averaged.

A P&ID showing the humidifier test rig can be found in Figure 3. The sensors used in our test rig and their respective uncertainties are given in Table 2. Based on those uncertainties the error propagation is calculated according to the guideline (Joint Committee for Guides in Metrology (JCGM) 2008).

Air is fed to the test rig from a pressurized air storage tank. Behind the air storage, the air flow is split into two streams: one for the wet and one for the dry side. The wet path simulates the exhaust gas from the fuel cell and the dry path the air supplied to the humidifier. A control valve at the inlet of each path is used to adjust each the wet and dry air mass flow rates, respectively. Next, both air streams are heated up by electrical heaters to reach an operating temperature typical for membrane humidifiers used in PEM fuel cell systems. The pressure of both streams can be controlled individually by two valves located at the outlet of the air paths. Vapor is fed from a vapor storage tank through a controlled valve to achieve the desired inlet humidity of the wet air stream. The tubes of the vapor supply line are heated to avoid condensation, which is required to get valid measurement results. The availability of heaters on all tubes is important because the humidity sensors installed can only measure water in gaseous form and therefore condensation has to be prohibited. If condensation of water occurs, the water transfer rates measured on the dry and wet side deviate from each other. Thus, the water transfer rates are continuously checked during the measurement.

2.4 Inputs and Parameters

The water transfer in the humidifier is governed by the inlet conditions of the wet and dry flow and the mass transfer capability. To mimic the operation in a fuel cell system, the following quantities can be adjusted:

- mass flow rates of both, dry and wet, streams;
- temperatures of both, dry and wet, streams;
- pressures of both, dry and wet, streams;
- relative humidity of the wet stream.

The parameter limits for the boundary conditions are listed in Tab. 3. Since both mass flows are conditioned to have nearly same temperature and the humidifier is isolated against the environment, heat transfer is considered to be of negligible effect.

Since the membrane permeability is not disclosed by the manufacturer it is considered to be a calibration parameter of the humidifier model that must be identified by a calibration process using the measurement data. This calibration result is than compared to a fitting result of a previous study that used a CFD model (Pollak et al. 2023).

3 NTU Membrane Humidifier Model for System Simulations

The humidifier models discussed in this study are set-up using the commercially available Modelica libraries TIL, TILMedia and the TIL3_Addon_HydrogenEnergySystems each in version 3.13.0 (TLK-Thermo GmbH 2022). Dymola 2023x is used as modeling and simulation environment (Dassault Systèmes SE 2022a). A novelty of our approach is to use a first order transfer function in the mass transfer model. The introduction of a first order transfer function for the permeating mass is motivated on the one hand numerically and on the other hand physically. A numerical benefit of the introduced state is that non-linear systems can be eliminated. From a physical perspective the first order transfer function can be used to reflect the dynamics of the mass transfer, which is commonly disregarded in mass exchanger models.

3.1 NTU Humidifier Model

The developed humidifier model is built upon the NTU approach as derived by Brandau et al. (Brandau, Heinke, and Koehler 2016) and derived from the class available in TIL3_Addon_HydrogenEnergySystems (TLK-Thermo GmbH 2022).

An overview of the humidifier model with its replaceable submodels and records, the used ports and the objects for thermophysical property calculations is displayed in Fig. 4. The model consist of two air paths (a and b). Both paths are separated by the membrane that is visualized as a dashed line in Fig. 4. Path a is connected to the outside by two connectors, $\mathrm{A_a}$ and $\mathrm{B_a}$. Just like path a, path b has two connections to the outside $\mathrm{A_b}$ and $\mathrm{B_b}$. The model can handle flow from port A to B and vice versa in both paths. At each port a gas object is located (cf. Fig. 4) that is used to calculate the thermophysical properties of the gas mixture at the ports. In both paths a replaceable model for the pressure drop is applied. Since no pressure drop Δp is

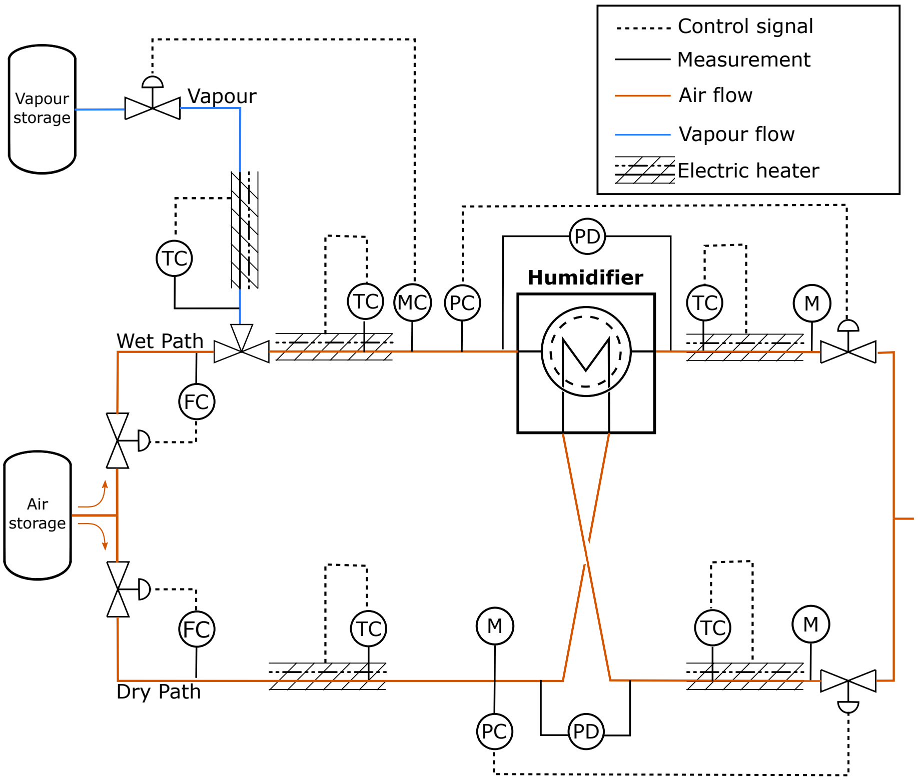


Figure 3. Piping and instrumentation diagram of the test rig used for the investigation of the water permeation in the hollow fiber membrane humidifier at various inlet conditions (Pollak et al. 2023). Flow (F), temperature (T), moisture (M) and pressure (P) sensors are installed. Control variables are marked by a 'C'. Pressure difference measurements are marked with a 'PD'.

Table 2. Used sensors and their measurement uncertainties.

Sensor	Measured Quantity	Output Unit	Uncertainty
Vaisala HMT-337	Humidity	%	$\pm(1.5 + 0.015\varphi)$
Omega FMA-1609A	Mass flow rate	g/s	$\pm(0.008\dot{m} + 0.00204)$
Omega PXM459	Differential pressure	Pa	± 56
WIKA P-30	Pressure	bar	± 0.068
WIKA TR-40	Temperature	K	$\pm 0.15 + 0.002(T - 273.15)$

Table 3. Limits of the boundary conditions for simulation and measurement of the membrane humidifier. The values of water mass fraction apply to the wet side only.

Parameter	Minimum	Maximum
Temperature	60 °C	80 °C
Pressure	1.5 bar	2.0 bar
Air flow rate	0.2 g/s	0.7 g/s
Water mass fraction	0.027	0.172

investigated in our study, the pressure drop is set to zero. The same is true for the heat transfer, therefore the heat transfer coefficient α is also set to zero. The calculation of the overall mass transfer coefficient β_{eff} is discussed in Sec 3.2. In the top right of Fig. 4 a record storing the geometry information of the humidifier is depicted.

The NTU approach for mass exchangers is formulated in analogy to the well known NTU approach for heat exchangers (Brandau, Heinke, and Koehler 2016). Three dimensionless numbers are derived from the inlet quantities given in Fig. 1 to describe the mass transfer between two fluid flows in the NTU model. The three dimensionless numbers described by Brandau et al. (Brandau, Heinke, and Koehler 2016) are the mass transfer efficiency, the ratio of volume flow rates and the number of transfer units. These three dimensionless numbers can be formulated for the dry and wet side of the humidifier, respectively. To calculate the results, only one set of the three dimensionless numbers must be calculated and is denoted by the subscript a in the following. The first dimensionless number is the mass transfer efficiency:

$$\eta_a = \frac{c''_{a,\text{H2O}} - c'_{a,\text{H2O}}}{c'_{b,\text{H2O}} - c'_{a,\text{H2O}}} \tag{3}$$

It describes the ratio of actual mass transfer to the theoretically possible mass transfer. Additionally, the ratio of

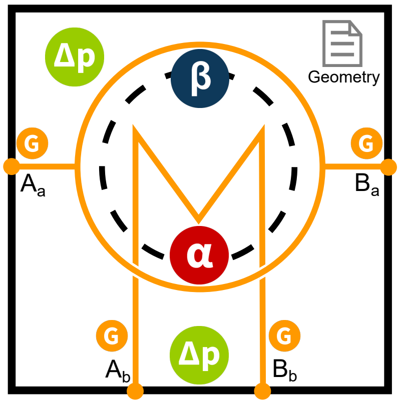


Figure 4. Overview of the humidifier model with submodels, ports and objects for property calculations.

volume flow rates is required:

$$VFR_a = \frac{\dot{V}_a'}{\dot{V}_b'} \tag{4}$$

Finally, the Number of Transfer Units (NTU) can be defined:

$$NTU_a = \frac{\beta_{\text{eff}} A}{\dot{V}_a} \tag{5}$$

To achieve high mass transfer efficiencies, the humidifier operates in counterflow. The following equations derived by Brandau, Heinke, and Koehler (2016) are used to calculate the mass transfer efficiency:

$$\eta_a = \begin{cases} \frac{NTU_a}{1+NTU_a}, & \text{if } VFR_a = 1 \\ \frac{1-exp[(VFR_a-1)NTU_a]}{1-VFR_a\,exp[(VFR_a-1)NTU_a]}, & \text{otherwise} \end{cases} \tag{6}$$

The transferred water is than calculated based on the mass transfer efficiency and the inlet quantities:

$$\dot{m}_{perm,H2O} = M_{H2O}\eta_a(c_{b,H2O}' - c_{a,H2O}')\dot{V}_a' \tag{7}$$

In order to allow a fitting of the suggested model to transient data and to break non-linear systems, a first order transfer function is introduced that can be selected by the user (TLK-Thermo GmbH 2022). The mass transfer dynamics are caused by a combination of effects i.e. sensor delay and the residence time of supplied water in the humidifier. In our approach, the ratio of permeation rate and the smaller air mass flow rate at a given time is introduced as state variable:

$$\Xi = \frac{\dot{m}_{\text{perm,H2O}}}{min(\dot{m}_a', \dot{m}_b')} \tag{8}$$

Ξ_{state} is introduced as state variable as motivated above to describe the dynamics of the mass transfer process defined by the equation

$$\frac{d\Xi_{\text{state}}}{dt} = \frac{\Xi - \Xi_{\text{state}}}{\tau_{\text{perm}}} \tag{9}$$

The permeation rate $\dot{m}_{perm,H2O,state}$ that includes the introduced dynamics is than calculated using the following equation:

$$\dot{m}_{\text{perm,H2O,state}} = \begin{cases} \Xi_{\text{state}}\dot{m}_a' & \text{if } \dot{m}_b' \geq \dot{m}_a \\ \Xi_{\text{state}}\dot{m}_b' & \text{otherwise} \end{cases} \tag{10}$$

The advantage of using Ξ_{state} instead of a state for the permeation rate $\dot{m}_{\text{perm,H2O}}$ is that Ξ_{state} is not directly linked to the inflowing mass flow rate $\dot{m}_a'$, but takes both air mass flow rates into account. It is linked to the smaller mass flow rate $min(\dot{m}_a, \dot{m}_b)$ so that the permeation rate cannot exceed the capacity of the smaller mass flow. Thus, the model becomes more robust. This is especially important for dynamic changes, e.g. when one of the mass flow rates is decreased dramatically.

If no first order transfer function is used, the following equation holds:

$$\dot{m}_{\text{perm,H2O,state}} = \dot{m}_{\text{perm,H2O}} \tag{11}$$

The calculated permeation rate is than introduced in the balance equations for mass and species. Those balance equations are formulated separately for each path. The mass balances of both paths are connected with the permeation rate and read:

$$\dot{m}_a'' = \dot{m}_a' - \dot{m}_{\text{perm,H2O,state}} \tag{12}$$

$$\dot{m}_b'' = \dot{m}_b' + \dot{m}_{\text{perm,H2O,state}} \tag{13}$$

Furthermore, the water balances for both paths read:

$$\dot{m}_{a,\text{H2O}}'' = \dot{m}_{a,\text{H2O}}' - \dot{m}_{\text{perm,H2O,state}} \tag{14}$$

$$\dot{m}_{b,\text{H2O}}'' = \dot{m}_{b,\text{H2O}}'' + \dot{m}_{\text{perm,H2O,state}} \tag{15}$$

The mass transfer is accompanied by an enthalpy flow that is calculated using the inlet states and the previously discussed permeation rate:

$$\dot{H}_{\text{perm}} = \begin{cases} \dot{m}_{\text{perm,H2O,state}}\, h_a & \text{if } c_{a,\text{H2O}}' \geq c_{b,\text{H2O}}' \\ \dot{m}_{\text{perm,H2O,state}}\, h_b & \text{otherwise} \end{cases} \tag{16}$$

The enthalpy flow rate of the permeation flow is included in the energy balance equation of both paths:

$$\dot{H}_a'' = \dot{H}_a' - \dot{H}_{\text{perm}} \tag{17}$$

$$\dot{H}_b'' = \dot{H}_b' + \dot{H}_{\text{perm}} \tag{18}$$

Furthermore, a heat transfer rate can be calculated in the model, too, but is not discussed here due to the selection of nearly isothermal operating conditions.

3.2 Mass Transfer in the Membrane Humidifier

To calculate the NTU in the presented model, the overall mass transfer coefficient is required. The calculation of this mass transfer coefficient takes place in a submodel, which can easily be replaced and adapted. In general, the effective mass transfer coefficient is the reciprocal of the overall mass transfer resistance:

$$\beta_{\text{eff}} A = \frac{1}{R_{\text{eff}}} \tag{19}$$

This effective mass transfer resistance can be split in three parts that are connected in series as given in:

$$R_{\text{eff}} = R_{\text{conv,wet}} + R_{\text{mem}} + R_{\text{conv,dry}} \tag{20}$$

The first term on the right-hand side describes the convective resistance to the mass transfer in the wet flow. Second, the membrane poses a resistance to the mass transfer. This membrane resistance is much lower for vapor than for other gas components. Therefore, we assume that only vapor is transferred in the humidifier. Last, another convective resistance is present on the dry side.

Both convective resistances are calculated using Sherwood correlations that were empirically determined in the literature (Costello et al. 1993; Gnielinski 2010). Using the determined Sherwood numbers, the mass transfer coefficients for the convective transfer can be calculated:

$$\beta_i = \frac{Sh\, D_{\text{H2O,Air}}}{l_{ch}} \tag{21}$$

For the calculation of the Sherwood number of wet air flow inside the fibers, the well known correlations for heat transfer in tube are adapted from Gnielinski (2010):

$$Sh_{\text{wet}} = [Sh_{\text{wet},1}^3 + 0.7^3 + ([Sh_{\text{wet},2} - 0.7)^3]^{1/3} \tag{22}$$

$$Sh_{\text{wet},1} = 3.66 \tag{23}$$

$$Sh_{\text{wet},2} = 1.615 \left(Re\, Sc \frac{d_{hyd}}{l_F} \right)^{1/3} \tag{24}$$

The hydraulic diameter equals the inner diameter of a single fiber. On the other hand, the shell side is more complicated due to more complex flow phenomena. Costello et al. (1993) proposed the following correlation that includes the packing density Φ :

$$Sh_{\text{dry}} = (0.53 - 0.58\Phi) Re^{0.53} Sc^{0.33} \tag{25}$$

This correlation is also used by Vu, Nguyen, and Yu (2022) to model the shell side convective mass transfer of a hollow fiber humidifier. The packing density is defined as ratio of shell cross sectional area to the cross sectional area occupied by the fibers:

$$\Phi = \frac{n_F d_{F,o}^2}{d_H^2} \tag{26}$$

The mass transfer resistance of the membrane is modeled with a constant diffusion coefficient and the thickness given in Tab. 1:

$$R_{\text{mem}} = \frac{\delta}{D_{\text{mem}} A_{\text{mem}}} \tag{27}$$

4 Calibration and Validation of the Humidifier Models

In this section, the calibration and validation of the presented model using measurements of the test rig are shown. At first, the membrane diffusion coefficient is calibrated and validated using steady state data of the humidifier reflecting the range of operating conditions (cf. Tab. 3). Next, the introduced time constant of mass transfer is calibrated using measurement data and the model with the previously calibrated mass transfer coefficient.

4.1 Steady State Operation

The available measurement data contains two data sets. Our first data set includes 105 steady state operating points. For this data set the volume flow rate ratio was kept close to one ($VFR \approx 1$). The points of this first data set are randomly split between the calibration and the validation set. We use 60 % of the data for the calibration and 40 % for the validation of the model. The second data set is used for validation only. It contains only eight points but in contrast to the operating points the ratio of volume flow rates differs significantly from one. With this data set the accuracy of the NTU model when predicting water transfer rates at operating points very different from the calibration data is assessed. The calibration of the model is done using the truncated Newton (TNC) optimization method of the SciPy library written in Python (Virtanen et al. 2020). For the calibration process, the model is exported from Dymola as functional mock-up unit (FMU) and simulated in Python using FMPy (Dassault Systèmes SE 2022b). The root mean square error (RMSE) of the simulated $\hat{\dot{m}}_{\text{perm}}$ and measured water permeation rate $\dot{m}_{\text{perm}}$ is used as objective function to be minimized:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} (\dot{m}_{\text{perm},i} - \hat{\dot{m}}_{\text{perm},i})^2}{n}} \tag{28}$$

The objective of the fitting process is to find the membrane diffusion coefficient $D_{\text{mem,fit}}$ that minimizes the previously defined RMSE of the model $\vec{X}_{a,b}$:

$$D_{\text{mem,fit}} = \text{argmin RMSE}(\vec{X}_{a,b}, D_{\text{mem}}) \tag{29}$$

Finally, the membrane diffusion coefficient for vapor $D_{\text{mem,fit}} = 3.63e-7\, m^2/s$ that minimizes the deviation of simulation and measurement is found. This fitting result is close to the value $D_{\text{mem,CFD}} = 4.02e-7\, m^2/s$, determined in a previous study where a smaller data set and a CFD

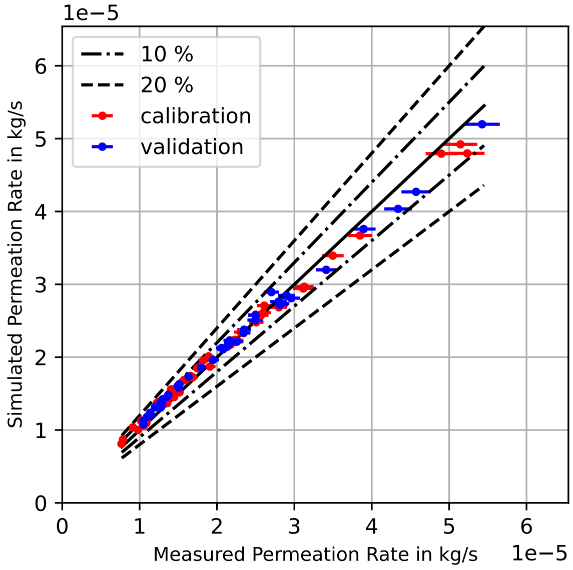


Figure 5. Comparison of the calibration and validation results of the NTU model with the steady state measurement data.

model of the same humidifier were investigated (Pollak et al. 2023). The simulation results of the calibrated model are shown in Fig. 5, marked by blue points. The depicted error bars represent the measurement uncertainty of permeation rate calculated according to the guideline (Joint Committee for Guides in Metrology (JCGM) 2008). For all investigated points, the deviation of the simulation is less than 20 % in comparison to the measurement. Moreover, the results of the validation are plotted in Fig. 5, too. Again, the deviation of the simulation results from the measurements is less than 20 % for the validation data. The deviation of the simulated from the measured permeation rate is less than 10 % for most operating points and often within the range of measurement uncertainty. In Tab. 4 an overview of the RMSE is presented. From the values shown in Tab. 4, only a small difference between calibration and validation can be identified.

Table 4. RMSE of the simulated and measured water permeation rate for calibration, validation and total data.

	Calibration	Validation	Total
RMSE in g/s	1.3242e-6	1.4036e-6	1.3560e-6

To test the capability of the NTU model to extrapolate, another data set is used. A special feature of this data set is that the ratio of volume flow rates is varied systematically from 0.37 to 2. To assess the model accuracy, the mass transfer efficiency of both measurement and simulation is plotted versus the NTU of the simulations in Fig. 6. One can see that the simulation results lie directly on the characteristic lines representing a fixed volume flow ratio. On the other hand, the measurement results do not directly match the characteristic lines, which is due to deviations of measurements and simulation results. Nonetheless, simulation results and measurements show a similar trend. Furthermore, the relative deviation of the simulated and measured mass transfer efficiencies is below 10 % for all considered operating points.

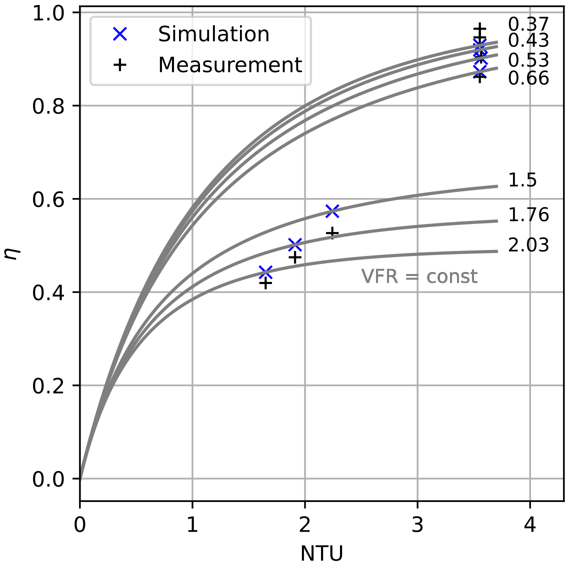


Figure 6. Comparison of simulation results and measurements on validation data for different volume flow rate ratios. The values of the investigated VFR_a are shown next to the lines.

In summary one can conclude that the developed NTU model can accurately predict the water permeation under various steady state operating conditions. The ability of the NTU model to extrapolate was demonstrated by using the NTU model that was calibrated for a $VFR_a \approx 1$, for a range of $0.35 < VFR_a < 2.0$.

4.2 Dynamic Simulations

Another goal of our study is to evaluate if the developed humidifier model is able to predict dynamic operation of the humidifier accurately. For this purpose, we use data from the test rig that was collected while switching from one operating point to another. Therefore, most of our investigations apply to the water mass fraction feed to the wet air inlet. To investigate the dynamics of the humidifier, the water mass fraction at the wet inlet was controlled manually to mimic a step response.

For the dynamic simulations we use time series data of a typical measurement session as input for the humidifier model. We extract the sensor values of the inlet quantities of the humidifier from the measurement data, described in Sec. 2.4, and feed those quantities to real input blocks of the humidifier model. The membrane permeability was kept at $D_{mem} = 3.63e - 7m^2/s$, as identified in the previous section.

In a first step, the time constant for the first order delay was calibrated manually to match the water diffusion over a period of 24000 s. The mean absolute error (MAE) was used as metric to evaluate the calibration

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\dot{m}_{\text{perm},i} - \hat{\dot{m}}_{\text{perm},i}| \tag{30}$$

That MAE was used in the following object function:

$$\tau_{\text{perm,fit}} = \text{argmin}\,\text{MAE}(\vec{X}_{a,b}, \tau_{\text{perm}}) \tag{31}$$

A value of $\tau_{\text{perm,fit}} = 6.1\,\text{s}$ was identified to yield the minimal absolute deviations between measurement and simulation results. At this point it shall be stated, that the dynamics of the measurements do not only originate from the humidifier itself but also from the time constants of the used sensors.

In Fig. 7 an exemplary step response of the water permeation rate caused by an abrupt decrease in the water mass fraction at the wet inlet from $\xi'_{H2O,wet} = 0.083$ to $\xi'_{H2O,wet} = 0$ is shown. All operating conditions of this scenario are given in Tab. 5. The measurement of the permeation rate is shown as a blue line. To show the influence of the introduced time constant, the humidifier model was simulated three times with differently parameterized values of 1.0 s, 6.1 s and 20 s for the time constant. In Fig. 7, it is obvious that a time constant of 20 s leads to a slow response and high deviations from the measurement. Furthermore, it can be observed that the green and blue line are close to each other. A minor deviation can be found in the last moments of the step response. In contrast, the model parameterized with a time constant of 1.0 s reacts too abruptly. Comparing the models to each other, one can see that all models predict the same steady state solution as expected and show a small deviation from the measurement.

Table 5. Boundary conditions before and after step shown in Fig. 7 used for the fitting of the time constant. Temperature and pressure apply to both sides.

Parameter	Before	After
Temperature	70 °C	70 °C
Pressure	1.5 bar	1.5 bar
Air flow rate, dry	0.3 g/s	0.3 g/s
Air flow rate, wet	0.65 g/s	0.6 g/s
Water mass fraction, wet	0.083	0.0

For the validation of the fitted time constant the data of another measurement session is used. Just as for calibration, a step response of the humidifier due to a change of water mass fraction at the wet inlet is also used for validation. The operating conditions are given in Tab. 6. In contrast to the calibration data, the focus lies on a step to higher water mass fractions. In Fig. 8 the measured and simulated water permeation rates are plotted over the time. One can easily identify that the dynamics of opening the vapor control valve are more complex than the closing when comparing Fig. 8 and Fig. 7. In Fig. 8 a first abrupt rise of the permeation rate is visible nearly instantly when the vapor control valve is opened. This first rise is followed by a plateau with a duration of about 30 s. After this phase a less steep rise of the permeation rate is observed. Opening the steam valve affects both the control of the steam generator and storage tank as well as the control of the air supply. This is the reason for the complex dynamics when opening the valve. Again, the results of the three NTU models with different time constants are shown. The measure response of the humidifier depicted in Fig. 8 is accurately reproduced by the NTU model with the fitted time constant. The NTU model with the lowest time constant of 1 s shows an overshoot to the first part of the step, whereas the rising time of the NTU model with the highest time constant is way to long. As discussed for the fitting results, a slight deviation of the steady state results between measurement and all simulation models can be observed.

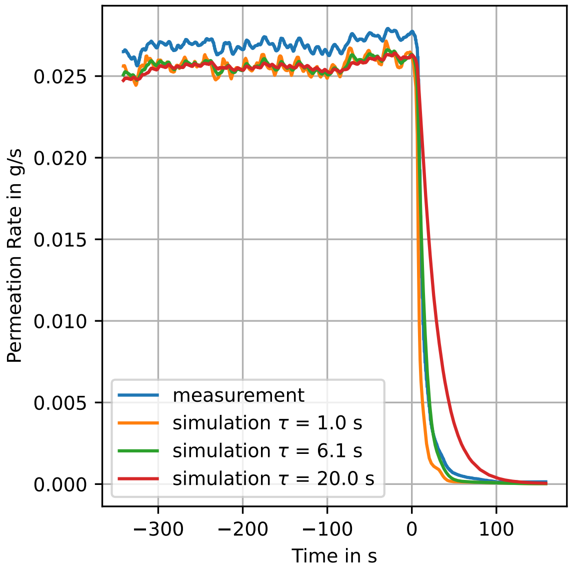


Figure 7. Comparison of the simulation results of differently parameterized NTU models with the measurement data of a step at t = 0 s to dry conditions. The NTU models use different time constants for the introduced first order behavior.

Table 6. Boundary conditions before and after step shown in Fig. 8 used for the validation of the fitted model. Temperature and pressure values apply to both sides.

Parameter	Before	After
Temperature	80 °C	80 °C
Pressure	2.0 bar	2.0 bar
Air flow rate, dry	0.4 g/s	0.4 g/s
Air flow rate, wet	0.4 g/s	0.43 g/s
Water mass fraction, wet	0.0	0.069

In summary, the developed model is suitable for dynamic simulations as well. It was shown that the model is able to reproduce the dynamics of the measured data accurately.

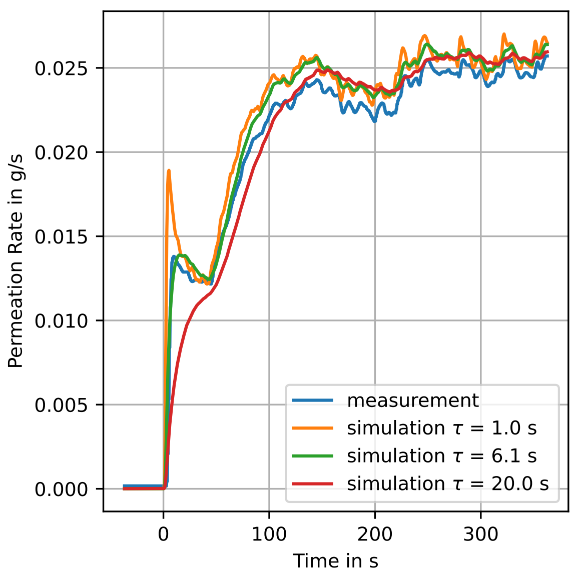


Figure 8. Comparison of the simulation results of differently parameterized NTU models with the measurement data of a step at t = 0 s from completely dry to wet conditions. The NTU models use different time constants for the introduced first order behavior.

4.3 Computational Times

The presented NTU humidifier model can be used for real-time predictions due to its fast computations resulting from the employed simple model structure. When the model is packaged into FMU format the simulation takes an average of 27.37 s computational time to simulate a full time series of 17428 s. The mean computational time to calculate a single time step of 1 s time is 0.00157 s. The calculations were done on an AMD Ryzen Threadripper 1900X.

5 Conclusion and Outlook

The results of our study show that the presented NTU model can accurately predict the water transfer occurring inside a hollow fiber humidifier under steady state as well as for dynamic operating. As a first result, the diffusion coefficient of water in the membrane was determined for the steady state operation. It was found that this fitting result agrees well with the results from a previous study (Pollak et al. 2023). In a next step, the model was fitted to dynamic measurement data while keeping the fitted value of the membrane diffusion coefficient. With the fitted model a validation step response can be predicted accurately. Furthermore, it was demonstrated that our developed model is capable of real time predictions on a desktop computer. For future studies, the effect of a simultaneously occurring heat transfer is an important topic to investigate. Another effect to be investigated in this context is the enthalpy of ad- and desorption of water on the membrane surfaces. Moreover, it should be assessed whether the humidifier model is able to capture the effects of liquid water on mass transfer as investigated by Mull et al. (2023) , that might be present in the exhaust gas of the fuel cell.

Acknowledgments

This work has been supported by both the German Federal Ministry of Education and Research and the German Federal Ministry of Economic Affairs and Climate Action as part of the projects AUTO-GEN (grant number 01IS20086B) and SKAiB (grant number 20M2101F).

Nomenclature

Abbreviations

CFD	Computational Fluid Dynamics
FMU	Functional Mock-up Unit
MAE	Mean absolute error
PEM	Proton Exchange Membrane
RMSE	Root Mean Square Error

Latin Symbols

A	Area, m^2
c	Concentration, $mol/(m^3)$
D	Diffusion coefficient, m^2/s
d	Diameter, m
h	Specific enthalpy, $W/(m^2K)$
$\dot{H}$	Enthalpy flow W
i	Number, *dimensionless*
l	Length, m
M	Molar mass, kg/mol
n	Number, *dimensionless*
NTU	Number of transfer units, dimensionless
$\dot{m}$	Mass flow rate, kg/s
p	Pressure, Pa
R	Mass transfer resistance, s/m^3
Re	Reynolds number, dimensionless
Sc	Schmidt Number, dimensionless
Sh	Sherwood Number, dimensionless
t	Time, s
T	Temperature, K
VFR	Ratio of volume flow rates, dimensionless
$\vec{X}$	State vector

Greek Symbols

β	Mass transfer coefficient, m/s
δ	Thickness, m
η	Mass transfer efficiency dimensionless
ξ	Mass fraction, kg/kg
Ξ	Ratio of transferred water, kg/kg
τ	Time constant, s
Φ	Packing density, dimensionless

Subscript

a	Side a
b	Side b
ch	Characteristic
conv	Convective
eff	Effective
fit	Result of the calibration
H2O	Water
hyd	hydraulic
mem	Membrane
perm	Permeation

Superscript

$\square'$	inlet quantity
$\square''$	outlet quantity

References

Brandau, N., S. Heinke, and J. Koehler (2016-08). “Analysis of mass exchangers based on dimensionless numbers”. en. In: *International Journal of Heat and Mass Transfer* 99, pp. 261–267. ISSN: 00179310. DOI: 10.1016/j.ijheatmasstransfer.2016.03.080.

Cahalan, T. et al. (2017). “Analysis of membranes used in external membrane humidification of PEM fuel cells”. In: *International Journal of Hydrogen Energy* 42.22, pp. 15370–15384.

Chen, Dongmei, Wei Li, and Huei Peng (2008-05). “An experimental study and model validation of a membrane humidifier for PEM fuel cell humidification control”. en. In: *Journal of Power Sources* 180.1, pp. 461–467. ISSN: 03787753. DOI: 10.1016/j.jpowsour.2008.02.055. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378775308003625 (visited on 2023-05-03).

Costello, M.J. et al. (1993-06). “The effect of shell side hydrodynamics on the performance of axial flow hollow fibre modules”. en. In: *Journal of Membrane Science* 80.1, pp. 1–11. ISSN: 03767388. DOI: 10.1016/0376-7388(93)85127-I. URL: https://linkinghub.elsevier.com/retrieve/pii/037673889385127I (visited on 2023-04-28).

Dassault Systèmes SE (2022a). *Dymola 2023x*. https://www.3ds.com/products-services/catia/products/dymola/.

Dassault Systèmes SE (2022b). *FMPy v0.3.12*. https://github.com/CATIA-Systems/FMPy.

FUMATECH BWT GmbH (2019). *ECOMATE Humidifiers*. URL: https://www.fumatech.com/.

Gnielinski, Volker (2010). “G1 heat transfer in pipe flow”. In: *VDI heat atlas*. Springer, pp. 691–700.

Joint Committee for Guides in Metrology (JCGM) (2008-09). *Evaluation of measurement data — Guide to the expression of uncertainty in measurement*. Guideline.

Mull, Sophie et al. (2023-07). “Membrane humidifier model for PEM fuel cell systems”. en. In: *European Fuel Cell Forum 2023*. Lucerne.

Nguyen, Xuan Linh, Hoang Nghia Vu, and Sangseok Yu (2021). “Parametric understanding of vapor transport of hollow fiber membranes for design of a membrane humidifier”. In: *Renewable Energy* 177, pp. 1293–1307. ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2021.06.003. URL: https://www.sciencedirect.com/science/article/pii/S0960148121008673.

Ozen, Dilek Nur, Bora Timurkutluk, and Kemal Altinisik (2016). “Effects of operation temperature and reactant gas humidity levels on performance of PEM fuel cells”. In: *Renewable and Sustainable Energy Reviews* 59, pp. 1298–1306. ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2016.01.040. URL: https://www.sciencedirect.com/science/article/pii/S1364032116000708.

Park, Sang-Kyun, Song-Yul Choe, and Seo-ho Choi (2008). “Dynamic modeling and analysis of a shell-and-tube type gas-to-gas membrane humidifier for PEM fuel cell applications”. In: *International journal of hydrogen energy* 33.9, pp. 2273–2282.

Pollak, Markus et al. (2023). “Analysis of Surrogate Models for Vapour Transport and Distribution in a Hollow Fibre Membrane Humidifier”. en. In: *Energies* 16.6. ISSN: 1996-1073. DOI: https://doi.org/10.3390/en16062578.

TLK-Thermo GmbH (2022). *TIL Suite*. https://www.tlk-thermo.com/index.php/de/software/til-suite.

Virtanen, Pauli et al. (2020). “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Vu, Hoang Nghia, Xuan Linh Nguyen, and Sangseok Yu (2022). “A Lumped-Mass Model of Membrane Humidifier for PEMFC”. In: *Energies* 15.6, p. 2113.

Wu, Di et al. (2020). “Review of system integration and control of proton exchange membrane fuel cells”. In: *Electrochemical Energy Reviews* 3, pp. 466–505.

Yun, Sungho et al. (2018-10). “Numerical analysis on the dynamic response of a plate-and-frame membrane humidifier for PEMFC vehicles under various operating conditions”. en. In: *Open Physics* 16.1, pp. 641–650. ISSN: 2391-5471. DOI: 10.1515/phys-2018-0081. URL: https://www.degruyter.com/document/doi/10.1515/phys-2018-0081/html.

Modeling Components of a Turbine-Generator System for Sub-Synchronous Oscillation Studies with Modelica

Eric Segerstrom[1] Luigi Vanfretti[2] Chetan Mishra[3] Kevin D. Jones[3]

[1]University of Vermont, Burlington, VT, USA, esegerstr@uvm.edu
[2]Rensselaer Polytechnic Institute, Troy, NY, USA, vanfrl@rpi.edu
[3]Dominion Energy, Richmond, VA, USA, {chetan.mishra,kevin.d.jones}@dominionenergy.com

Abstract

In power systems, sub-synchronous oscillations associated with the interaction between a mechanical rotor shaft and electrical system can lead to equipment damage if left unmitigated. This paper describes the development of a scalable, multi-mass torsional shaft model and a synchronous machine model that includes DC offset torque components using Modelica. When coupled, these models can be used to perform shaft torsional studies. Two methods of coupling the shaft with the rest of the turbine-generator system are devised and analyzed. A single-machine, infinite-bus test system using the torsional shaft model and generator model developed in this paper is proposed to observe the penetration of sub-synchronous oscillations throughout an electrical system. The test system is then modified to model sub-synchronous resonance leading to system instability. Analysis of the models described in this paper highlights the value of the `Modelica_LinearSystems2` library in determining the torsional mode shapes and frequencies associated with a turbine-generator system model, which is not feasible with most power system simulation tools.

Keywords: OpenIPSL, power systems, turbine-generator, torsional shaft, SHAF25, GENDCO, sub-synchronous resonance, sub-synchronous oscillations, Modelica_LinearSystems2

1 Background

In the power system dynamic performance analysis of a turbine-generator system, the rotor shaft is generally represented as a single, lumped mass. While this is sufficient for many studies, the rotor of a turbine-generator system is much more complex. A more detailed representation employs several masses to characterize various components along the rotor, such as the turbine blades of different pressure stages, connected by shafts of varying cross-sectional diameters. This representation can aid in understanding the electromechanical dynamics resulting from torsional oscillations occurring between rotor shaft segments, with oscillations below the synchronous frequency translating to potentially disastrous interactions between the electrical system and mechanical rotor if unmitigated. These effects can include sub-synchronous resonance between the generator and series capacitor compensated lines (“Reader’s guide to subsynchronous resonance” 1992) or torsional fatigue and material damage due to accumulated oscillations.

While the study of sub-synchronous resonance has been of practical interest following two shaft failures at the Mohave Generating Station in Nevada in 1970 and 1971 (Walker et al. 1975), challenges associated with these complex dynamics continue to emerge. For example, in July 2015, sub-synchronous oscillations originating from a wind farm in the Xinjiang Uygur Autonomous Region of China excited torsional vibrations in the shaft of a synchronous generator 300 km away. This caused torsional stress relays to trip three large thermal generation units offline and ultimately resulted in a sudden loss of approximately 1,280 MW of power (Shi, Nayanasiri, and Li 2020). In November 2015, torsional vibrations along the rotor shaft of a thermal generation unit in Ha Tinh province in Vietnam were exacerbated by sub-synchronous resonance. The unstable oscillations resulted in several cracks throughout the turbine-generator shaft (Duc Tung, Van Dai, and Cao Quyen 2019). More recently, in an isolated industrial area of Russia, torsional oscillations and sub-synchronous resonance have repeatedly caused protection systems to shut down all operating gas turbine generators in the area and led to widespread outages (Ilyushin and Kulikov 2021).

Incidents such as the ones listed above indicate how important it is to be able to adequately model the behavior of a turbine-generator rotor shaft system when investigating torsional oscillations or sub-synchronous oscillations. While a complete continuum model of the rotor subdivided into dozens of minute masses and connecting shafts is needed to capture the entire range of torsional oscillations, it is generally sufficient to represent the rotor as a lumped, multi-mass model if the oscillations of interest are sub-synchronous (Ong 1998). The oscillations associated with the torsional rotor shaft can also be exacerbated by the DC braking torque effect of the generator. When a symmetrical fault occurs in close proximity to a generator, the sudden disturbance has a tendency to cause the generator rotor to back swing. This effect could influence the angular displacement of the rotor and alter the torsional modes expressed by the rotor shaft (Shackshaft 1970).

DOI
10.3384/ecp204541

While sub-synchronous resonance studies are generally performed in three-phase, electromagnetic transient simulation programs, there are still some instances in which a detailed positive sequence electrical system model with a simplified model of the interactions between the shaft and electrical dynamics is desirable. In these cases, a Modelica-based implementation can provide several beneficial features that are not available in other modeling languages or simulation tools. For example, the `Modelica_LinearSystems2` Library enables linearization and eigenvalue analysis of models to verify the modal quantities of a shaft without requiring the development of a separate model for linear analysis. Additionally, programs such as Dymola include functionality to easily compare the performance of different model implementations.

1.1 Motivation and Objectives

The Open-Instance Power System Library (OpenIPSL) was developed in part to provide Modelica implementations of standard phasor-domain power system models for research and teaching activities with a transparent library development framework. With the recent release of version 3.0.1, several models from the PSS®E Model Library that were missing from previous versions of OpenIPSL were added to the library (DeCastro Fernandes 2023). However, two component models critical to shaft torsional studies and sub-synchronous oscillation studies that are widely used in the industry have yet to be added. These models are a torsional shaft model with up to 25 masses, `SHAF25`, and a round rotor generator with a DC offset torque component, `GENDCO`. By developing Modelica implementations of these models, OpenIPSL can be used to develop power system models with richer dynamics suitable for investigating the effects of sub-synchronous oscillations and sub-synchronous resonance. After additional validation and testing, the models proposed in this paper will be added to a future release of OpenIPSL.

Finally, this work addresses a significant gap in power system dynamic modeling. According to the PSS®E Program Operation Manual, the `SHAF25` model is classified as a turbine-governor model (Siemens 2015a). It is therefore not possible to model turbine governor dynamics and the torsional dynamics of the `SHAF25` model simultaneously. Neglecting the turbine governor of a system restricts analysis of the impact of sub-synchronous resonance to the rotor shaft of the turbine-generator. By using components from OpenIPSL, the modeling approach proposed in this work allows for the dynamics of turbines, boilers, and governors to be modeled alongside the torsional dynamics of the shaft.

To summarize, the objectives of this work were to:

- Develop and validate a scalable, lumped mass torsional shaft model using the Modelica modeling language. This approach will allow for the dynamics of the shaft to be modeled simultaneously with the the dynamics of the turbine, boiler, and governor of the turbine-generator system, which is not possible with existing domain-specific tools.
- Develop and verify the behavior of a round rotor generator with quadratic saturation that includes a DC offset torque component.
- Demonstrate the functionality of the developed models by using an illustrative, single-machine infinite-bus (SMIB) test system to demonstrate the penetration of sub-synchronous resonance throughout a power grid.

1.2 Contributions and Organization

The primary contributions of this work are:

- The proposal and assessment two methods of coupling the electrical and mechanical dynamics of a turbine generator system.
- The development of a flexible implementation for modeling a torsional shaft with any number of masses that. The implementation enables the simultaneous modeling of turbine, boiler, and governor dynamics alongside the dynamics of the shaft.
- The implementation of a synchronous machine model with quadratic saturation and DC offset torque components for shaft torsional studies.
- A demonstration of the usefulness of the Modelica modeling language in developing power system models capable of simulating the effects of sub-synchronous resonance.

The remainder of the paper is organized as follows. Sections 2 and 3 detail the process of modeling and assessing two turbine-generator components respectively: 1) a scalable multi-mass torsional shaft; and 2) a round rotor generator model with a DC offset torque component. Section 4 demonstrates the use of these components to develop an illustrative power system example model to observe the penetration of sub-synchronous resonance throughout the grid. Finally, Section 5 summarizes the contributions of this work with concluding remarks and outlines planned future work for the models described throughout the paper.

2 Scalable Multi-Mass Shaft

2.1 SHAF25 Implementation Using Modelica

The `SHAF25` model found in the PSS®E Model Library (Siemens 2015b) is a lumped-mass torsional shaft model capable of representing up to 25 masses connected by weightless springs. Using the `Modelica Standard Library`, it is straightforward to assemble shaft models for a set number of masses by connecting an alternating sequence of rotational mechanical `inertia` and `springDamper` components. However, to be able to represent shafts with up to 25 masses, this approach would require maintaining a library with 25 separate shaft models. Alternatively, by using concepts from the

ScalableTestSuite Library (Casella et al. 2015), a single shaft model can be developed with the number of masses on that shaft determined by a single integer parameter.

Listing 1 provides an excerpt of the text layer for the multi-mass torsional shaft model showing its scalability. The relevant parameters and components are all instantiated as arrays, and a for loop in the equation section iteratively connects the components in the correct order. As an added benefit, this scalable modeling approach allows the model to represent a shaft with any number of masses, effectively bypassing the 25 mass limit of the original SHAF25 model.

Listing 1. Excerpt of text layer for the scalable multi-mass torsional shaft model illustrating its scalability.

```
model ScalableShaft;
// Parameter declaration
parameter Integer N = 5 "Number of masses";
parameter SIunits.Inertia H[N] "Vector of p
    .u. moment of inertia values";
parameter SIunits.Inertia J[N] = H.*(
    SysData.S_b/wo^2);
parameter SIunits.RotationalSpringConstant
    K[N-1] "Vector of stiffness
    coefficients";
parameter SIunits.RotationalSpringConstant
    C[N-1] = K.*p^2*(SysData.S_b/(4*wo)) "
    Vector of stiffness coefficients in N-m
    /rad";
parameter SIunits.RotationalDampingConstant
     D[N-1] = fill(Modelica.Constants.small
    , N-1) "Vector of damping constant
    values in Nms/rad";
// Class Instantiation
Modelica.Mechanics.Rotational.Components.
    Inertia inertia[N](J=J);
Modelica.Mechanics.Rotational.Components.
    SpringDamper springdamper[N-1](c=C, d=D
    );
Modelica.Mechanics.Rotational.Sensors.
    RelAngleSensor relAngle[N];
// Model Equations
equation
  connect(inertia[1].flange_a, flange_a);
  for i in 1:(N-1) loop
    connect(inertia[i].flange_b,
        springdamper[i].flange_a);
    connect(springdamper[i].flange_b,
        inertia[i+1].flange_a);
    connect(relAngle[i].flange_b, inertia[i
        ].flange_a);
    connect(relAngle[i].flange_a, inertia[i
        ].flange_b);
  end for;
  connect(flange_b, inertia[N].flange_b);
  connect(relAngle[N].flange_b, inertia[N].
      flange_a);
  connect(relAngle[N].flange_a, inertia[N].
      flange_b);
end ScalableShaft;
```

In this work, and many other works in the literature, the damping coefficients of the springDamper components connecting the masses of the shaft are assumed to be negligible. It is generally acceptable to assume that the damped and undamped natural frequencies of the shaft are within 0.1 Hz of each other (**anderson:1999sub-synchronous**). As such, the damping coefficients for the shaft model shown here are all set to an arbitrarily small value, Modelica.Constants.small.

2.2 Model Validation

To verify the behavior of the multi-mass shaft model, eigenvalue analysis was used to observe the relative angular displacement between the segments of a five mass implementation of the component. The five mass shaft was parameterized using data from a four pole nuclear unit found in Section 15.1 of Prabaha Kundur's textbook, *Power System Stability and Control* (Kundur 1994). This procedure is aided and simplified through the use of the Modelica_LinearSystems2 library to generate a linearized electromechanical state-space model of the torsional shaft system. This library is also used to determine the frequencies of the modes exhibited by the system. From this model, it is straightforward to obtain an eigenvector matrix that describes the magnitude of the mode shapes. The resulting mode shapes are shown in Figure 1a. For comparison and validation, Figure 1b shows the mode shapes obtained in *Power System Stability and Control*(Kundur 1994).

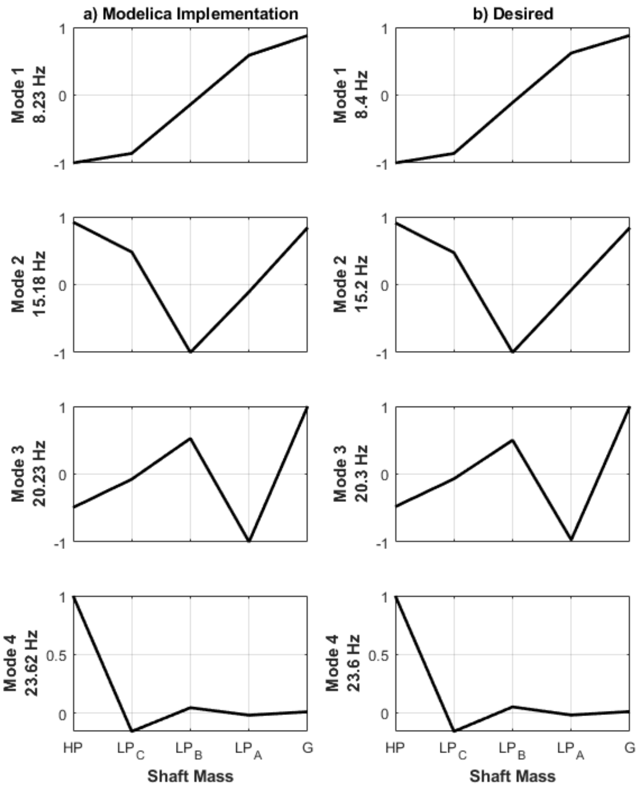


Figure 1. Mode shapes and frequencies obtained from a) the Modelica implementation of a five-mass torsional shaft, and b) the implementation found in *Power System Stability and Control* (Kundur 1994).

After normalizing each of the eigenvectors such that the magnitude of the largest element is 1.0, the overall trajectory of the mode shapes and modal frequencies of the two implementations are nearly identical, indicating that the Modelica multi-mass shaft implementation exhibits the expected behavior. While there are some minor discrepancies in the modal frequencies of the two implementations, this can likely be attributed to initialization differences in the Modelica implementation, as the states' precise initial values of the model from *Power System Stability and Control* are not known.

2.3 Coupling Mechanical Shaft Dynamics with Electrical System Dynamics

As a purely rotational mechanical model in Modelica, the multi-mass shaft has mechanical torque flange inputs and outputs. The PSS®E generator models in OpenIPSL through their extension of a `baseMachine` class, however, only accept real inputs for mechanical power and field voltage. Therefore, two methods were devised to interface the shaft model with the electrical system.

2.3.1 Modified Base Machine Class

The first approach to couple the mechanical and electrical dynamics of the system involved creating a variant of the `baseMachine` class included in OpenIPSL. Each of the PSS®E generator models in OpenIPSL extends the same `baseMachine` class. By altering this class to replace the mechanical power input with a mechanical torque flange input, the torsional shaft model can be simply connected to the rest of the electrical system through the electrical pin of the generator. For comparison, the icon layer of the original base machine and the modified base machine are shown in Figure 2.

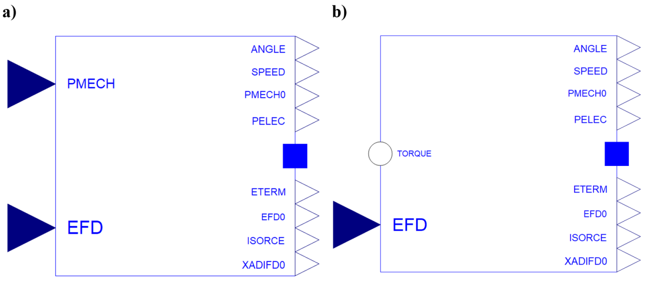


Figure 2. Comparison of the icon layer of a) the original base machine model from OpenIPSL, and b) the modified base machine model to accept a torque input from a torsional shaft model.

To facilitate this connection, two mathematical equations to explicitly calculate the mechanical power input to the generator from the mechanical torque input had to be added to the text layer of the model.

The first equation, shown in Equation 1, converts the per unit speed deviation of the rotor into mechanical speed:

$$\omega_m = \omega_b(1+\omega) \tag{1}$$

where ω_m is the mechanical speed of the rotor in radians per second, ω_b is the synchronous speed of the system in radians per second, and ω is the per unit mechanical speed deviation of the generator rotor.

The second equation, shown in Equation 2 uses this mechanical speed to convert the input mechanical torque into mechanical power:

$$P_m = \frac{\omega_m T_m}{M_b}, \tag{2}$$

where P_m is the per unit mechanical power input to the generator, T_m is the mechanical torque input to the generator from the torsional shaft in Newton-meters, and M_b is the system base power in volt-amperes.

By redefining the variable for mechanical power, the swing equation of the original `baseMachine` model can be left unaltered:

$$2H\frac{d\omega}{dt} = \frac{\omega_m - D\omega}{\omega+1} - T_e \tag{3}$$

2.3.2 Torque-to-Mechanical Power Interface

While the previous method to couple the electrical and mechanical dynamics of the turbine-generator system is relatively straightforward, it would require pervasive changes to OpenIPSL to implement, as individual models for each generator extending the new multi-domain base machine class would have to be developed. This would effectively require two of each generator model to be maintained.

An alternative approach is to create a standalone interface model that accepts a mechanical torque flange input and produces a real mechanical power output as shown in Figure 3 (F. J. Gómez et al. 2018; Aguilera, Vanfretti, Bogodorova, et al. 2019; Aguilera, Vanfretti, and F. Gómez 2018).

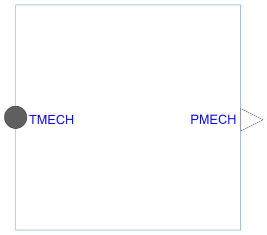


Figure 3. Icon layer of an interface to couple the mechanical dynamics of a torsional shaft with the electrical dynamics of a turbine-generator system.

Using this method, the per unit speed deviation of the rotor must be explicitly calculated by taking the derivative of the angular position of the shaft connected to the input of the interface:

$$\omega = \frac{d\phi}{dt}\frac{1}{\omega_b} \tag{4}$$

where ω is the per unit mechanical speed deviation of the rotor, ϕ is the relative angular position of the rotor in radians, and ω_b is the synchronous speed of the system in radians per second.

The per unit mechanical speed deviation can then be used to calculate the mechanical speed of the rotor:

$$\omega_m = \omega_b(1+\omega) \tag{5}$$

where ω_m is the mechanical speed of the rotor in radians per second.

Finally, the mechanical speed of the rotor and torque input to the interface can be used to determine the mechanical power input to the generator:

$$P_m = \frac{\omega_m T_m}{M_b} \tag{6}$$

where P_m is the per unit mechanical power output from the interface, T_m is the mechanical torque input to the generator from the torsional shaft in Newton-meters, and M_b is the system base power in volt-amperes.

2.3.3 Comparison of Implementations

For comparison, Figure 4 shows the graphical layers for both methods of coupling the mechanical and electrical dynamics of a turbine-generator system. Both implementations include a representation of the boiler, turbine, and speed-governor modeled by an `IEEEG1` component; an IEEE Type 1 excitation system modeled using an `IEEET1` component; and a round rotor synchronous generator with quadratic saturation represented by a `GENROU` model.

To compare the efficacy of the two methods of interfacing the shaft model with the remainder of the turbine-generator model, the two models were initially simulated with an identical simulation configuration for 30 seconds using `DASSL`, a variable time-step solver. As shown in Figure 5, the error of the mechanical power input to the generator calculated between the two implementations appears to accumulate throughout the simulation. Using `Rkfix4`, a fixed step, fourth-order Runge-Kutta method solver, however, the mechanical power plotted for both implementations were identical, as shown in Figure 6.

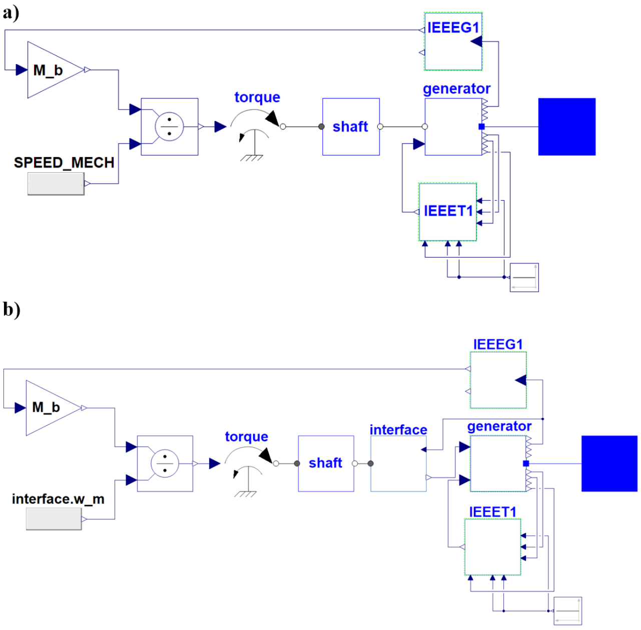


Figure 4. Comparison of the graphics layer of two methods to interface a torsional shaft model with an electrical system: a) A modified base machine directly accepts mechanical torque from the shaft and internally calculates mechanical power, b) An interface accepts a mechanical torque input from the shaft and produces a mechanical power output for the generator. Both approaches notably allow for a turbine-governor model, represented by the `IEEEG1` model, to be simulated in conjunction with the shaft model, which is not possible with many power system simulation tools.

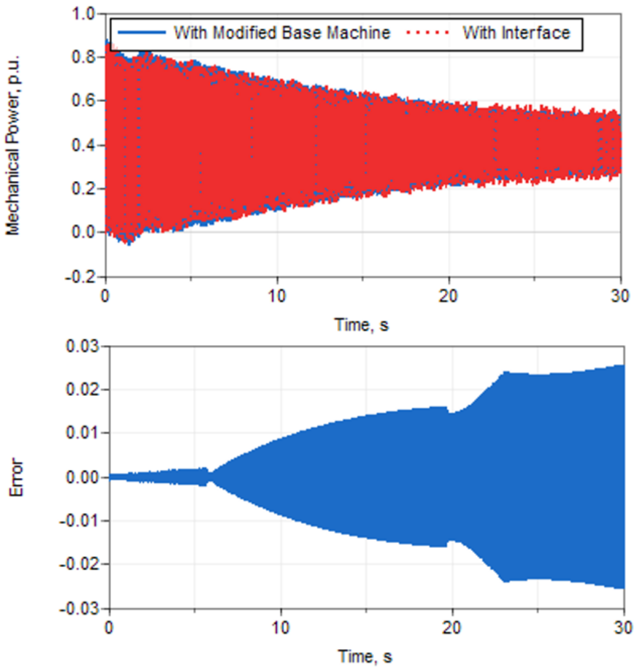


Figure 5. Plot of mechanical power input to the generator for both methods of interfacing the shaft model with the remainder of the turbine-generator model when simulated using the DASSL solver in Dymola and the error between the two.

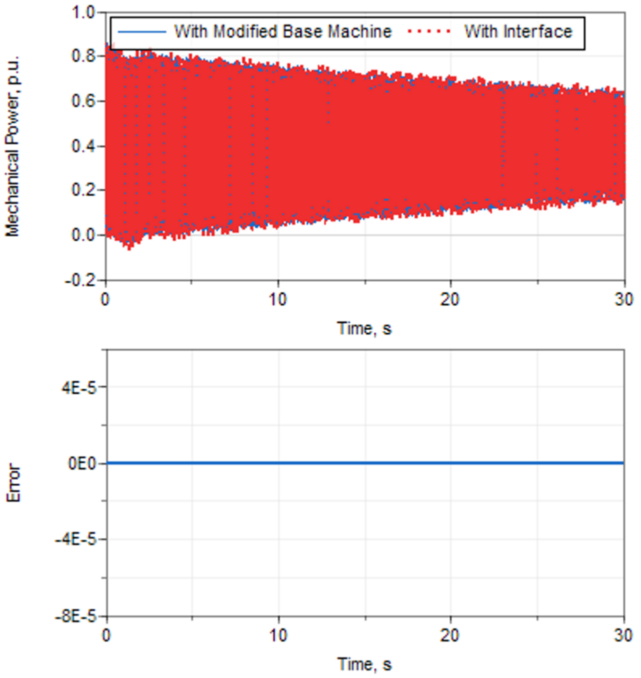


Figure 6. Plot of mechanical power input to the generator for both methods of interfacing the shaft model with the remainder of the turbine-generator model when simulated using the Rkfix4 solver with an integration step size of 0.00001 s in Dymola and the error between the two.

Generally, solutions obtained from variable-step solvers are more accurate, as the solver dynamically adjusts the size of the integration step size to match the speed at which the states of the model change. Fixed-step solvers, however, maintain the same integration step-size throughout the simulation. With a large step size, the simulation time can be greatly reduced, however, the accuracy of solutions may suffer as the solver cannot adjust its step size to match the stiffness of the system.

While the two implementations are modeled using the same equations and should therefore, in theory, produce identical solutions. However, the integration step size when using the DASSL solver does not vary identically for the implementations. As such, the solutions acquired by both implementations are not identical, as observed in Figure 5. When the same step size is enforced for both models, however, the solutions become identical, as shown in Figure 6. While the solution results may be identical for the two implementations, the accuracy of this solution is not guaranteed. Figure 7 shows a plot of the mechanical power for both implementations using an integration step size of 0.01 seconds.

While the plots of mechanical power from implementations agree with each other, it can be inferred that they are both inaccurate with respect to the actual result by comparing the resulting plot with the plots acquired from a variable-step solver or a fixed-step solver with a very small step size.

By enabling the `Generate block timers` flag in the simulation setup dialog of Dymola, the simulation

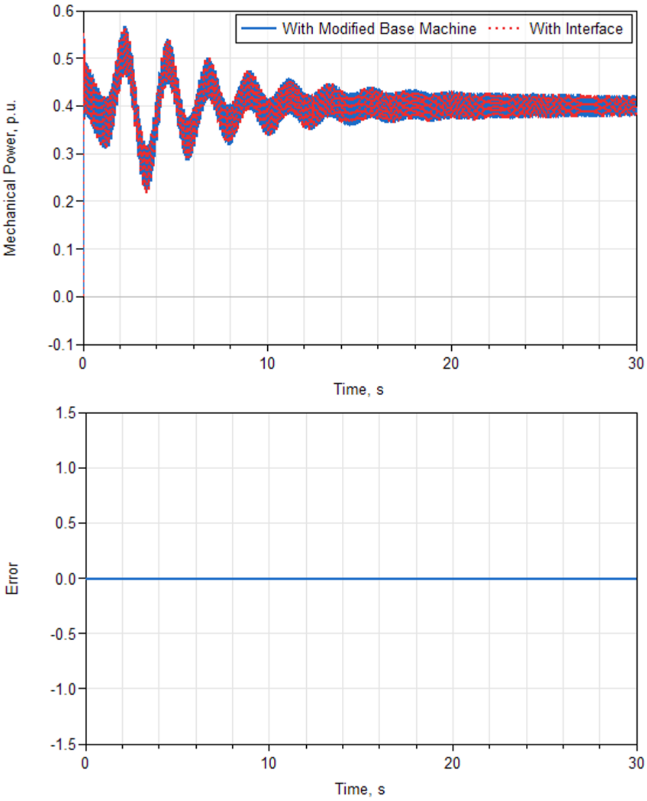


Figure 7. Plot of mechanical power input to the generator for both methods of interfacing the shaft model with the remainder of the turbine-generator model when simulated using the Rkfix4 solver with an integration step size of 0.01 s in Dymola and the error between the two.

time of the two implementations can be compared as shown in Table 1. Using the `Rkfix4` solver, the range of time steps that both implementations could successfully complete simulation for was between 0 seconds and 0.01 seconds. For each time step, it can be observed that the modified base machine implementation is slightly more computationally efficient. Using the variable time-step `DASSL` solver results in an even greater discrepancy between the computational cost of the two implementations, with the modified base machine method remaining the more efficient option.

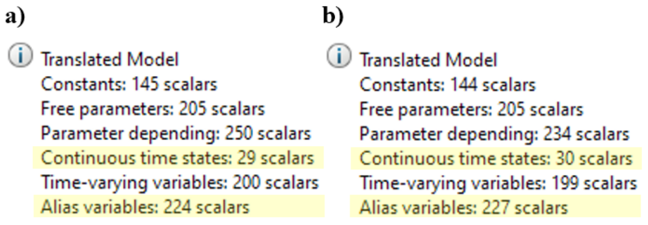
The discrepancy in simulation time between the two implementations can be explained by examining the statistics of the translation log in Dymola, shown in Figure 8.

When translating a model, Dymola uses its state variables to create differential, linear, and non-linear systems to solve. As such, models with more states generally will take longer to simulate (Fish and Harrison 2017). The number of states can be approximated by examining the `Time-varying variables` and `Alias variables` entries of the translation statistics log (Horn 2020). Also of interest is the `Continuous time states` entry of the log which indicates the overall size

Table 1. Comparison of simulation time for two methods of coupling mechanical and electrical dynamics of a turbine-governor system using the Rkfix4 solver with various time steps and the DASSL solver.

Solver	Step Size (s)	Coupling Method	Simulation Time (s)
Rkfix4	0	Base Machine	328.835
		Interface	341.871
	0.0001	Base Machine	134.925
		Interface	135.096
	0.01	Base Machine	1.391
		Interface	1.638
DASSL	–	Base Machine	75.946
		Interface	80.161

of the model. From Figure 8 it can be observed that the modified base machine implementation contains fewer alias variables and continuous time states than the torque-to-power interface implementation, corresponding with the former's faster simulation time.



a)

ⓘ Translated Model
Constants: 145 scalars
Free parameters: 205 scalars
Parameter depending: 250 scalars
Continuous time states: 29 scalars
Time-varying variables: 200 scalars
Alias variables: 224 scalars

b)

ⓘ Translated Model
Constants: 144 scalars
Free parameters: 205 scalars
Parameter depending: 234 scalars
Continuous time states: 30 scalars
Time-varying variables: 199 scalars
Alias variables: 227 scalars

Figure 8. Comparison of excerpts from the translation statistics logs in Dymola for a) the modified base machine implementation, and b) the torque-to-power interface implementation. Highlighted statistics affect the simulation time of the implementations.

3 GENDCO

When performing power system studies involving shaft torsional dynamics in PSS®E, the SHAF25 torsional shaft model must be coupled with a GENDCO generator model (Siemens 2015c). The GENDCO model is a round rotor generator model with quadratic saturation and DC offset torque components included. To model the effect of these DC offset components, the direct- and quadrature-axis armature-winding voltage equations of the generator were modified. The original equations used in the GENROU model are shown in Equation 9 and Equation 10 (Baudette et al. 2018):

$$v_d = -R_a i_d - \Psi_q \tag{7}$$

$$v_q = -R_a i_q + \Psi_d \tag{8}$$

where R_a is the machine armature resistance, i is the direct- or quadrature-axis current, and Ψ is the direct- or quadrature-axis stator flux linkage. The modified equations for the GENDCO model include the rate of change of stator flux linkages as shown in Equation 9 and Equation 10 (Dandeno et al. 2003):

$$v_d = -R_a i_d - \Psi_q + \frac{1}{\omega_0}\frac{d\Psi_d}{dt} \tag{9}$$

$$v_q = -R_a i_q + \Psi_d + \frac{1}{\omega_0}\frac{d\Psi_q}{dt} \tag{10}$$

where ω_0 is the synchronous electrical speed. These equations assume that the rotor speed never deviates from the synchronous rotor speed.

For GENROU and other generator models that ignore the effects of DC offset components, the air-gap torque of the generator following a disturbance will consist primarily of a unidirectional step change caused by stator resistance losses (Kundur 1994). For the GENDCO model, however, an additional decaying oscillatory component representing the DC offset component of current induced in the stator by the disturbance will be present. Figure 9 illustrates the difference in the air-gap torque response of a GENROU generator model from OpenIPSL that neglects the rate of change of stator flux linkages in the armature-winding voltage equations and the GENDCO generator model that includes them.

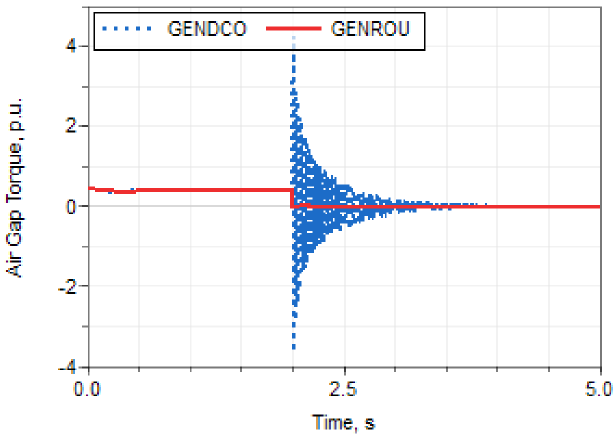


Figure 9. Plot comparing air-gap torque over time for identically parameterized GENROU and GENDCO models. A fault is applied at t = 2 s for 0.15 s.

The additional oscillatory component in the air-gap torque response of the GENDCO generator model can also be confirmed through linear eigenvalue analysis. Table 2 shows the pole pair and modal frequency obtained through linear analysis using the

`Modelica_LinearSystems2` library in Dymola for a simple test system including the `GENROU` generator model. Table 3 shows the pole pairs and modal frequencies for the same test system when the generator model was changed to the `GENDCO` model. In comparing the two tables, it can be observed that as a result of the DC offset components, the `GENDCO` model contains an additional, higher frequency pole while retaining the same 0.814 Hz pole exhibited by the `GENROU` model.

While the `GENDCO` model and the inclusion of DC offset components is critical in obtaining the correct dynamic characterization of any sudden changes in the electromagnetic torque of the generator, it is important to note that DC offset approximation effects are only valid when analyzing the effects of symmetrical faults and imbalances. Consequently, the model should only be used for studies involving shaft torsional dynamics (Siemens 2015c).

Table 2. Pole and Modal Frequency of `GENROU` generator model

Eigenvalue	Frequency (Hz)
$-0.26835 \pm j5.1092$	0.8143

Table 3. Poles and Modal Frequencies of `GENDCO` generator model

Eigenvalue	Frequency (Hz)
$-0.2684 \pm j5.1087$	0.8142
$-0.021085 \pm j5605.7$	892.7985

4 Example of Subsynchronous Resonance Analysis

The interaction between the sub-synchronous oscillations created by the torsional shaft of a turbine-generator system and the rest of the electrical system can lead to equipment failure and damage due to torsional fatigue if improperly damped (Walker et al. 1975). As such, it is important to be able to analyze how the resonance resulting from these interactions can penetrate into the power system.

As a preliminary investigation into how the models developed in this paper can be applied to sub-synchronous resonance studies, a single-machine, infinite-bus (SMIB) test system was developed as shown in Figure 10.

The generator unit on the left hand side of the test system in Figure 10 models a turbine-generator consisting of a six-mass implementation of the torsional shaft model described in Section 2 and the `GENDCO` model described in Section 3. The `GENCLS` generator model at the far-right of the test system is parameterized as an infinite bus, representing the reminder of the power grid. A fault was configured to be applied to the system at $t = 2$ seconds and cleared at $t = 2.15$ seconds. The turbine-generator system and lines were parameterized using parameters from the IEEE first benchmark model for computer simulation of sub-synchronous resonance (Farmer 1977). With these parameters, the turbine-generator system is able to return to a stable state following a disturbance. Figure 11 shows the air-gap torque response of the `GENDCO` model in the test system when the torsional shaft model is included in the turbine-generator system and when it is omitted.

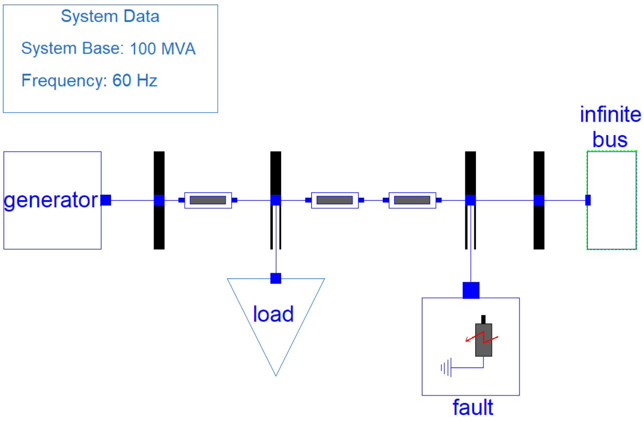


Figure 10. Single-machine, infinite-bus test system used to analyze the penetration of sub-synchronous oscillations throughout an electrical system.

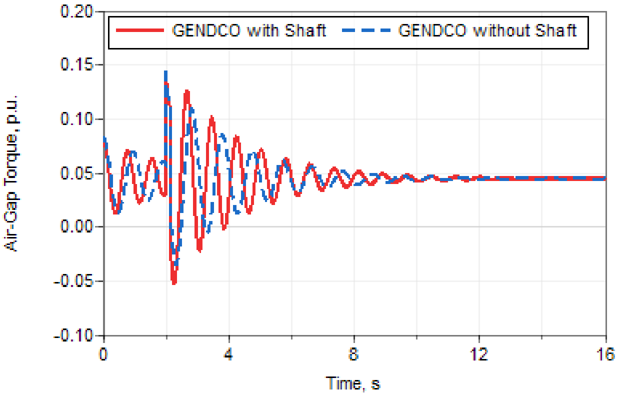


Figure 11. Plot of the air-gap torque of the test system turbine-generator model when the torsional shaft model is omitted and included.

The response of the two model variants of the system are similar, with a slight discrepancy in the frequency of the oscillations preceding and following the fault being the primary difference. To further illustrate this difference, Figure 12 shows a detailed view of the air-gap torque response of the test system after the turbine-generator system has returned to a steady state after the fault was cleared. The turbine-generator system with the shaft omitted exhibits an approximately 60 Hz oscillation, corresponding with the expected behavior due to the effect of the DC offset components. The turbine-generator system with the shaft included contains a sum of several other oscillations, representing the torsional modes of the shaft.

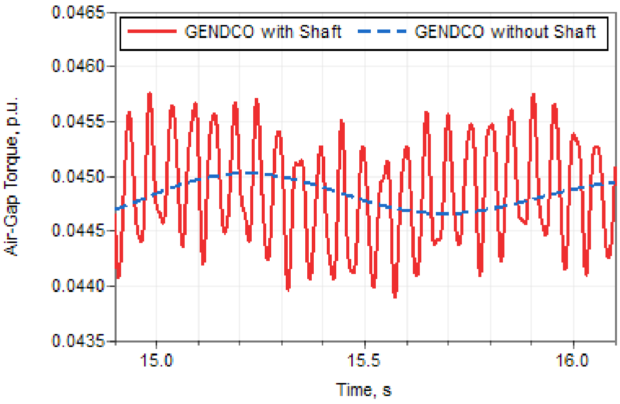


Figure 12. Detailed view of the steady-state portion of the air-gap torque response of the test system following a fault when the torsional shaft is included and omitted from the turbine-generator system.

The time constant controlling the decay of the DC offset torque components roughly corresponds to the network resistance-to-reactance ratio as seen by the generator (Siemens 2015c). As such, if this ratio is excessively large, the effects of sub-synchronous resonance may cause the system to become unstable. Figure 13 compares the bus voltage at the location of the fault in the test system for the line parameters used in the IEEE first benchmark model (Farmer 1977) and for arbitrary line parameters that greatly increased the ratio of resistance-to-reactance seen by the generator. The bus voltage response is similar for the two sets of line parameters before and during the fault, however, following the fault, if the resistance-to-reactance ratio is excessively large, the oscillations following the fault will persist as shown by the unstable case in Figure 13.

5 Conclusions and Future Work

The scalable, multi-mass torsional shaft model and the GENDCO synchronous machine model with DC offset torque components developed in this paper enable a flexible method for performing shaft torsional studies using Modelica. The torsional shaft model enables the ability to model the mechanical dynamics of any number of boilers, turbine pressure stages, and governor simultaneously. Two methods for coupling the shaft model to the rest of the turbine-generator system were explored. While modifying the OpenIPSL base machine class of the generator model to accept a mechanical torque input was shown to be more computationally efficient, an interface to convert the mechanical torque from the shaft to mechanical power input to the generator model would require less pervasive changes to the library to maintain. When coupled with the torsional shaft model, the GENDCO generator model with DC offset torque components enables modeling sudden changes in the electromagenetic torque of the generator more accurately for studies involving shaft torsional dynamics. By employing these models in an SMIB power system model, the effects of sub-synchronous oscillations can be observed. Consequently, adequate line parameters and lenient operating conditions for the modeled system can be determined to limit the impact of sub-synchronous resonance.

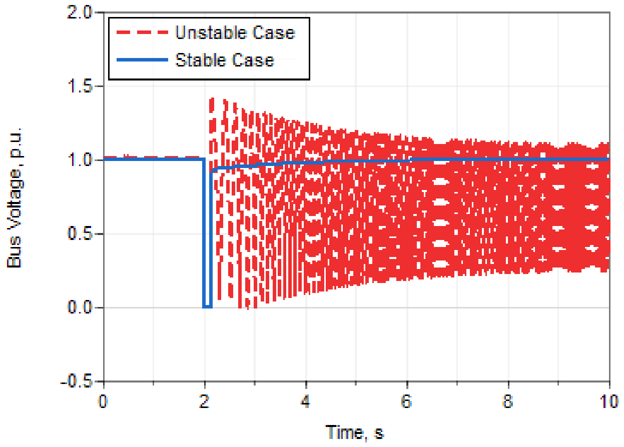


Figure 13. Plot of the bus voltage at the location the fault was applied to when the test system was parameterized with two different sets of transmission line parameters.

Future work includes validating the behavior of the GENDCO model developed in this paper. There is currently no openly accessible PSS®E dynamic model parameter data for an existing GENDCO unit. However, if this data was obtained, software-to-software validation could easily be performed to confirm that the Modelica implementation of the model behaves identically to the equivalent model in PSS®E. Additionally, while the IEEE first benchmark model for the computer simulation of sub-synchronous resonance was used to parameterize the test system developed in Section 4, implementing the first and second benchmark models in Modelica to further explore the effects of sub-synchronous resonance on an electrical system remains the subject of future work (Farmer 1977; Farmer 1995). Finally, the torsional shaft model developed in this paper assumes negligible damping throughout the shaft. The damping of torsional oscillations is generally very small, but difficult to predict without performing field tests on a specific shaft (Kundur 1994). By obtaining PSS®E dynamic data for an existing SHAF25 unit, the viscous damping of each mass with respect to the rotor and the damping between each mass could be accurately modeled in the equivalent Modelica implementation. Upon the completion of these tasks, the integration of the torsional shaft model and GENDCO model into OpenIPSL will be pursued.

Acknowledgements

This work was supported in part by Dominion Energy, the National Science Foundation Award No. 2231677, and the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office, Award No. DE-EE0009139.

The authors would like to thank Dr. Leonardo Lima of Kestrel Power and Mr. James Feltes of Siemens PTI for their modeling guidance.

References

Aguilera, Miguel, Luigi Vanfretti, Tetiana Bogodorova, et al. (2019). "Coalesced gas turbine and power system modeling and simulation using Modelica". In: *Proceedings of the American Modelica Confererence 2018*. Vol. 154. 10. Linköping University Electronic Press, pp. 93–102. DOI: 10.3384/ecp19157617.

Aguilera, Miguel, Luigi Vanfretti, and Francisco Gómez (2018). "Experiences in power system multi-domain modeling and simulation with modelica & FMI: The case of gas power turbines and power systems". In: *2018 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. IEEE, pp. 1–6.

Baudette, Maxime et al. (2018). "OpenIPSL: Open-instance power system library—update 1.5 to "iTesla power systems library (iPSL): A Modelica library for phasor time-domain simulations"". In: *SoftwareX* 7, pp. 34–36.

Casella, Francesco et al. (2015). "Simulation of large-scale models in modelica: State of the art and future perspectives". In: *Linköping electronic conference proceedings*, pp. 459–468.

Dandeno, P et al. (2003). "IEEE guide for synchronous generator modeling practices and applications in power system stability analyses". In: *IEEE Std*, pp. 1110–2002.

DeCastro Fernandes, Marcelo et al. (2023-02). "Version [OpenIPSL 2.0.0] - [iTesla power systems library (iPSL): a Modelica library for phasor time-domain simulations]". In: *SoftwareX* 21, p. 101277. DOI: 10.1016/j.softx.2022.101277.

Duc Tung, Doan, Le Van Dai, and Le Cao Quyen (2019). "Subsynchronous resonance and FACTS-novel control strategy for its mitigation". In: *Hindawi Journal of Engineering* 2019. ISSN: 2314-4904. DOI: 10.1155/2019/2163908.

Farmer, R. G. et al. (1977). "First benchmark model for computer simulation of subsynchronous resonance". In: *IEEE Transactions on Power Apparatus and Systems* 96.5, pp. 1565–1572.

Farmer, R. G. et al. (1995). "Second benchmark model for computer simulation of subsynchronous resonance". In: *IEEE Trans. on Power Apparatus and Systems* 104.5, pp. 1057–1066.

Fish, Garron and Sas Harrison (2017). *Introduction to the model translation and symbolic processing*. Ed. by Claytex. URL: https://www.claytex.com/tech-blog/model-translation-and-symbolic-manipulation/.

Gómez, Francisco J. et al. (2018). "Multi-domain semantic information and physical behavior modeling of power systems and gas turbines expanding the common information model". In: *IEEE Access* 6, pp. 72663–72674. DOI: 10.1109/ACCESS.2018.2882311.

Horn, Nate (2020). *The Dymola translation log*. Ed. by Claytex. URL: https://www.claytex.com/blog/the-dymola-translation-log/.

Ilyushin, Pavel V. and Aleksandr L. Kulikov (2021). "On the occurrence of subsynchronous torsional oscillations of gas turbine units in an isolated energy area with an industrial load". In: *International Ural Conference on Electrical Power Engineering*.

Kundur, Prabha S. (1994). *Power system stability and control*. McGraw-Hill Education.

Ong, Chee-Mun et al. (1998). *Dynamic simulation of electric machinery: using MATLAB/SIMULINK*. Vol. 5. Prentice hall PTR Upper Saddle River, NJ.

"Reader's guide to subsynchronous resonance" (1992). In: *IEEE Transactions on Power Systems* 7.1, pp. 150–157. DOI: 10.1109/59.141698.

Shackshaft, G. (1970). "Effect of oscillatory torques on the movement of generator rotors". In: *Proceedings of the Institution of Electrical Engineers*. Vol. 117. 10. IET, pp. 1969–1974.

Shi, Tong, Dulika Nayanasiri, and Yunwei Li (2020). "Subsynchronous oscillations in wind farms – an overview study of mechanisms and damping methods". In: *IET Renewable Power Generation* 14.19, pp. 3974–3988.

Siemens, PTI (2015a). "Program operation manual of PSS/E-34". In: *Schenectady, NY, USA*.

Siemens, PTI (2015b). "PSS/E 34 model library". In: *Siemens PTI: Schenectady, NY, USA*.

Siemens, PTI (2015c). "PSS/E 34.0 program application guide". In: *Siemens PTI: Schenectady, NY, USA*.

Walker, D.N. et al. (1975). "Results of subsynchronous resonance test at Mohave". In: *IEEE Transactions on Power Apparatus and Systems* 94.5, pp. 1878–1889.

`DroneLibrary`: Multi-domain Drone Modeling in Modelica

Meaghan Podlaski[1] Luigi Vanfretti[2] Dietmar Winkler[3]

[1]GE Research, Niskayuna, NY, United States, meaghan.podlaski@ge.com
[2]Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY, USA, vanfrl@rpi.edu
[3]Department of Electrical Engineering, Information Technology and Cybernetics, University of South-Eastern Norway, Porsgrunn, Norway, dietmar.winkler@usn.no

Abstract

In the development of complex, novel electrified aerial systems such as Unmanned Aerial Vehicles (UAVs) and electric vertical take-off and landing (eVTOL) systems, multi-domain modeling and simulation studies can provide indispensable insight on system design and performance. In this paper, a Modelica library that can be used to model multi-domain drones is introduced. This library models a drone in the electrical, mechanical, and control domains, with examples for applications such as battery-power analysis, virtual reality simulation and user interaction. *Keywords: drone, quadcopter, eVTOL, UAV*

1 Introduction

The `DroneLibrary` was created in response to the growing need for simulation-based studies in Unmanned Aerial Vehicle (UAV) drone and electric vertical take-off and landing (eVTOL) electric aircraft for Urban Air Mobility (UAM) (Doo et al. 2021). Multi-engineering domain, simulation-based studies are valuable in determining which design concepts and methods for eVTOL systems can best meet design requirements and specifications (Podlaski, Niemiec, et al. 2021; Podlaski, Vanfretti, et al. 2022). There are limited opportunities for creating and testing physical prototypes for such systems due to time and monetary constraints. These systems also have limitations on which physical device qualities are recorded during testing, making well-defined reliable models valuable at early design stages for eVTOL systems, such as those seen in conventional fixed-wing distributed electrical propulsion physical prototype systems (Borer et al. 2016)

To bridge this gap, the authors' prior efforts in this area (M. Podlaski 2020; M. Podlaski 2021) have resulted in a Modelica library that includes multi-domain models to represent each subsystem of a quadcopter, specifically focusing on the mechanical, electrical, and control domains. This library contains examples of the quadcopter model at multiple levels of detail under different operating conditions. The component models are developed in a manner to easily replace them for different simulation applications, creating replaceable models that are easy to maintain with broad application scope.

1.1 Related Works

This library builds off of the works in (M. Podlaski 2020), which show initial models and simulation results for the `DroneLibrary`. The aerodynamic and mechanical behavior of these models are derived from (Bresciani 2008) and (Luukkonen 2011).

System-level drone models created using Modelica have previously been discussed in (Kuric, Osmic, and Tahirovic 2017). However, these systems mainly focused on multirotor aerial vehicle (MAV) dynamics modeling while assuming ideal and constant power consumption. All dynamics in this MAV Modelica model are reduced to a single domain, linear model. In this library, non-idealities in the electrical system are introduced.

A drone PID controller developed using Modelica is discussed in (Ma, Li, and Nae 2016). This model assumes that the drone body is rigid and symmetrical with the force of each propeller proportional to the square of the angular speed of the propeller. The models in the `DroneLibrary` build off the ideal assumptions made in this work by adding flexibility to accommodate different airframe structures through the multibody mechanics.

In contrast with the aforementioned previous works that aims in modeling system dynamic aspects of specific drone or eVTOL systems, the work in (Coïc, Budinger, and Delbecq 2022) targets drone sizing and trajectory optimization aspects. While the library proposed could be used for these purposes too, this is out of the scope of the present paper.

Finally, it is necessary to note that the present work has its origin as a course project by Hao Chang in 2018 at Rensselaer Polytechnic Institute that was the basis of the start of the `DroneLibrary`. At that time there were no commercial or open source libraries offering the capabilities that have been now consolidated into the `DroneLibrary`. To the authors' knowledge, the `DroneLibrary` is the only open source license free library with the described scope of applications. In the case of commercial and proprietary libraries, with the release of Dymola 2022, Dassault Systèmes included the Aviation Systems Library (AVL), which offers a similar scope for multi-copter and extends it to fixed wing aircraft. Similarly, Claytex released a UAV Dynamics Library in 2020[1],

[1] There is limited publicly available information about these li-

DOI
10.3384/ecp204551

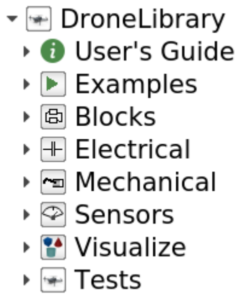
that now has a similar scope and application areas as the AVL[2]. While these commercial libraries have a broader scope, the DroneLibrary provides an open-source option that reduces the barrier of entry for drone modeling and can be seen as a starting point before working with these specialized libraries if the need arises.

2 Library Overview

The package structure of the DroneLibrary is shown in Figure 1. Test cases of the quadcopter are located in the Examples package. The Blocks package has models used to create the controllers and input signal commands for the quadcopter. The Electrical and Mechanical contain models at varying levels of modeling fidelity representing the electrical and mechanical components used in the drone, such as motors, chassis frames, and power sources. The Sensors package contains sensor models used to track the drone's location and acceleration during flight to assist in the control. The Visualization package utilizes the DLR Visualization library to simulate the quadcopter in interactive virtual reality environments. This package contains multiple examples for interacting with the drone model via hardware-in-the-loop (HIL) using keyboard and joystick commands.

The library is dependent on the other following libraries:

- Modelica Standard Library, v4.0.0 (Modelica Association 2023)
- DLR Visualization Library Professional Edition, v1.6.0 (Hellerer, Bellmann, and Schlegel 2014), visualization examples only.
- Modelica Device Drivers Library, v2.1.1 (Thiele et al. 2017), visualization examples only.



DroneLibrary
User's Guide
Examples
Blocks
Electrical
Mechanical
Sensors
Visualize
Tests

Figure 1. DroneLibrary package structure.

3 Examples

All of the components described herein are configured to create the simple quadcopter model shown in Figures 2 and 3. Figure 2 shows the system when an ideal power source is used, while Figure 3 shows the system with a battery power source. The quadcopter chassis in Figure 3 also has the frame_a1 connector that can be linked to additional external payloads, such as a camera.

These quadcopter variants are tested in models configured in the Examples package. The examples include using different input signals to control the inputs xcoord, ycoord, and zcoord, which can be signals provided in the DroneLibrary, Modelica Standard Library, experimental data, and custom signal functions defined by the user. The inputs for the quadcopter model can also be left disconnected from any inputs and compiled as a Functional Mock-up Unit (FMU). By selecting this option, the model can be exported to other software tools for analysis and simulation.

4 Electrical Models

The models for the drone's motors and power sources are located in the Electrical package.

4.1 Source Models

The Modelica Standard Library includes a battery stack model that is included in the Examples.DroneWithoutIdealPower package, which is used when a battery model is desired.

The battery model is configured to interface with the rest of the drone's power system using two different power electronic typologies. One of these topologies is shown in Figure 4, where the battery model in the Modelica Standard Library is connected to a step-down DCDC converter to provide power to the drone's main controller unit (MCU). The other topology in the package includes an additional set of electrical pins to connect the battery to each of the drivetrains.

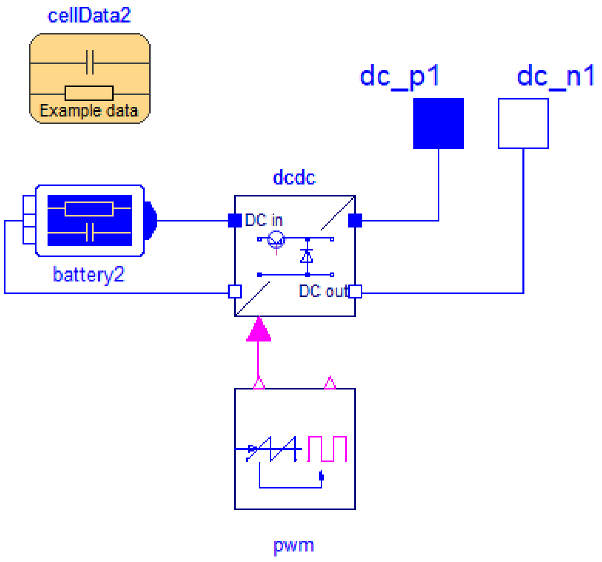


Figure 4. Battery with DCDC converter to interface with drone controller.

In the battery topology that connects the batteries to

braries, a description of the AVL is available online at: https://tinyurl.com/DS-AVLib and for the UAVDL at: https://tinyurl.com/CT-UAVDLib

[2] See: https://tinyurl.com/CT-UAVDLibUpgrade

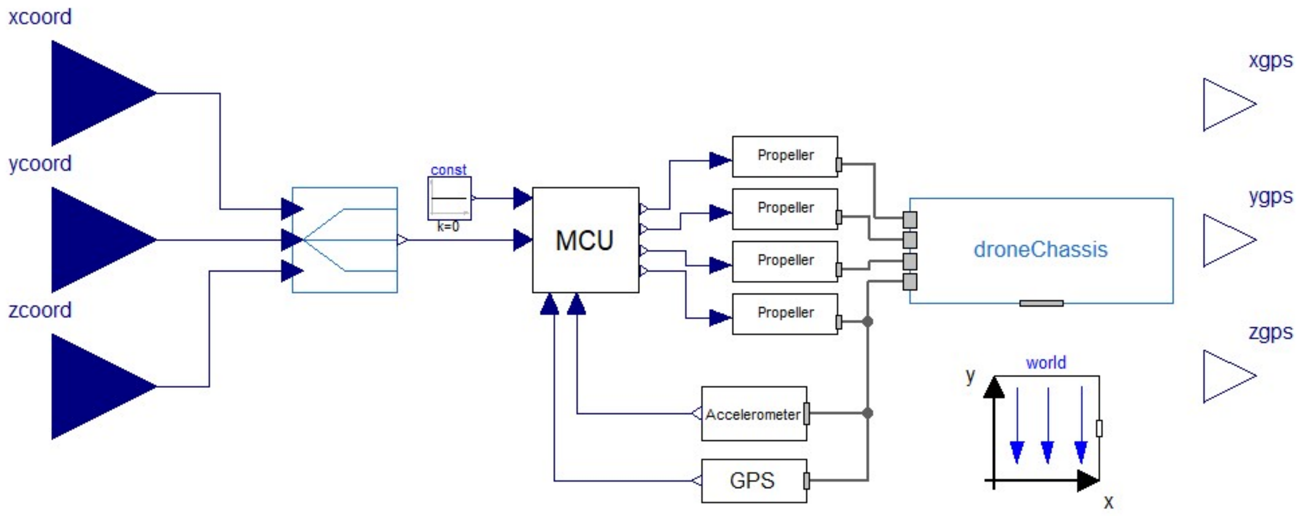


Figure 2. Quadcopter model used in example cases.

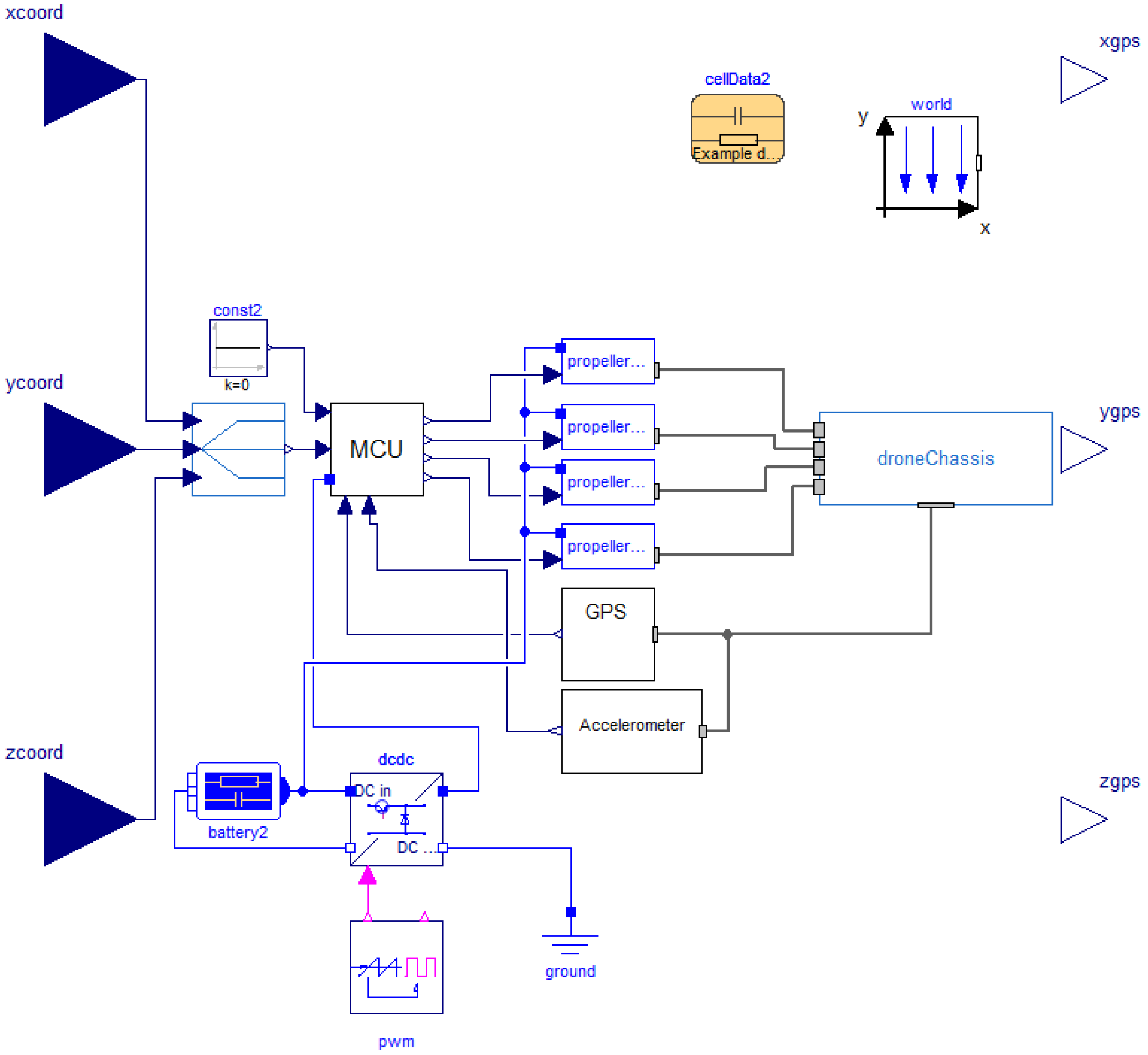


Figure 3. Complete drone model consisting of propellers, motor, controller, and chassis with battery power system. Inputs come from x, y, and z coordinate location commanded by the user.

each of the drivetrains, additional control to scale the voltage applied to the drivetrain is necessary. This is dependent on the SOC of the battery as well as the signal from the MCU. This component, `PowerControl`, is shown in Figure 5.

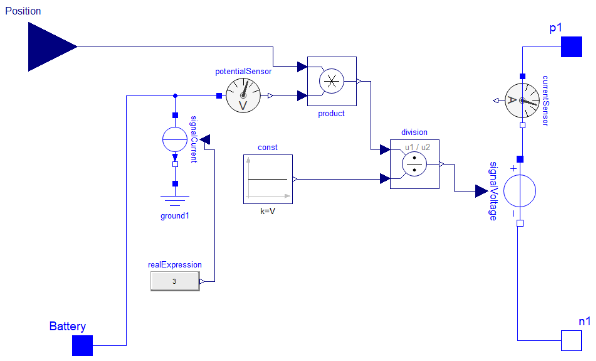


Figure 5. Control component to scale voltage applied to drivetrain based on battery SOC and MCU signal.

4.2 Machine Models

The library contains multiple machine models at varying degrees of modeling fidelity. The simplest representation of the motor is defined as `SimpleDCMotor`, which follows Equations (1)-(3) below. This model only considers torque τ, linear force f, motor speed ω, and current i to control the motor with no electrical, thermal, or mechanical losses considered in the model.

$$\tau = K_\tau \cdot i \tag{1}$$

$$J_p \frac{d\omega}{dt} = \tau - b \cdot \omega \tag{2}$$

$$f = K_f \cdot \omega \tag{3}$$

Most quadcopter systems use brushless DC motors in drivetrains due to their high efficiency and power-to-size ratio. To this end, the library includes an example using the permanent magnet DC machine model `DC_PermanentMagnet` from the Modelica Standard Library in the `Examples` package.

4.3 Control Modules

Each of the drivetrains in the quadcopter relies on a signal from a controller to enable flight. There are multiple representations of the control module in the library considering both the power consumption of the controller and different sampling methods of the controller's PID blocks. There are five representations of the control module:

- Discrete PID
- Discrete PID and electrical load
- Discrete PID using Synchronous Library
- Discrete PID using Synchronous Library and electrical load
- Continuous PID

An example of the control module is shown in Figure 6, which uses an XYZ-coordinate position command, XYZ-coordinate current position GPS measurement, 3-dimensional acceleration measurement, and a one-dimensional yaw command to control the quadcopter's position. These signals are fed into discrete PID blocks to determine the voltage command sent to each drivetrain, denoted by `y`, `y1`, `y2`, and `y3`. For the model variations that include an electrical load, the `pin`, `resistor`, and `ground` components are included in the model to represent a constant power draw.

5 Mechanical Models

The mechanical package includes the following subpackages to model the quadcopter in the mechanical domain:

- Blades
- Propeller
- Chassis
- Rotor
- Motor

The components in the mechanical package use a `replaceable` template structure, where the components use the same base model that is parameterized for different quadcopters and levels of modeling fidelity. Inside of each of these subpackages, there is a package called `Templates` where the template for the component is stored.

5.1 Blade Models

The blade models utilize the Modelica Standard Library's multibody functionalities to link with the rest of the system mechanically. The blade model uses a template defined in `DroneLibrary.Mechanical.Blades.Templates`, which includes the multibody connector `Input` that interfaces a 3-dimensional force and torque vector with the rest of the drivetrain. The blade model consists of two-point body masses representing the individual wings in the model, denoted as the `bodyShape` components in Figure 7. Since the drone models can be animated, the blade model utilizes the `fixedShape` component from the `MultiBody.Visualizers.FixedShape` package from the Modelica Standard Library. The blade model can be configured to be animated using a `.stl` file when the model is simulated, that is stored in the `A_Modelica/DroneLibrary/Resources/Images` folder of the library. The library also includes blade models parameterized for quadcopters such as the Mavic Air and Phantom. This entails defining the blade's mass m and relative distance (defined as an XYZ-coordinate) to the quadcopter body as a point body mass as shown in

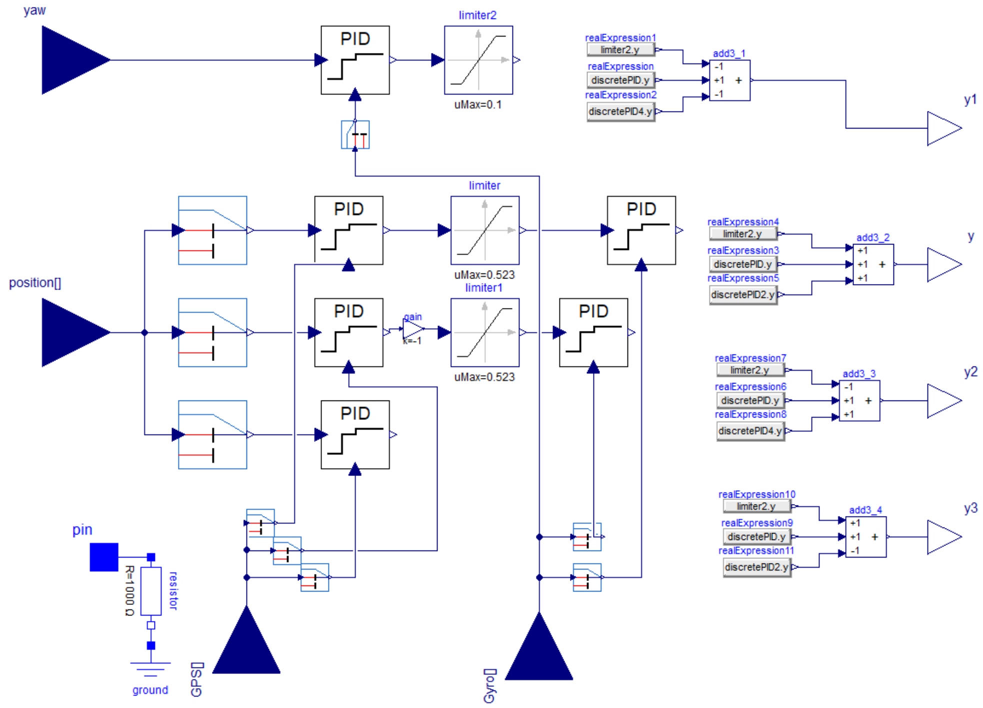


Figure 6. Main controller unit.

Figure 7.

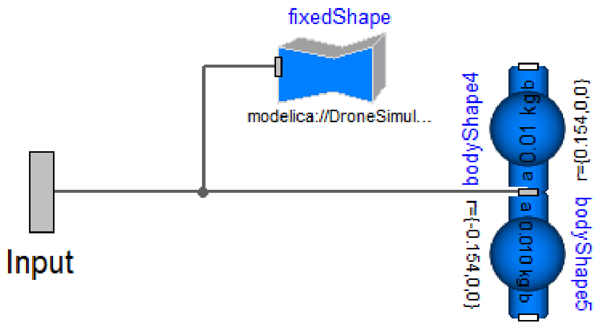


Figure 7. Blade model.

5.2 Propeller Models

The propeller template is shown in Figure 8. The propeller model combines the motor, rotor, and blade models using a replaceable model structure. Each propeller has a `position` input from the controller and has an `Airframe` mechanical connector that communicates a 3-dimensional force and torque value with the quadcopter's chassis.

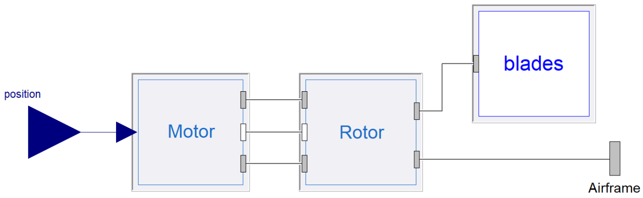


Figure 8. Propeller model.

The propeller model can also be configured to account for a system that has a non-ideal power source, e.g., a battery that changes voltage as it discharges. This is shown in Figure 9, where the electrical pin connector `p1` electrically links the voltage and current drawn by the motor to the quadcopter's power source.

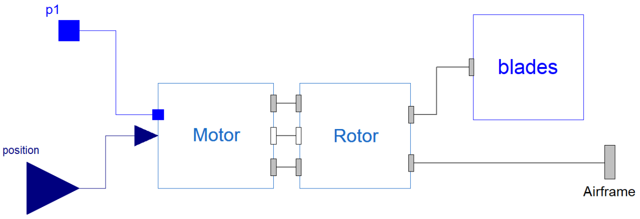


Figure 9. Propeller model with electrical power input.

5.3 Chassis Models

The quadcopter's body is modeled in the chassis subpackage. The template of the chassis model is shown in Figure 10, which connects mechanically to each of the propellers using the `frame_a` connectors. Each arm of the quadcopter's chassis is modeled as a point body mass `bodyShape`, and the centralized body of the quadcopter is represented by the `bodyCylinder` and `pointMass` components. The `frame_a4` interface connects the quadcopter chassis to various sensors that control the drone's positioning, pitch, and yaw. The chassis model is configured to be animated in Figure 11, which contains a `fixedShape` component that links a `.stl` file representing the chassis to be used in the animation of the quadcopter.

The chassis multibody masses follow the mathematical model in Equation (4). Each inertia tensor can be defined individually in this system, which reduces to Equation (5) as defined in the Modelica Standard Library (MSL)'s Multibody Library (Otter, Elmqvist, and Mattsson 2003).

$$\tau = I_{XYZ} * \alpha = \begin{bmatrix} I_{XX} & I_{XY} & I_{XZ} \\ I_{YX} & I_{YY} & I_{YZ} \\ I_{ZX} & I_{ZY} & I_{ZZ} \end{bmatrix} \begin{bmatrix} \dot{\omega}_X \\ \dot{\omega}_Y \\ \dot{\omega}_Z \end{bmatrix} \qquad (4)$$

$$\tau = \begin{bmatrix} I_{XX} & 0 & 0 \\ 0 & I_{YY} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix} \begin{bmatrix} \dot{\omega}_X \\ \dot{\omega}_Y \\ \dot{\omega}_Z \end{bmatrix} \qquad (5)$$

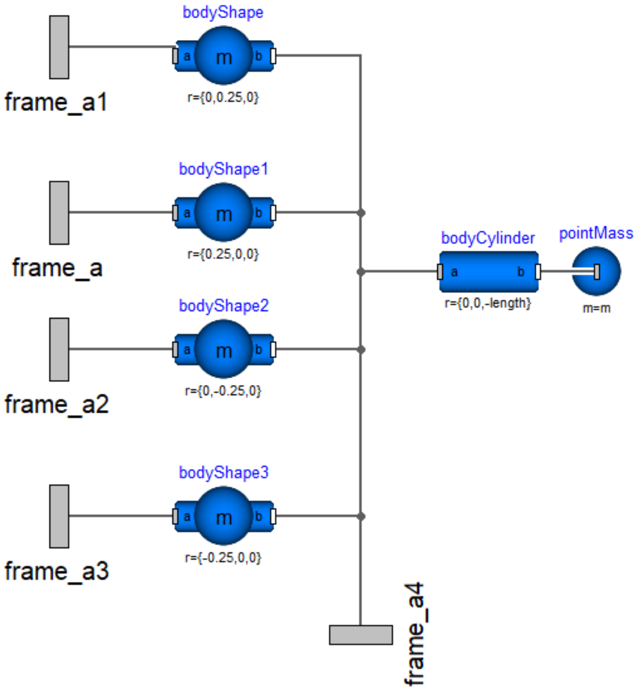


Figure 10. Chassis model template.

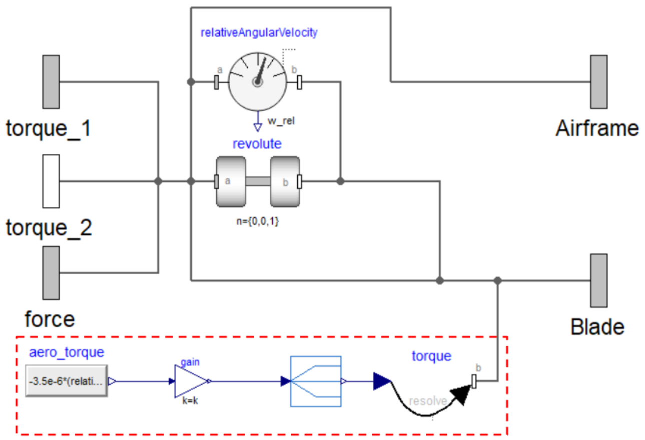


Figure 12. Rotor model. The red box in the lower left corner of the model represents the calculations needed to determine aerodynamic forces applied to the rotor.

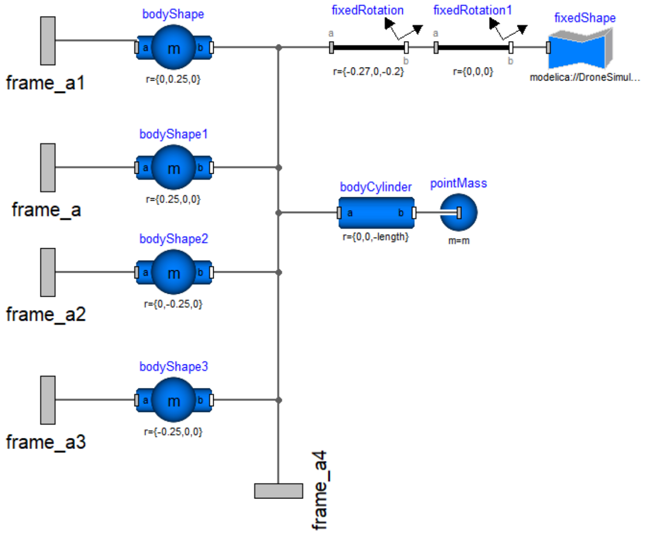


Figure 11. Chassis model with visualization functionalities.

5.4 Rotor Models

The rotor model is represented in Figure 12, which mechanically links to the motor, chassis, and blades. The multibody interfaces `torque_1` and `torque_2` link the torque from the machine to the revolute. The revolute's speed is determined by a scaled measurement of the relative angular velocity between `torque_1` and `torque_2`.

The aerodynamic forces are applied to the rotor using the ω^2 model. Equation (6) calculates the aerodynamic torque, `aero_torque`. More information about the derivation of the simplified model can be found in (Luukkonen 2011).

$$\tau_o = (3.5 \times 10^{-6})\omega^2 \qquad (6)$$

5.5 Motor Models

The motor subpackage contains two templates, `DCMotor` and `DCMotor_Power`. These templates can be configured

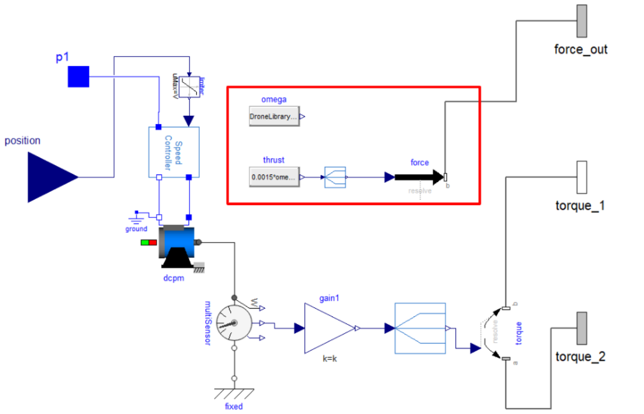


Figure 13. Motor model. The red box contains the components used to calculate the thrust.

with any of the models from the `Electrical.Motors` subpackage. The motor models receive a voltage command from the controller and link the rotor speed and torque mechanically with the rest of the system. In the case of the `DCMotor_Power` models such as the one shown in Figure 13, the electrical pin `p1` can be connected to an external power source such as a battery model. The voltage command is scaled according to the maximum voltage the power source provides using the `Electrical.Sources.PowerControl` component. For the motor models using the `DCMotor` template, the `Electrical.Sources.PowerControl` and `p1` components are removed to represent the electrical system at a lower level of modeling fidelity.

In the motor models, the aerodynamic forces applied to the quadcopter are calculated using the ω^2 model. Equation (7) determines the torque applied from the motor. The thrust provided by the motor is calculated in the component `thrust` in Figure 13. The thrusts are coupled to the motor component using the mechanical multibody connector `thrust`. More information about the derivation of the simplified model can be found in (Luukkonen 2011).

$$\tau_h = 0.0015\omega^2 \tag{7}$$

6 Sensor Models

The sensors package includes models used in the quadcopter to monitor its status and provide measurements for control. This package contains accelerometer models, gyroscope models, and GPS models to track the location of the quadcopter. Figure 14 shows the GPS sensor model, which uses the relative position sensor from the Modelica Standard Library to track the quadcopter's relative position to an origin point. It links mechanically to the quadcopter chassis through interface `frame_a` and communicates a three-dimensional position vector signal to the controller through real output `y[]`.

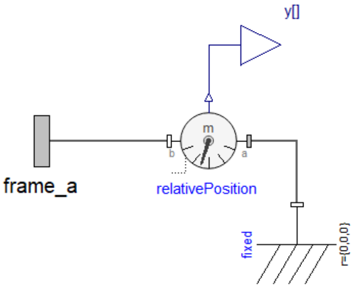


Figure 14. GPS sensor model.

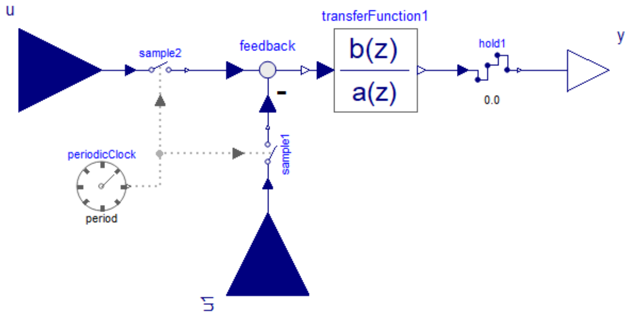


Figure 15. Discrete PID block using Modelica synchronous clocked components.

7 Other Packages

The library also contains various subpackages that provide ancillary functionalities for simulation and analysis.

7.1 Blocks

The `Blocks` subpackage contains models for sources, controls, and signal routing used in the quadcopter components. The `Sources` subpackage contains two models that can be used as XYZ-coordinate inputs to the quadcopter. The `circlePath` component gives a flight command to fly the quadcopter in a circular motion and the `linePath` command applies a ramp path along an axis to fly the quadcopter in a straight line.

In the `Control` subpackage, there are PID transfer function blocks that are used to model the main controller units using both discrete and continuous controls. The discrete PID transfer function component is shown in Figure 15, which uses the clocked components from the Modelica Standard Library to model the PID controls. These models are typically used in virtual reality applications. The purpose of including multiple representations of the PID control blocks is to support a broad range of simulation studies and applications where different integration and simulation methods are used.

7.2 Visualize

The `Visualize` package includes models that allow the user to use visualization functionalities provided both by the MSL and those that depend on the DLR Visualization Library (Hellerer, Bellmann, and Schlegel 2014). It provides component models for the user to interact in real-time with the model using functionalities of the Modelica Devices Library, enabling keyboard and joystick input. Examples that use these functionalities can be found in `Examples.Visualize`. Before using the examples, see the information included in `Examples.Visualize.HowToRun`.

7.3 Tests

The `Tests` subpackage contains test model examples for individual subsystems in the quadcopter. These models verify the functionality of the discrete control elements, the command path models, and the quadcopter components incrementally. These models can also be useful for helping users through the debugging process when developing new models.

8 Examples

8.1 Basic Example

Several example model can be found within the package `Examples.DroneWithIdealPower`. The package contains a `TestSystem`, as shown enclosed in blue in Figure 16. The goal is to simulate the drone's take-off and hoovering by providing a ramp input to the z-axis reference. The figure also shows how the model can be quickly reconfigured to analyze diverse model variants by using the replaceable models, as illustrated in Figure 16 where the simplest representation of the drivetrain's machine is chosen.

The simplest representation, i.e., when selecting `Drone_IdealMachine` as shown in Figure 16, the drone is modeled with an ideal motor and voltage source power system. This model uses a constant voltage power source that can draw as much current as needed to satisfy any command applied to the motors. This means all electrical dynamics from the battery are also neglected in the model.

The drone is simulated under ideal conditions in Figure 17 with a ramp signal applied in the Z-direction from the ground to $5m$. The drone overshoots the $5m$ hovering height by 5.43 % in this test. These results can be compared by the user for other cases, e.g., by running the models with the discrete PID or the discrete PID using the synchronous components by selecting the desired model variant as shown in Figure 16.

8.2 Battery Powered Drone Example

In the `Examples.DroneWithoutIdealPower` package, the `TestSystem` model is configured similarly to that described for the previous example. However, here the drone variants/classes can be changed to study the drone's behavior when using a battery power source. This model uses a battery power source where the voltage and current are affected by the battery's state of charge. This means the electrical dynamics from the battery are also included in the model.

This model configuration also uses a DC permanent magnet machine, using the model in Figure 13. The battery is parameterized with a nominal voltage of $12.1V$ and the motor is parameterized with a nominal voltage of $12V$. This model variant considers the non-ideal behaviors that were neglected in the previous section. The system is given the same $5m$ ramp command in the Z-direction, with the response shown in Figure 18. Since the simulation is only for a 10-second period, the power consumption will have a negligible impact on the electrical dynamics. The overshoot and steady-state error in the Z-direction are $1.36\,\%$ and $0.13\,\%$ respectively. The lower error and overshoot compared to the simplified case is due to the higher damping modeled in the motors.

8.3 Example using DLR's `Visualization` Library and the `Modelica_DeviceDrivers` Library

The drone can be simulated using HIL and virtual reality software programs using the DLR Visualization Library. The drone is configured for HIL simulation using the DLR Visualization Library in Figure 19[3].

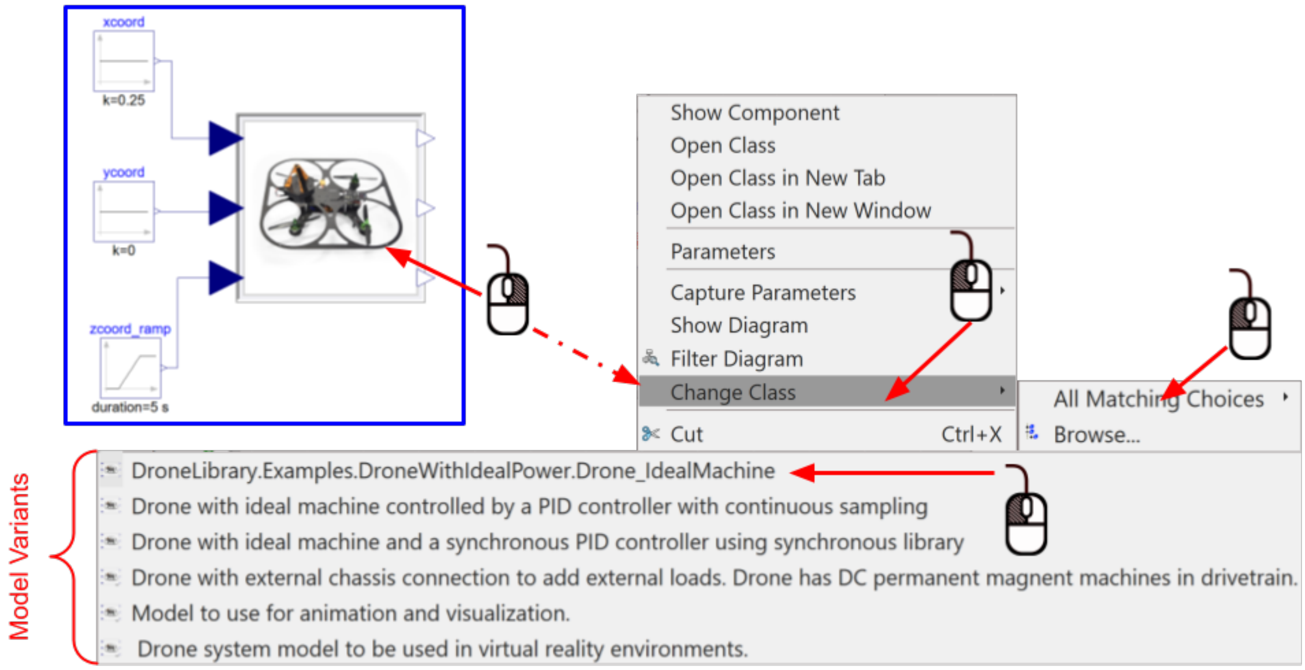
In the package, `Examples.Visualize` the model `ModuleTest_SimVis` is configured to use computer keyboard arrow inputs to direct the drone's flight path by using functionalities from the `Modelica_DeviceDrivers` library. The components from DLR's Visualization Library include `camera1` and `camera2` components that are connected to the `frame_a` interface on the chassis to follow the drone component around the environment during the simulation. The `world` component defines gravity and a reference frame to all moving components in the environment. This `world` component (from the MSL) is connected to `shape1` (also from from DLR's Visualization Library), which defines the physical terrain (i.e. the scene) that the drone is interacting with.

When the model is compiled and simulated, it appears in the environment in Figure 20. The user can then utilize the keyboard commands to maneuver the drone around the environment. The drone can also be configured to be controlled by a joystick or in other virtual reality environments, where more information can be found in (M. Podlaski 2021).

9 Conclusions

The `DroneLibrary` introduces a basis for open-source, multi-domain drone models at varying levels of detail and complexity. The models included in the `DroneLibrary` are more complex than those in previous literature with the capability to interface components between multiple engineering domains. Many of these systems previously only considered mechanical behaviors and aerodynamics, while the `DroneLibrary` considers electrical dynamics in the motors and power systems. The models in the `DroneLibrary` allow for animation of the system for a

[3] This requires a Professional Edition license of the Visualization Library, to obtain one, see:`https://visualization.ltx.de/`.



DroneLibrary.Examples.DroneWithIdealPower.Drone_IdealMachine

Drone with ideal machine controlled by a PID controller with continuous sampling

Drone with ideal machine and a synchronous PID controller using synchronous library

Drone with external chassis connection to add external loads. Drone has DC permanent magnent machines in drivetrain.

Model to use for animation and visualization.

Drone system model to be used in virtual reality environments.

Figure 16. `Examples.DroneWithIdealPower.TestSystem` example model and how to modify it to use the different available variants.

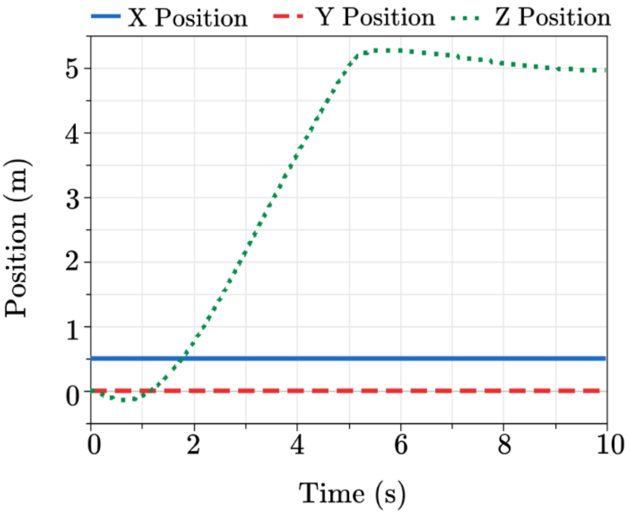


Figure 17. Drone with an ideal motor and ideal power system XYZ position under a ramp signal input.

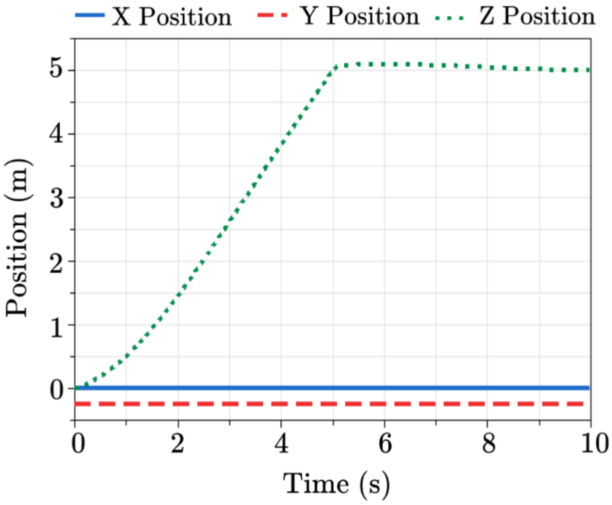


Figure 18. Drone with an DC permanent magnet motor and battery XYZ position under a ramp signal input.

given input, enhancing the insight, analysis, and communication between domain specialists.

The models in the library are designed in a manner that encourages future development, i.e., allowing to both increase the complexity and the flexibility of the component and system models. In addition, these models can be integrated with other software using the FMI Standard (Modelica Association 2014), especially with virtual reality software and game engines such as those in (M. Podlaski 2021).

The library has been developed in Dymola and the developers have successfully checked and simulated the models. In addition, the library was also tested with OpenModelica where currently the models pass checks but may not simulate.

The library has been released as open source software under the BSD 3-Clause License and is available online at: `https://github.com/ALSETLab/Modelica-Drone-3D-FMI`

Acknowledgements

This work was supported in part by the National Aeronautics and Space Administration under award number 80NSSC19M0125 as part of the Center for

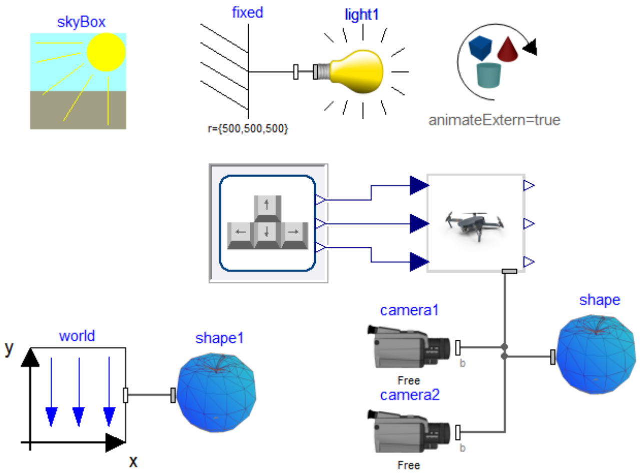


Figure 19. Drone model configured for HIL interaction using the DLR Visualization Library.

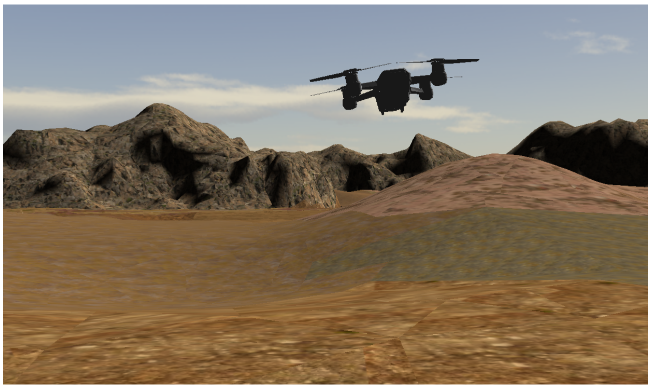


Figure 20. Drone simulated in desert terrain environment using DLR Visualization Library. A video can be found at: `https://youtu.be/1OfGjCZVH5o`.

High-Efficiency Electrical Technologies for Aircraft (CHEETA).

The work of M. Podlaski was supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 1744655 and the Chateaubriand Fellowship of the Office for Science & Technology of the Embassy of France in the United States.

The work of L. Vanfretti was supported in part by Dominion Energy Virginia.

The authors would like to sincerely thank the following former Rensselaer Polytechnic Institute students for their contributions to the library and this research: Hao Chang, Jake Montenieri, Jordan Grey, Eric Segall and James Lewin.

References

Borer, Nicholas K. et al. (2016). “Design and Performance of the NASA SCEPTOR Distributed Electric Propulsion Flight Demonstrator”. In: *16th AIAA Aviation Technology, Integration, and Operations Conference*. DOI: 10.2514/6.2016-3920. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2016-3920. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2016-3920.

Bresciani, Tommaso (2008-10). “Modelling, Identification and Control of a Quadrotor Helicopter”. MSc Thesis. Lund, Sweden: Department of Automatic Control, Lund University.

Coïc, Clément, Marc Budinger, and Scott Delbecq (2022). “Multirotor drone sizing and trajectory optimization within Modelon Impact”. In: *Proc. American Modelica Conference 2022*. doi: 10.3384/ECP2118656, pp. 56–63.

Doo, Johnny T. et al. (2021). *NASA Electrical Vertical Takeoff and Landing (eVTOL) Aircraft Technology for Public Services - A white Paper. NASA Transformative Vertical Flight Working Group 4 (TVF4)*. White Paper NASA Document Number 20205000636. NASA. URL: https://www.sti.nasa.gov/.

Hellerer, Matthias, Tobias Bellmann, and Florian Schlegel (2014-03). “The DLR Visualization Library - Recent development and applications”. In: *Proc. 10th Int. Modelica Conf.* doi: 10.3384/ecp14096899, pp. 899–911.

Kuric, Muhamed, Nedim Osmic, and Adnan Tahirovic (2017-07). “Multirotor Aerial Vehicle modeling in Modelica”. In: *Proc. 12th Int. Modelica Conf.* doi: 10.3384/ecp17132373, pp. 373–380.

Luukkonen, Teppo (2011-08). “Modelling and Control of Quadcopter”. Independent research project in applied mathematics. Espoo, Finland: School of Science, Aalto University.

M. Podlaski, et al (2020-10). “UAV Dynamic System Modeling and Visualization using Modelica and FMI”. In: *VFS 76th Annu. Forum & Tech. Display*. doi: doi:10.4050/F-0076-2020-16289., pp. 1058–1072.

M. Podlaski, et al (2021). “Towards a VR-Based Early Design Interaction for Electric Vertical Take-Off & Landing (eVTOL) Cyber-Physical Models”. In: *2021 NAFEMS World Congr.* NAFEMS.

Ma, X., Z. Li, and C. Nae (2016). “Multi-domain modeling and Simulation of a Quad-rotor aircraft based on Modelica”. In: *Proc. Intl. Conf. Modeling, Simulation and Visualization Methods (MSV)*, pp. 120–124.

Modelica Association (2014-07). *Functional Mock-up Interface for Model Exchange and Co-Simulation Version 2.0*. Tech. rep. Linköping: Modelica Association. URL: https://fmi-standard.org.

Modelica Association (2023). *Modelica Standard Library v4.0.0 (2020-06-04)*. URL: https://github.com/modelica/ModelicaStandardLibrary/releases/tag/v4.0.0 (visited on 2023-06-08).

Otter, Martin, Hilding Elmqvist, and Sven Erik Mattsson (2003-11). “The New Modelica MultiBody Library”. In: *Proc. 3rd Int. Modelica Conf.* Pp. 311–330.

Podlaski, Meaghan, Robert Niemiec, et al. (2021-05). “Multi-Domain Electric Drivetrain Modeling for UAM-Scale eVTOL Aircraft”. In: *Proceedings of the 77th Annual Forum*. The Vertical Flight Society. Virtual.

Podlaski, Meaghan, Luigi Vanfretti, et al. (2022). “Extending a Multicopter Analysis Tool using Modelica and FMI for Integrated eVTOL Aerodynamic and Electrical Drivetrain Design”. In: *Proc. American Modelica Conference 2022*. doi: 10.3384/ECP2118647, pp. 47–55.

Thiele, Bernhard et al. (2017-05). “Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library”. In: *Proc. 12th International Modelica Conference*. doi: 10.3384/ecp17132713, pp. 713–723.

Optimal Scheduling of IES Considering Thermal Transmission Delay Based on Modelica and Julia

Yong Qiu Jin Wang Shubin Zhang Yuan He Haiming Zhang
Ji Ding Fanli Zhou
Suzhou Tongyuan Software & Control Technology Co., Ltd, China,
{Yong Qiu, Jin Wang, Shubin Zhang, Yuan He, Haiming Zhang, Ji Ding, Fanli Zhou}@tongyuan.cc

Abstract

The Integrated Energy System (IES) enables integrated control and coordinated optimization of multiple energy flows. Due to the complexity of dynamic characteristics of multiple energy flows and the significant differences in time scales, thermodynamic problems occur during the operation of the system. In this paper, we propose an IES operation method that comprehensively considers thermodynamics to reduce the impact of thermal transmission delay (TDD) on the system's operational strategy, including modeling, evaluation, and scheduling programs. Firstly, an IES model is established to describe the dynamic characteristics of the energy supply network. Secondly, a two-stage optimization scheduling model considering TDD is established to reduce the impact of TDD on the operation decisions of IES, and the thermal power imbalance rate index is proposed to measure the impact of thermodynamics. Finally, the proposed method's effectiveness is validated by utilizing a comprehensive energy system as an example and implementing it on the MWORKS platform using the Modelica and Julia languages.

Keywords: integrated energy system, optimal dispatch, heat dynamics, Modelica, Julia

1 Introduction

An IES integrates green and clean energy on the production side, couples energy conversion equipment with energy networks, and provides various forms of energy such as electricity, gas, heat/cooling to the consumption side, effectively reducing energy waste and improving energy utilization efficiency (Liu, et al., 2019; Song et al., 2022). Due to the different flow characteristics of in an IES, modeling IES is different from traditional energy systems. In the IES model, it is necessary to include modeling of the same type of energy transmission network (such as heat network or natural gas network) as well as modeling of the mutual conversion and coupling of heterogeneous energy sources, which increases the difficulty of modeling the IES (Cui, et al., 2022; Liu, et al., 2020; Pan, et al., 2016). The authors in (Zhai, et al., 2021) applied the characteristic method for hyperbolic partial differential equation analysis to analyze the dynamic changes of natural gas systems in IES. The authors in (Chen X, et al., 2020) established an IES model that comprehensively considers the dynamic characteristics of heat and gas transmission, which can be used for simulation analysis of IES and the synergistic optimization among different energy subsystems, providing a basis for improving the economic and security of IES services. The authors in (Liu et al., 2016) defined the meaning of multi-energy complementarity in different energy networks and conducted a comprehensive analysis of IES containing three types of energy networks, including electricity, gas, and heat, to achieve a comprehensive modeling and simulation of the three energy systems. In (Wang et al., 2019) the authors were inspired by the individual-based modeling method in bacterial ecology. An individual-based modeling method for IES was proposed by them, which decouples the entire system into multiple independent individuals and utilizes input and output sets for each individual to achieve uniform interaction.

In order to dig deeper into the dispatching value of electricity, heat/cooling, and gas loads on the demand side, plenty of research on integrated energy coordination control and dispatching optimization considering demand response has been conducted to realize the optimal utilization of various energy resources. In (Dou, et al., 2020), an elastic demand side response model in the electric-thermal coupling system is established to optimize the load scheduling, showing significant effect in cutting down the system operating cost and improving the absorption capacity of PVs. A multitime scale flexible resources coordination optimization scheme is presented in (Yan, et al., 2020), which considers the smart loads' participation and presents a multi-time-scale power dispatch model that considers coordination and interaction between resources and electrical loads. (Shen, et al., 2020) focused on regional IES and proposed a thermal-electrical coupling energy optimization method that includes virtual energy storage resources. This method effectively improves the matching degree between energy production output and user energy consumption demand, ensuring the economic and flexible operation of the system. In (Gu, et al., 2017), the thermal storage capacity of both the heating network and buildings is applied to participate in the dispatch of the IES to decrease the operational cost and increase the wind power consumption. Besides, the flexible heating/cooling demand of buildings is also integrated with the flexible electrical and gas demand, to form an integrated demand response programs and participate in the energy management of energy systems, with the aim of improving economic performances and enhancing operational reliability (Wang, et al., 2017; Gu, et al., 2017). However, in almost all the existing researches, the slow heat dynamics in the heating network during the dispatch interval is usually simplified into a transient process. This simplification will result in considerable errors in the heat dynamics. It has not been investigated what impacts this simplification will bring about on the operation of the system and how to measure these impacts. Fundamentally, the heat dynamics in the heating network is governed by partial differential equations (PDE) (Jie, et al., 2012). Many numerical methods such as the finite difference method based on the characteristic path and the implicit upwind method are used to discretize the PDE into algebraic equations with proper time and spatial steps (Wang, et al., 2017).

In summary, although there have been many advances in modeling, simulation, and optimization scheduling for IES, the dynamic transmission characteristics of the energy supply network are rarely considered in the research on optimization scheduling (Lu, et al., 2020). However, it has a significant impact on the operation of IES. Therefore, in this paper, we propose an IES operation method that comprehensively considers thermal dynamic, including modeling, evaluation, and scheduling programs. For the intra-day optimization scheduling of IES, we implemented the "operation decision-simulation evaluation" of IES on the same platform framework based on MWORKS (including Modelica and Julia). The main contributions of this paper are summarized as follows.

(a) An IES model that considers network dynamics was developed in response to the dynamic transmission characteristics of the energy supply network. The dynamic characteristics of energy transmission in the network were simulated, and the possible impact of transmission delays on IES operational decisions was analyzed.

(b) The thermal power imbalance rate was proposed as a metric to measure the impact of time resolution on IES operation. The index was calculated based on scheduling decisions and simulation results, and was used as a basis for making operational decisions.

(c) An optimization scheduling model for IES considering TDD was proposed to reduce the impact of TDD on operational decisions.

2 Integrated Energy System Model

There are significant differences between different subsystems. Therefore, in analyzing the dynamic processes of the IES, mathematical models for each subsystem should be established. Furthermore, by combining these subsystem models with the coupling element models, equipment models and network models of various types can be coupled together to form a multi-energy coupled IES model.

2.1 Natural Gas Network Model

The flow of natural gas in pipelines always follows the three laws of mass conservation equation, momentum conservation equation, and energy conservation equation (Vitaliy, et al., 2019).

Mass conservation equation:

$$\frac{\partial}{\partial \tau}\left(A\rho\right)+\frac{\partial}{\partial x}\left(\rho w A\right)=0 \tag{1}$$

where A is cross-sectional area of pipeline; ρ is gas density; w is gas flow rate.

Momentum conservation equation:

$$\frac{\partial}{\partial \tau}\left(\rho w A\right)+\frac{\partial}{\partial x}\left(\rho w^2 A\right)=-g\rho A\sin\alpha-\frac{\partial}{\partial x}\left(pA\right)-\frac{\lambda}{D}\frac{w^2}{2}\rho A \tag{2}$$

where λ is the friction resistance coefficient of the gas pipeline; D is the inner diameter of the pipeline; α is the inclination angle of the pipeline.

Energy conservation equation:

$$\begin{aligned}-\frac{\partial Q}{\partial \tau}\left(\rho w A\right)&=\frac{\partial}{\partial \tau}\left[\left(\rho A\right)\left(u+\frac{w^2}{2}+gs\right)\right]+\\ &\frac{\partial}{\partial x}\left[\left(\rho w A\right)\cdot\left(H+\frac{w^2}{2}+gs\right)\right]\end{aligned} \tag{3}$$

where Q is heat exchange; u is internal energy; H is enthalpy.

2.2 Heating Network Model

In order to study the dynamic changes in heat transfer processes inside heating system pipelines, a dynamic model of hydraulic-thermal coupling for the heating pipeline is formed by coupling a thermal calculation model with the hydraulic calculation model of the heating pipeline.

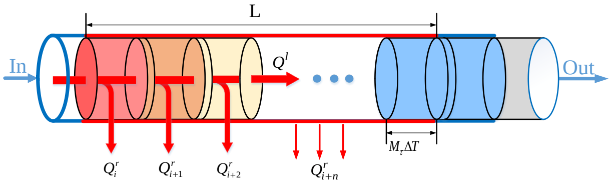


Figure 1. Physical model of pipe heat transfer.

The variation of heat transfer in the heating network can be expressed as the spatiotemporal distribution of the temperature and mass flow rate of the hot water in the system. Figure1 shows the process of heat transfer along the axial direction and radial loss in the hot water. The pipeline is discretized into several units, each of which has three associated heat flows that determine the energy storage change of each unit. The radial heat transfer of each unit is independent and the heat is dissipated to the external environment through the pipe wall and insulation layer. The axial flow is the connection condition of the radial temperature field of the entire pipe section, that is, the input of each unit is the output of the previous unit, and the output is the input of the next unit (Qiu, et al., 2022).

To describe the situation of considering both axial flow and radial heat dissipation simultaneously, a dynamic model of hydraulic-thermal coupling for the heating pipeline is established. The continuity equation and motion equation of pipeline flow are the same as those of gas pipelines, shown in Equations 1 and Equations 2, while the energy equation considers the radial heat dissipation of the pipeline on the original basis, as shown in Equation 4.

$$\frac{d\left(\dot{m}_i \cdot u_i\right)}{dt} = \dot{m}_i h_i - \dot{m}_{i+1} h_{i+1} + Q_i^r \tag{4}$$

where m is the mass of the cell heating medium; $\dot{m}$ is the mass flow rate of the cell heat medium; Q_i^r is the radial heat dissipation of each unit, it can be expressed as

$$Q_i^{\mathrm{r}} = \frac{T_{i,\tau} - T_0}{R_{all}} \tag{5}$$

where $T_{i,\tau}$ is the hot water temperature; T_0 is the ambient temperature; R_{all} represents the total thermal resistance between hot water and the external environment.

2.3 Power System Load Flow Model

Compared with gas and heating systems, the power system responds quickly to disturbances, generally within milliseconds. Therefore, this paper ignores the dynamic changes in power system flow. In power system flow simulation, the power system flow simulation model can be established based on the node voltage equation, and then the solution of various state parameters of the power system can be achieved.

This article adopts a classic power flow calculation model, where the basic equation for power flow calculation is:

$$\frac{P_i - jQ_i}{\overset{*}{U}_i} = \sum_{j=1}^{n} Y_{ij} \dot{U}_j \left(j = 1, 2, \cdots, n\right) \tag{6}$$

where P_i is the active power of the node i; Q_i is the reactive power of the node i; U_i is the voltage vector of the node; $\dot{U}_j$ is the conjugation of node j voltage; Y_{ij} is the admittance between node i and node j.

2.4 Coupling Element Model

Combined Heat and Power (CHP) unit is composed of gas turbines, waste heat boilers, internal combustion engines, and external combustion engines. It is a production method that combines heating and power generation, and it can produce both electricity and heat. The model of CHP unit can be expressed by the following formula.

The mathematical model for the output power of the CHP unit can be expressed as:

$$m_{\mathrm{chp},\tau}^{\mathrm{load}} = \frac{P_\tau^{\mathrm{chp}} + H_\tau^{\mathrm{chp}}}{\eta_{\mathrm{chp}} H_{GV}} \tag{7}$$

where H_τ^{chp} is the thermal power of CHP at time τ; P_τ^{chp} is the electrical power of CHP at time τ; $m_{\mathrm{chp},\tau}^{\mathrm{load}}$ is the mass flow rate of natural gas consumed by CHP; η_{chp} is the efficiency of the CHP unit; H_{GV} is the high calorific value of natural gas.

The relationship between heat power and electrical power of CHP units can be expressed as:

$$k_{\mathrm{chp}} = \frac{H_\tau^{\mathrm{chp}}}{P_\tau^{\mathrm{chp}}} \tag{8}$$

where k_{chp} is the thermoelectric ratio of the CHP unit.

The CHP unit outputs heat to the heating system, and its heat output and water temperature meet the following requirements:

$$H_\tau^{\mathrm{CHP}} = c_p m_\tau^{\mathrm{CHP}} \left(T_\tau^{\mathrm{s}} - T_\tau^{\mathrm{r}}\right) \tag{9}$$

where T_τ^{s} is the water supply temperature at the time τ; T_τ^{r} is the return water temperature at the time τ.

3 Optimization Scheduling Model for IES Considering Thermal Transmission Delay

Due to the multi-time scale characteristics of energy transmission in IES, such as electricity, gas, and heat, there is a delay in response to dispatch instructions. Compared with the power system and natural gas system, the heat dynamics of the heating system is a slowly changing process, and its response time to dispatch instructions is longer. In the determination of the operating strategy of the IES, the evolution process of heat dynamics cannot be ignored. In this paper, the framework of operation decision-making and simulation evaluation is adopted to study the optimization scheduling of IES.

3.1 Optimization Scheduling Model

The optimal dispatch of the IES aims to determine the short-term (typically, day-ahead) optimal output of devices in the system in order to minimize the cost. The dispatchable variables include the state and output of each device, the networkenergy power, and the energy power supplied to end users. The constraints include the operational constraints of the electrical power system, heating system, and nature gas system.

The total operating costs during the scheduling phase of the IES include the costs of purchasing and selling electricity, the costs of purchasing gas, and the costs of participating in the operation and maintenance of the energy supply equipment during the scheduling phase. The objective of the optimal dispatch model of the IES is as follows.

$$\begin{aligned}
&\min\left(C_{\text{ex}}^{\text{power}}+C_{\text{buy}}^{\text{gas}}+C_{\text{om}}^{\text{equi}}\right)\\
&=\sum_{t=1}^{24}\left(\frac{c_{\text{buy}}^{t}+c_{\text{sell}}^{t}}{2}P_{ex}^{t}+\frac{c_{\text{buy}}^{t}-c_{\text{sell}}^{t}}{2}\left|P_{ex}^{t}\right|\right)\cdot\Delta t\\
&+\sum_{t=1}^{24}\left(m_{\text{gas}}^{t}\cdot c_{\text{gas}}^{t}\right)\cdot\Delta t\\
&+\sum_{t=1}^{24}\left(c_{\text{CHP}}\cdot Q_{\text{CHP}}^{t}+c_{\text{PV}}\cdot P_{\text{PV}}^{t}+c_{\text{WT}}\cdot P_{\text{WT}}^{t}+c_{\text{BS}}\cdot P_{\text{BS}}^{t}\right)\cdot\Delta t
\end{aligned}\tag{10}$$

where $C_{\text{ex}}^{\text{power}}$ is the costs of purchasing and selling electricity; $C_{\text{buy}}^{\text{gas}}$ is the costs of purchasing gas; $C_{\text{om}}^{\text{equi}}$ is the costs of participating in the operation and maintenance.

The Constraints of the optimal dispatch model of the IES is as follows.

Constraint on electrical power balance:

$$P_{\text{WT}}^{t}+P_{\text{PV}}^{t}+P_{\text{NP}}^{t}+P_{\text{CHP}}^{t}+P_{\text{ex}}+P_{\text{BS}}=P_{\text{Load}}\tag{11}$$

where P_{Load} is the power system load.

Constraint on heat power balance:

$$\sum_{i=1}^{n}P_{\text{h,CHP},i}^{t}=P_{\text{h,Load}}\tag{12}$$

where $P_{\text{h,Load}}$ is the thermal load. $P_{\text{h,CHP},i}^{t}$ is thermal power of CHP unit.

Constraint on gas mass flow rate balance:

$$M_{\text{gas}}^{t}-\sum_{i=1}^{n}M_{\text{pipe,in}}^{t}+\sum_{i=1}^{n}M_{\text{pipe,out}}^{t}=M_{\text{Load}}\tag{13}$$

where M_{Load} is the natural gas load.

In addition to satisfying the electricity, gas, and heat power balance constraints mentioned above, the IES scheduling model also needs to satisfy operational constraints such as energy storage device constraints, interconnection power transmission constraints, and CHP output constraints. Among these constraints, Equations 14-16 represent the energy storage device constraints, Equation 17 represents the interconnection power transmission constraint, and Equation 18 represents the CHP output constraint.

$$S_{\text{BS}}^{\min}\le S_{\text{BS}}^{t}\le S_{\text{BS}}^{\max}\tag{14}$$

$$P_{\text{BS,ch}}^{\min}\le P_{\text{BS,ch}}^{t}\le P_{\text{BS,ch}}^{\max}\tag{15}$$

$$P_{\text{BS,dis}}^{\min}\le P_{\text{BS,dis}}^{t}\le P_{\text{BS,dis}}^{\max}\tag{16}$$

$$P_{\text{EX}}^{\min}\le P_{\text{EX}}^{t}\le P_{\text{EX}}^{\max}\tag{17}$$

$$P_{\text{CHP}}^{\min}\le P_{\text{CHP}}^{t}\le P_{\text{CHP}}^{\max}\tag{18}$$

where S_{BS}^{t} is battery capacity; $P_{\text{BS,ch}}^{t}$ is battery charging power; $P_{\text{BS,dis}}^{t}$ is battery discharge power; P_{EX}^{t} is the purchasing and selling power of the tie-line.

3.2 Influence of Thermal Transmission Delay

When performing optimization scheduling of an IES, the transmission delay of the heating pipeline often results in a significant time difference between our scheduling command response and the thermal load. In order to illustrate the impact of heating pipeline transmission delay on the day-ahead optimization scheduling of the IES, a simulation is performed using a specific heating pipeline (L=1000m, D=0.3m, m=50kg/s, T(start)=90℃) as an example. Three load demand changes are set for three time periods: ① T(t=0)=95℃, ② T(t=3600s)=80℃, ③ T(t=7200s)=90℃. The simulation model based on Modelica is shown in Figure 2.

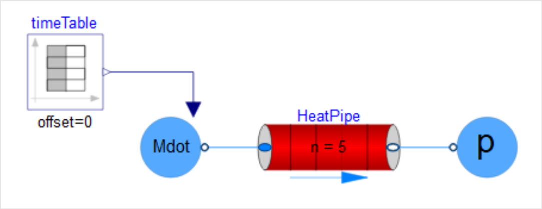


Figure 2. Heating pipeline simulation model.

The simulation results are shown in Figure 3, where "T_in" represents the temperature of the heat source adjusted based on the load demand, which is numerically equal to the user's required water supply temperature. "T_out" represents the actual temperature of the hot water supplied to the user. From the simulation results, it can be seen that heat network transmission delay can have a serious impact on supply-demand balance. The demand of the heat network is mainly met through the response of the heat source, and there is a delay of about one hour from the time the heat is emitted from the heat source to the time it reaches the load, leading to a serious supply-demand imbalance. Therefore, from the perspective of operating decision, it is necessary to consider heat network transmission delay in the day-ahead scheduling stage and control this supply-demand error within a certain range.

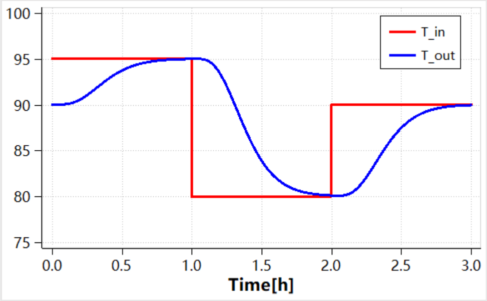


Figure 3. Simulation results of a specific pipeline.

The above results have verified the serious impact of heat network transmission delay. However, how to measure and overcome its impact has not yet been resolved. To address this issue, a further proposal is made for the heat power imbalance rate indicator and a two-stage process method for operating strategy.

3.3 Two-Stage Optimization Scheduling Process for IES Considering Thermal Transmission Delay

The simulation results from the previous section indicate that errors in the heat power network will lead to a supply-demand imbalance of thermal energy in the IES 's day-ahead optimization scheduling. Therefore, this section proposes the heat power imbalance rate indicator to quantify the thermal dynamic errors in the operating strategy, reflecting the supply-demand imbalance of the heat network in each time period. The specific calculation formula is as follows:

$$\Delta H_{\%}^{i,t} = \sum_{i=1}^{n}\left(\frac{\left|H_{\mathrm{s}}^{i,t}-H_{\mathrm{l}}^{i,t}\right|}{H_{\mathrm{s}}^{i,t}}\times 100\%\right) \qquad (19)$$

where $\Delta H_{\%}^{i,t}$ is heat power balance rate index; $H_{\mathrm{s}}^{i,t}$ is the actual supplied thermal power of each node; $H_{\mathrm{l}}^{i,t}$ represents the actual thermal power demand of each node.

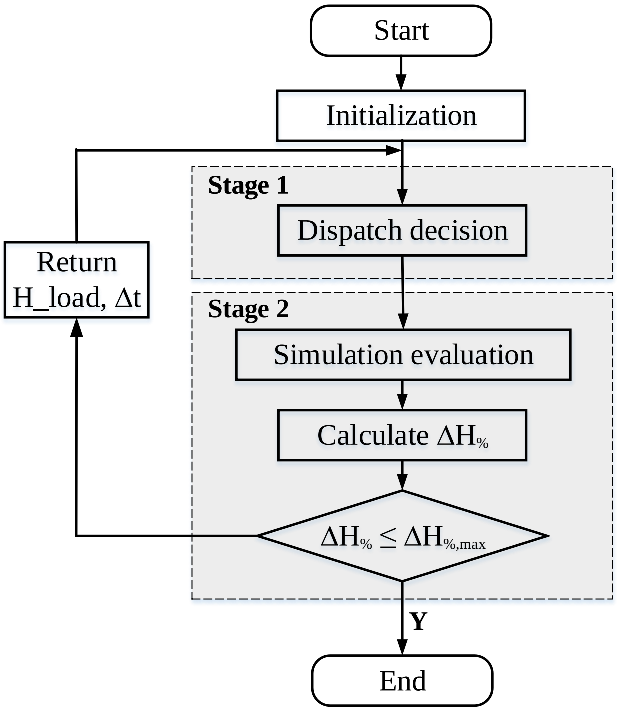


Figure 4. Two-stage dispatch procedure.

It should be noted that when the heat power imbalance rate exceeds the limit, it indicates that the current operating strategy does not meet the thermal dynamic control accuracy. At this point, simulation evaluation is needed to determine the delay amount, and then re-evaluate the thermal load and adjust the time resolution. The two-stage optimization scheduling process for the IES based on the "operating decision-simulation evaluation" framework is shown in Figure 4. The Stage 1 is the dispatch decision process and the Stage 2 is the posterior evaluation of the supply and demand situation of thermal energy. The detailed steps are as follows:

A. Initialization stage: Based on the IES model established in the second section, the IES simulation model is built. (using MWORKS.Sysplorer)

B. "Operating decision" stage: Based on the established IES operation optimization model, the operating decision is made to obtain the operation optimization results. (using MWORKS.Syslab)

C. "Simulation evaluation" stage: The operating decision results are input into the IES model for simulation calculation, and the heat power imbalance rate indicator is calculated based on the simulation results. (using MWORKS.Syslab and MWORKS.Sysplorer)

D. Validation stage: Determine whether the operating strategy meets the thermal dynamic control accuracy requirements. If the requirements are met, the process ends; otherwise, update the day-ahead thermal load and time resolution and proceed to step B.

4 Case Studies

This paper investigates a IES with an integrated network of electricity, gas, and heat as a case study, whose topology is illustrated in Figure 5. The power grid, nature gas network, and heat network are interconnected through a CHP unit. The power grid comprises 30 nodes, which include a wind turbine, a photovoltaic power station, a small nuclear power station, and a battery unit. The gas grid comprises 17 nodes and 16 pipelines. The heat network includes 20 nodes, with 10 of them serving as user node.

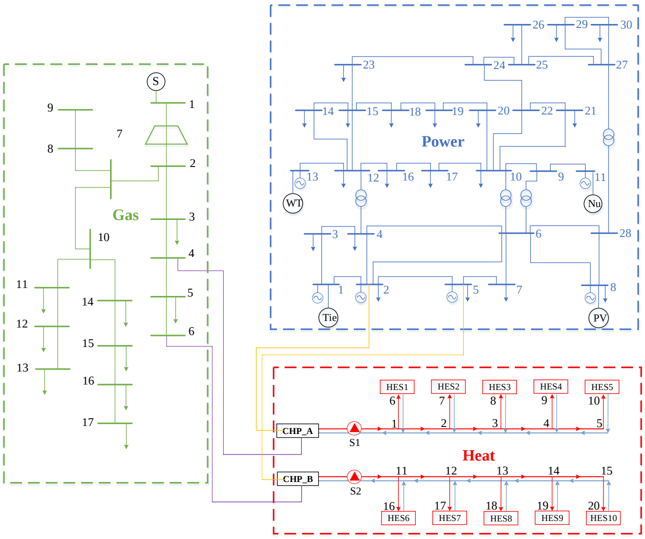


Figure 5. Topological structure diagram of IES.

Building upon the aforementioned work, the scheduling strategy and system simulation model for the comprehensive energy system have been implemented using the Modelica and Julia languages. The program was developed utilizing the MWORKS.Sysplorer and MWORKS.Syslab software tools, along with their unified simulation capabilities. The system model is illustrated in Figure 6. From the figure, it can be seen that the simulation model consists of two parts: a simulation module and an optimization module. The electric power system part in the simulation module is developed based on the Julia language, while the other models are developed based on the Modelica language. The optimization module is developed based on the Julia language and the Julia code is included in the Sysplorer model through the SyslabFunction of the MWORKS platform, enabling the combination of simulation and optimization.

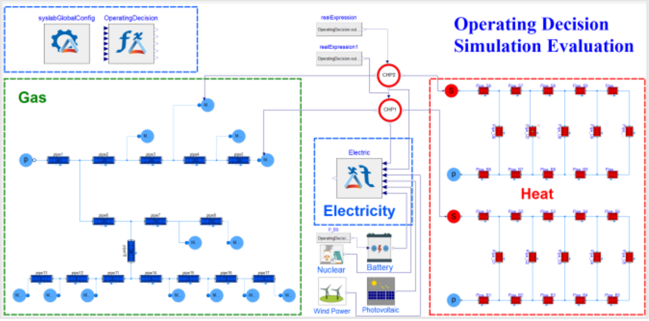


Figure 6. IES operating decision and simulation evaluation model.

Two optimization running strategies were simulated and compared, including a two-stage optimization scheduling method (TSOSM) that considers TDD and an optimization scheduling method (TOSM) that does not consider TDD. As shown in Table 1, the heat power imbalance, temporal resolution, and operating cost of the two methods were compared, and the heat power imbalance of TSOSM was significantly lower than that of TOSM, indicating a significant improvement in supply-demand balance, and thermal comfort is greatly improved. This is our primary focus. The cost calculation results indicate that considering thermal dynamics reduces operating costs by approximately ¥10400, which is significant when viewed from an annual perspective. However, using TSOSM requires a higher time resolution, which increases computational complexity.

Table 1. Comparison of results on heat power imbalance rate, time resolution, and operating cost.

Parameter	*TOSM*	*TSOSM*
$\triangle h_{\%}$	9.58	0.97
$\triangle t$ (min)	60	30
Cost (10^3RMB ¥)	4011.48	4001.08

Figure 7 displays the heat demand of the most disadvantageous user node and the supply water temperature change curves of two methods. It can be clearly seen that TSOSM has a better response to the heat demand, while TOSM fails to meet the heat demand of users for a long period of time after the demand changes, which greatly reduces the users' thermal comfort and causes a very unfavorable impact.

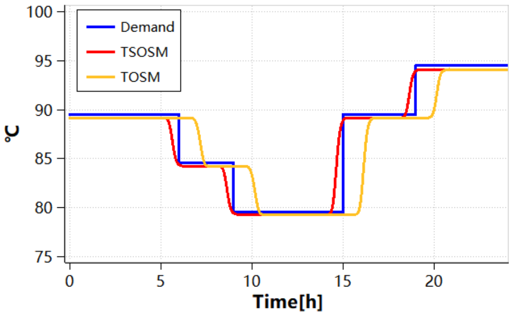
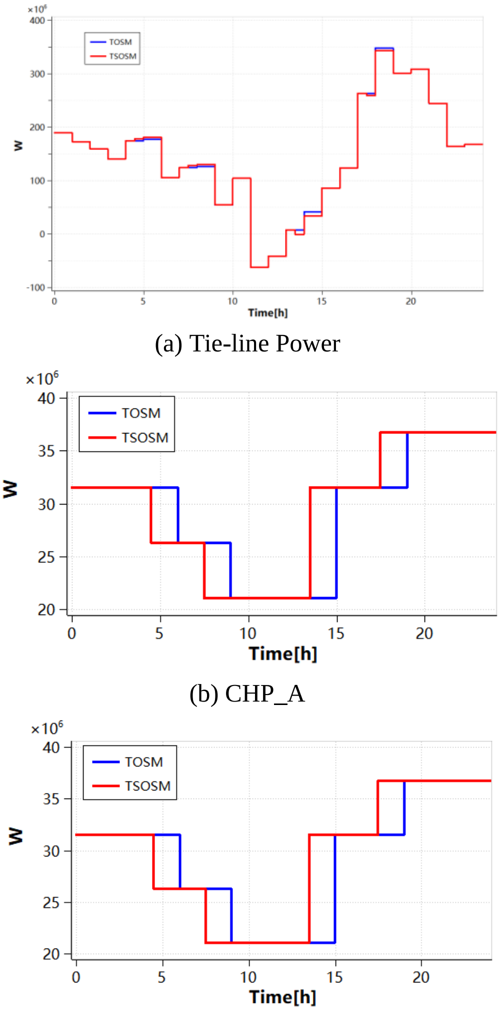


Figure 7. Temperature comparison of the most unfavorable nodes.

The comparison of the day-ahead scheduling results of the IES using TSOSM and TOSM is shown in Figure 8. For tie-line, the positive values indicate purchasing electricity, and the negative values indicate selling electricity. For the batteries, the positive value represents discharging (releasing), and the negative value represents charging (storing). The output of the battery under the two operating strategies is almost the same. The difference between the output of the CHP unit is evident in both cases. The results indicate that considering TDD has a relatively small effect on the battery's output, but a significant impact on the CHP's output. In addition, under different operating strategies, the wind power, photovoltaic, and nuclear energy outputs are the same and equal to the predicted power generation. The power of the tie-line also varies in some periods under different operating strategies. In conclusion, considering TDD not only has a considerable impact on the thermal scheduling results, but also affects the power scheduling results, although the impact is relatively small.

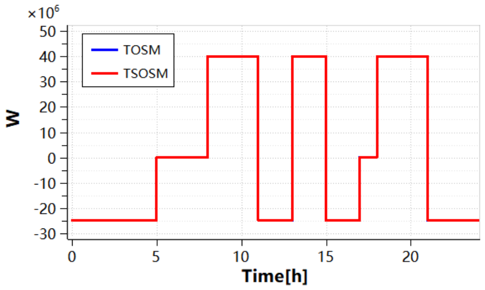
Substantially, TSOSM focuses on characterizing the thermal dynamic response, which leads to differences in the heat power injected into the heating network. Therefore, the scheduling results of equipment related to heat power (i.e., CHP units) will be directly affected.



(a) Tie-line Power

(b) CHP_A

(c) CHP_B



(d) Battery

Figure 8. Comparison of scheduling results.

Under the framework of "operation decision-simulation evaluation", further analysis of the system's state can be conducted based on determining the operational decision, achieving precise control of the system, and providing a basis for scientific management in the intra-day scheduling stage.

The supply and demand response of the power system is shown in Figure 9. The CHP unit, wind power generation, photovoltaic, nuclear energy, and battery jointly provide electricity for the power system. Despite the system's high demand for electricity, it still needs to purchase electricity from the main grid to meet the shortfall in load. Under this operational strategy, the supply and demand of the system are balanced, and clean energy such as wind and photovoltaic provide most of the electricity, saving the system's operational costs. In addition, the battery discharges during periods of high load (such as 9:00-11:00 and 14:00-15:00) and charges during periods of light load (such as 23:00-4:00), to achieve peak shaving and valley filling.

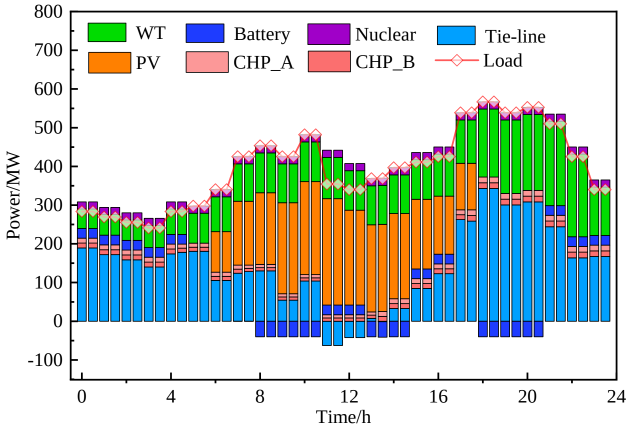
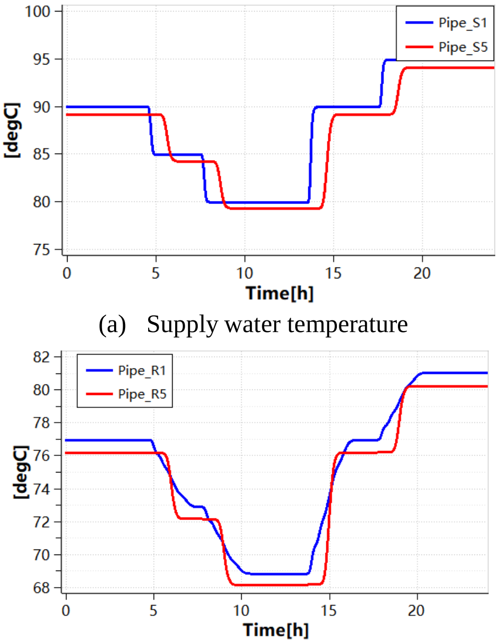


Figure 9. Supply and demand response of the power system.

Figure 10 illustrates the temperature change curves of the supply and return pipes at the beginning and end of the heating network. As shown in the figure, the outlet temperatures of each supply and return pipe also change correspondingly with the variation of the heat load. Due to the influence of thermal inertia, the response time is about 90 minutes. Meanwhile, due to the coupling of hydraulic transmission and thermal transmission, there is a time delay of about 45 minutes between the response of the first pipe and the response of the fifth pipe, reflecting the delay characteristic of transmission, which is particularly important in the optimization and scheduling of heating systems.



(a) Supply water temperature

(b) Return water temperature

Figure 10. Temperature variation curve of heating network supply water and return water.

Changes in the state of the heating network can also trigger responses in the natural gas network, firstly affecting the load side, causing a rapid change in gas load, and then leading to the redistribution of pressure and transmission flow rate in each branch pipe. The mass flow rate changes of some sections of the natural gas network are shown in Figure 11, which shows that the flow rate of the pipe fluctuates with the load and reaches a stable state again after a period of time (about 10 minutes).

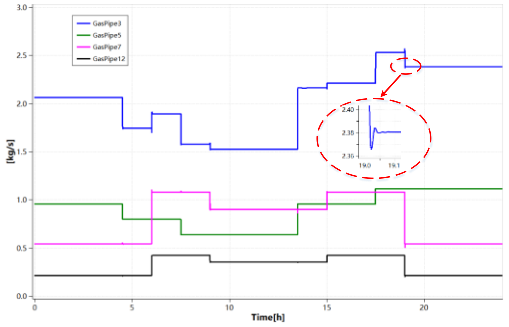


Figure 11. Mass flow rate variation curve of natural gas pipeline network

In summary，the above case verify both the necessity of considering heat dynamics of the heating network and the effectiveness of the proposed dispatch procedure. The simulation results show that considering thermal dynamics has different degrees of impact on the scheduling results of the power system, heating system, and natural gas system. Since the essence of considering thermal dynamics lies in correcting the heat load, the impact on the heating system is the greatest. However, due to the characteristics of multi-energy coupling in IES, the impact on the heating system will also be coupled to the power system and natural gas system, but to a relatively lesser extent.

5 Conclusions

In this paper, we propose an IES operation optimization method that comprehensively considers thermal dynamics, including modeling, evaluation, and scheduling programs. The heat power imbalance rate index is introduced to evaluate the satisfaction level of heat demand. A two-stage optimization scheduling model for IES considering transmission delay is proposed to reduce the impact of TDD on the system's operational strategy. Finally, the proposed method is validated using an IES.

Based on theoretical analysis and simulation results, the following conclusions can be drawn:

1) TSOSM can significantly mitigate the impact of thermal supply-demand imbalance caused by TDD, and improve users' thermal comfort. In addition, compared with the TOSM, the operating cost of TSOSM has been reduced.
2) The scheduling results of TSOSM have an impact on the heating system, power system, and natural gas system. During simulation evaluation, the dynamic characteristics of the system were also well simulated, such as when the load changes, the response time of the heating network is about 1.5 hours, and the response time of the gas network is about 10 minutes.
3) The integration of optimization scheduling of IES and multi-agent power flow analysis into a unified framework is achieved in this study. Effective methods and tools are provided for optimizing the operation and scientific management of IES.

References

Liu L, Wang D and Hou K (2020). “Region model and application of regional integrated energy system security analysis”. In: *Applied Energy*. Vol. 2020.114268. DOI:10.1016/j.apenergy.2019.114268.

Song D, Meng W. and Dong M (2022). “A critical survey of integrated energy system: Summaries, methodologies and analysis”. In: *Energy Conversion and Management.* Vol. 115863. DOI:10.1016/j.enconman.2022.115863.

Cui Z., Chen J and Liu C (2022). “Time-domain continuous power flow calculation of electricity–gas integrated energy system considering the dynamic process of gas network”. In: Energy Reports. Vol.8:597-605. DOI:10.1016/j.egyr.2022.02.238.

Pan Z., Sun H. and Guo Q (2016). “A static security analysis method for energy internet based on multiple energy streams”. In: Power Grid Technology. Vol. 40 (06): 1627-1634. DOI:10.13335/j.1000-3673.pst.2016.06.004.

Zhai J, Zhou X. and Li Y (2021). “Analysis of natural gas flow reversal in integrated energy system based on dynamic simulation”. In: *Energy Reports*. Vol: 1149-1158.DOI: 10.1016/j.egyr.2021.09.152.

Chen X, Wang C and Wu Q (2020). “Optimal operation of integrated energy system considering dynamic heat-gas characteristics and uncertain wind power”. In: *Energy,* Vol: 117270. DOI: 10.1016/j.energy.2020.117270.

Liu X., Pierluigi M (2016). “Modelling, assessment and Sankey diagrams of integrated electricity-heat-gas networks in multi-vector district energy systems”. In: *Applied Energy*. Vol: 167:336-352.DOI:10.1016/j.apenergy.2015.08.089.

Wang L., Zheng J. and Li M (2019). “ Multi-time scale dynamic analysis of integrated energy systems: an individual-based model”. In: *Applied Energy*. Vol: 237: 848-861. DOI: 10.1016/j.apenergy.2019.01.045.

Dou C., Zhou X.and Zhang T (2020).“Economic optimization dispatching strategy of microgrid for promoting photoelectric consumption considering cogeneration and demand response”. In: *Journal of Modern Power Systems and Clean Energy*. Vol: 8,557-563. DOI: 10.35833/MPCE.2019.000214.

Yan X., Li R (2020). “Flexible coordination optimization scheduling of active distribution network with smart load”. In: *IEEE Access*, Vol: PP (99).1-1. DOI:10.1109/ACCESS.2020.2982692.

Shen Y , Liang X , Hu W (2020). “Optimal Dispatch of Regional Integrated Energy Syst, et alem Based on a Generalized Energy Storage Model”. In: *IEEE Access.* Vol: 1546-1555. DOI:10.1109/ACCESS.2020.3046743.

Gu W, J. Wang and S. Lu (2017). “Optimal operation for integrated energy system considering thermal inertia of district heating network and buildings.” In: *Appl. Energy*. Vol:199pp.234–246.Aug. DOI:2017.10.1016/j.apenergy.2017.05.004.

Wang J, Zhong H and Ma Z (2017). “Review and prospect of integrated demand response in the multi-energy system”. In: *Appl. Energy*. Vol: 202.pp.772–782. DOI: 2017.10.1016/ j.apenergy.2017.05.150.

Gu W (2017). “Residential CCHP microgrid with load aggregator: Operation mode, pricing strategy, and optimal dispatch”. In: *Appl. Energy*. Vol: 205.pp.173–186. Nov. 2017.DOI:10.1016/j.apenergy.2017.07.045.

Jie P.F., Z. T. S. and Zhu N.(2012).“Modeling the dynamic characteristics of a district heating network.” In: *Energy*. Vol:39.No.1.pp.126–134.Mar. DOI:2012.10.1016/j.energy.2012.01.055

Wang Y (2017). “Thermal transient prediction of district heating pipeline: Optimal selection of the time and spatial steps for fast and accurate calculation”. In: *Appl. Energy.* Vol: 206.pp. 900–910.Nov.15. DOI:2017.10.1016/j.apenergy.2017.08.061

Wang YR. et al. (2017) “Thermal transient prediction of district heating pipeline:Optimal selection of the time and spatial steps for fast and accuratecalculation”. *Appl. Energy*, vol. 206, pp. 900–910, Nov. 15. DOI:10.1016/j.apenergy.2017.08.061

Lu G , Gu W , Zhou S (2020). “High-Resolution modeling and decentralized dispatch of heat and electricity Integrated Energy System.” In: *IEEE TRANSACTIONS ON SUSTAINABLE ENERGY.* Vol: 11, NO. 3, JULY 2020.

Vitaliy G , Anatoly Z (2019). “An explicit staggered-grid method for numerical simulation of large-scale natural gas pipeline networks”. In: *Applied Mathematical Modelling.* Vol: 65.pp. 34–51. DOI:10.1016/j.apm.2018.07.0510

Qiu Y, Li Gj, Wei Lin, Zhang T (2022). “Dynamic thermal performance of district heating network under typical safety accidents”. In: *Journal of Building Performance Simulation.* Vol: pp. 678-690 DOI:10.1080/19401493.2022.2078407

Simulation Study of Flow Instability in Parallel Multi-Channel Systems Based on Modelica

Qiushi Tong[1] Xing Lv[1] Kangjie Deng[2] Xiaokang Zeng[2] Yanlin Wang[2] Haiming Zhang[1] Fanli Zhou[1]

1. Suzhou Tongyuan Soft & Ctrol Technology Co., Ltd., China;
2. CNNC Key Laboratory on Nuclear Reactor Thermal Hydraulics Technology, Nuclear Power Institute of China,China;

Abstract

In parallel channels of a nuclear reactor core, flow instability can cause a significant decrease in critical heat flux (CHF) or mechanical oscillation of the fuel components, endangering the normal operation of the reactor. Nuclear reactor Modeling and Analysis Platform, NUMAP, developed based on the two-fluid six-equation theory and using the Modelica language, is a multi-domain unified modeling and simulation platform for nuclear power plants. In this paper, a parallel dual-channel system model was constructed based on the NUMAP, referencing a high-temperature and high-pressure steam-water two-phase thermohydraulic experimental device, to simulate flow instability phenomena. The comparison with experimental data validated the transient analysis ability of the NUMAP for flow instability phenomena. Based on this, the flow instability boundary of a parallel multi-channel system was calculated under the same operating conditions. When the number of parallel channels was 2, 3, and 4, the calculated flow instability boundary error did not exceed ±5%, verifying that a parallel dual-channel structure can be used to obtain the flow instability boundary when there are multiple parallel heating channels.

Keywords: parallel channels, flow instability, Modelica, NUMAP

1 Introduction

Flow instability refers to the phenomenon where a system deviates from its stable state and experiences non-periodic drift or periodic oscillation in thermal parameters after being subjected to instantaneous disturbances (Xu 2001). For systems with multiple parallel heating channels, flow instability may occur between the parallel channels, manifested as periodic flow pulsation between the heating channels. When flow instability occurs between parallel channels, it can significantly reduce the critical heat flux density or cause mechanical vibration of the equipment, thereby endangering equipment safety.

The mechanism of flow instability is shown in Figure1 which shows the pressure drop and flow rate curve of water in a straight pipe with constant heating power.

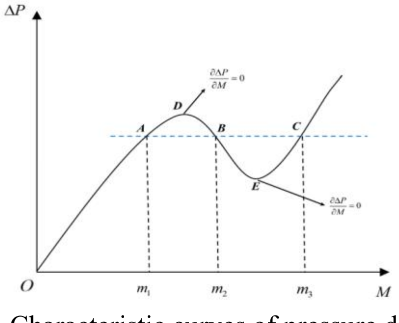


Figure 1. Characteristic curves of pressure drop and flow rate in the heating channel

Within the entire range of mass flow rate (M), when the flow rate decreases to the point E in a flow pipeline with constant total heating power, two-phase flow occurs due to the generation of a large amount of steam, leading to an increase in pressure drop (ΔP) in the pipeline as the flow rate decreases. At point D, the flow rate is small enough to form single-phase steam flow in the pipeline, and the D-O region is the pressure drop characteristic curve of single-phase steam. In Figure 1, we can see that when the differential pressure (ΔP) between the inlet and outlet is between point D and point E, for the same differential pressure (ΔP), there may be three flow rates in the channel, which may lead to flow instability. For parallel channels, periodic flow pulsations may occur in the flow rates in each channel in this situation.

Many scholars have conducted experimental research on flow instability in parallel channels. Cheng et al. (Cheng 2018) studied the flow instability of natural circulation rod bundle parallel channels through experiments. The study mainly focused on the experimental phenomenon and mechanism of pressure drop flow instability when the stabilizer was connected to the upstream of the heating channel, and the effect of inlet subcooling and heat flux density on flow instability was also studied. Tang Yu et al. (Tang 2014) conducted experimental research on flow instability in rectangular parallel channels under forced circulation conditions, considering the effects of system pressure, mass flow rate, and inlet subcooling on the flow instability boundary, and obtained a non-dimensional relationship for flow instability through experimental data fitting.

Currently, experimental methods are mainly used to study flow instability in parallel channels (Peng 2021; Zang 2014). However, experimental research has the disadvantages of long cycle length, limited experimental conditions, and inability to capture the internal mechanism of flow instability phenomena. Therefore, it is necessary to use simulation methods to simulate flow instability phenomena, study the flow instability mechanism under different operating conditions of parallel heating channels, and determine the flow instability boundary to provide a reference for experimental research.

In engineering practice, one-dimensional system simulation programs are commonly used to simulate and analyze flow instability phenomena in parallel heating channels. Since the fluid state in parallel heating channels under forced circulation changes dramatically, the flow path is generally divided into several control volumes. This method reduces the complexity of the calculation compared to the three-dimensional numerical simulation method for studying the flow and temperature fields in the flow path, and achieves the expression of the thermal-hydraulic characteristics of flow instability, which is

DOI
10.3384/ecp204569

widely used in modeling and simulation analysis of flow instability (Qian 2014; Xiong 2015). Commonly used simulation programs are developed using Fortran, C language, Matlab/Simulink, or directly using nuclear engineering professional simulation software such as RELAP5, THEATRE (Xia 2011; Xing 2010). These programs and software use card-based modeling, and the modeling process lacks a graphical interface, making it difficult for users to obtain the composition of the system equipment, equipment connection relationships, and equipment parameter information intuitively. Graphical simulation software such as MMS in the United States, APROS in Finland, and STAR-90 in China have been widely used in the field of dynamic system simulation. Their drag-and-drop modeling and visualization of equipment parameters greatly reduce the workload of modeling, but the code of these commercial software is relatively closed and not conducive to users for secondary development according to their needs.

Modelica is a physical modeling language that is object-oriented, equation-based, and uses reusable hierarchical component models (Zhao 2006). Its numerical model is open-source and visible, and is written in equation form, which has strong comparability with empirical relationships obtained from experiments and theoretical equations. Users can efficiently and accurately construct and optimize numerical models using the advantages of the Modelica language. The Modelica language supports graphical drag-and-drop modeling, which can correspond one-to-one between the topological structure design of the numerical model and the system schematic diagram, greatly improving the modeling efficiency and model readability (Zhang 2022). Based on the advantages of the Modelica language, Suzhou Tongyuan Soft & Control Technology Co., Ltd. and China Nuclear Power Research and Design Institute have constructed a two-phase thermal-hydraulic characteristic model architecture, established a hierarchical structure from the basic control equations and closed equations to the system model, and developed a nuclear reactor unified modeling and analysis platform NUMAP based on Modelica, which is multi-disciplinary, multi-level, and multi-system (Huang 2021).

2 Two-Fluid Six-Equation Modeling Based on Modelica

The thermal hydraulic model library of NUMAP is described using the Modelica language by two-fluid six-equation formulation. The two-fluid six-equation model is a commonly used mathematical equation model for thermal-hydraulic system analysis of nuclear reactors, which can accurately simulate the mass and heat transfer behavior of the two-phase flow in normal and accident operating conditions of the nuclear reactor system. It can be used for nuclear reactor system design verification, operation simulation, safety analysis, and other scenarios. Based on the Modelica language, the two-fluid six-equation model considers the different properties, flow rates, temperatures, and mass, energy, and momentum exchange between the two-phase fluids in the actual two-phase fluid flow process。Mass, momentum, and energy conservation equations are established separately for the vapor and liquid phases. In order to make the control equation group closed, constitutive equations such as interfacial friction, interfacial heat transfer, interfacial mass exchange, wall friction, and wall heat transfer are added.

2.1 Conservation equation model

The mass conservation equation of two phase:

$$\frac{\partial \alpha_k \rho_k}{\partial t} + \frac{\partial \alpha_k \rho_k \mu_k}{\partial t} = \Gamma_k \tag{1}$$

The momentum conservation equation of two phase:

$$\frac{\partial \alpha_k \rho_k u_k}{\partial t} + \frac{\partial \alpha_k \rho_k \mu_k^2}{\partial z} + \alpha_k \frac{\partial p}{\partial z} = \Gamma_k u_{ik} + \alpha_k \rho_k g + F_{wk} + F_{ik} + F_a \tag{2}$$

The thermal energy conservation equation of two phase:

$$\frac{\partial \alpha_k \rho_k h_k}{\partial t} + \frac{\partial \alpha_k \rho_k \mu_k h_k}{\partial t} = \alpha_k \frac{\partial p}{\partial t} + \Gamma_k h_{ik} + q_{ik} + q_{wk} + F_{ik}\mu_{ik} + \alpha_k \rho_k \mu_k g \tag{3}$$

where the following quantities and symbols appear:

k	*vapor phase g or liquid phase f*，
i	*interface*，
α	*volume percentage*，
μ	*flow rate*，
h	*enthalpy value*，
Γ_k	*interface mass transfer*，
F_{wk}	*wall friction*，
F_{ik}	*interface friction*，
F_a	*action of external force field*，
q_{ik}	*interface heat transfer*，
q_{wk}	*wall heat transfer*.

2.2 Closed constitutive equation model

In Section 2.1, the thermal-hydraulic conservation equations used by NUMAP are presented. To solve these equations in a mathematically closed manner, sufficient closed constitutive equation models need to be supplemented, including Flow Regime Maps Model, Wall-to-Fluid Heat Transfer Model, Interfacial Heat Transfer Model, Interphase Friction Model, Wall Drag Model, water and steam property calculation Models, etc. (Rockville 2001).

2.2.1 Flow Regime Maps Model

Two flow-regime maps for two-phase flow are used in the NUMAP: a horizontal map and a vertical map for flow regime in pipes.

(1) The horizontal flow regime map

The horizontal flow regime map is for the flow pipeline whose elevation angle θ is such that $0 \le \theta \le 45$

degrees. The map contain Bubbly Flow, Slug Flow, Annular Mist Flow, Mist Flow, Horizontal Stratification Flow and Transition Flow pattern.

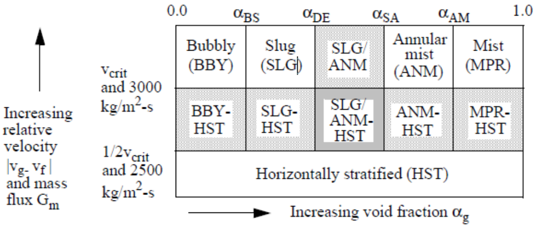

0.0
α_{BS}
α_{DE}
α_{SA}
α_{AM}
1.0
Bubbly (BBY)
Slug (SLG)
SLG/ ANM
Annular mist (ANM)
Mist (MPR)
v_{crit} and 3000 kg/m^2-s
Increasing relative velocity $|v_g - v_f|$ and mass flux G_m
BBY-HST
SLG-HST
SLG/ ANM-HST
ANM-HST
MPR-HST
1/2v_{crit} and 2500 kg/m^2-s
Horizontally stratified (HST)
Increasing void fraction α_g

Figure 2. Schematic of horizontal flow regime map with hatchings, indicating transition regions.

(2) The vertical flow regime map

The vertical flow regime map is for the flow pipeline whose elevation angle θ is such that $45 < \theta \le 90$ degrees. The map contain Inverted Annular Flow, Inverted Slug Flow, Mist Flow, Bubbly Flow, Slug Flow, Annular mist Flow, Vertically Stratification Flow and Transition Flow pattern.

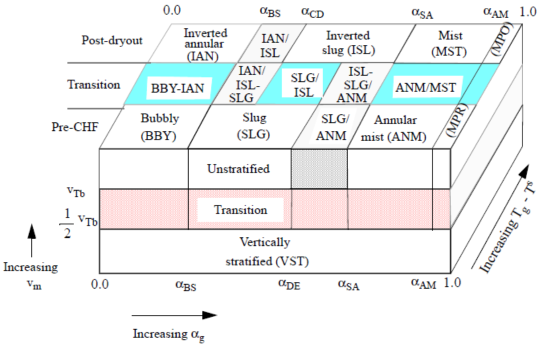


Figure 3. Schematic of vertical flow regime map with hatchings, indicating transition regions.

2.2.2 Wall-to-Fluid Heat Transfer Model

the total wall heat flux (q) is the heat flux to the vapor plus the heat flux to the liquid.

$$q = htcg(T_w - T_{reg}) + htcf(T_w - T_{ref}) \tag{4}$$

where the following quantities and symbols appear:

$htcg$ *heat transfer coefficient to gas*

$htcf$ *heat transfer coefficient to liquid*

T_w *wall temperature*

T_{ref} *liquid reference temperature*

T_{reg} *gas reference temperature*

Heat transfer coefficient *htcg* and *htcf* used in different wall convection heat transfer models are shown in Table 1.

Table 1. Wall convection heat transfer models

Heat transfer model	Correlations
single-phase gas or liquid	Dittus-Boelter, Shah, McAdams
nucleate boiling	Chen
transition boiling	Chen-Sundaram-Ozkaynak
film boiling	Bromley, Sun-Gonzales-Tien

2.2.3 Interfacial Heat Transfer Model

In NUMAP, the interfacial heat transfer between the gas and liquid phase actually involves both heat and mass transfer. The form used in defining the heat transfer correlations for superheated liquid (SHL), subcooled liquid (SCL), superheated gas (SHG), and subcooled gas (SCG) is that for a volumetric heat transfer coefficient (W/(m$^3\cdot$K)). Since heat transfer coefficients are often given in the form of a dimensionless parameter (usually Nusselt number, Nu), the volumetric heat transfer coefficients are coded as follows:

$$H_{ip} = \frac{k_p}{L} Nua_{gf} = h_{ip} a_{gf} \tag{5}$$

where the following quantities and symbols appear:

H_{ip} *volumetric interfacial heat transfer coefficient for phase p* $(\mathrm{W/m^3 \cdot K})$

k_p *thermal conductivity for phase p* $(\mathrm{W/\,m \cdot K})$

L *characteristic length* (m)

a_{gf} *interfacial area per unit volume* $(\mathrm{m^2/m^3})$

h_{ip} *interfacial heat transfer coefficient for phase p* $(\mathrm{W/\,m^2 \cdot K})$

p *phase p (either f for liquid for g for gas)*

2.2.4 Interphase Friction Model

The interface friction in the phasic momentum equations (2) is expressed in terms of phasic interfacial friction coefficients as:

$$F_{ig} = \alpha_g \rho_g FIG(v_g - v_f) \tag{6}$$

$$F_{if} = a_f \rho_f FIF(v_g - v_f) \tag{7}$$

where F_{ig} is the magnitude of the interfacial friction force per unit volume on the vapor and F_{if} is the magnitude of the interfacial friction force per unit volume on the liquid. *FIG* and *FIF* are the phasic interfacial friction coefficients.

2.2.5 Wall Friction Model

The pressure loss of fluid flow includes wall frictional pressure drop, gravity pressure drop, and accelerated pressure drop. The wall frictional pressure drop is calculated by The Wall Friction Model which include Frictional resistance of single-phase flow and two-phase flow.

(1) single-phase flow

Darcy's formula is used for the frictional pressure drop of single-phase flow

$$\Delta P = f \frac{L}{De} \frac{\rho \mu^2}{2} \tag{6}$$

where f is the frictional resistance factor.

(2) two-phase flow

The HTFS correlation is used to calculate the two-phase friction multipliers. This correlation was chosen because it is correlated to empirical data over very broad ranges of phasic volume fractions, phasic flow rates and phasic flow regimes. The correlation has also been shown to give good agreement with empirical data.

The HTFS correlation for the two-phase friction multiplier is expressed as:

$$\Phi_f^2 = 1 + \frac{C}{X} + \frac{1}{X^2} \tag{7}$$

$$\Phi_g^2 = 1 + CX + X^2 \tag{8}$$

where C is the correlation coefficient and X^2 is the Lockhart-Martinelli ratio.

In order to calculate the frictional pressure drop of two-phase flow， NUMAP represents the two-phase pressure drop as the product of the single-phase frictional pressure drop and the HTFS correlation.

2.2.6 Water and Steam Property Calculation Models

In thermal hydraulic simulation software, it is necessary to accurately and quickly calculate all thermodynamic parameters of water and steam. When calculating the thermodynamic parameters of water and steam through fitting formulas, high-order polynomials are necessary to improve calculation accuracy, but high-order polynomials can slow down the calculation speed and consume time when the number of control volume is large.

In order to improve calculation accuracy and speed, NUMAP uses the lookup table method to solve the water and steam parameters, using the 1967 ASME Steam Table. The water vapor table consists of three tables, namely the steam property table (two-dimensional), the water property table (two-dimensional), and the saturated property table (one-dimensional).

2.3 Discrete solution of equations

The conservation equations and closed constitutive equations form a nonlinear equation system for solving thermal-hydraulic problems. In the process of solving the equation system, numerical methods are used to solve the coupled "pressure-velocity" relationship in the model. In the discretization process, a semi-implicit method is used, which was proposed by Patankar and Spalding in 1972 and is commonly known as the SIMPLE (Semi-Implicit Method for Pressure Linked Equations) method, which is a semi-implicit method for solving the pressure coupling equation. The semi-implicit solution method is based on the coupling of velocity and pressure on a staggered grid (Tao 2001).

For one-dimensional fluids, a staggered grid refers to storing the velocity and pressure separately on two different grids. The pressure is stored at the control volume center (node), and the velocity is stored at the interface between two adjacent control volumes. When calculating the velocity correction value, the influence of the velocity correction value on the adjacent interface is not considered, that is, the velocity correction value on the interface is only determined by the pressure correction value between adjacent control volume nodes.

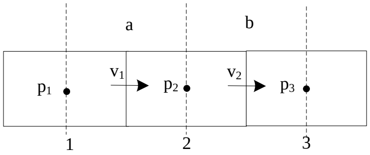


Figure 4. Storage of pressure and velocity.

3 Development and engineering validation of a parallel channel system model

3.1 Development and validation of a parallel dual-channel system model.

The parallel heating dual-channel experiment was conducted on a high-temperature and high-pressure two-phase thermal-hydraulic experimental device, which has a design pressure of 17.2 MPa, a design temperature of 350°C, and an experimental mass flow rate range of 100-5000 kg/(m^2.s) (Wang 2021). Referring to the experimental platform, a parallel heating dual-channel system model was built based on NUMAP, and the system model is shown in Figure 5.

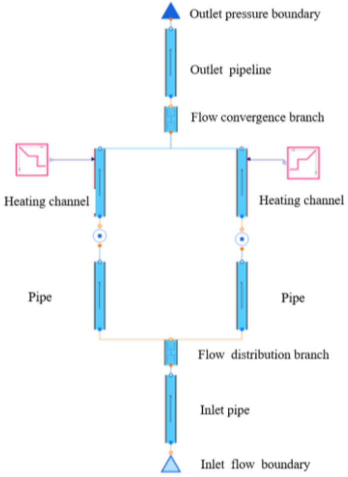


Figure 5. Model of parallel heating dual-channel system.

The way to carry out simulation calculation conditions is to gradually increase the heating power in a step-by-step manner while keeping the system pressure and inlet parameters unchanged. After each power increase, the heating power is kept constant for 2 minutes. When the flow instability occurs, the heating power is kept constant, and the quality gas rate at the channel outlet is recorded. Experiments and simulation calculations were conducted using 10 groups of uniform heating conditions and 16 groups of non-uniform heating conditions. The comparison between the simulation results and the experimental results is shown in Figures 6 and 7.

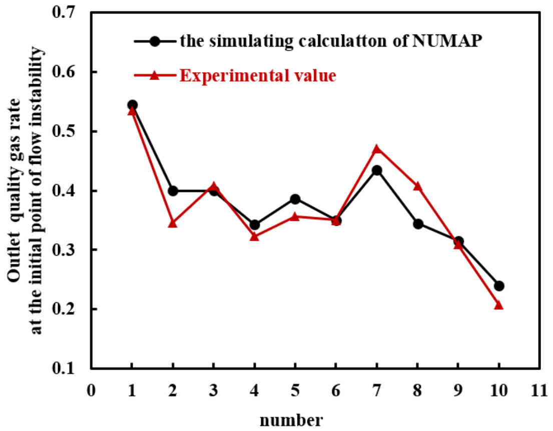


Figure 6. Calculation error of uniform heating.

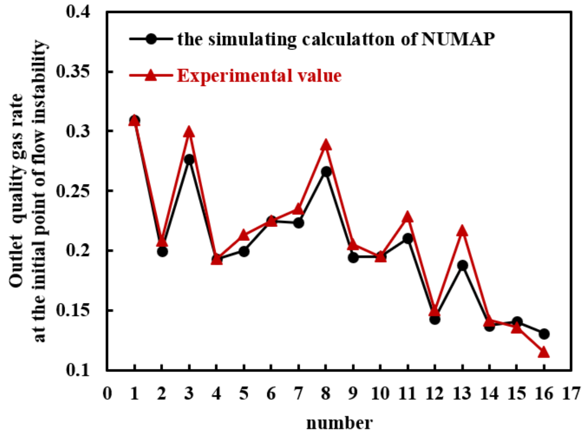


Figure 7. Calculation error of non-uniform heating.

Based on the reference experimental conditions, simulations were carried out for the flow instability of uniform and non-uniform parallel heating dual-channel systems. The simulation results of the quality gas rate at the channel outlet when the flow instability occurs were selected and compared with the experimental values. In terms of calculation error, the maximum errors between the simulation results and the experimental results for uniform and non-uniform heating conditions were 18.87% and 15.7% respectively, which met the error requirement of 20% for this experimental condition; In terms of calculation speed, the simulation setting time for both uniform and non-uniform heating test cases is 400 seconds, and the actual average simulation time is 356 seconds, meeting the real-time requirements of simulation calculation. This indicates that the NUMAP can be used for simulation analysis of the transient characteristics of flow instability in parallel channels.

3.2 The effect of different numbers of heating channels on the unstable boundary of parallel multi-channel flow.

Parallel multi-channel systems exist in various nuclear power equipment, and their flow instability can lead to a significant decrease in critical heat flux (CHF) or mechanical oscillation of the equipment, affecting the normal operation of nuclear power plants. Therefore, how to avoid the occurrence of flow instability in parallel multi-channel systems during the operation of nuclear power plants has always been a hot research topic. After verifying the transient calculation capability of NUMAP for the flow instability of parallel heating dual-channel flow, this section will simulate the flow instability boundary of parallel multi-channel flow.

3.3 Calculation method for flow instability boundary.

Ishii (Ishii 1942) showed that the subcooling number N_{sub} and the dimensionless number of phase change N_{pch} are the two most important dimensionless numbers for characterizing flow instability under forced circulation conditions. Most researchers prefer to use the N_{pch} - N_{sub} diagram to plot the flow instability boundary and region, and this method is also used in this study.

The dimensionless subcooling number N_{sub} and the dimensionless phase change number N_{pch} and are defined as follows:

$$N_{pch} = \frac{Qv_{fg}}{Wh_{fg}v_f} \tag{4}$$

$$N_{sub} = \frac{\Delta h_{sub} v_{fg}}{h_{fg} v_f} \tag{5}$$

where the following quantities and symbols appear:

Q *the total heating power, W.*

W *the total mass flow rate at the inlet of the channels, kg/s,*

h_{fg} *the latent heat of vaporization, J/kg,*

Δh_{sub} *the inlet subcooling enthalpy, J/kg,*

v_{fg} *the difference in specific volume between saturated vapor and liquid,m^3/kg,*

v_f *the specific volume of the saturated liquid phase, m3/kg。*

3.2.1 Simulation calculation of flow instability boundary in multi-channel flow.

The flow instability boundary of parallel dual, triple, and quadruple channel systems was studied using the experimental conditions selected in Section 3.1. The typical simulation curves obtained are shown in Figures 6-8:

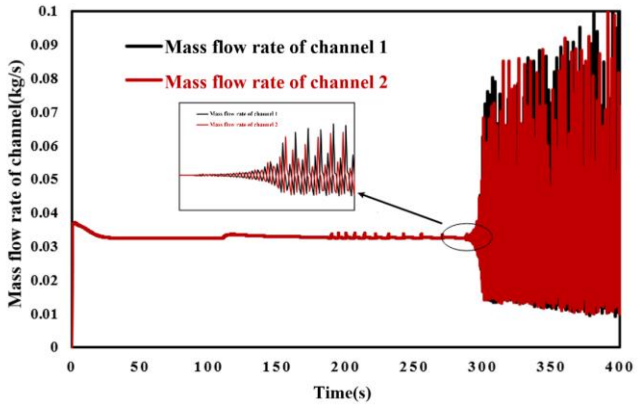


Figure 8. Typical flow rate curve for parallel dual-channel system with symmetric uniform heating.

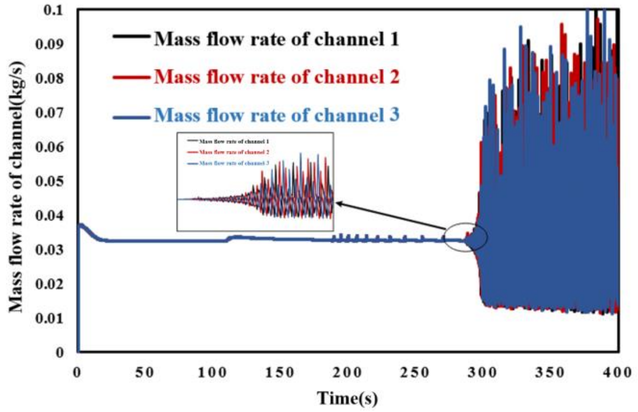


Figure 9. Typical flow rate curve for parallel triple-channel system with symmetric uniform heating.

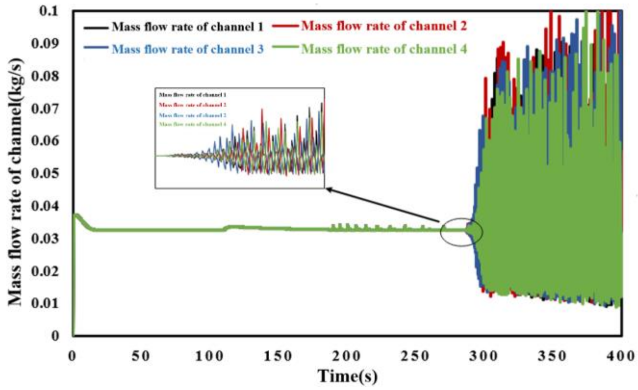


Figure 10. Typical flow rate curve for parallel quadruple-channel system with symmetric uniform heating.

In the simulation of parallel heating multi-channel system, as the heating power increases, two-phase flow of gas and liquid begins to appear when the outlet flow of the heating channel reaches saturation. As the outlet gas content of two-phase flow gradually increases, the flow rate of each channel in the parallel multi-channel system shows periodic flow pulsation when the outlet gas content reaches a certain level, which indicates the occurrence of flow instability, as shown in the local enlarged figures in Figures 8-10.

Under different operating conditions, the values of the subcooling degree number N_{sub} and phase change number N_{pch} at the point of flow instability were calculated for 2, 3, and 4 heating channels, and the flow instability boundaries of the parallel multi-channel system were compared based on these values as shown in Figure 11. It can be seen clearly from the figure that the flow instability boundaries are basically consistent with differences within ±5%. The simulation results are consistent with the experimental results of Wang Yanlin et al. (Wang 2021), indicating that when there are multiple parallel heating channels, adopting a parallel dual-channel structure can obtain the flow instability boundary.

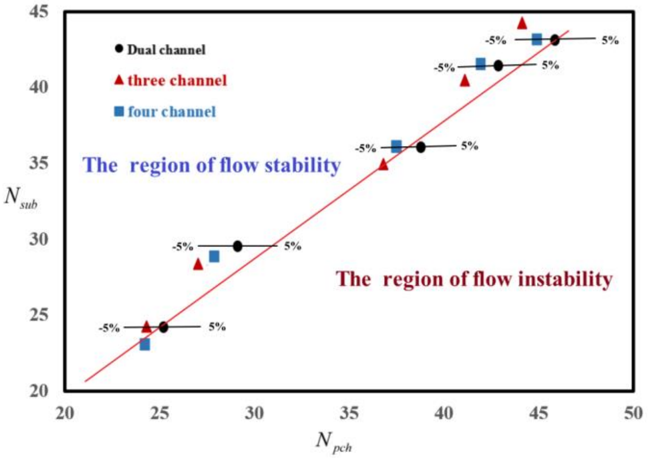


Figure 11. Flow instability boundaries under different heating channel numbers

4 Conclusion

(1) Flow instability is one of the important research topics in nuclear reactor thermal safety, and simulation analysis can serve as an important means for conducting flow instability research. In this paper, a parallel dual-channel heating model was built using the NUMAP developed based on Modelica language. Simulations and analysis of flow instability by this model were conducted. The simulation results met the error requirements of the experimental conditions, verifying the transient calculation and analysis ability of NUMAP for flow instability phenomena.

(2) Simulation study of a parallel multi-channel heating model was conducted using the NUMAP. The values of subcooling degree number N_{sub} and phase change number N_{pch} at the onset of flow instability were calculated for 2, 3, and 4 heating channels. The difference of the flow instability boundary was within ±5%. This indicates that when there are multiple parallel heating channels, a parallel dual-channel structure can be used to obtain the flow instability boundary.

5 prospects

Simulation analysis software is one of the indispensable core technologies for Nuclear Reactor System research and development, and also an important cornerstone of Nuclear Reactor System innovation. The NUMAP is developed by giving full play to the characteristics of Modelica physical Modeling language's

DOI
10.3384/ecp204569

object-oriented, equation based, reusable and reconfigurable hierarchical component model technology and its advantages in multi physical and multi-scale collaborative simulation。

Compared to traditional nuclear thermal hydraulic software, the NUMAP developed based on Modelica language has the following advantages: in terms of user experience, the drag and drop modeling method has greatly improved interactivity and intuitiveness in the modeling process; In terms of post processing of results, NUMAP provides real-time monitoring of variable parameter processes based on data points and XY curves, which can clearly and intuitively obtain the variation patterns of parameters and the linkage relationships of various parameters during the simulation process; In terms of application, the characteristics of Modelica multi domain unified modeling endow NUMAP with the ability of joint simulation in different fields of Mechanical, electrical, hydraulic and control systems. It provides not only thermal hydraulic system model library for nuclear power direction simulation, but also mechanical, electrical, control and other multi domain model libraries.

In the next stage, we will use Modelica language to continuously develop nuclear domain model libraries on the NUMAP, such as nuclear Nuclear reactor core model, pump model, steam generators model， Steam turbine model and other thermal hydraulic models, as well as nuclear instrumentation and control, mechanical and electrical system models. We will build NUMAP into a large multi discipline, multi-level and multi system unified modeling and simulation platform based on Modelica that promote high-quality research and development of nuclear power systems.

References

Xu Jiyun. Boiling Heat Transfer and Gas-Liquid Two-Phase Flow [M]. Beijing: China Atomic Energy Press, 2001.

Cheng K, Meng T, Tian C, et al. Experimental investigation on flow characteristics of pressure drop oscillations in a closed natural circulation loop[J]. International Journal of Heat and Mass Transfer, 2018, 122: 1162-1171.

Tang Y, Chen B, Xiong W, Wang Y. Experimental study on flow instability of the first kind of density wave in rectangular parallel dual-channel[J].Nuclear Power Engineering, 2014,35(03):26-30. DOI:10.13832/j.jnpe.2014.03.0026.

Peng C, Zan Y, Yuan D, et al. Study on flow instability of parallel channel flow drift[J].Nuclear Power Engineering, 2021,42(S1):17-20.DOI:10.13832/j.jnpe.2021.S1.0017.

Zang J, Yan X, Huang Y. Numerical simulation of flow instability in parallel channels with supercritical water[J]. Nuclear Power Engineering, 2021, 42(02): 72-76. DOI:10.13832/j.jnpe.2021.02.0072.

Qian L, Ding S, Qiu S. Research on flow instability model of parallel rectangular channels[J]. Nuclear Power Engineering, 2014, 35(02): 41-46.

Xiong W, Tang Y, Chen B. Numerical simulation of flow instability of parallel rectangular dual-channel density wave based on one-dimensional drift flow model[J]. Atomic Energy Science and Technology, 2015, 49(11): 1989-1996.

Xia G, Dong H, Peng M, Guo Y. Analysis of pulsation instability between narrow gap channels[J]. Atomic Energy Science and Technology, 2011, 45(09): 1034-1039.

Xing L, Guo Y, Zeng H. Analysis of flow instability in single-channel natural circulation based on RELAP5[J]. Atomic Energy Science and Technology, 2010, 44(08): 958-963.

Zhao J, Ding J, Zhou F, et al. Modelica language and its unified modeling and simulation mechanism in multiple fields[C]// China System Simulation Society. Proceedings of the Fifth National Congress of the China System Simulation Society and the 2006 National Academic Annual Meeting. 2006: 578-581.

Zhang H. Graphical modeling and simulation of marine steam turbine based on Modelica/Mworks[J]. Mechanical and Electrical Equipment, 2022, 39(04): 97-102. DOI:10.16443/j.cnki.31-1420.2022.04.018.

Huang Y, Zeng X, Ding J. Simulation model architecture and conceptual verification of two-phase thermohydraulic characteristics based on Modelica[J]. Nuclear Power Engineering, 2021, 42(01): 1-7. DOI:10.13832/j.jnpe.2021.01.0001.

NuclearSafety Analysis Division. RELAP5/MOD3.3 code manual, volume V: User's guidelines [M]. Rockville, Maryland, Idaho Falls, Idaho: Nuclear Safety Analysis Division, Information Systems Laboratories, Inc., 2001.

Tao W. Numerical Heat Transfer [M]. 2nd edition. Xi'an Jiaotong University Press, 2001, 264-275.

Wang Y, Zhou L, Zan Y, et al. Experimental study on flow instability of parallel multi-channel[J]. Nuclear PowerEngineering,2021,42(01):15-17.DOI:10.13832/j.jnpe.2021.01.0015.

Ishii. Thermal induced flow instabilities in two-phase mixtures in thermal equalibrium[D]. Geogia Institute of Technology, Mechanical, 1942.

Integration of Heat Flow through Borders between Adjacent Zones in AixLib's Reduced-Order Model

Philip Groesdonk[1,2] David Jansen[3] Jacob Estevam Schmiedt[1] Bernhard Hoffschmidt[1,2]

[1]Institute for Solar Research, German Aerospace Center (DLR), Germany,
{philip.groesdonk,jacob.estevamschmiedt}@dlr.de
[2]Chair of Solar Components, RWTH Aachen University, Germany
[3]Institute for Energy Efficient Buildings and Indoor Climate, E.ON Energy Research Center, RWTH Aachen University, Germany, david.jansen@eonerc.rwth-aachen.de

Abstract

For dynamically simulating the thermal behavior of a building, the reduced-order model (ROM) implemented in the Modelica IBPSA and AixLib libraries provides a time-efficient calculation method based on the standard VDI 6007-1. Additionally, the Python package TEASER features a possilibity to fill the model parameters with automatically generated typical and/or enriched building data. So far, both have not been capable of modelling heat flow through borders between thermal zones. In this contribution, we present the integration of this feature into the open-source software combination. Additional new features include non-constant soil temperatures and a new approach to estimate interior building elements in cases without proper knowledge. Calculation results are presented for an exemplary application and show satisfactory agreement with measured values. The respective code (including the example presented here) is in the process of being published as part of the AixLib and TEASER open-source repositiories.

Keywords: AixLib, TEASER, building simulation, archetype, BIM, BEM

1 Introduction

The building sector faces the need to reduce carbon emissions and increase energy efficiency drastically. As a consequence, building energy simulation has become an important tool to investigate the possible effects of planned measures in single-building applications as well as to get an overview of the building stock with limited data availability on an urban scale. Among others, Modelica libraries for building energy simulation have been created in recent years. For example, important advances in open-source libraries for these purposes have been reached in the framework of IBPSA Project 1 and the previously completed project IEA EBC Annex 60 (Wetter, Treeck, et al. 2019). The library AixLib (Müller et al. 2016) is also connected to this framework and is being continuously developed further. In this publication, we present the integration of a feature for heat flow through borders between adjacent thermal zones into the so-called `ReducedOrder` model (ROM) of AixLib and the complimentary Python tool TEASER (Remmen et al. 2018). The motivation for this work was the possibility to enrich geometrically available data of existing buildings with typical thermal properties using TEASER, thereby enabling a workflow to create fully parameterized simulation models quickly in cases of limited (digital) data availability. For this use case, the feature has been previously identified as a shortcoming of the ROM (Jansen et al. 2021).

This paper is structured as follows: Section 2 introduces the state of the art regarding the workflow that the ROM is usually applied in, the model concept itself, and previous work on the topic treated here. In Section 3, we describe the methodology for the new implementation in AixLib and TEASER. An exemplary application and its results are presented in Section 4. We discuss them further in Section 5. In Section 6, we conclude the paper and give an outlook to future work.

2 State of the Art

The foundations for this work were created by the developers of the open-source Modelica library AixLib and the complimentary Python tool TEASER. Therefore, relevant aspects of these tools that this publication builds up on are introduced in the following. Additionally, this section summarizes a previous approach to introduce heat flow between adjacent zones into the combination of the two tools and what is recommended in this regard by the guideline VDI 6007-1, which is the theoretical basis for the used model.

2.1 Automatically Parameterized Modelica Simulation Models

Figure 1 shows the workflow of TEASER to generate dynamic building simulation models and the corresponding simulation results. The main strength of TEASER is the automation of the time-consuming parameterization of the model described in VDI 6007-1. TEASER allows creating parametric building models in Python and to transfer them directly into a Modelica model, automating the complete calculation of the relevant variables. The availability of statistical data in TEASER enables a creation of so-

DOI
10.3384/ecp204577

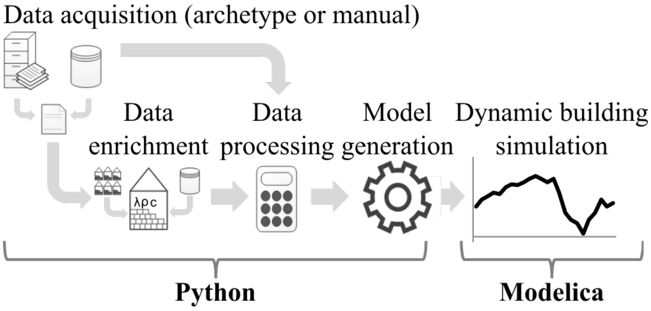

Data acquisition (archetype or manual)
Data enrichment
Data processing
Model generation
Dynamic building simulation
Python
Modelica

Figure 1. Generation process of Modelica models using TEASER (Remmen et al. 2018).

called archetype models with a minimal data set of input data. In addition, detailed creation of concrete buildings is also possible. In this case, the modeler specifies the entire building, all thermal zones including the usage conditions of these zones, and the assignment of the respective components (outer walls, inner walls, windows, etc.) to them. Each component can then be given layer structures and materials. In this process, data enrichment can be used at any time to enrich missing information, such as wall structures or information about usage profiles.

2.2 AixLib's Reduced-order Model

AixLib is an open-source Modelica library for building energy simulation based on the Modelica IBPSA Library (Wetter, Blum, and Hu 2019). It is currently (June 2023) available in version 1.3.2. AixLib contains a wide range of models covering heating, ventilation, and air conditioning (HVAC) equipment as well as two packages for thermal zone models with different level of detail (`HighOrder` and `ReducedOrder` (ROM)). The latter is relevant for this contribution. Its development is described in detail by Lauster (2018). In the following, some relevant aspects of the model structure and its calculation approach are explained briefly.

The hierarchical concept of the ROM is visualized in Figure 2. A building is represented by a `Multizone` object. This object mainly serves to collect externally defined boundary conditions, such as weather data, setpoint temperatures, and internal gains. Furthermore, it optionally contains a model of an air handling unit (e.g. for ventilation systems). Within the `Multizone` environment, an array of n_{zones} `ThermalZone` objects is specified. The boundary conditions are passed to these objects. Each thermal zone consists of a core resistance-capacitance (RC) module and supplementary components.

Core RC modules are available in different levels from `OneElement` to `FourElement`. With decreasing number of elements, more building components (roof and floor plate, in that order) are lumped into the element for exterior walls. For the final step from `TwoElement` to `OneElement`, the inner walls, i.e. solid interior masses, are neglected. As roofs and exterior walls do not differ in their description, presenting the `ThreeElement` model is sufficient here. Figure 3 shows a visually adjusted version of the thermal network representation by Lauster (2018).

In the network, nodes represent temperatures. If they are connected to capacities, a thermal mass with that temperature is present. Resistances govern the heat flow between temperature nodes. Some heat flows, represented by arrows in the figure, are prescribed boundary conditions. Blue-coloured boxes are parts of the network that may be repeated in a series connection. However, this feature is not used for the scope of this paper.

The centre-right node in the network represents the air inside the zone with temperature ϑ_{air}. $\dot{Q}_{\text{g,cv}}$ is the sum of convective heat gains, including the convective share of heat flow from solar gains through windows, heating and cooling, machines, lights, and humans. Radiative flows from the same sources are directly applied to the different lumped elements' surfaces. On the far side of the elements, a fixed soil temperature ϑ_{soil} (floor plate element) and equivalent temperatures merging convection and both longwave and shortwave radiation (window, exterior wall, and roof elements) are set. For the simulation, this means that the different zones are not interconnected, which reduces calculation complexity. In practice, the `TwoElement` model has shown to be a good trade-off between calculation times and accuracy (Remmen et al. 2018). Lumping to two elements is also suggested by VDI 6007-1 (2015-06), the standard on which the modelling approach of ROM is based.

2.3 Heat Exchange with Adjacent Zones According to VDI 6007-1

As mentioned in Section 2.2, AixLib's ROM so far does not feature a possibility to model heat flow between zones. However, the underlying standard VDI 6007-1 mentions adjacent zones. It suggests lumping heat flow through borders to adjacent rooms with the heat flow to the exterior using an equivalent temperature

$$\vartheta_{\text{eq,NR}} = \vartheta_{\text{air,NR}} + \frac{\dot{Q}_{\text{rad,se,NR}}}{\alpha_{\text{cv,se,NR}} \cdot A_{\text{se,NR}}}, \tag{1}$$

where $\vartheta_{\text{air,NR}}$ is the air temperature in the adjacent room, $\dot{Q}_{\text{rad,se,NR}}$ is the sum of the radiant heat sources and sinks onto the adjacent room's wall surface $A_{\text{se,NR}}$, and $\alpha_{\text{cv,se,NR}}$ is the convective heat transfer coefficient on that surface. $\vartheta_{\text{eq,NR}}$ is supposed to be merged into the equivalent temperature calculation of the 2-element model mentioned above.

2.4 Previous Work on the Topic

Previous efforts to include interzonal heat transfer into the TEASER/AixLib tool chain were made in connection with measurements at an exemplary single-family building (Gorzalka et al. 2021). Instead of following Equation 1 and introducing adjacent rooms into the equivalent temperature, a fifth element was inserted into the ROM's

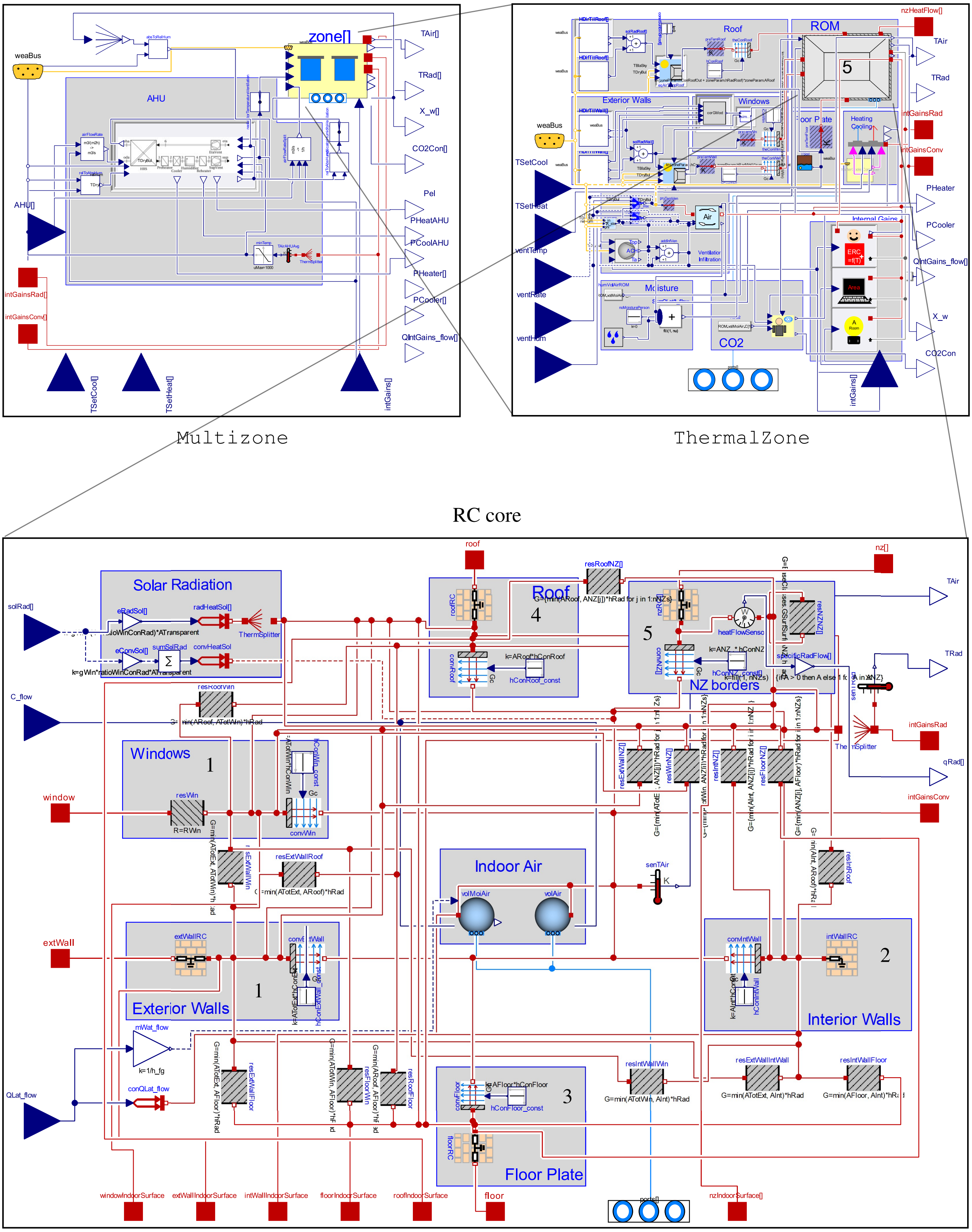


Figure 2. Visualization of the ROM concept with the three model levels. Numbers in the RC core indicate which elements are added from `OneElement` up to the `FiveElement` model introduced here. Zoom in for details.

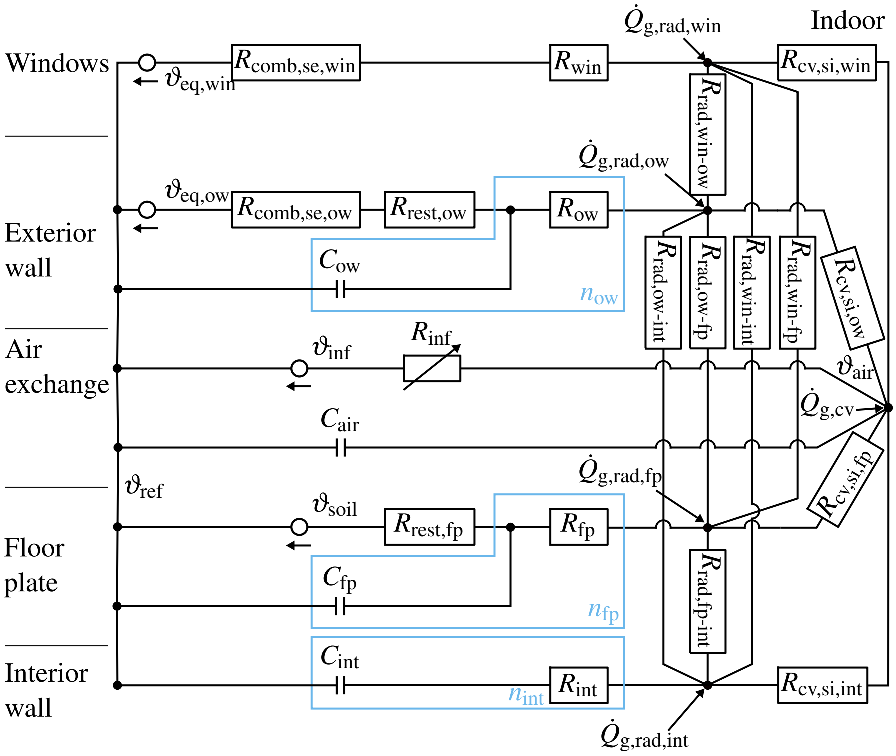

Windows
Exterior wall
Air exchange
Floor plate
Interior wall
Indoor
$\vartheta_{\mathrm{eq,win}}$
$R_{\mathrm{comb,se,win}}$
R_{win}
$\dot{Q}_{\mathrm{g,rad,win}}$
$R_{\mathrm{cv,si,win}}$
$R_{\mathrm{rad,win\text{-}ow}}$
$\vartheta_{\mathrm{eq,ow}}$
$R_{\mathrm{comb,se,ow}}$
$R_{\mathrm{rest,ow}}$
R_{ow}
$\dot{Q}_{\mathrm{g,rad,ow}}$
C_{ow}
n_{ow}
$R_{\mathrm{rad,ow\text{-}int}}$
$R_{\mathrm{rad,ow\text{-}fp}}$
$R_{\mathrm{rad,win\text{-}int}}$
$R_{\mathrm{rad,win\text{-}fp}}$
$R_{\mathrm{cv,si,ow}}$
ϑ_{inf}
R_{inf}
ϑ_{air}
$\dot{Q}_{\mathrm{g,cv}}$
C_{air}
ϑ_{ref}
$\vartheta_{\mathrm{soil}}$
$R_{\mathrm{rest,fp}}$
R_{fp}
$\dot{Q}_{\mathrm{g,rad,fp}}$
$R_{\mathrm{cv,si,fp}}$
C_{fp}
n_{fp}
$R_{\mathrm{rad,fp\text{-}int}}$
C_{int}
n_{int}
R_{int}
$R_{\mathrm{cv,si,int}}$
$\dot{Q}_{\mathrm{g,rad,int}}$

Figure 3. Thermal network representation of the AixLib `ThreeElement` model according to Lauster (2018), visually adjusted. Refer to Tables 3 and 4 for an explanation of the variables and indices.

core RC model. This fifth element and the RC elements for exterior walls and roofs were defined as Modelica arrays instead of single RC chains such that not all building parts, but only those with equal azimuth and tilt, were lumped into one element. With this and additional modifications e.g. regarding non-constant soil temperatures, a satisfactory agreement between the room air temperature measurements within the example building and the results of the simulation model was reached. In total, a largely automated thermal modelling workflow for existing buildings based on drone images was demonstrated.

3 Methodology

Building up on the works presented in the previous section, we developed a methodology to integrate heat flow through borders between adjacent zones into the AixLib ROM as well as its counterpart in the Python package TEASER. This section describes both aspects.

3.1 `FiveElement` ROM

We initially considered three possible approaches for the integration of heat exchange with adjacent zones into the ROM:

1. The equivalent temperature approach described in Section 2.3
2. The vectorized approach described in Section 2.4
3. The introduction of a fifth element without modifications to the rest of the ROM

Trying to stick as much as possible to the design principles previously applied in the model, we ruled out option 1: Equation 1 does not fit to the implementation of a largely simplified equivalent temperature calculation entirely based on boundary conditions. Furthermore, interconnecting $\dot{Q}_{\mathrm{rad;se;NR}}$ from within the core ROM with the far side of the other zone and vice-versa would result in modelling the same building element twice, but lumped with different other elements and therefore with different boundary conditions. This would not only risk physically false results, but also simulation crashes.

Option 2 would require significant changes to the core RC model and affect the established IBPSA core library. Additionally, the benefits of a more detailed model were considered not worth the increase in calculation time, as the ROM is mostly used for simplified modelling.

Option 3 seemed most promising as a consequence. In our implementation, we keep the thermal network (shown for three elements in Figure 3) and add a possibility to connect multiple thermal zones, in this regard following the vectorization idea presented by Gorzalka et al. (2021). Each zone border element is modelled as part of the RC model of the zone with the lower index through the `Multizone` model (see Figure 2) to avoid double modelling. In the adjacent zone, the heat flow is directly connected to the surface area. The resulting thermal network is shown in Figure 4. In the figure, array connections are represented by dashed lines. The dotted line between the interior surface nodes stands for the pairwise connection of the nodes by resistances for radiation heat exchange.

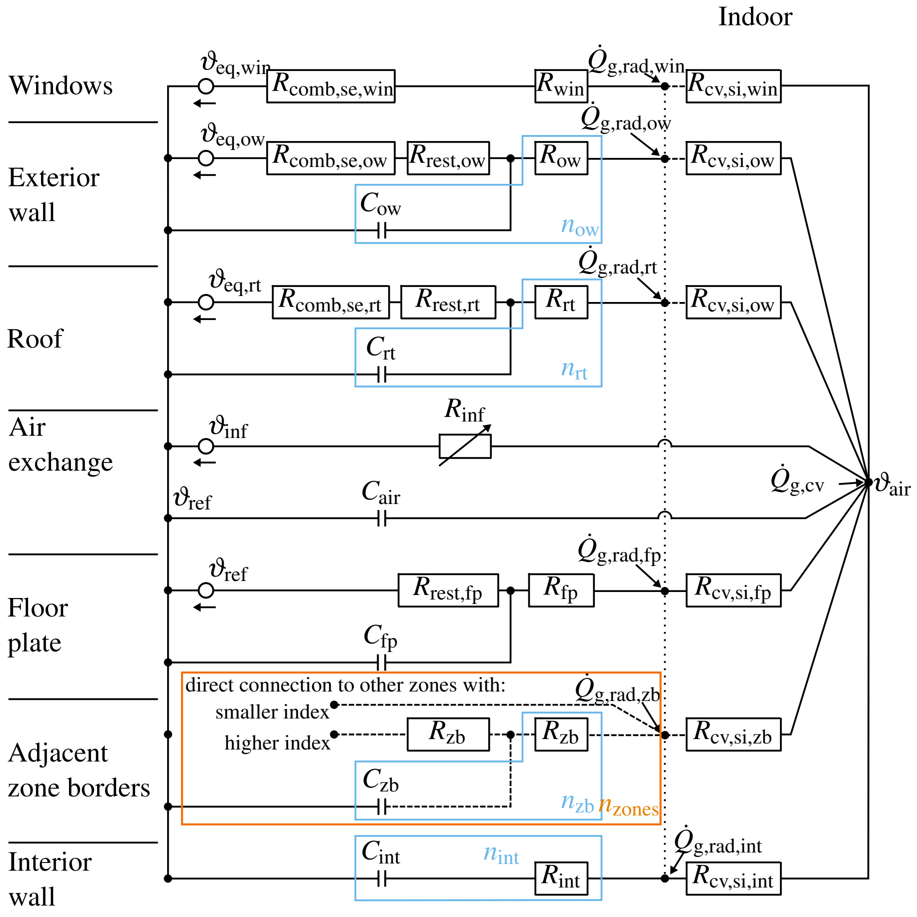


Figure 4. Thermal network representation for the new `FiveElement` implementation in AixLib, with the fifth element modelling heat exchange with adjacent zones. Refer to Tables 3 and 4 for an explanation of the variables and indices.

Other than in the original implementation, the temperature on the outer surface of the ground floor element ϑ_{soil} is not necessarily a constant here. It can also be connected to a table in a file, a sine function, or the `AixLib.BoundaryConditions.GroundTemperature.GroundTemperatureKusuda` model that was already a part of AixLib.

3.2 Complimentary Features in TEASER

TEASER is an integral part of the automatic model creation workflow shown in Section 2. As a consequence, features of the ROM should also be represented there. In this case, we added three features to TEASER: (i) a representation of borders between adjacent zones including the interface to the `FiveElement` ROM; (ii) a possibility to adapt the boundary conditions of the exported Modelica model, including non-constant soil temperatures and a partly customizable interface; and (iii) a new estimation approach for interior thermal masses that accounts for the newly added zone borders.

3.2.1 Zone borders

So far, TEASER has featured `OuterWall`, `Rooftop`, `GroundFloor`, `Window`, and `Door` elements modelling building elements between a zone and the exterior. Additionally, the `InnerWall`, `Ceiling`, and `Floor` elements are used to describe the vertical and horizontal interior thermal masses for the zone. Following the principle for the inner elements, we introduce `InterzonalWall`, `InterzonalCeiling`, and `InterzonalFloor` for modelling borders to other zones on the same floor, on a floor above, and on a floor below respectively. Upon export to the AixLib ROM, they are lumped to a single element per adjacent zone using the established algorithms of TEASER. The workflow of enriching data of an only geometrically described building in TEASER and exporting it to Modelica is described in the following and visualized in Figure 5.

One of the arguably most important features of TEASER is the availability of default layers and thermal properties for the building elements, e.g. from the TABULA typology (Loga, Stein, and Diefenbach 2016). This enables the user to add energetically relevant data for a building for which only the envelope geometry was known before. So far, all boundaries to the exterior had their counterparts in TABULA and were mapped to default layers and U-values from that database. For inner elements, although not covered by TABULA as such, typical el-

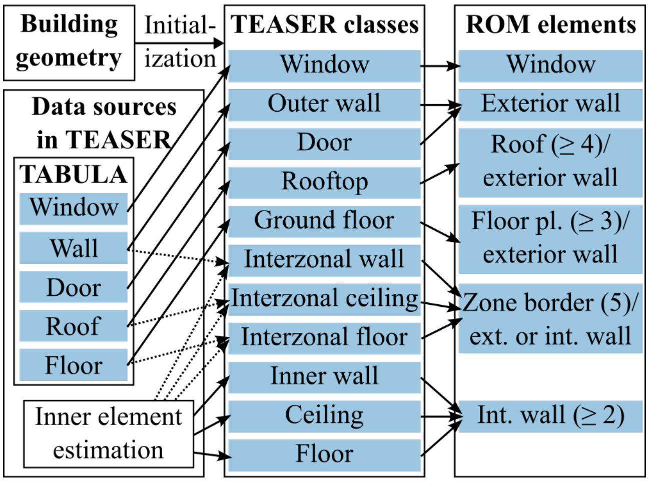


Figure 5. Visualization of the data sourcing process for the ROM used by TEASER described in Section 3.2. Numbers in parentheses represent the number of ROM elements, e.g. "Floor pl. ($\geq$ 3)/exterior wall" means the ground floor is exported to a floor plate element from `ThreeElement` on and is lumped to the exterior wall element for lower-order models.

ements for each of TABULA's building age periods are available in TEASER, too. Lacking a proper state-of-the-art approach, Lauster (2018) implemented an algorithm estimating their size based on the number of floors and usual room lengths and widths for the given use conditions, assuming that the uppermost border of a zone is the rooftop, the lowermost is the ground floor, and that each room has one outer wall.

As they can either be borders to unconditioned zones or to other conditioned zones, the new interzonal elements do not have a direct counterpart in TABULA. Therefore, depending on whether the associated thermal zones are equally conditioned or not, we map the elements either to the default respective outer or inner type element. Due to the hierarchical system of TEASER, each zone border is created for each of both zones. The resulting Python objects are assigned equal properties. Default layer sequences are reversed for the border elements of unconditioned thermal zones. This is also considered when calculating the RC parameters for the ROM using the asymmetrical algorithm of VDI 6007-1. Although the creation of two separated element objects for one physical zone border means that care has to be taken to keep the model consistent if changes are made after data enrichment, the export to AixLib uses only one of the two elements due to the RC component not being modelled in the higher-indexed zone as visible in Figure 4.

3.2.2 Interface to the AixLib ROM

In addition to the previously existing Modelica file templates for `OneElement` to `FourElement` models, we added a `FiveElement` zone parameter template covering the parameters of the new interzonal elements. The user can choose for which pairs of zones (e.g. depending on heating and cooling setpoints) interzonal heat transfer should be considered. Other elements are treated as inner elements on each side. If exporting to a ROM with less than five elements, interzonal elements are lumped to the exterior wall element if an unheated zone is on the other side or to the interior wall element otherwise. The sine model and table options for the soil temperatures were included in all five element templates. Furthermore, we added an option to introduce custom `Multizone` templates, which allows more individual boundary condition settings like custom weather file readers, internal gains, or setpoint tables.

3.2.3 Interior Thermal Mass Estimation

As a matter of fact, the previously available approach to estimate the size of inner walls, floors, and ceilings (see Section 3.2.1) does not consider interzonal elements. Keeping it would increase the tendency of TEASER to overestimate interior thermal masses. However, the typical length and width of a room defined by the usage as introduced by Lauster (2018) is still the best base for the calculation that we could find. Using the number of floors and rooms that the zone should have depending on its area and height, the new `'typical_minus_outer'` option in TEASER estimates the area of inner elements by subtracting all bordering elements (considering their tilt) from the overall surface area of the typical rooms separately for walls, floors, and ceilings.

4 Exemplary Application and Results

With the methodology presented above, a `FiveElement` model was created for the exemplary building used by Gorzalka et al. (2021). This section presents the results of simulating the about three weeks of time with changing heat load within the building (warm-up, approximately constant temperature, free cooling) for which measured air temperatures are available.

4.1 Model Setup

The exemplary building is a vacant single-family house in Morschenich, Germany. It consists of two heated floors interconnected with an open staircase and unheated floors (basement and attic) below and above. Geometry and temperature measurements are sourced from an actual building. The thermal properties of the building elements are based on the best knowledge after on-site assessment for one of the model variations (D) and typical values for a single-family house in Western Germany built in the 1960s with windows exchanged in 1995 for the others (A to C) with different properties of the interior masses.

For modelling, the two heated floors were considered as a single zone. The Modelica model was created in four different variations:

- With the purpose of assessing the influence of model parameterization, three variations (A to C) cover the different estimation approaches for interior thermal masses. All building elements were given the typical properties as mentioned above through TEASER. With the areas for the three zones listed in Table 1, variation A is based on the one previously implemented in TEASER, variation B uses the newly implemented approach, and variation C uses the known area of the building. The attic has no inner walls, so its interior element surface area is always 0 m^2.

- For demonstrating the applicability of the model in a real-world case, one variation (D) uses the actual thermal properties of the building as far as they are known from Gorzalka et al. (2021): ("[The variation] contains the best knowledge from building plans and on-site investigations about the walls (exterior and interior) and the roof of the building. As the actual compositions of the building parts in contact to soil and of the basement ceiling are unknown, it falls back to TABULA values there.").

Table 1. Surface area of the interior elements of each zone and source for materials and thermal properties for the four model variations.

V.	*Interior element surface in* m^2			*Material/layer*
	Attic	*Basement*	*Main zone*	*data source*
A	0	171.15	501.52	TABULA
B	0	136.62	388.69	TABULA
C	0	81.68	265.05	TABULA
D	0	81.68	265.05	Best knowledge

For all variations, heating setpoints (to very high temperatures) and usage profiles (to zero) were set in such a way that they do not influence the simulation. Instead, the measured loads of the installed heaters were defined as internal gains. Air exchange rates were kept at the default value for the conditioned zone. For the unconditioned zones, they were set to 10 h^{-1} for the attic and 1 h^{-1} for the basement, following the recommendations in Table 7 of ISO 13789 (2017-06). Table 8 in the same standard is the source for the interior surface heat transfer coefficients of non-vertical surfaces. They are set to $5.0\,\mathrm{W\,m^{-2}\,K^{-1}}$ for upwards and $0.7\,\mathrm{W\,m^{-2}\,K^{-1}}$ for downwards heat flow. The default value in TEASER is $1.7\,\mathrm{W\,m^{-2}\,K^{-1}}$ for both because the coefficients are constants in the ROM and the direction of the heat flow changes over the course of a full year.

Weather data (temperature and solar radiation) as well as the temperature of the surfaces in contact to soil were sourced from measurements recorded on site during the test period.

4.2 Temperature Comparison

The model was simulated from January 18, 8:00 to March 1, 16:00 (simulation time 1 497 600–5 155 200 s, where 0 s is the beginning of the year 2019). This left enough time for the model to stabilize under constant weather conditions (the actual building was unheated at the time) before the comparison period starts in the evening of February 4. Simulated temperatures and the volumetric mean of roomwise temperature measurements are compared for the conditioned zone in Figure 6 and for basement and attic in Figure 7.

For the conditioned zone, the results show that the two variations with automatically estimated interior surfaces areas (A and B) fit very well to the measurement in periods without a steep increase or decrease of the temperature (February 11 to 16 and 22 to 26). Variation C with the actual surface areas shows a tendency to overheat (February 7 to 12) and cool down too much (February 20 to 26). However, given that the thermal properties of the building were taken from the typology rather than from actual values, differences to the measured temperatures are not relevant for an evaluation of the model.

Variation D containing the best knowledge of the actual building shows an overall good agreement to the measured temperatures. The root-mean-square error (RMSE) for the hourly temperature difference between February 5, 15:00 and February 26, 24:00 is 1.13 K. Obvious deviations occur during warm-up (too slow until February 8, slight overheating afterwards; overall RMSE 1.51 K between February 5, 15:00 and February 13, 11:00) and cooldown (overall RMSE 0.90 K between February 17, 1:00 and February 26, 24:00). In the period of approximately constant temperature between February 13, 12:00 and February 16, 24:00, the simulated temperature is mostly overestimating the measured temperature to a minor degree with an RMSE of 0.59 K.

The sensitivity to changing interior masses is also interesting. Here, the reduction by 22.5% from variation A to B has an only minor impact. The additional reduction by 31.8% (approximately the same absolute reduction) to variation C changes the model behaviour to a far larger extent. Although this calls for further investigations into the sensitivity of the ROM, the different variations are comparable in times with dynamic loads in the magnitude of those appearing in usual application cases, i.e. interior temperatures of conditioned zones being kept within a range of a few K. A similiar observation can be made for the unconditioned zones. Here, the difference in daily fluctuations of the basement temperature between measured (almost no fluctuation) and all four simulated air temperatures (about 1–5 K) might be the result of an overestimated air exchange rate or of the model neglecting the vertical temperature distribution in the heated zone. For the attic, a poor performance of variation D is apparent between February 13 and February 18 during night times. Possible reasons are the same as for the deviations in basement temperatures. Variation C compensates these issues by a higher-than-actual U-value ($0.84\,\mathrm{W\,m^{-2}\,K^{-1}}$ instead of $0.52\,\mathrm{W\,m^{-2}\,K^{-1}}$) for the interzonal ceiling.

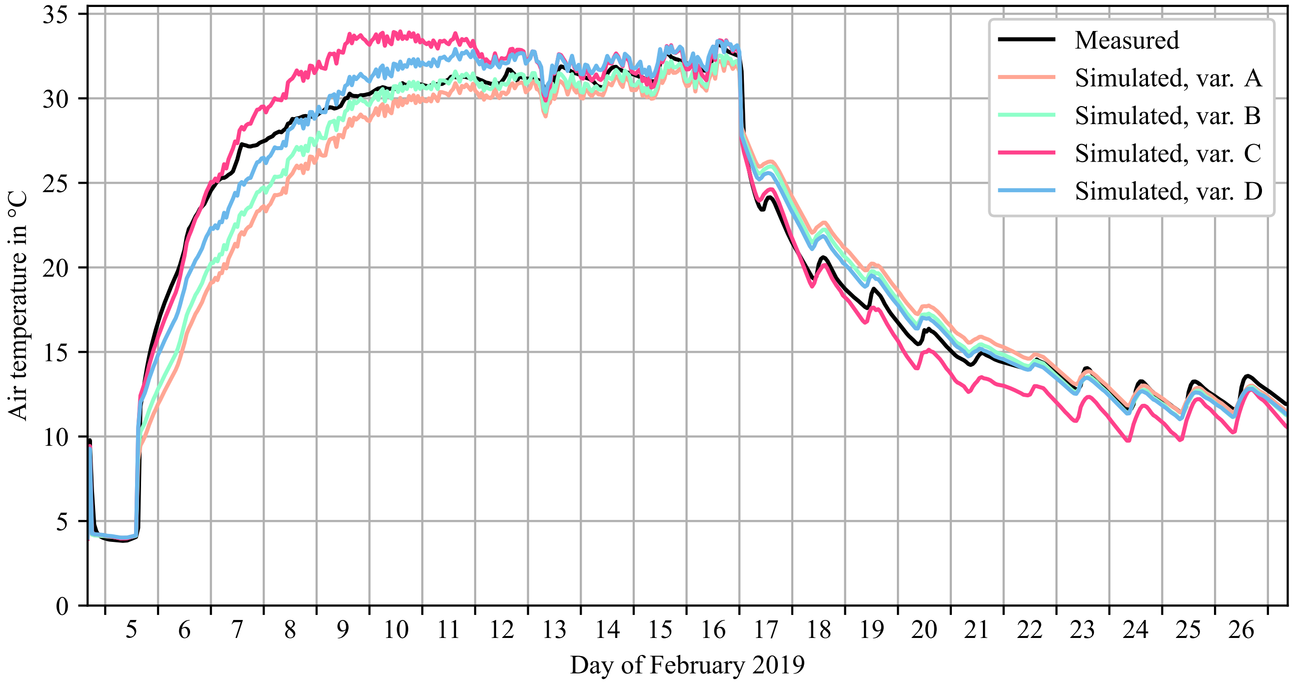


Figure 6. Measured and simulated mean air temperatures for the conditioned zone of the exemplary building.

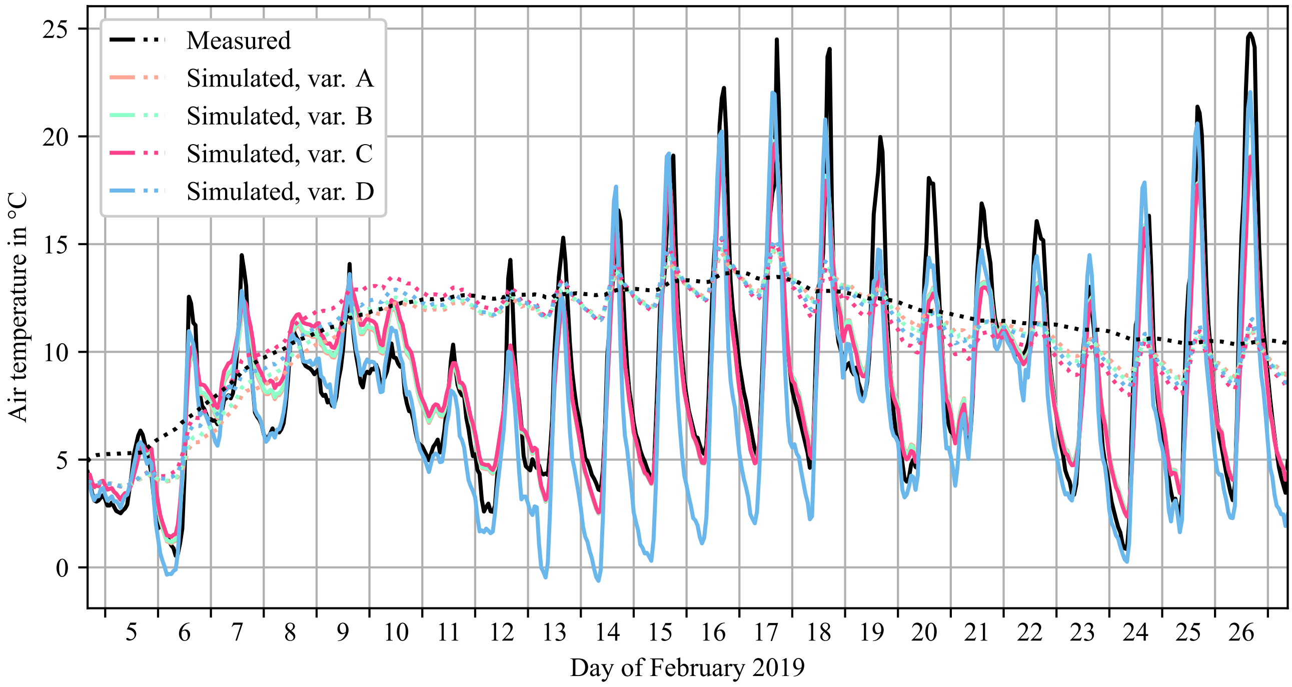


Figure 7. Measured and simulated mean air temperatures for basement (dashed lines) and attic (solid lines) of the exemplary building.

4.3 Comparison to `TwoElement` and `FourElement`

To evaluate the influence of the `FiveElement` model on simulation results and its impact on calculation time, we created models for the exemplary building with two, four, and five elements. In the `TwoElement` and `FourElement` model, unheated zones were not included. Instead, we replaced the borders to the attics by a `Rooftop` element with reduced outer convection and the borders to the basements by a `GroundFloor` element. The temperature of the soil in contact to the outer surface of these and the `FiveElement` basement's `GroundFloor` elements was set to the TEASER default value of 13 °C. Results for a simulation of the test period showed that the fit to the measured values depends heavily on this soil temperature value. As a consequence, we decided to simulate a full year under comparable conditions, which is a more useful indicator for model performance. In these full-year models, default use conditions for residential zones were taken from TEASER, with the following exceptions:

- No internal gains and neither heater nor cooler in the unheated zones of the `FiveElement` model
- Cooling active with setpoint 25 °C
- Heating setpoint 20 °C

Table 2 shows the calculation times on a Windows notebook (Dymola 2020x, 32-bit compiler, solver Radau, tolerance 1×10^{-4}) as well as the heating and cooling energy used for the main zone from the simulation run.

Table 2. Duration of the calculation and integrated heating and cooling power for the full-year simulations of the `TwoElement`, `FourElement`, and `FiveElement` representations of the exemplary building.

Elements	*Duration*	*Heating*	*Cooling*
2	13 s	204 MWh	39 kWh
4	11 s	203 MWh	51 kWh
5	35 s	201 MWh	263 kWh

5 Discussion

The previously presented results have shown an acceptable agreement to measured values, given the dynamic boundary conditions and the reduced order of complexity of the model. This demonstrates and verifies the ability of the ROM to model actual building operation. To our knowledge, validating the dynamic model of the interzonal heat transfer is not possible with the applicable standards. All test examples in VDI 6007-1 consider only one room. In ANSI/ASHRAE Standard 140 (2020), the tests relevant for the ROM are those of class I. They comprise a test case with interzonal heat transfer (case 960). Upon contribution of the presented model to the open-source version of AixLib, we plan to add it in addition to the already implemented VDI 6007-1 and ASHRAE 140 cases for single-zone applications. However, there are only annual and no hourly validation results available for case 960. So, the validity of the dynamic calculations cannot be tested.

Regarding the interior thermal masses, the results show their importance in application cases with highly dynamic loads. However, this is rarely the case for use cases of the `ReducedOrder` model. As a consequence, the new estimation approach for interior masses improves the consistency of the overall workflow, but is most likely not essential for reliable results.

Although interzonal heat transfer does not need to be considered for the use case of urban-scale simulations, it is very important when considering specific buildings. The prior lack of interzonal heat transfer for the ROM was already discussed by Jansen et al. by comparing simulation results of the ROM to results of the well-established simulation tool EnergyPlus (Jansen et al. 2021). With the implemented changes, this shortcoming of the ROM was solved. However, the expansion is also accompanied by an increasing parameterization effort, since it must be known which zones are in contact with each other via which components. Nevertheless, this effort is put into perspective, especially when automated approaches are used for model creation. For example, if Building Information Modeling (BIM) is used as a data source, the contact points of the zones can be automatically identified and forwarded to TEASER. The existing approach BIM2SIM[1], which also uses TEASER, can therefore use the presented changes to create more realistic ROMs based on BIM data.

The calculation time for a full-year `FiveElement` simulation showed to be three times the duration of a run without the two unheated zones and without interzonal elements (see Table 2). This shows that the added complexity of connecting the zones did not increase computation effort significantly more than simulating them in parallel, which is in line with past findings that "simulation time [...] is correlated to the number of state variables resp. thermal capacitances" (Lauster and Müller 2019). For unknown reasons, we also did not find an increased simulation time for `FourElement` in comparison to `TwoElement` although the number of state variables increases from 10 to 12. Regarding energy demand, the interzonal heat transfer influenced heating and cooling loads mainly in summer. In particular, replacing the constant soil temperature by a simulated basement zone caused a reduction of heat flows through the floor of the main zone.

6 Conclusion

In the previous sections, we presented a new feature of the AixLib `ReducedOrder` (ROM) model and its compli-

[1] `https://github.com/BIM2SIM/bim2sim`

mentary Python tool TEASER. It is now possible to simulate heat flow through borders between adjacent zones with the help of an additional RC component for these building elements available in the `FiveElement` model of the ROM. Furthermore, TEASER can now automatically source thermal properties of these building parts from the German TABULA typology. All contributions to the two software tools are currently in the process of being published open-source[2].

In an exemplary application to a single-family house, we have seen an acceptable agreement between measured and simulated air temperatures—interestingly not only with the best knowledge of the actual building, but also with statistical data for its age. Although this cannot replace the test of the new ROM features with an appropriate test case for validation, it demonstrates that the ROM is able to simulate the highly dynamic loads to the largely simplified single-family house, resulting in an overall RMSE of 1.13 K.

The importance of interior masses for dynamic simulation has been investigated by comparing two approaches for the estimation of inner wall sizes and the actual interior geometry of the building. The results show large differences during highly dynamic loads, but these loads rarely appear in the use cases of the ROM. This led to the conclusion that inner wall sizes are mostly a question of parameter consistency.

A modest increase in simulation time was predominantly caused by the added zones rather than by the interzonal elements. Therefore, we recommend to check which zones need to be explicitly simulated when using the feature.

Regarding the overall workflow, the new developments are embedded into other work regarding the BIM2SIM approach towards an automated toolchain from BIM via TEASER to the AixLib ROM that can create more realistic ROMs based on BIM data.

Used Symbols

Table 3. List of variables.

Symbol	*Meaning*	*Unit*
α	Heat transfer coefficient	$\mathrm{W\,m^{-2}\,K^{-1}}$
ϑ	Temperature	°C
A	Area	$\mathrm{m^2}$
C	Thermal capacity	$\mathrm{J\,K^{-1}}$
$\dot{Q}$	Heat flow	W
R	Thermal resistance	$\mathrm{K\,W^{-1}}$

Table 4. List of abbreviations in variable indices.

Index	*Meaning*
comb	Combined (convective and radiative)
cv	Convective
eq	Equivalent
fp	Floor plate
g	Gains
inf	Infiltration
int	Interior walls
NR	Adjacent (neighbouring) room
ow	Exterior (outer) wall
rad	Radiative
ref	Reference
rt	Rooftop
se	Exterior surface
si	Interior surface
win	Window
zb	Zone border

Acknowledgements

Parts of this work have been supported by the German Federal Ministry for Economic Affairs and Energy in the framework of the “Building Tomograph” (Gebäudetomograph, Gtom) project (grant number 03ET1405A).

References

ANSI/ASHRAE Standard 140 (2020). *Method of Test for Evaluating Building Performance Simulation*. Standard. American Society of Heating, Refrigerating and Air-Conditioning Engineers.

Gorzalka, Philip et al. (2021). “Automated Generation of an Energy Simulation Model for an Existing Building from UAV Imagery”. In: *Buildings* 11.9, p. 380. DOI: 10.3390/buildings11090380.

ISO 13789 (2017-06). *Thermal performance of buildings: Transmission and ventilation heat transfer coefficients: Calculation method*. Technical standard. International Organization for Standardization.

Jansen, David Paul et al. (2021). “Examination of Reduced Order Building Models with Different Zoning Strategies to Simulate Larger Non-Residential Buildings Based on BIM as Single Source of Truth”. In: *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*. Ed. by Martin Sjölund et al. Linköping, Sweden: Modelica Association and Linköping University Electronic Press, pp. 665–672. DOI: 10.3384/ecp21181665.

Lauster, Moritz (2018). “Parametrierbare Gebäudemodelle für dynamische Energiebedarfsrechnungen von Stadtquartieren”. Dissertation. Aachen, Germany: RWTH Aachen. DOI: 10.18154/RWTH-2018-230258.

Lauster, Moritz and Dirk Müller (2019). “Characterization of Linear Reduced Order Building Models Using Bode Plots”. In: *Proceedings of the 13th International Modelica Conference*. Linköping University Electronic Press, pp. 25–32. DOI: 10.3384/ecp1915725.

[2] see `https://github.com/RWTH-EBC/AixLib/issues/1080` and `https://github.com/RWTH-EBC/TEASER/issues/679`

Loga, Tobias, Britta Stein, and Nikolaus Diefenbach (2016). "TABULA building typologies in 20 European countries—Making energy-related features of residential building stocks comparable". In: *Energy and Buildings* 132, pp. 4–12. DOI: 10.1016/j.enbuild.2016.06.094.

Müller, Dirk et al. (2016). "AixLib - An Open-Source Modelica Library within the IEA-EBC Annex 60 Framework". In: *Proceedings of the CESBP Central European Symposium on Building Physics and BauSIM 2016*. Ed. by John Grunewald. Stuttgart, Germany: Fraunhofer IRB Verlag, pp. 3–9. URL: http://www.ibpsa.org/proceedings/bausimPapers/2016/A-02-3.pdf.

Remmen, Peter et al. (2018). "TEASER: An open tool for urban energy modelling of building stocks". In: *Journal of Building Performance Simulation* 11.1, pp. 84–98. DOI: 10.1080/19401493.2017.1283539.

VDI 6007-1 (2015-06). *Calculation of transient thermal response of rooms and buildings - Modelling of rooms*. Guideline. The Association of German Engineers.

Wetter, Michael, David Blum, and Jianjun Hu (2019). *Modelica IBPSA Library v1*. DOI: 10.11578/dc.20190520.1.

Wetter, Michael, Christoph van Treeck, et al. (2019). "IBPSA Project 1: BIM/GIS and Modelica framework for building and community energy system design and operation – ongoing developments, lessons learned and challenges". In: *IOP Conference Series: Earth and Environmental Science* 323.1, p. 012114. DOI: 10.1088/1755-1315/323/1/012114.

Electrode boiler model for ancillary service simulation

Rene Just Nielsen[1] Thomas Egsgaard Pedersen[1]

[1]Added Values P/S, Denmark, {rjn,tep}@AddedValues.eu

Abstract

A generic component-based model of an industrial electrode boiler with internal control systems is presented. A mechanistic modelling approach was taken to include as much process and control information as possible and to generate detailed simulation results. The model is intended for *qualitative* studies of electrode boiler dynamics in the context of district heating generation and power grid ancillary services in collaboration with other electric power consuming units.

An example boiler control scheme is designed and included in the simulation model as this is paramount to the dynamic response of the system. Simulations of standstill, load changes, and startup from hot and cold state show that the strictest ancillary service requirements can be fulfilled when the boiler is kept at operating temperature.

Keywords: electrode boiler, district heating, ancillary services

1 Introduction

Increasing penetration of renewable energy sources, such as wind and solar power, increases the demand for ancillary service provisions to stabilize the power grid frequency around its nominal value, for example 50 Hz (Energinet 2021). At the same time, district heating (DH) production from electrical power is gaining ground as it is becoming increasingly economically competitive with combustion-based DH generation. Furthermore, by including thermal energy storage, DH production and consumption can be decoupled, thus making heat production during times with low electric-power prices possible.

On the DH production side, the combination of heat pumps and electric boilers is particularly interesting in terms of ancillary service provision. Typically, the merit order will dispatch heat pumps before electric boilers, as the heat pumps produce "COP times" more thermal power than the boilers. Thus, this leaves room for the electric boilers to *increase* power consumption and for the heat pumps to *reduce* power consumption. Translated to ancillary service terms, the units can provide *downward regulation* and *upward regulation*, respectively.

In the western part of the Danish power grid — part of the European ENTSO-E area — the ancillary services are divided into three groups with different requirements to response times (ENTSO-E 2018).

- **Frequency Containment Reserve (FCR)** which has the fastest requirements of maximum 30 seconds from activation to full engagement. This is typically based on the producer's (or consumer's) own frequency measurement and proportional controller.
- **Automatic Frequency Restoration Reserve (aFRR)** which currently has a response requirement of 15 minutes but will be reduced to 5 minutes in the near future (Energinet 2022). The transmission system operator (TSO) has one balancing (PID) controller which closes the loop around the power imbalance and all its engaged producers/consumers in the grid. Hence the term "automatic".
- **Manual Frequency Restoration Reserve (mFRR)** which also has a response requirement of 15 minutes but is engaged manually — typically via the balance responsible party (BRP).

Electric boilers are simple and compact, and offer fast load changing abilities (Danish Energy Agency 2023). Some manufacturers promise power consumption responses that comply with all three mentioned ancillary service categories, e.g., (PARAT 2020). The purpose of the work behind this paper is to qualify this promise using a mechanistic modelling approach.

No publications on physical modelling of electric boiler models in this context are known to the authors of this paper. (Nielsen et al. 2016) make an economic assessment of electric boilers in a DH production context but do not describe the dynamic behaviour. In (Zhi et al. 2017) a (data driven) neural network model of a 20 MW electrode boiler is presented but without internal controllers and physical phenomena.

This paper presents a dynamic component-based model of an electrode boiler with its internal controls. Focus is on dynamic closed-loop behaviour during standstill heating, start-up, and load changes. Although the presented model is technically speaking a *water heater*, it will be referred to as a *boiler* as this seems to be the common term in the context of district heating systems.

The outcome of the work is a generic, component-based model of an electric boiler for general studies of internal

DOI
10.3384/ecp204589

phenomena — not to exactly replicate the operating characteristics of one specific boiler model since this would require complete insight into the controller code from the manufacturer. The controlled model is used to verify the plausibility of FCR compatible responses.

2 Electrode boilers — working principle

Electrode boilers are a sub-type of electric boilers in which the medium to be heated (water) acts as an electric resistance. Various types of electrode boilers of different designs are available from various manufacturers. The type of electrode boiler presented in this work is inspired by Norwegian manufacturer PARAT Halvorsen AS and is illustrated in Figure 1. It consists of two concentric tanks — an inner/upper and an outer/lower — containing purified, conditioned water. Electrodes connected to a medium voltage AC supply are immersed in the upper tank, in which the water level is controlled with a valve allowing to drain the upper tank to the lower tank. A pump circulates water from the lower tank, via a DH heat exchanger (HX) to the upper tank. The water surrounding the electrodes acts as an electric heating resistance and the thermal power transferred from the electrodes to the water is proportional to the water level. Increasing the water level covers more of the electrode surface and lowers the resistance between the electrodes. Water on the DH-side of the HX removes the generated heat.

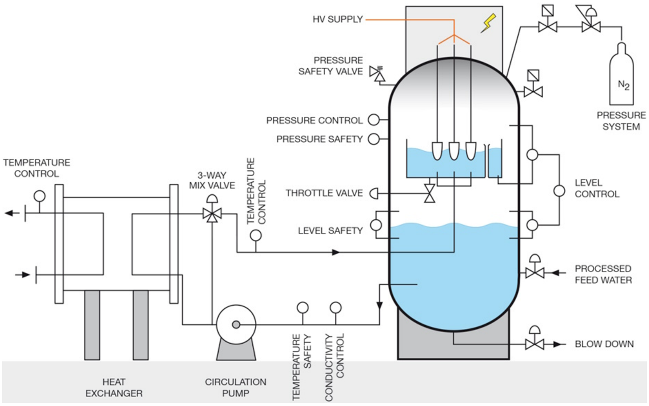


Figure 1. electrode boiler working principle (PARAT 2020).

3 Model

The boiler *system* model shown in Figure 2 is constructed from components from the Modelica Standard Library (MSL) (Modelica Association 2020) and from the inhouse developed District Heating Library presented in section 3.1. Components include pumps, valves, a heat exchanger, sensors, internal control, and the electrode boiler vessel itself.

On the left side of the HX the DH pump (top) circulates cold return water from port_a to port_b through the HX during DH production. A control valve (bottom left) and a recirculation valve (middle left) allows zero net DH flow even when the pump runs at its minimum speed, which is typically around 20 %. The other pump (bottom left) circulates DH supply water backwards through the HX and the top-left isolation valve during standstill heating. Both pumps have internal check valves to prevent reverse flow when the other pump is running.

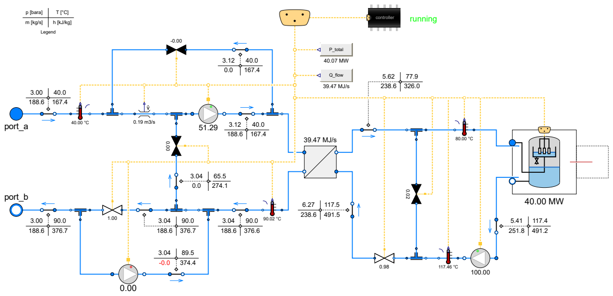


Figure 2. diagram view of the electrode boiler system model.

On the right side of the HX, the boiler circulation pump circulates water through the boiler. The HX admission and shunt valves can be utilized to maintain a fixed or load dependent boiler inlet temperature.

Sensors and actuators are connected to busses through an expandable connector to allow for a centralized and replaceable control system, separate from the process model itself. The external bus inputs to pumps and valves are conditional, and the setpoints can be set within these actuator components. This makes it possible to simulate the process without any feedback control which can be useful as an initial validation of the process. For example, setting valve positions and pump speeds to their nominal values brings the “steady-state” values on the system level close to the expected nominal values. The quotation marks suggest that the tank models create a marginally stable open-loop response which does not have a steady state until they overflow or become empty.

3.1 District Heating Library

The inhouse developed library District Heating Library (DHL) contains a collection of components for hydronic heating systems and is briefly presented here. Many of the components are based on Modelica Buildings Library (MBL) (Wetter et al. 2014) which offers free, robust and user friendly thermo-hydraulic components. The graphical appearance is an essential feature of the DHL, and we like the diagram view of the models to resemble process flow and heat balance diagrams in which the key simulation values are presented to the user directly in the diagram view. Examples of this can be seen in Figure 2 where

- “Value crosses” display pressure, temperature, mass flow rate and specific enthalpy of the corresponding fluid stream.

- Display units are bara, °C, and kJ/kg instead of Pa, K, and J/kg
- Boolean variables are indicated with red/green lamps, for instance if a pump is running
- Fluid stream connection lines have double width to emphasize the main flow paths

This is also extremely useful in debugging situations where the displayed values and Boolean states give an easy overview in the different model layers.
Additionally, by default, component instance names are hidden to avoid cluttering of the diagram view.

3.2 Electrode boiler model

The electrode boiler model itself is shown in Figure 3 and consists of two open tank models. The upper tank is connected to the lower tank via a throttle valve and can also overflow to the lower tank.

The lower tank is thermally connected to the ambient temperature (either fixed or via a conditional heat port) to simulate external heat loss. Using Newton's law of cooling, the thermal conductance, G_l, is calculated from a desired heat loss settling time, $t_{loss} \approx 5\tau_{loss}$, as

$$G_1 = \frac{m_w c_{p,w}}{\tau_{loss}} = 5 \cdot \frac{m_w c_{p,w}}{t_{loss}} \tag{1}$$

with the time constant, τ_{loss}, the specific heat capacity, $c_{p,w}$, and total mass of water in the boiler, m_w. The heat capacity of the metal part is not considered explicitly but is "absorbed" in the calculated conductance. Additionally, the two tank volumes are thermally connected by a conductive heat transfer element to ensure steady-state equilibrium temperature. The thermal conductance is arbitrarily set to 5 times the conductance of the ambient heat loss to prioritize the internal temperature equilibrium over the external heat loss.

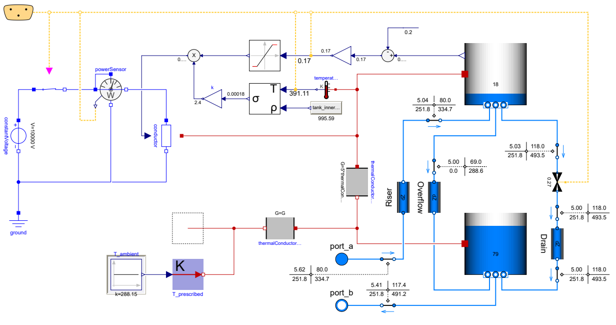


Figure 3. electrode boiler model.

For the sake of simplicity, the electric circuit is modelled with a constant DC source, a switch (as the main circuit breaker, MCB), and a variable conductance. Heat transfer from the electrodes to the water is simulated with a variable electrical conductor from MSL. According to Joule's law the relationship between supply voltage, U, electric conductance, G_e, (the inverse of electric resistance, R), and electric power dissipation in the water is

$$P = \frac{U^2}{R} = G_e U^2 \tag{2}$$

We assume that the conductance is proportional to the normalized water level (0–1) and the conductivity, σ, of the boiler water, i.e.,

$$G_e \propto L \cdot \sigma \tag{3}$$

The normalized level, L, is calculated as

$$L = \frac{L_w - z_e}{L_e} \tag{4}$$

With the measured water level L_w, the electrode length L_e, and the elevation of the electrodes above the upper tank bottom z_e. L is limited to values between zero and one, indicating zero to full electrode coverage.

The composition of the *conditioned* boiler water — and its thermophysical properties — is only known by the manufacturer but for the sake of simplicity we assume that σ is proportional to the conductivity of pure water, $\sigma_w(\rho, T)$, which can be calculated using equations published by the International Association for the Properties of Water and Steam (IAPWS 1990) and (IAPWS 2019).

$$\sigma \propto \sigma_w(\rho, T) \tag{5}$$

The density, ρ, can be calculated from the water pressure and temperature and combining equations (3) and (5) we get

$$G_e = k \cdot L \cdot \sigma_w(\rho, T) \tag{6}$$

The fitting coefficient, k, can now be derived from equations (2) and (6), and the nominal values of power, voltage, water level ($L = 1$), temperature, and pressure

$$k = \left(\frac{U^2}{P \cdot \sigma_w(\rho(p,T), T)} \right)_{nominal} \tag{7}$$

As medium model, liquid incompressible water from MBL is used. Actuators and measurements are connected to a signal bus, implemented with an expandable connector.

3.3 Pump

The icon and diagram view of the DHL pump model is shown in Figure 4. It is based on the MBL model `SpeedControlled_y` and is augmented with conditional external inputs (bus, on/off or analogue speed setpoint), temperature and pressure sensors, and optional check valve and variable speed drive (VSD) to simulate ramp limits, minimum speed and start/stop/coastdown behaviour. All relevant pump information and setpoints are exposed on the `PumpBus` expandable connector which can also be conditionally enabled.

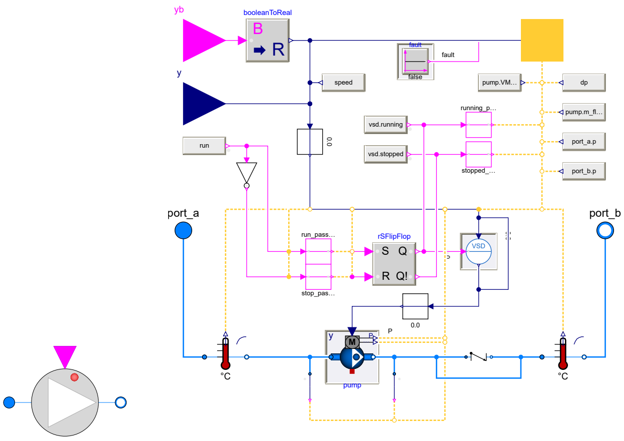


Figure 4. Icon and diagram view of pump model.

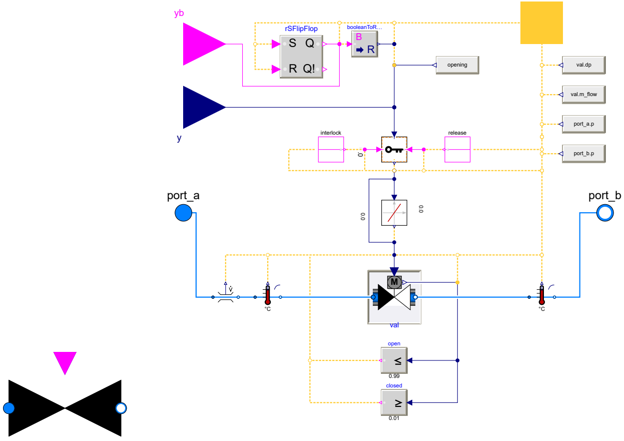


Figure 5. Icon and diagram view of valve model.

3.4 Valve

The valve model in Figure 5 is based on a replaceable MBL linear valve model and the position can be set internally or through different external connectors (Boolean/Real/bus).

An optional slew rate limiter can be used to simulate the valve stroke time and optional release/interlock expressions can force opening/closing of the valve. All relevant information is exposed on the `ValveBus` expandable connector.

3.5 Heat exchanger

The heat exchanger model contains a replaceable instance of the MBL `PlateHeatExchangerEffectivenessNTU`. The port placement and icon of the DHL heat exchanger can be changed through a parameter. Figure 6 shows different appearances of the heat exchanger model.

3.6 Tee junction

The tee junction is an extension of the MBL `Junction` model with unchanged functionality. However, the graphical annotation `primitivesVisible=false` was used to omit the icon annotation of the `Junction` model and redraw it in the DHL version as shown in Figure 7.

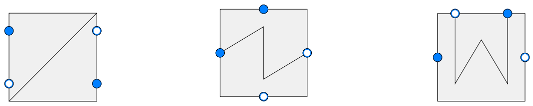


Figure 6. Different appearances of the heat exchanger model.

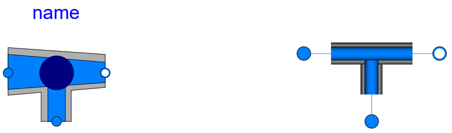


Figure 7. Change of icon using the `primitivesVisible` annotation.

4 Control

To study the transient behaviour of the boiler during activation of ancillary services, the internal controller models are just as important as the models of the physical process. Since the actual controller code is an intellectual property of each boiler manufacturer, and thus unavailable to the authors of this paper, a comparative plausible control structure is set up based on experience and on the required controlled variables for the exemplary boiler.

The following controlled variables are defined

- Electric power consumption
- DH heat flow rate
- DH supply temperature
- Water level of the inner tank
- Maximum internal temperature
- Boiler inlet temperature
- Standstill temperature

Controllers for water conditioning (conductivity, make-up, blowdown), pressurization, and inertization are omitted in the context of this work.

In addition to the continuously controlled variables, the boiler system can transit between three operating states, described by the sequential function chart in Figure 8.

- **Running** — the boiler controls water level, electrical power consumption (or thermal output), and DH supply temperature.
- **Stopped** — pumps are stopped, controllers deactivated, upper tank drained, and MCB open.
- **Standstill heating** — 90 °C DH supply water is consumed to maintain an internal temperature of 60–80

°C, water level is controlled slightly *below* the electrode tips to prevent arcing from water surface ripples, and power consumption is approximately zero. This enables short start-up time.

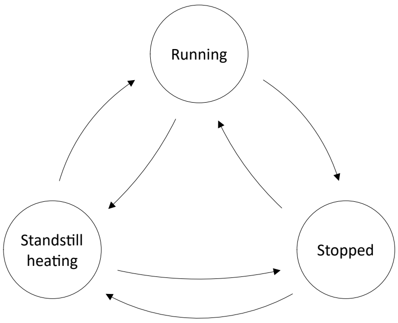


Figure 8. Electrode boiler state chart

The continuous and sequential controllers are organized in one controller model as shown in Figure 2 and Figure 9.

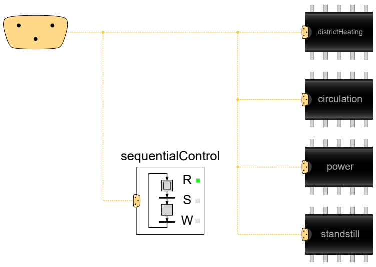


Figure 9. Continuous and sequential controllers.

4.1 Sequential control

The implementation of the three operating states and six corresponding transitions, using `Modelica StateGraph` (Otter et al. 2005), is shown in Figure 10. The states are connected to the run/stop and auto/manual signals of the relevant pumps, valves, and continuous controllers in the system model. To mention a few transitions:

- Transitions to **running** state are enabled if
  - the thermal/electrical setpoint is above the minimum continuous load, e.g., 0.5 MW and
  - the system **run** command is true and
  - the boiler water temperature is below the maximum limit
- Transitions to **standstill heating** state are enabled if
  - the run command is true and
  - the temperatures are below maximum limits but
  - the power setpoint is below minimum continuous load

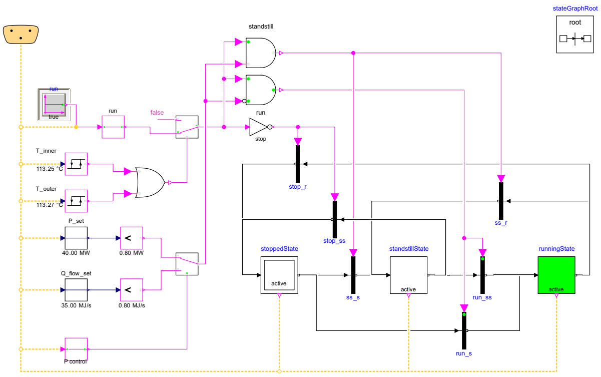


Figure 10. sequential controller.

4.2 Power and level control

The power and level controllers are shown in Figure 11. The level controller controls the water level in the upper tank with the throttle valve. When the controller is deactivated, the valve is opened to drain the tank to put the boiler into an operationally *safe* state. A temperature limiter forces the throttle valve to open if the maximum internal operating temperature is exceeded. This would be the case if, for instance, cooling of the dissipated power with DH return water fails. The controller manipulates the lower output limit of the level controller since this ensures direct "contact" with the level controller output and prevents integrator windup.

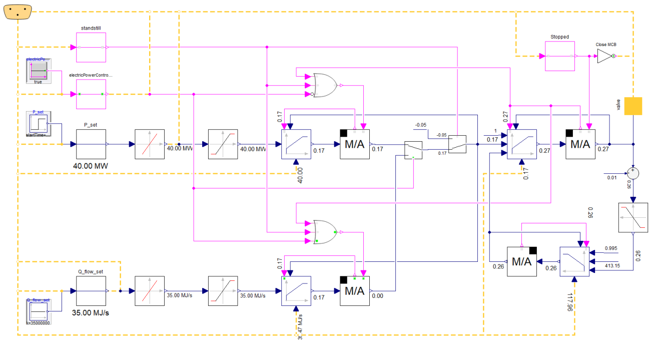


Figure 11. Power and level controllers.

The level setpoint is provided by the thermal or electrical power controllers in a cascade configuration. If the thermal power controller is activated, the electric power controller is in manual mode with its output tracking the output of the thermal power controller. The same applies to the opposite case and enables bumpless switching between the two controllers. The power and heat flow rate setpoints are max/min limited to the minimum-to-maximum continuous load and rate limited such that a change from zero to maximum load takes 15 seconds (fast enough to comply with the FCR requirements). The logic circuitry around the controllers ensures that the power controllers are deactivated if the level controller is deactivated (or set to manual mode) or if the boiler is not in

running state.
Each PID controller is succeeded by a *control station* block — which can be used to simulate an operator intervention — supplying a tracking reference value and Boolean track signal to the controllers.

4.3 DH supply temperature control

Figure 12 shows the DH supply temperature controller.

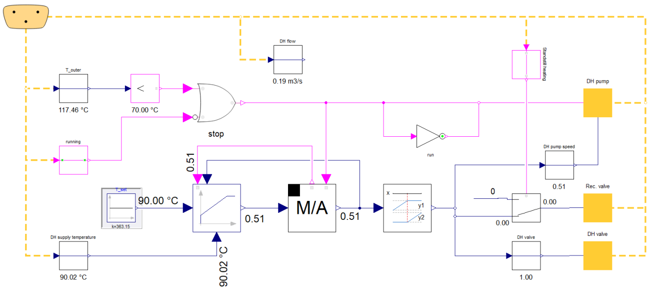


Figure 12. DH supply temperature controller.

When the boiler is in **running** state and the internal temperature is above 70 °C the controller is activated, and the DH pump is started. The DH pump, control valve and recirculation valve are controlled in a split-range configuration that produces a smooth DH flow from zero to maximum. Figure 13 shows the pump speed and valve positions as a function of the temperature controller output (top) and the resulting DH flow (bottom). From zero to 20 % controller output the pump runs at minimum speed (20 %) and the valves control the flow. From 20 to 100 % the pump controls the flow.

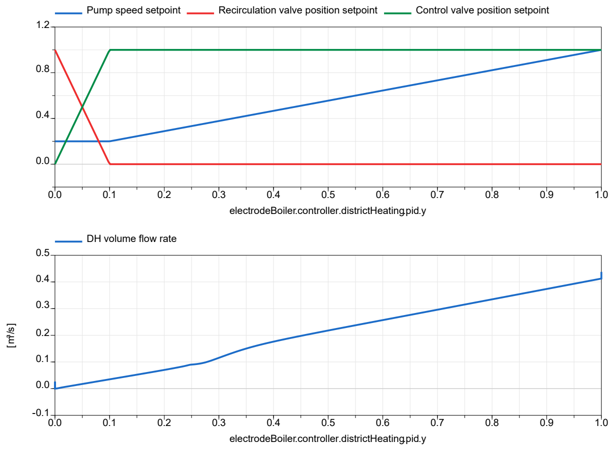


Figure 13. Split-range control of DH pump and valves (top) to produce an almost linear flow (bottom).

4.4 Boiler circulation control

The boiler circulation controller, shown in Figure 14, controls the boiler circulation pump speed and the boiler inlet temperature. The pump speed setpoint is proportional to the electric power or heat flow rate setpoint — depending on the mode of operation. This load dependence was chosen to reduce the recirculation rate and throttling loss at low boiler load. The boiler inlet temperature is controlled with the HX admission and shunt valves. When one is opened, the other closes.

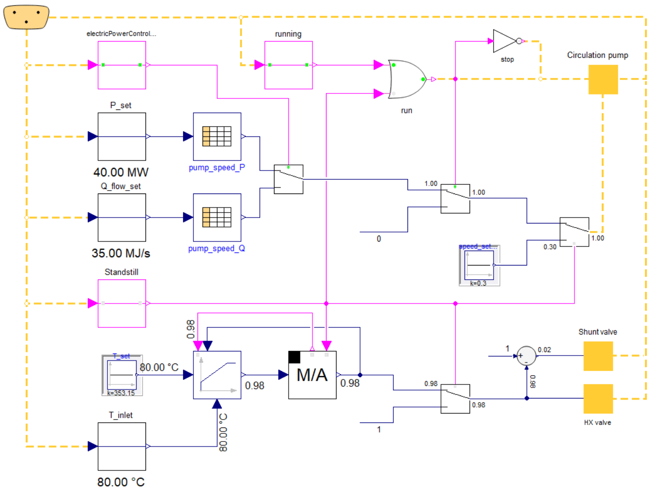


Figure 14. Boiler circulation control.

4.5 Standstill temperature control

The standstill temperature controller (Figure 15) controls the internal boiler temperature during standstill to maintain a high electric conductivity of the boiler water and, thus, a enables fast start-up response. When the standstill heating state is active, the isolation valve is opened, and the standstill pump controls the temperature in an on/off fashion.

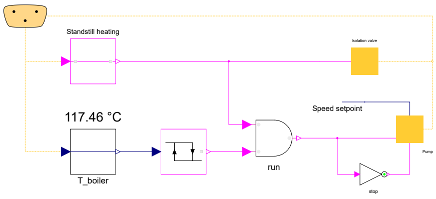


Figure 15. Standstill temperature controller.

5 Simulation results

Figure 16 shows the `BaseCase` simulation model in which the system is simulated to steady state with nominal conditions. The block to the right contains the electrode boiler system (shown in Figure 2) with its internal controllers. Setpoints are set inside the controller blocks but can optionally be taken from the system bus — for example when the boiler is controlled by an outer master controller. All simulation variations (step-responses, part load, failure conditions etc.) extend from the base case with modified boundary conditions (setpoints etc.), cf. Listing 1.

Listing 1. Exemplary part-load simulation.

```
model PartLoad "Part-load simulation"
  extends BaseCase(electrodeBoiler(
    controller(power(P_set(k=5000000)))));
end PartLoad;
```

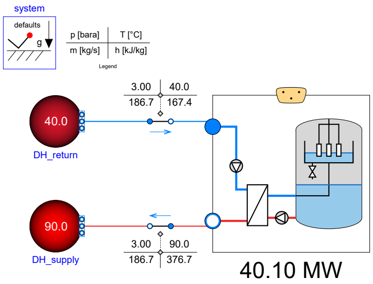


Figure 16. Simulation model.

The model is parameterized as a 40 MW boiler. Based on information from (PARAT 2020) the boiler height and diameter are set to 6 and 3 metres, respectively. Nominal mass flow rates are calculated from nominal temperatures and heat flow rates. Valve stroke times and pump data are estimated from experience. The key model parameters are summarized in table 1.

Table 1. Key model parameters.

Quantity	*Value*
Heat flow rate	40 MJ/s
DH return temperature	40 °C
DH supply temperature	90 °C
Boiler inlet temperature	80 °C
Boiler outlet temperature	120 °C
DH mass flow rate	191 kg/s
Boiler mass flow rate	119 kg/s
Stroke time (external valves)	30 s
Stroke time (throttle valve)	5 s
Total pressure drop, boiler circuit	1.0 bar
Total pressure drop, DH circuit	0.5 bar

The following simulation results show the behaviour of the model in different situations. The main interest is to study internal behaviour of the boiler model and to evaluate its response during activation of ancillary services. The simulation models contains 7,366 equations, the translated model has 89 differentiated variables, and the sizes of nonlinear systems are $\{4, 3, 2, 2\}$. The integration time of the presented models using DASSL is approximately 0.5 seconds in Dymola 2023x Refresh 1 running on a normal laptop PC.

5.1 Cold start

In the first simulation the boiler system is initialized in **stopped** state without standstill heating. All initial temperatures are 20 °C and the upper tank is empty. At five minutes, the boiler receives a start signal and an electric power setpoint of 40 MW.

Figure 17 (top) shows the electric power consumption and its setpoint (blue) together with the thermal power output delivered to the DH system (red). The power consumption takes 3–4 minutes to reach its setpoint. The five-minute offset between the electrical and thermal power transient is because the DH temperature controller is not activated until the internal boiler temperature reaches 70 °C.

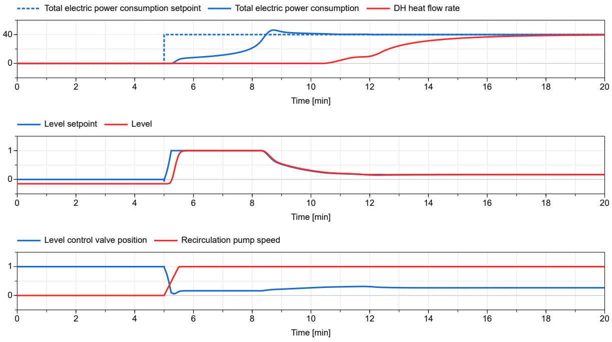


Figure 17. Power and level during cold start.

The middle figure shows the level setpoint (blue) and the corresponding water level (red). The blue slope is caused by rate limitation of the power setpoint and the integral time of the power controller providing the level setpoint. The ~20 second delay of the level is a combination of

- The water level starting at $-z_e$ (electrode elevation above the tank bottom)
- The throttle valve closing (bottom figure, blue)
- The circulation pump starting (bottom figure, red)

The fact that the throttle valve position settles on about 25 % to maintain the steady-state level suggests that the valve sizing is a bit off. Instead, 50 % is preferable as this would give a symmetric process gain during level increase (valve closing) and decrease (valve opening) for a constant pump speed.

Figure 18 (top) shows the DH supply temperature (red) and its setpoint. The bottom figure shows the temperature controller output (blue), the DH valves (green and magenta), and the DH pump speed (red). When the DH temperature controller is activated at approximately 10 minutes, the pump ramps to its minimum speed (20 %) After another couple of minutes, the DH temperature approaches its setpoint and the DH valves start to actively control the supply temperature. Subsequently, as the heat flow rate increases, the pump speed increases to keep the temperature on its setpoint.

Figure 19 (bottom) shows the boiler inlet temperature (red) and its setpoint (blue) which are controlled by

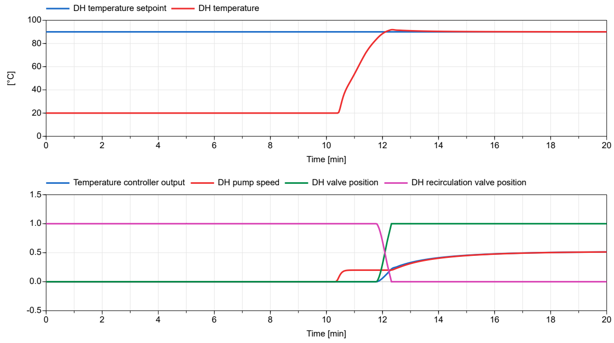


Figure 18. DH supply temperature (top) and DH pump, valves and temperature controller output (bottom).

the HX admission and shunt valves (red and green, middle) and the boiler pump speed (blue, middle). The temperatures of the upper (blue) and lower tank (red) are shown in the top figure.

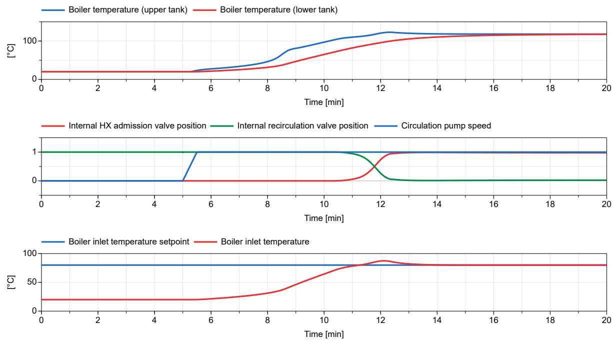


Figure 19. Internal temperatures and circulation.

As expected, the electric power response, following a *cold start*, is not fast enough to comply with the 30 second FCR requirements. However, it is fully capable of fulfilling aFRR and mFRR requirements.

5.2 Standstill

The "slow" response during cold start is mainly attributed to the relatively low electric conductivity at low temperatures. Figure 20 shows the conductivity of pure water (at 5 bara) as a function of temperature.

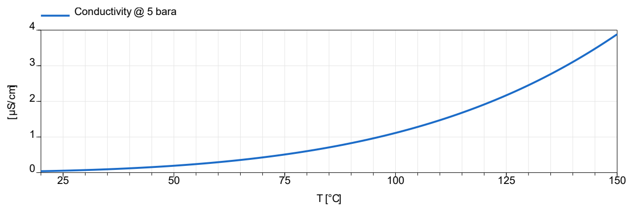


Figure 20. Conductivity of pure water at 5 bara as a function of temperature.

Standstill heating is activated by setting the boiler **run** command to true and the electric power setpoint to zero. Figure 21 shows the on/off control of the boiler temperature during standstill (top, red) to a setpoint of 80 °C (top, blue). The standstill pump starts when the temperature drops below 77 °C and stops when the temperature reaches 83 °C. With t_{loss} set to 7 days the 6 °C cooling (heat loss) phase takes 4–5 hours. The bottom figure shows the standstill valve (red) and pump (blue), and the boiler circulation pump (green). As the boiler temperature measurement is located on the pipe leaving the boiler, it can only provide a representative reading of the (uniform) boiler temperature when water flows through it. For that reason, the boiler circulation pump runs continuously at low speed.

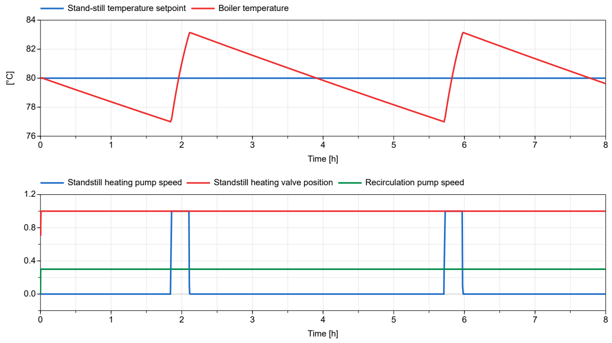


Figure 21. Standstill temperature control.

5.3 Warm start

Figure 22 shows the electric power and level responses during a warm start (red) compared to a cold start (blue). Now the power response is only approximately 15 seconds compared to 3–4 minutes.

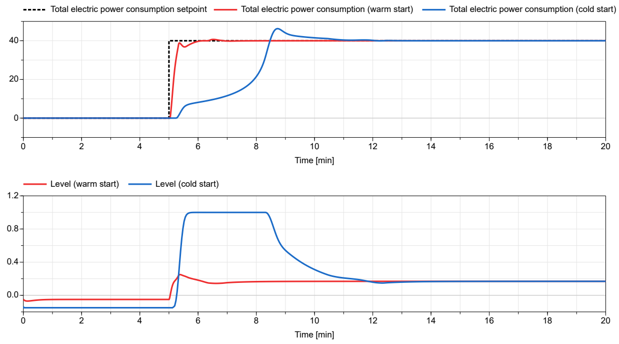


Figure 22. Power and level during warm start (red) compared to cold start (blue).

The improvement can be explained by:

1. The water conductivity is improved by the standstill heating preceding the warm start.
2. During standstill heating the water level is maintained slightly below the electrodes, whereas in

stopped stated (cold), the upper tank is emptied. This causes a startup penalty while the boiler circulation pump fills up the inner tank.

5.4 Load reduction

The FCR requirements apply to both load increases and reductions. The following figures show the simulated response during a power setpoint reduction to minimum stable load (0.5 MW).

Figure 23 shows the power and heat flow rate (top), the level (middle) and throttle valve position, and circulation pump speed (bottom). The power consumption reaches and undershoots its setpoint within 15 seconds. The level undershoot causes the electrodes to be unconvered, resulting in zero power consumption. As the boiler circulation pump speed is low at minimum power, the water level only recovers slowly.

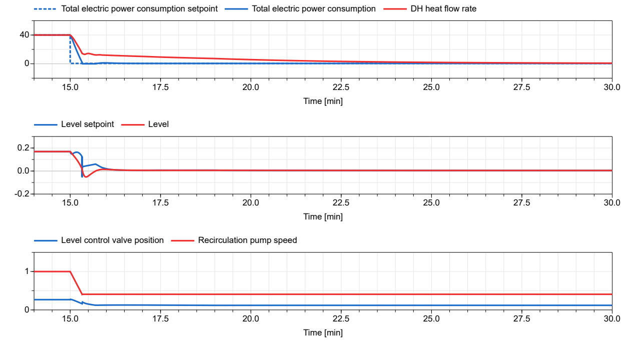


Figure 23. Power and level during a load reduction.

During the load transition, the DH supply temperature drops to 84 °C as shown in Figure 24 (top). The DH pump and valves (bottom figure) are clearly not fast enough to keep the temperature on it setpoint during a 15 s power transition and the 30 s valve stroke time should be re-considered in an improved design.

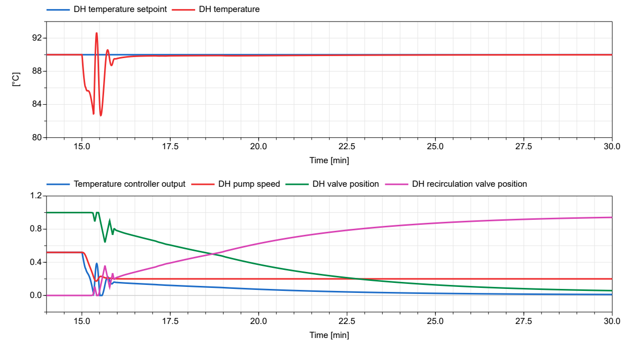


Figure 24. DH supply temperature drop during load reduction.

Figure 25 (top) shows the internal boiler temperatures decreasing from 118 °C at 40 MW to approximately 90 °C at 0.5 MW. At 19 minutes the controlled boiler inlet temperature (bottom) suddenly starts deviating fromt its setpoint. This can be explained by the 20 % minimum speed of the DH pump, resulting in a high recirculation rate to keep the net DH flow low.

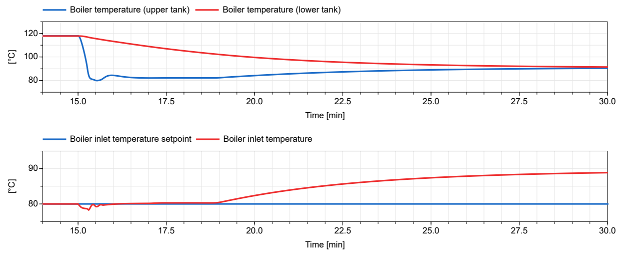


Figure 25. Internal boiler states during load reduction.

Figure 26 shows a part of the diagram view of the simulation. The DH recirculation results in an HX inlet temperature of 87.2 °C. Since the boiler inlet temperature is the mixture temperature of the boiler-side HX inlet (91.5 °C) and outlet (88.9 °C) temperatures it is not possible to reach the 80 °C setpoint.

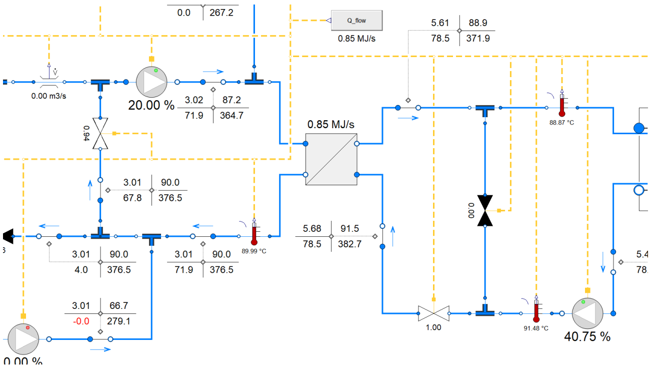


Figure 26. DH recirculation causing a high boiler inlet temperature.

6 Discussion

The mechanistic, component-based modelling approach generates a lot of interesting results, not attainable with data-driven approaches. It reveals several details of the closed-loop behaviour which could not easily be found by prior scrutiny.

Although the simulations show good results, improvements to the model and the control strategy are always possible.

During normal operation, level control is strictly speaking unnecessary. The electric/thermal power controllers could just as well actuate the throttle valve directly, and this would enable a faster power control, as the controller cascade would be omitted. However, during standstill the level needs to be controlled slightly below the electrode tips. This could be arranged by placing the level controller "in parallel" with the power and heat flow rate controllers, switching to level control in the relevant situations.

The boiler circulation pump speed was made load-dependent to reduce the throttling loss at low loads. However, this results in load-dependent process gain from throttle valve to tank level and slow process response at low load. This could be changed, for instance, to a mid-range control scheme in which the circulation pump keeps the valve position around 50 %. This would give a more symmetric level increase/decrease response.

Also, more emphasis could be given to refining the relationship between the water properties and the conductance. This would require more insight into the chemical part of the boiler and probably the involvement of the boiler manufacturer.

The electric part of the model was made with a DC circuit for simplicity. To connect the model to a power grid model this must be extended — for example by using AC components from libraries such as (Franke and Wiesmann 2014) or (Baudette et al. 2018).

Finally, matching actual operating data from a real electrode boiler, geometry, valve stroke times, pump curves, and the detailed control scheme would have to be replicated as this is paramount in that context.

7 Conclusion

A component-based model of an electrode boiler and its related internal controllers was presented. It is easy to parameterize, and simulations between zero and full load show good agreement with expected responses. With the presented parameters an exemplary 40 MW boiler model complies with the FCR ancillary service requirements both during load reduction and load increase from warm conditions.

The simulated model provides great insight into the process and automation details of the electrode boiler system. It consists of several sub-components making it easy to study its behaviour in detail. The short simulation time makes the model suitable for integration in a larger system simulation model.

References

Baudette, Maxime et al. (2018). "OpenIPSL: Open-Instance Power System Library – Update 1.5 to "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations"". In: *SoftwareX*. DOI: 10.1016/j.softx.2018.01.002.

Danish Energy Agency (2023). *Technology Data – Generation of Electricity and District Heating*. Tech. rep. URL: https://ens.dk/sites/ens.dk/files/Analyser/technology_data_catalogue_for_el_and_dh.pdf.

Energinet (2021). *SCENARIERAPPORT 2022 – 2032 Forventninger til fremtidens Systemydelser*. Tech. rep. URL: https://energinet.dk/media/k4kb5i3x/scenarierapport-2022-2032.pdf.

Energinet (2022). *Prækvalifikation af anlæg og aggregerede porteføljer*. Tech. rep.

ENTSO-E (2018). *Electricity Balancing in Europe*. Tech. rep. URL: https://eepublicdownloads.entsoe.eu/clean-documents/Network%20codes%20documents/NC%20EB/entso-e_balancing_in%20_europe_report_Nov2018_web.pdf.

Franke, Rüdiger and Hansjürg Wiesmann (2014). "Flexible modeling of electrical power systems – The Modelica Power Systems library". In: *Proceedings of the 10th International Modelica Conference*. DOI: 10.3384/ecp14096515.

IAPWS (1990). *Electrolytic Conductivity (Specific Conductance) of Liquid and Dense Supercritical Water from 0°C to 800°C and Pressures up to 1000 MPa*. Tech. rep. International Association for the Properties of Water and Steam. URL: http://www.iapws.org/relguide/conduct.pdf.

IAPWS (2019). *Revised Release on the Ionization Constant of H_2O*. Tech. rep. International Association for the Properties of Water and Steam. URL: http://www.iapws.org/relguide/Ionization.pdf.

Modelica Association (2020). *Modelica Standard Library*. Tech. rep. URL: https://github.com/modelica/ModelicaStandardLibrary.

Nielsen, Maria G. et al. (2016). "Economic Valuation of heat pumps and electric boilers in the Danish energy system". In: *Applied Energy*. DOI: 10.1016/j.apenergy.2015.08.115.

Otter, Martin et al. (2005). "StateGraph — A Modelica Library for Hierarchical State Machines". In: *Proceedings of the 4th International Modelica Conference*, pp. 569–578. DOI: 10.1080/19401493.2013.765506.

PARAT (2020). *High Voltage Electrode boiler for Steam and Hot water*. Tech. rep. URL: https://www.parat.no/en/products/industry/parat-ieh-high-voltage-electrode-boiler.

Wetter, Michael et al. (2014). "Modelica Buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Zhi, Xia et al. (2017). "Research on mathematical model of electrode boiler based on neural network". In: *2017 13th IEEE International Conference on Electronic Measurement Instruments (ICEMI)*. DOI: 10.1109/ICEMI.2017.8265792.

Heat Exchanger Surrogates for a Vapor Compression System

Nasrulloh Ratu Bagus Satrio Loka[1] Nicolás Ablanque Mejía[2] Santiago Torras Ortiz[2] Sriram Karthik Gurumurthy[3] Antonello Monti[3] Joaquim Rigola[2] Carles Oliet[2] Ivo Couckuyt[1] Tom Dhaene[1]

[1]IDLab, Ghent University - IMEC, Belgium, {nasrulloh.loka, ivo.couckuyt, tom.dhaene}@ugent.be
[2]Universitat Politècnica de Catalunya - Barcelona Tech (UPC), Heat and Mass Transfer Technological Center (CTTC), Spain, {nicolas.ablanque, santiago.torras, joaquim.rigola, carles.oliet}@upc.edu
[3]ACS, EONERC, RWTH Aachen University, Germany, {sgurumurthy,amonti}@eonerc.rwth-aachen.de

Abstract

Given the computationally intensive nature of heat exchanger simulators, utilizing a data-driven surrogate model for efficiently computing the heat exchanger outputs is desirable. This study focuses on developing integrated surrogate models of heat exchangers for a vapor compression system in Modelica. The surrogate models are designed to serve as steady-state equivalents based on an efficient physics-based model calibrated using reference data from a more advanced simulation model. Subsequently, the calibrated model was employed to generate the training and testing data for developing Gaussian Process (GP) and Multi-Layer Perceptron (MLP) surrogates. The findings indicate that GPs exhibit high accuracy when applied to the heat exchanger's outputs with smooth behavior. GPs also demonstrate excellent data efficiency compared to MLPs. In cases where the GP struggles to model specific outputs effectively, MLPs are able to capture the more complex behavior. Moreover, hyperparameter optimization is employed to identify optimal MLP topologies. Finally, the fast and compact surrogate model was integrated into the Modelica/Dymola environment. This adaptation allowed the surrogate models to be directly combined with the physical model of the heat exchanger.

Keywords: Heat Exchanger, Surrogate Model, Gaussian Process, Multi-Layer Perceptron, Hyperparameter Optimization

1 Introduction

In thermal systems, amid the most common components, heat exchangers are arguably the most challenging units to simulate from the numerical point of view due to the complex thermal and fluid-dynamic phenomena involved (e.g. fluid phase changes). Different approaches exist to model heat exchangers, but in general, the level of accuracy should increase with the level of detail considered in the model.

This work explores the development of heat exchanger surrogate models to be used within the Modelica environment. In this sense, a flexible heat exchanger model (Ablanque et al. 2022), adapted to simulate an air-to-refrigerant condenser, has been used to train and evaluate surrogate models. The specific condenser studied is part of a vapor compression system which, in turn, is included in an aircraft Environmental Control System (ECS).

Various neural networks and deep learning techniques have been implemented to model different aspects of heat exchangers. (Abbassi and Bahar 2005) use a shallow neural network to model the thermodynamics of an evaporative condenser. (Romero-Méndez et al. 2016) model convective heat transfer rate that occurs during the evaporation of a refrigerant flow using a neural network.

In this work, surrogate models have been developed for each target variable of the heat exchanger. Gaussian Process (GP) regression models have been constructed for target variables exhibiting smooth behavior due to GP's data efficiency and strong interpolation abilities for smooth functions. A Multi-Layer Perceptron (MLP) Neural Network Model has been employed for more challenging target variables with highly non-linear behavior. To enhance the performance of the MLP, hyperparameter optimization on the network architecture has been conducted. The performance of the proposed surrogate models has been demonstrated, and the advantages of hyperparameter tuning have been highlighted in the context of surrogate modeling.

Finally, the fast and compact surrogate model was successfully integrated into the Modelica/Dymola environment. This adaptation enabled the direct integration of the surrogate models with the physical model of the heat exchanger.

2 Heat Exchanger Model

2.1 Description

The structure implemented for the heat exchanger model is aimed to provide high flexibility in terms of geometries, participating fluids (i.e., liquids, gases, and two-phase refrigerants), and phenomenologies such as evaporation and condensation. The model layout consists of two specific sub-components for calculating fluid flows which are thermally linked via an additional sub-component that stands for the solid parts. Figure 1 shows the scheme for an air-to-refrigerant condenser.

DOI
10.3384/ecp204599

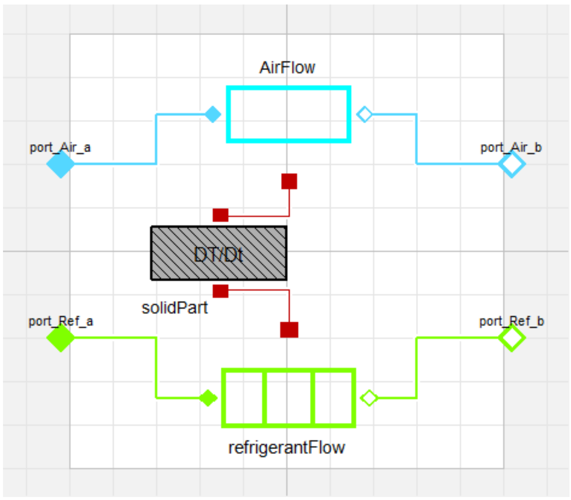


Figure 1. Heat exchanger main structure (air-to-refrigerant condenser).

The calculation of the fluid flow sub-component is based on a steady-state approach. The model discretization distinguishes three different zones, namely, superheated gas, two-phase, and sub-cooled liquid, as shown in Figure 2 for a condensation case. The aforementioned model zones can exist or not depending on the fluid inlet and outlet conditions so that the heat exchanger switches between different operating modes.

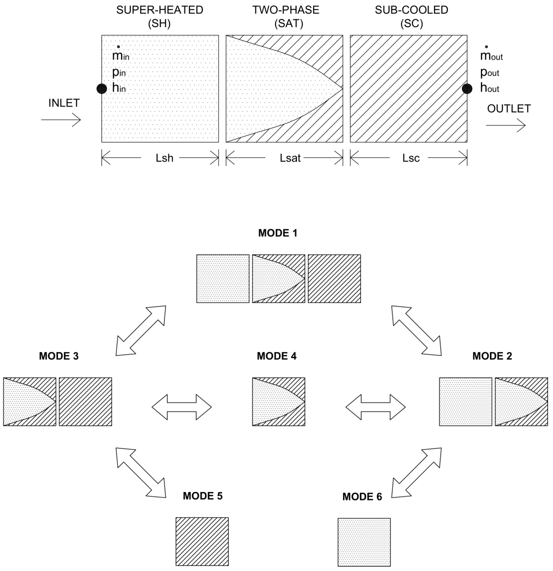


Figure 2. Condenser discretization and operating modes.

The pressure drop for the whole heat exchanger is calculated from a traditional approach (i.e., $\dot{m}$=KΔP$^{\alpha}$) where ΔP represents the pressure drop. The parameters K and α are previously determined from reference data.

The energy conservation equation is solved considering constant pressure and constant solid temperature. The calculation is conducted sequentially from zone to zone (if the heat calculated for the current zone is higher than the maximum heat allowed for this zone, the calculation will continue to the next zone. Otherwise, the calculation will terminate in the current zone). For single-phase zones, the heat flow rate between the fluid and the solid part $\dot{Q}_{single}$ is calculated based on an ε-NTU method (Incropera and DeWitt 1996) in order to optimize the calculation speed and to avoid unrealistic temperature values:

$$\dot{Q}_{single} = \varepsilon C(T_{solid} - T_{fluid,in}) \tag{1}$$

where C stands for the thermal capacity ratio and ε corresponds to the heat exchange effectiveness. For two-phase zones, the heat flow rate ($\dot{Q}_{two}$) is calculated from a standard approach based on a heat transfer coefficient (α), the temperature difference between the solid and the saturated fluid, and the heat transfer area (A):

$$\dot{Q}_{two} = \alpha(T_{solid} - T_{fluid,sat})A \tag{2}$$

The calculation of the solid sub-component is based on a transient approach. It is calculated considering a unique solid temperature (T), the solid mass (M), the solid mean specific heat capacity (c_p), and the heat rate transferred with the two fluids:

$$Mc_p\frac{dT}{dt} - \dot{Q}_{fluid,1} - \dot{Q}_{fluid,2} = 0 \tag{3}$$

The complete resolution is carried out by means of the default differential/algebraic system solver of Dymola. The heat exchanger's overall thermal response is dynamic as it combines the steady-state approach used for both flows with the dynamic approach considered for the solid part. Artificial relaxations can also be applied to the energy conservation equation of both flows to further overcome the negative impact of the absence of dynamic terms. The pressure drop equation is not only used to calculate the mass flow rate but also to approximate the phase saturation limits needed for the energy conservation equation.

2.2 Numerical Assessment and Tests

The current model has been developed to simulate different types of heat exchangers included in large thermal architectures consisting of multiple systems. Therefore, the need to meet demanding numerical requirements was crucial for its successful use in the aforementioned environments. The main requirements include robustness at initialization, robustness to any boundary conditions and/or signal types, robustness to conduct simulations at any particular simulation set-up parameter (e.g., interval length), the capacity of both fluids to handle null mass flow rate as well as reversed flows, and ability to handle changes of the expected heat flow direction. The model is a suitable platform to generate large quantities of training data and tests for the surrogate models due to its numerical characteristics.

2.2.1 Initialization and steady-state tests

A complete data set of cases has been generated to test the robustness of the model during initialization and the correct resolution for steady-state conditions. The data set has been built-up taking into account different values for all the boundary conditions (i.e., air and refrigerant inlet parameters) covering the whole physical range of possibilities and all its possible combinations. The data set also takes into consideration different fluid boundary condition types (see Figure 3) and different values for the interval length.

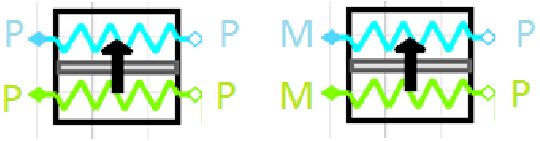


Figure 3. Boundary condition types: pressure - pressure (left) and Mass flow rate - pressure (right).

The data set consists of 1296 cases (216 cases for each combination of boundary condition type and interval length). The simulation stop time is set at 2000 seconds so that the steady-state condition can be reached. The results have shown that the model initialization is successful for all the cases without being affected by the combination of boundary values or the interval length value.

2.2.2 Transient tests

Similarly to the initialization studies, many tests have been conducted to assess the model's numerical robustness for other crucial transient conditions. Some illustrative examples are presented. Figure 4 shows the results for a test where both the null mass flow rate and the reversed flow capacities are tested. This particular case corresponds to a transient simulation where the refrigerant is operating at a particular mode and is forced to experience flow direction changes and null mass flow rate at different moments.

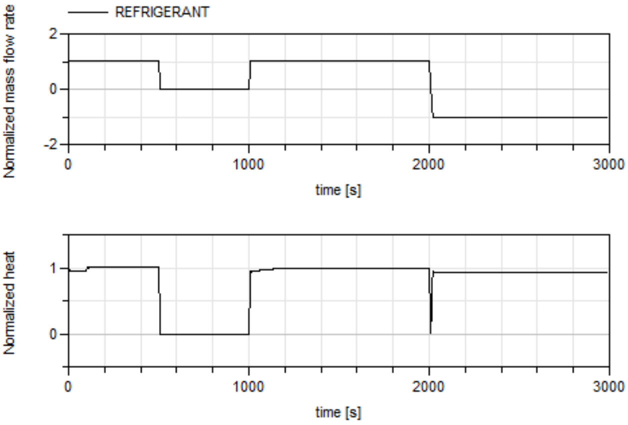


Figure 4. Null mass flow rate and reversed flow test example.

Figure 5 shows illustrative results for a test where many boundary conditions are provided as sine signals to force mode transitions in the heat exchanger (see Figure 1).

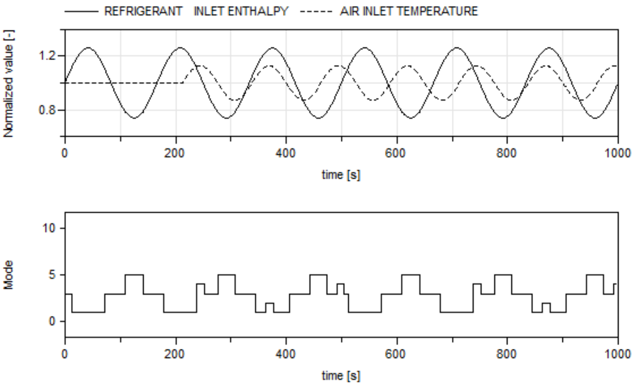


Figure 5. Sines signals test example.

2.2.3 Model validation

The accuracy of the condenser model has been comprehensively assessed. In this sense, a full set of reference data of 4766 cases has been provided from an advanced heat exchanger numerical model and used to compare the predictions. The parameter used for the comparisons is the so-called Prediction Error (PE) which characterizes the difference between the model-predicted value and the reference value of a particular variable. The local PE is a percentage value, and for the heat flow, it is evaluated as follows:

$$\mathrm{PE} = \frac{|\dot{Q}_{model} - \dot{Q}_{reference}|}{\dot{Q}_{reference}} \times 100 \tag{4}$$

To assess the accuracy regarding the whole data, an averaged PE is used, the so-called Mean Prediction Error (MPE), which is defined as follows:

$$\mathrm{MPE} = \frac{1}{N}\sum_{i=1}^{i=N} \mathrm{PE}_i \tag{5}$$

Table 1 shows the mean prediction error for the heat flow predicted by the condenser model. The results show good accuracy as the MPE for the whole data is 2.49 (this value decreases significantly as the less accurate results are not being considered).

3 Surrogate Modeling

Surrogate modeling aims to develop data-driven regression models that emulate complex systems or processes. This compact surrogate can then be used for real-time analysis, optimization, or prediction without the need for resource-intensive direct simulation of the system. In a regression problem, an input set is denoted as $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$, $\mathbf{x}_i \in \mathbb{R}^d$, where d is the dimensionality

Table 1. Condenser model accuracy assessment (heat flow prediction).

Data	%	MPE	MaxPE	std
4766	100	2.49	42.0	3.7
4671	98	2.12	15.6	2.6
4528	95	1.78	10.4	1.9
4289	90	1.43	5.8	1.2
4051	85	1.23	4.0	0.8
3575	75	0.99	2.2	0.6

of the input. A corresponding set of continuous observations is denoted as $Y = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_N)]$, with $f(\mathbf{x}_i) \in \mathbb{R}$. The goal is to construct a model that can predict the value of $y := f(x_*)$ for an unobserved point x_*. The prediction is denoted as $\hat{y}$. In this study, two techniques are employed to build predictive models of the heat exchanger system, namely Gaussian Processes and Multi-Layer Perceptrons.

3.1 Gaussian Process Regression

Gaussian Process (GP) (Rasmussen and Williams 2018) is a common data-efficient surrogate for regression problems. A GP is specified by a mean $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$. Given pair of input sets X and its evaluation on the system simulations, GP can be defined as: $f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$.

For the choice of the kernel function, the Matérn 5/2 kernel (Handcock and Stein 1993) is used as it does not put strong smoothness assumptions on the unknown function to be approximated (Genton 2002). The Matérn 5/2 kernel is defined as:

$$k\left(\mathbf{x}, \mathbf{x}'\right) = \gamma\left(1 + \sqrt{5}r + \frac{5}{3}r^2\right)\exp(-\sqrt{5}r), \qquad (6)$$

$$r = \sqrt{\sum_{m=1}^{d} \frac{\left(x_m - x_m'\right)^2}{l_m^2}} \qquad (7)$$

where γ is a scale parameter, and l is a lengthscale parameter for the kernel function.

Training the GP model involves estimating the hyperparameters $\hat{\theta}$. In this case, $\hat{\theta}$ contain the parameters of $k(\mathbf{x}, \mathbf{x}')$. Maximum Likelihood Estimation (MLE) is used to estimate the hyperparameters:

$$\hat{\theta} = \arg\max_{\theta} \log p(\mathbf{f} \mid X, \theta) \qquad (8)$$

$$= \arg\max_{\theta} -\frac{1}{2}\left(\log|2\pi K_{\mathbf{xx}}| + \mathbf{f}^T K_{\mathbf{xx}}^{-1}\mathbf{f}\right) \qquad (9)$$

The predictive mean $\mu\left(X_\star\right)$ and the predictive variance $\sigma^2\left(X_\star\right)$ of a new, untested data point X_* is calculated as:

$$\mu\left(X_\star\right) = K_{\star\mathrm{x}} K_{\mathrm{xx}}^{-1} Y \qquad (10)$$

$$\sigma^2\left(X_\star\right) = K_{\star\star} - K_{\star\mathrm{x}} K_{\mathrm{xx}}^{-1} K_{\star\mathrm{x}}^T \qquad (11)$$

where $K_{\mathrm{xx}} = k(\mathbf{x}_i, \mathbf{x}_j)$, $K_{\star\mathrm{x}} = k(\mathbf{x}_{\star i}, \mathbf{x}_j)$, and $K_{\star\star} = k(\mathbf{x}_{\star i}, \mathbf{x}_{\star j})$. The predictive mean of the GP is used as the surrogate model prediction $\hat{y}$. Moreover, the predictive variance could prove beneficial for quantifying uncertainty, which increases trust in the outcome and enables decision-making, optimization, or anomaly detection.

For Gaussian process regression, the GPFlow library (G. Matthews et al. 2017) is used, and the MLE is optimized using the Limited memory Broyden–Fletcher–Goldfarb–Shanno Bounded (LBFGS-B) optimizer (C. Zhu et al. 1997).

3.2 Multi Layer Perceptron

The *Multi Layer Perceptron* (MLP) (Hinton 1989) is a popular class of deep learning neural network architectures used for regression tasks. While it is not as data efficient as GP, it can capture more complex, non-linear relationships between input and output variables. It also scales well to the size of the data set compared to GP w.r.t. computational complexity. MLP is a versatile choice for surrogate modeling when a larger data set is available.

An MLP comprises an Input Layer, L Hidden Layers, and an Output Layer. The Input Layer matches the dimensionality of the input data, while the Output Layer matches the dimensionality of the target function. Generally, an MLP (Prince 2023) can be described as:

$$\begin{aligned}
\mathbf{h}_1 &= \mathbf{a}\left[b_0 + \mathbf{W}_0\mathbf{x}\right] \\
\mathbf{h}_2 &= \mathbf{a}\left[b_1 + \mathbf{W}_1\mathbf{h}_1\right] \\
\mathbf{h}_3 &= \mathbf{a}\left[b_2 + \mathbf{W}_2\mathbf{h}_2\right] \\
&\vdots \\
\mathbf{h}_L &= \mathbf{a}\left[b_{L-1} + \mathbf{W}_{L-1}\mathbf{h}_{L-1}\right] \\
\hat{\mathbf{y}} &= b_L + \mathbf{W}_L\mathbf{h}_L.
\end{aligned} \qquad (12)$$

where b_l and $\mathbf{W}_l$ are the bias term and weight parameters of the network at l^{th} layer, and $\mathbf{a}$, is the activation function. Rectified Linear Unit (ReLU) (Fukushima 1969) is used for $\mathbf{a}$ and defined as $\mathbf{a}(x) = \max(0, x)$

To find the optimal parameters (b_l and $\mathbf{W}_\mathbf{l}$), the ADAM (Kingma and Ba 2015) optimizer is used. Specifically for this study, The base architecture of the MLP was set to 150, 100, and 50 neurons for each of the three hidden layers respectively, as illustrated in Figure 6. The Scikit-learn machine learning library (Pedregosa et al. 2011) is used to train the MLP surrogates.

4 Heat Exchanger Surrogate Models

Steady-state equivalent surrogate models for the heat exchanger have been developed and tested using data sets derived from the physical simulator. A heat exchanger model incorporating mass flow rate and pressure approach was used in this case. The input-output diagram of the surrogate models is illustrated in Figure 7.

DOI
10.3384/ecp204599

Table 2. Heat exchanger surrogate model inputs domain.

Variable	Input	Lower Bound	Upper Bound	Unit
Pa_ref	Refrigerant pressure at refrigerant flow port A	150000	1800000	Pa
Pa_air	Air pressure at refrigerant flow air port A	20000	110000	Pa
m_ref	Refrigerant mass flow rate	-0.16	0.16	kg/s
m_air	Air mass flow rate	-1.8	1.8	kg/s
Tin_air	Air inlet temperature	-20	60	°C
Hin_ref	Ref. inlet enthalpy	210	490	kJ/kg

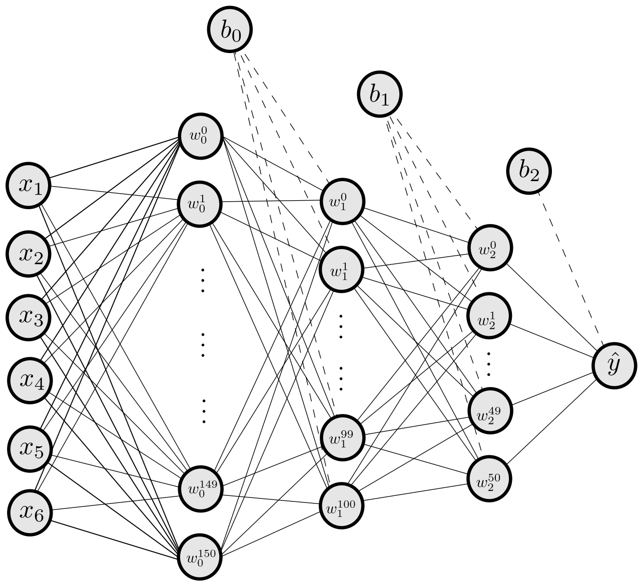


Figure 6. Architecture of the MLP used in this paper.

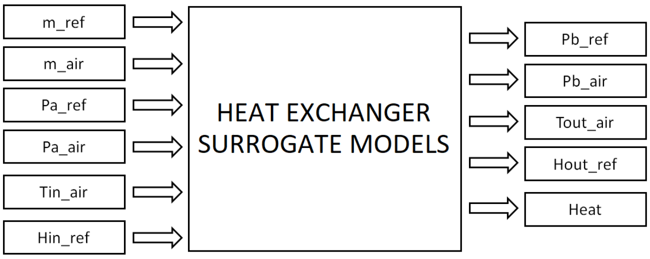


Figure 7. Heat exchanger surrogate model Inputs and Outputs.

Table 3. Specification of the heat exchanger surrogate.

Variable	Output	Unit
Pb_ref	Refrigerant pressure at port B	Pa
Pb_air	Air pressure at port B	Pa
Tout_air	Air outlet temperature	°C
Hout_ref	Ref. outlet specific enthalpy	kJ/kg
Heat	Heat transferred between fluids	W

4.1 Generating the data sets

Data collection for the heat exchanger surrogate models has been performed by evaluating the physical model developed in the Modelica framework. The data sets are described in Table 2 and 3 respectively. In total, four data sets were prepared. The first two data sets, consisting of 150 and 80,000 points, were drawn using the Halton random sequence (Owen 2017). The remaining two data sets consist of 40,000 and 100,000 random points for hyperparameter optimization and validation of the surrogate model, respectively (Gramacy 2020).

5 Surrogate Modeling Results

The performance of the surrogate models is evaluated in terms of their predictive accuracy. Furthermore, hyperparameter optimization was conducted for the Multi-Layer Perceptron (MLP) to improve the performance of the surrogate models.

5.1 Performance Comparison

Surrogate models have been developed and benchmarked for all of the outputs of the heat exchanger. In particular, four different surrogate model scenarios are executed to select the surrogate model with the best performance. The considered surrogate model scenarios are:

- MLP trained on 80,000 points (MLP-80K).
- MLP trained on 150 points (MLP-150).
- GP trained on 150 points (GP-150).
- Random Forest (RF) trained on 150 points (RF-150).

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \qquad (13)$$

The Root Mean Squared Error (RMSE) is used as a metric to evaluate the surrogate models on testing data sets of 100,000 points. The RMSE formula is presented in equation 13. The full result of the benchmark is shown in Table 4. The surrogate model with the lowest RMSE is used for each output as the final surrogate mode. Almost all MLP-80K models have the best RMSE compared to the other methods, except for the Pb_air output. In this case, GP is superior to MLP. Thus, all outputs are modeled using MLP with 80,000 data points except for Pb_air.

Table 4. Testing Root Mean Squared Error (RMSE) of all compared methods.

Output	RMSE			
	MLP-80K	MLP-150	GP-150	RF-150
Pb_ref	**8.54e+3**	4.55e+5	1.30e+5	5.06e+4
Pb_air	2.33e+2	7.36e+3	**4.24e+1**	4.14e+2
Tout_air	**1.51e+0**	1.26e+2	2.27e+1	3.49e+1
Hout_ref	**5.20e+3**	5.59e+5	8.02e+4	1.07e+5
Heat	**4.52e+2**	2.56e+4	8.98e+3	7.86e+3

5.2 Neural Network Hyperparameter Optimization

To explore the potential of MLP models in more detail, a hyperparameter optimization step was performed.

HyperParameter Optimization (HPO) (Yu and H. Zhu 2020; Morales-Hernández, Van Nieuwenhuyse, and Rojas Gonzalez 2022) for the MLP has been conducted using the Optuna framework (Akiba et al. 2019). The optimized hyperparameters are the learning rate and the architecture: the number of the hidden layer, the number of neurons of the hidden layers, and the activation function. Full specification of the MLP hyperparameter search space is presented in Table 5.

Table 5. Hyperparameter search space for the HPO.

Variable	Domain	Type
Number of hidden layer	[2, 6]	Integer
Number of neurons	[8, 1024]	Integer
Learning rate	[0.0001, 0.01]	Float
Activation function	{identity, tanh, logistic, ReLU}	Function

The HPO was conducted with a budget of 250 iterations. The training data consisted of 80,000 points, while the cost function was defined as the loss value on a randomly sampled validation data set of 40,000 points. The optimal hyperparameters identified through the HPO process are presented in Table 6.

Finally, the complete comparison of the Base-MLP and the optimal architecture found by the HPO (i.e., HPO-MLP) is presented in Table 7. It should be noted that the same hyperparameter setting from the previous section (MLP-80K) was used for the Base-MLP.

6 Modelica integration of the surrogate models

The resulting surrogate models have been integrated in the Modelica framework. To accomplish this, the optimal parameters obtained during training are stored as matrices on MATLAB (’*.mat*’) files, as they are compatible with the Modelica framework. For the GP model, these files store the training data, kernel parameters, and precomputed terms that are independent of newly observed data (i.e., $K_{XX}^{-1}Y$). This speeds up the computation of predictions since the expensive matrix inverse operation does not need to be recalculated every time a new point needs to be predicted. Additionally, for the MLP model, the resulting $\mathbf{W}_l$ and b_l terms are saved, along with the activation functions at each layer for the HPO-MLP.

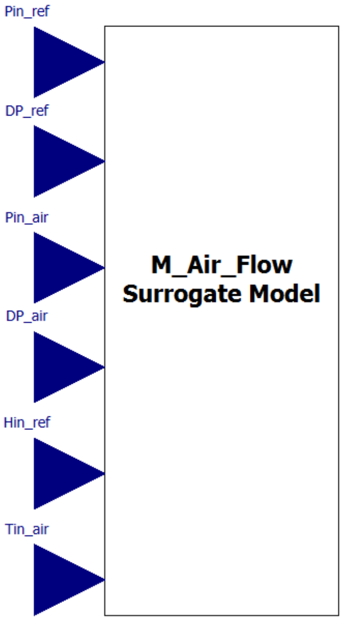


Figure 8. Surrogate model adaption in Open Modelica.

The integration process utilizes OpenModelica (Fritzson et al. 2005) and can also be adapted to Dymola. The command ’*readRealMatrix*’ imports the matrix, and ’*readMatrixSize*’ is used to retrieve the dimensions of the matrices. The surrogate model prediction routine was then implemented in Modelica using equations 10 and 12 for GP and MLP, respectively. The adapted Modelica surrogate prediction function block is shown in Fig. 8 where the six inputs required by the function are shown. The only input that this function block requires is the path where the ’*.mat*’ file can be found.

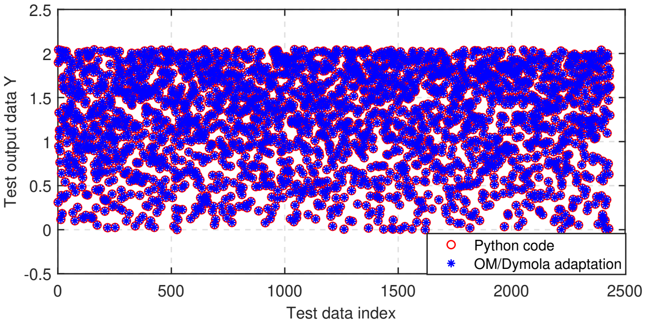


Figure 9. Testing the Open Modelica (OM)/Dymola adaptation.

2,500 test data points have been evaluated using the physical model for validating the Modelica implementation. We compared the resulting Modelica outputs with

Table 6. Multi-layer perceptron hyperparameter optimization result.

Variable	N hidden layer	Number of neurons	Learning rate	Activation function
Pb_ref	4	(1012, 252, 470, 392)	0.00119	ReLU
Tout_air	4	(410, 607, 906, 280)	0.00010	ReLU
Hout_ref	5	(752, 862, 122, 226, 221)	0.00098	ReLU
Heat	4	(895, 501, 667, 670)	0.00054	ReLU

Table 7. Root Mean Squared Error (RMSE) of Base-MLP and HPO-MLP. Ten repetitions have been conducted to validate the robustness of the optimized model.

Surrogate Name	RMSE ± Standard Deviation		Improvement (%)
	Base-MLP	HPO-MLP	
Pb_ref	9.380e+03 ± 5.608e+02	7.767e+03 ± 6.671e+02	17%
Tout_air	1.518e+00 ± 1.039e-01	1.266e+00 ± 1.016e-01	16%
Hout_ref	4.977e+03 ± 3.388e+02	4.374e+03 ± 5.778e+01	12%
Heat	4.685e+02 ± 4.408e+00	4.094E+02 ± 4.741E+01	13%

the original Python code for the surrogate model prediction. The results of this test are presented in Fig. 9, and it can be seen that the outputs are exactly matching. This generates trust to integrate the surrogate model in future applications where the heat exchanger will be used.

7 Conclusion

A heat exchanger model implemented in Modelica and adapted to simulate an air-to-refrigerant condenser has been validated and used to train and evaluate different surrogate models to mimic their steady-state behavior. The surrogate models are developed using Gaussian Process (GP) and Multi-Layer Perceptron (MLP) models. GPs are employed to capture the linear behavior of some heat exchanger outputs, while MLPs are utilized to handle other outputs with more complex, non-linear behavior. Additionally, hyperparameter optimization for the MLP architecture has been conducted, which led to significant improvements compared to the standard architecture. As a proof of concept, the surrogate models were also integrated in the Modelica/Dymola environment such that they can be directly augmented with physical models.

In this specific study, the surrogate model does not exhibit a substantial improvement in calculation time compared to the physical model. This limitation can be attributed to the fast nature of the physical model employed here. However, in cases with more complex physical models such as the distributed method, it will provide a large reduction in calculation time. Nonetheless, the surrogate model played a crucial role in ensuring calculation stability. Some of the factors that contribute to instability include operating modes transitions, empirical and corrector factor transitions, as well as thermophysical properties near the saturation dome, among others.

Future research will concentrate on constructing transient surrogate models to represent the transient behavior of the heat exchanger model explicitly. This can be accomplished by employing more suitable surrogate modeling techniques, such as non-stationary Gaussian Processes or Autoregressive Models like Long Short-Term Memory Neural Networks.

Acknowledgements

This project has received funding from the Clean Sky 2 Joint Undertaking (JU) under grant agreement No 886533. The JU receives support from the European Union's Horizon 2020 research and innovation program and the Clean Sky 2 JU members other than the Union.

Horizon 2020
European Union Funding
for Research & Innovation

Clean Sky 2
JOINT UNDERTAKING

This work has also been supported by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" and the "Fonds Wetenschappelijk Onderzoek (FWO)" programs.

Carles Oliet, as a Serra Húnter Associate Professor, acknowledges the Catalan Government for the support through this Programme.

Disclaimer: The contents presented in this article reflect only the author's point of view: the authors and Clean Sky JU are not responsible for any use that may be made of the information it contains.

References

Abbassi, A. and L. Bahar (2005). "Application of neural network for the modeling and control of evaporative condenser cooling load". In: *Applied Thermal Engineering* 25.17, pp. 3176–3186. ISSN: 1359-4311. DOI: https://doi.org/10.1016/j.applthermaleng.2005.04.006.

Ablanque, Nicolás, Santiago Torras, Carles Oliet, and Joaquim Rigola (2022). "Thermal Systems Oriented Two-Phase Heat Exchanger Models. Focus on Numerical Robustness". In: *19th International Refrigeration and Air Conditioning Conference at Purdue*, pp. 1–10. URL: https://docs.lib.purdue.edu/iracc/2405/.

Akiba, Takuya, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama (2019). "Optuna: A Next-Generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, pp. 2623–2631. ISBN: 9781450362016. DOI: 10.1145/3292500.3330701.

Fritzson, Peter, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman (2005-01). "The OpenModelica Modeling, Simulation, and Development Environment". In.

Fukushima, Kunihiko (1969). "Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements". In: *IEEE Transactions on Systems Science and Cybernetics* 5.4, pp. 322–333. DOI: 10.1109/TSSC.1969.300225.

G. Matthews, Alexander G. de, Mark van der Wilk, Tom Nickson, Keisuke Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman (2017). "GPflow: A Gaussian Process Library using TensorFlow". In: *Journal of Machine Learning Research* 18.40, pp. 1–6. URL: http://jmlr.org/papers/v18/16-537.html.

Genton, Marc G. (2002-03). "Classes of Kernels for Machine Learning: A Statistics Perspective". In: *J. Mach. Learn. Res.* 2, pp. 299–312. ISSN: 1532-4435. DOI: 10.5555/944790.944815.

Gramacy, Robert B. (2020). *Surrogates: Gaussian Process Modeling, Design and Optimization for the Applied Sciences*. Boca Raton, Florida: Chapman Hall/CRC. DOI: 10.1201/9780367815493.

Handcock, Mark S. and Michael L. Stein (1993). "A Bayesian Analysis of Kriging". In: *Technometrics* 35.4, pp. 403–410. ISSN: 00401706. DOI: 10.2307/1270273.

Hinton, Geoffrey E. (1989). "Connectionist learning procedures". In: *Artificial Intelligence* 40.1, pp. 185–234. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(89)90049-0.

Incropera, Frank P. and David P. DeWitt (1996). *Fundamentals of Heat and Mass Transfer*. Wiley. ISBN: 978-1-119-35388-1.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization." In: *ICLR (Poster)*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://dblp.uni-trier.de/db/conf/iclr/iclr2015.html#KingmaB14.

Morales-Hernández, Alejandro, Inneke Van Nieuwenhuyse, and Sebastian Rojas Gonzalez (2022). "A survey on multi-objective hyperparameter optimization algorithms for machine learning". In: *Artificial Intelligence Review*, pp. 1–51. DOI: 10.1007/s10462-022-10359-2.

Owen, Art B. (2017-06). "A randomized Halton algorithm in R". In: URL: http://arxiv.org/abs/1706.02808.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830. DOI: https://dl.acm.org/doi/10.5555/1953048.2078195.

Prince, Simon J.D. (2023). *Understanding Deep Learning*. MIT Press. URL: https://udlbook.github.io/udlbook/.

Rasmussen, Carl Edward and Christopher K. I. Williams (2018). *Gaussian Processes for Machine Learning*. Cambridge, Massachusetts: The MIT Press. DOI: 10.7551/mitpress/3206.001.0001.

Romero-Méndez, Ricardo, Patricia Lara-Vázquez, Francisco Oviedo-Tolentino, Héctor Martín Durán-García, Francisco Gerardo Pérez-Gutiérrez, and Arturo Pacheco-Vega (2016). "Use of Artificial Neural Networks for Prediction of the Convective Heat Transfer Coefficient in Evaporative Mini-Tubes". In: *Ingeniería, Investigación y Tecnología* 17.1, pp. 23–34. ISSN: 1405-7743. DOI: https://doi.org/10.1016/j.riit.2016.01.003.

Yu, Tong and Hong Zhu (2020). "Hyper-Parameter Optimization: A Review of Algorithms and Applications". In: *CoRR* abs/2003.05689. arXiv: 2003.05689. URL: https://arxiv.org/abs/2003.05689.

Zhu, Ciyou, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal (1997-12). "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization". In: *ACM Trans. Math. Softw.* 23.4, pp. 550–560. ISSN: 0098-3500. DOI: 10.1145/279232.279236.

5th Generation District Heating and Cooling Modelica Models for Prosumer Interaction Analysis

Angelidis, O.[1], Zinsmeister, D.[2] , Ioannou, A.[1], Friedrich, D.[3], Thomson, A.[4], Falcone, G.[1]

[1] James Watt School of Engineering, University of Glasgow, Glasgow G12 8QQ, UK,
2072680a@student.gla.ac.uk

[2]Technical University of Munich, Arcisstrasse 21, Munich, 80333, Germany, d.zinsmeister@tum.de

[3] School of Engineering, The University of Edinburgh, Edinburgh EH9 3FB, UK, D.Friedrich@ed.ac.uk

[4] Ramboll, 240 Blackfriars Rd, London SE1 8NW, UK, Alan.Thomson@ramboll.com

Abstract

5th Generation District Heating and Cooling (5GDHC) provides a promising pathway for decarbonising the thermal sector. To quantify the synergies between heating, cooling, and electricity, complex thermofluid models are required. Modelica offers a potential solution for developing such models but despite recent research efforts, there is a lack of bespoke 5GDHC component models in literature. This paper addresses this gap by presenting a comprehensive set of Modelica models for key elements of 5GDHC systems and their interactions: prosumers, balancing units, and hydraulic interfaces. The models comprise some commercial libraries. To facilitate accessibility, Functional Mock-up Units (FMU) are generated for these models, which can be opened by any Modelica environment using Functional Mock-up Interface (FMI). Component design, relevant controls, and the applicability of Power Hardware-in-the-Loop (PHIL) setups are discussed. A theoretical use case exemplifies hardware minimisation, using only heat exchangers to investigate prosumer behaviour. The paper concludes with a discussion on the potential use of these models, opportunities for improvement, and the need for further research and experimental investigations in understanding 5GDHC systems.

Keywords: 5th Generation District Heating and Cooling, Power Hardware-in-the-Loop, Energy Systems

1 Introduction

Among the efforts to limit the impact of climate breakdown and rise of global temperature levels, the decarbonisation of thermal networks represents a crucial challenge, especially while trying to maintain security of supply and low costs (IEA 2021). A system that is attracting increasing attention is 5th Generation District Heating and Cooling (5GDHC) which offers opportunities for synergies between heating and cooling loads, low temperature waste heat utilisation and sector coupling with the electricity grid through the use of heat pumps (Gjoka, Rismanchi, and Crawford 2023). This system utilises an ambient network for meeting both heating and cooling demands with decentralised energy stations. They feature water source heat pumps, boosting the temperature for meeting heating or cooling needs and thus commonly referred to as Booster Heat Pumps (BHP), Thermal Energy Storage (TES) and hydraulic pumps. Since buildings are feeding heat/coolth into the ambient network while they are using coolth/heat, they are referred to as prosumers. The thermal and hydraulic balance is provided to the system by a balancing unit, which adds heat or coolth depending on the demand requirement of the network (Buffa et al. 2019).

However, this pumping and energy unit decentralisation leads to a bidirectional flow regime in the network when heating and cooling demands are present. This may in turn cause thermodynamic subcycles, hydraulic misbalances such as “pump hunting” depending on the topology of the network and the transient behaviour of the network medium (Angelidis et al. 2023). To capture the operational complexity of such systems, it is key to accurately model thermofluid behaviour. Detailing the hydraulic and energy flow interaction coupled with overarching controls is a challenge that fits the multi-engineering scope of the Modelica simulation language (Abugabbara 2021). Modelica allows for accurate simulation of the system dynamics including bidirectionality of flow, pressure constraints, flow characteristics and energy interactions between heating and cooling. It is recognised by the International Energy Agency as one of the key computational tools for building system modelling (Wetter and Treeck 2017). Modelica features multiple open access libraries with validated components for buildings and community heating and cooling energy systems, including the Buildings (Wetter et al. 2014) and AixLib (Mueller et al. 2016) libraries, summarised in one library under BESMod (Wüllhorst et al. 2022).

Regarding 5GDHC systems, publications have focused on describing modelling methodologies and subcomponent development, aimed mainly at studying particular elements (Blacha et al. 2019; Abugabbara, Javed, and Johansson 2022; van der Heijde et al. 2017). However, these studies have limitations. The developed models are not provided for reuse, nor include a comprehensive explanation of the interplay between control regimes and prosumer, balancing unit, and decentralised pumping station interaction. Furthermore, they have been mostly case-specific, with only some Buildings library components providing limited insights on BHP and TES interaction and overarching control. Finally, prosumer interaction, the function of the balancing unit and the effects of decentralised pumping to system performance has not been experimentally validated. This is mainly due to the large number of units and hardware components required to study such interactions. Power Hardware-in-the-Loop (PHIL) provides a method for combining simulation tools with real hardware, interfacing through digital and analogue input/output signals, that could facilitate system-wide experiments with the use of minimal hardware. Facilitating such experiments through the provision of bespoke Modelica models for 5GDHC would be a step forward in understanding and quantifying the complex behaviour of such systems.

The aim of this paper is to present a set of comprehensive Modelica models, including experimentally validated subcomponents from the ProHMo library[1] for prosumers, hydraulic interface, and balancing unit to accurately simulate 5GDHC systems. The models have been developed to facilitate PHIL implementations, enabling experimental analyses of prosumer interactions in 5GDHC. A methodology for utilising only a heat exchanger (HEX) to replicate prosumer behaviour is presented along with a discussion on usability of the models using Functional Mock Up Interface (FMI). This feature allows the presented components to be used in any Modelica environment or in combination with Energy Management Systems (EMS) from other coding environments such as Python.

The library design is discussed in section 2, with a detailed investigation of the system components along with rule-based control strategies implemented. Section 3 includes an exemplary use case of the components for a simple 5GDHC system with two prosumers and a balancing unit. In section 4, the methodology for PHIL setups is discussed for experimental analysis of prosumer interaction or developed digital twins with minimal hardware use. Section 5 includes a discussion on strengths and limitations of the presented models along with the areas for further research. Finally, section 6 concludes with future use cases and research options.

2 Component Design

The development of the Modelica components is guided by five key guiding principles, namely usability, scalability, accuracy, flexibility & validity (Wetter and Treeck 2017). The prosumer and balancing unit models were based on equipment from the thermal Prosumer House Model (ProHMo) library (Zinsmeister and Perić 2022). The ProHMo library includes experimentally validated components from the Center for Combined Smart Energy Systems (CoSES) lab that are scalable. It is based on the Green city library from the commercial Modelica environment Simulation X (Zinsmeister and Perić 2022). The library uses a thermal only approach to simplify the models and shorten simulation time, where pressure influences are neglected. This simplification is valid for heating systems within houses (Zinsmeister and Perić 2022; Zinsmeister et al. 2023).

To model the interaction of prosumers in a district heating network with several prosumers, it is important to represent the network in detail, including pressure losses and bidirectionality of flow. For this purpose, the building models of ProHMo are coupled with hydraulic components through a communication interface submodel, referred to in this paper as hydraulic interface. The hydraulic interface serves as an accurate and comprehensive representation of the hydraulic components within the system, their behaviour and interaction. It comprises interconnected hydraulic elements (pumps, valves, sensors, pipes and elements of hydraulic resistance), facilitated by hydraulic connectors, and replicates all relevant elements encountered in real-world applications.

Furthermore, fitting control strategies are needed for all components for different grid operations. In this section, the development of bespoke components for 5GDHC is presented, allowing the setup for creating digital twins, an example of which is shown in Figure 1. In this figure, two prosumers are connected with a balancing unit through a thermal grid. The hydraulic interfaces allow for the hydraulic connection of the only thermal connector models. The prosumer and balancing unit models, as well as the hydraulic interfaces, are discussed below.

2.1 Prosumers

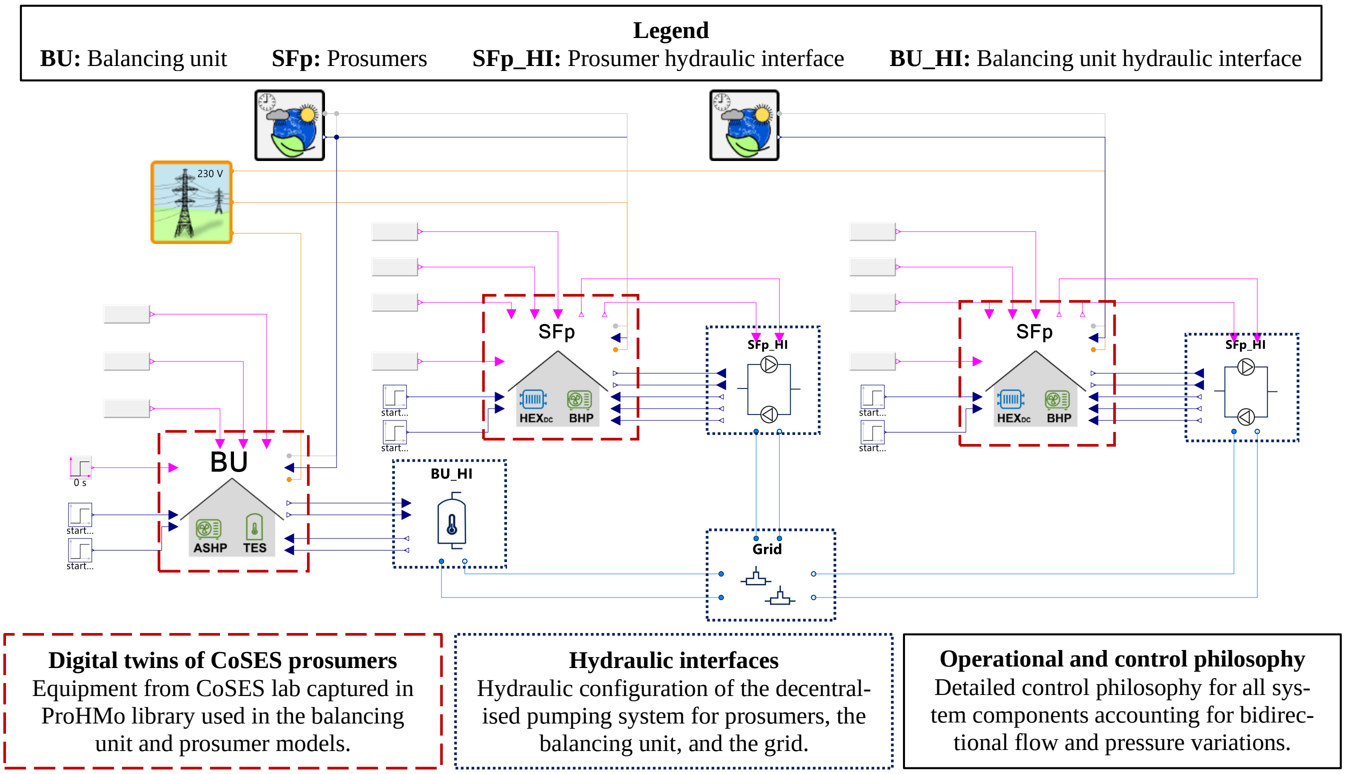
The prosumer model includes energy transformation units, thermal stores and demands. It can represent both Space Heating (SH) and Space Cooling (SC) demand along with Domestic Hot Water (DHW). The Modelica model is shown in Figure 2.

2.1.1 Model Description

The operation of the BHP and a Direct Cooling Heat Exchanger ($\mathrm{HEX_{DC}}$) is the focal point in the prosumer component. $\mathrm{HEX_{DC}}$ allows for direct utilisation of the coolth from the network's cold pipe (if low enough) without upscaling it via a BHP. It has been shown that their use in 5GDHC is instrumental to the system's efficient operation (Wirtz et al. 2021). For SH and DHW, the load is to be supplied mainly from the BHP with any additional loads supplied by an auxiliary heater (electric resistance) placed within the BHP unit. For heating, the energy transformation units are connected in series with the TES which is discharged by the heat sinks. Cooling is directly supplied by the energy transformation units ($\mathrm{HEX_{DC}}$ or BHP) without going through the TES.

The BHP model is based on measurements of a commercial BHP found in the CoSES lab, reproducing its efficiency and dynamics. The TES model has also been experimentally validated (Zinsmeister and Perić 2022) and is represented by a one dimensional stratified model, where the TES is split into multiple layers of constant size. 10 temperature layers are used in the ProHMo library to match the number of temperature sensors in the physical unit in the lab. The maximum temperature, seen at level

[1] Available online at: https://gitlab.lrz.de/energy-management-technologies-public/coses_prohmo



Legend

BU: Balancing unit **SFp:** Prosumers **SFp_HI:** Prosumer hydraulic interface **BU_HI:** Balancing unit hydraulic interface

Digital twins of CoSES prosumers
Equipment from CoSES lab captured in ProHMo library used in the balancing unit and prosumer models.

Hydraulic interfaces
Hydraulic configuration of the decentralised pumping system for prosumers, the balancing unit, and the grid.

Operational and control philosophy
Detailed control philosophy for all system components accounting for bidirectional flow and pressure variations.

Figure 1. Library components used for 5GDHC system development.

10, is set to 60°C. This value satisfies both DHW supply and legionella avoidance requirements (Chartered Institution of Building Services Engineers (CIBSE) 2020). A hydraulic switch, namely a 3-Way Valve (3WV), can change the charging levels based on temperature in the TES. Discharching for SH is from layers 5 (Flow) and 1 (Return) since there is a low temperature heating system (underfloor heating) and layer 10 (Flow) and 1 (Return) for DHW. The discharge of the TES is modulated by a pump valve setup based on temperature and flow requirements from the heat sinks.

The SH and SC demands are captured by adapted Green City library models which allow for different number of residents, construction characteristics, building type and terminal units. The default is set to new buildings with underfloor heating/cooling systems which is most relevant for 5GDHC prosumers with heating and cooling demands (Angelidis et al. 2023). The flow and return temperature depend on the flowrate supplied by the tertiary pumps (variable flowrate pumps in the building) but are designed for 40-30°C for heating and 15-20°C for cooling. Both SH and SC are modulating around a temperature setpoint (21°C for heating and 23°C for cooling) by varying the request inlet flowrate. Similarly, DHW is modelled, requiring a temperature of 60°C and, based on the consumption, returning a cooled down water at varying flowrates. There is a heat exchanger between the end DHW consumption and the water from the TES. DHW is dependent on the number of residents and can be switched off during

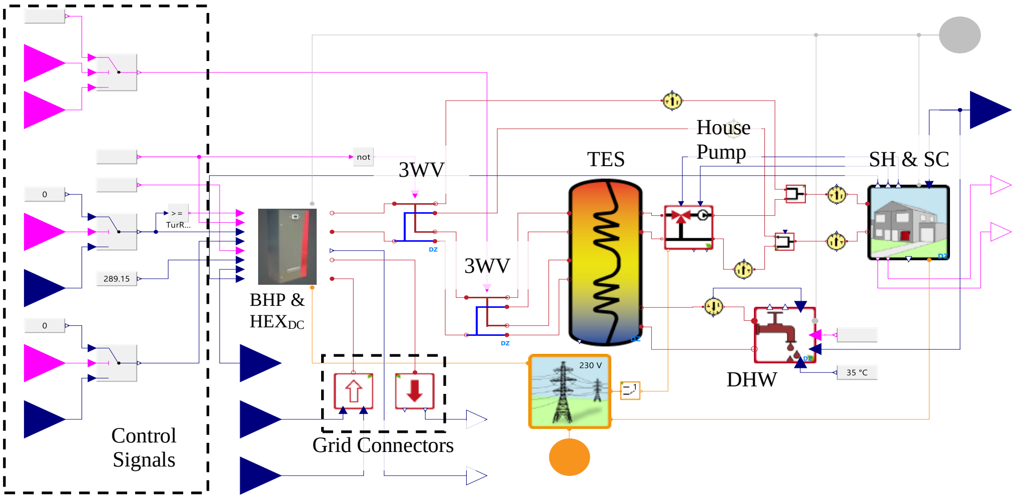


Figure 2. Prosumer Modelica Model

cooling operation (if no DHW is required during cooling periods). At each time step, there can only be heating or cooling demands with a 3WV alternating between BHP or $\mathrm{HEX}_{\mathrm{DC}}$ when in cooling mode.

2.1.2 Control Strategy

The control strategy for modulating the BHP in heating mode is built around the discharging rate of the TES. The goal for the control is to keep a stratified TES, minimise the starts and stops of the BHP, keep a minimum temperature of 55°C on the TES at layer 9 and maximise system efficiency. Based on these objectives, the control uses a 3WV to charge the top or middle of the TES, with priority given to charging the top layer. To avoid on/off control with hysteresis (system lagging to the input signal), a novel control method is proposed with the modulation of the BHP as a function of the reference TES temperature layer. Equation 1 shows how the modulation factor is determined by the ratio between the actual and maximum temperature difference for the TES temperature layer against set maximum and minimum values.

$$mf = (((\max\,(0, \min\,(1, (1 - \frac{T_{reference} - T_{sup,minT}}{T_{sup,maxT} - T_{sup,minT}})))) \tag{1}$$

where mf is the modulation factor for the BHP, $T_{reference}$ is the reference temperature layer, $T_{sup,minT}$ is the minimum temperature value for the reference temperature layer and $T_{sup,maxT}$ the maximum temperature value for the layer. When the reference temperature is equal to the maximum allowed temperature, the modulation factor is zero. Conversely, when the temperature matches the minimum allowed temperature, the modulation factor is 1. To ensure the modulation factor stays within the bounds of 0 and 1, a max-min definition is applied. This approach accounts for cases that the temperature levels in the TES exceed the upper limit (e.g., on start-up).

To maintain TES stratification, the prosumer component utilizes two modulation factors: one for the top for DHW and one for the middle for SH, as shown in Figure 3. Depending on the setting of the 3WV, the respective modulating factor is used, with the reference temperature layer set to layer 7 for charging of the top of the TES and layer 4 for the middle. These layers are chosen to limit hysteresis and the impact of water inflow to/outflow from the TES.

It is seen that the higher layer modulation factor mf_h is utilising a temperature band between the start and stop temperature setpoints, $T_{StartHP,h}$ and $T_{StopHP,h}$ respectively. In a similar manner, the lower layer modulation factor mf_l is determined by a lower temperature range $T_{StartHP,l}$ and $T_{StopHP,l}$. This control strategy allows for a stratified TES, maximisation of BHP operation and abiding to top level minimum temperature requirements. An operation example for 1 day is shown in Figure 4. The difference between layer 5 and 6 occurs due to the water outflow from the TES for SH demands occurring at layer 5.

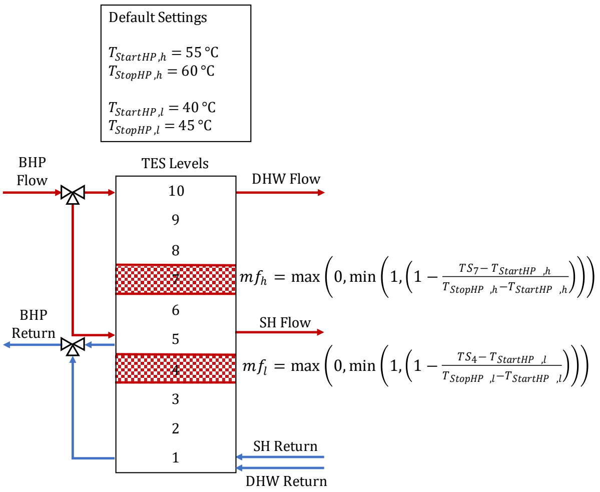

Default Settings
$T_{StartHP,h} = 55\,°\mathrm{C}$
$T_{StopHP,h} = 60\,°\mathrm{C}$
$T_{StartHP,l} = 40\,°\mathrm{C}$
$T_{StopHP,l} = 45\,°\mathrm{C}$
BHP Flow
TES Levels
DHW Flow
10
9
8
7
$mf_h = \max\left(0, \min\left(1, \left(1 - \frac{TS_7 - T_{StartHP,h}}{T_{StopHP,h} - T_{StartHP,h}}\right)\right)\right)$
BHP Return
6
SH Flow
5
4
$mf_l = \max\left(0, \min\left(1, \left(1 - \frac{TS_4 - T_{StartHP,l}}{T_{StopHP,l} - T_{StartHP,l}}\right)\right)\right)$
3
2
1
SH Return
DHW Return

Figure 3. Schematic of control methodology for BHP.

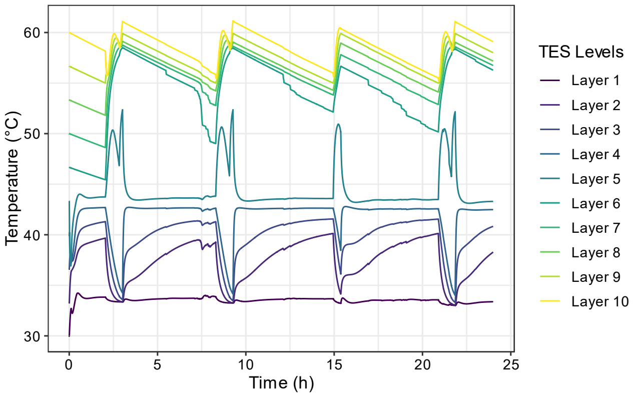


Figure 4. TES operation under modulation of the BHP

For SC, at default settings, priority is given to $\mathrm{HEX}_{\mathrm{DC}}$ over the BHP (in cooling mode). The choice of switching to the use of the BHP if the room is not cooled after a designated time (defined by the user) is also provided.

Finally, a further control option has been added for the operation of the BHP. This allows for operation of the evaporator and/or the compressor under constant temperature difference or flowrate, both of which are available for commercial BHP units. Depending on the operation, the power modulation is achieved by varying the non-fixed variable within limits set by the user. The equations governing these behaviours have been modified in the models utilizing conditional functions ("if" statements) to adapt their operation accordingly. By implementing these adjustments, the BHP and grid inlets can be dynamically controlled, enabling greater flexibility in their operation. This adaptability allows for improved system performance and optimization tailored to the specific use case, with due consideration given to external factors such as flowrate and temperature differences.

2.2 Balancing unit

The balancing unit is responsible for providing thermal and hydraulic balance to the network. The Modelica

DOI
10.3384/ecp204607

model is captured in Figure 5 and described in the following sections.

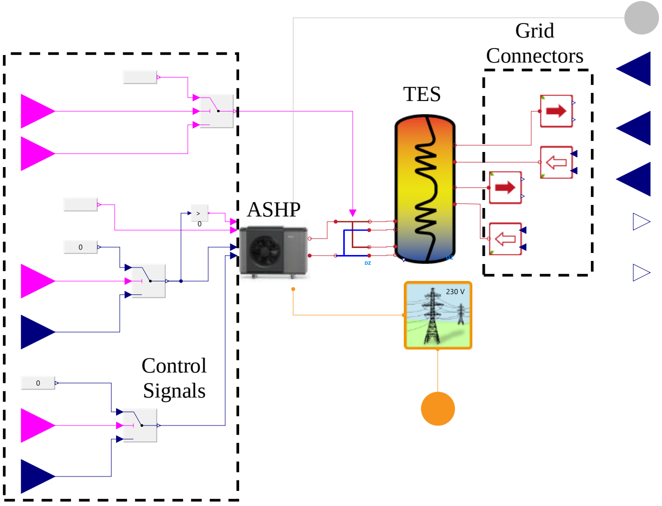


Figure 5. Balancing Unit Modelica Model

2.2.1 Model Description

An Air Source Heat Pump (ASHP) is connected in series with a TES that acts as a passive interface between the hot and cold pipes. This setup with the TES directly connected to the hot and cold pipe of the network (hot grid pipe at the top of the TES and cold grid pipe at the bottom), provides a passive hydraulic balance, critical for the operational integrity of the system featuring decentralised pumps and energy transformation units. The TES is therefore suppling heat or cold depending on the energy misbalance. The hot grid pipe is connected to the top of the TES (layer 10) while the cold pipe to the bottom (layer 1), allowing for a stratified TES with the hot pipe temperature at the top (e.g., 20°C) and the cold pipe temperature at the bottom (e.g., 15°C). Depending on the thermal balance needed by the network, the TES is cooling down (during heating balance needed) or heating up (during cooling balance needed). The ASHP needs to keep the TES temperature within the operational limits by recharging the top or bottom of the TES with heat or coolth respectively.

2.2.2 Control Strategy

To achieve this operational strategy, the ASHP is connected in series with the TES where a 3WV can change the TES charging levels based on mode of operation of the ASHP. Therefore, charging for heating uses level 9-6 for flow and return and level 2-5 for flow and return for cooling operation. This setup allows for unidirectional flow through the ASHP while keeping a stratified TES without mixing when variations between heating to cooling dominant system operation occurs. The mode of the ASHP depends on the flow direction of the grid, with cooling activated when the flow leaves the bottom of the TES, and heating when the flow leaves from the top.

The ASHP operation is following the same rule-based control for the modulation factor as the one described in equation (1). The operation of the balancing unit is captured in Figure 6, with an explanation of operation during heating and cooling dominant network operations described in the following paragraphs.

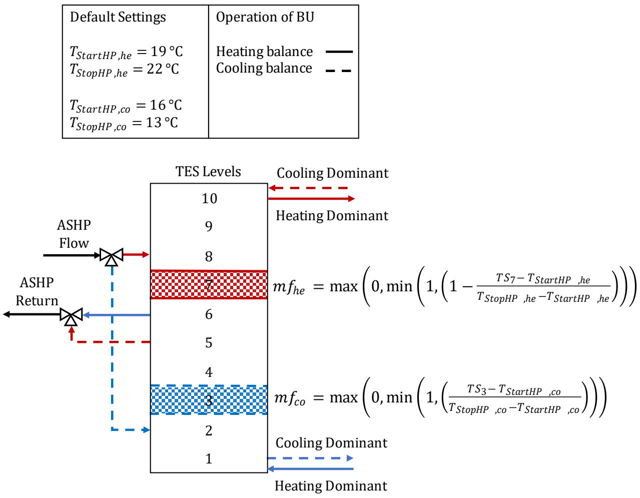

Default Settings
$T_{StartHP,he} = 19\,°C$
$T_{StopHP,he} = 22\,°C$
$T_{StartHP,co} = 16\,°C$
$T_{StopHP,co} = 13\,°C$
Operation of BU
Heating balance
Cooling balance
TES Levels
10
9
8
7
6
5
4
3
2
1
ASHP
Flow
ASHP
Return
Cooling Dominant
Heating Dominant
$mf_{he} = \max\left(0, \min\left(1, \left(1 - \frac{TS_7 - T_{StartHP,he}}{T_{StopHP,he} - T_{StartHP,he}}\right)\right)\right)$
$mf_{co} = \max\left(0, \min\left(1, \left(\frac{TS_3 - T_{StartHP,co}}{T_{StopHP,co} - T_{StartHP,co}}\right)\right)\right)$
Cooling Dominant
Heating Dominant

Figure 6. Balancing unit setup and connection schematic

Like the BHP heating setup, there are two modulation factors for the ASHP, in this case depending on the operation mode (heating or cooling). During heating, the flow going through the TES is from the bottom to the top with the ASHP in heating mode. The heating modulation factor mf_{he} is used which is calculated based on equation (1) with the upper and lower temperature bands being $T_{StartHP,he}$ and $T_{StopHP,he}$. For cooling, flow is reversed in the grid, with hot water coming in at the top of the TES and cold one coming out at the bottom. Therefore, the ASHP is in cooling mode, cooling down the lower half of the TES. For the modulation factor during cooling mf_{co} there is no need to subtract the ratio of the reference temperature from 1 since it directly responds to the cooling power requirements. This operation also allows for a stratified TES that can respond to dynamic changes in heating/cooling balance requirements.

2.3 Hydraulic Interface

The hydraulic interface is needed for the connection of Modelica components with thermal connectors to a system with hydraulic connectors that can capture bidirectional flow as well as pressure variations.

The hydraulic interface can avoid utilising library components that are only available in Simulation X, therefore open access Modelica standard library components are preferred. The functionality of the interface follows the methodology presented in the ProsNet library (Elizarov and Licklederer 2021), where the primary and secondary side communicate through a set of input/output signals. The key novelty in the approach developed in this paper, is the introduction of a thermal volume to represent the prosumer, considering thermal inertia and pressure variations of the system. Therefore, we can combine the benefits of utilising thermal only connectors in the prosumer and balancing unit components (low computational times and lower complexity) without compromising the hydraulic performance of the system. At the same time, this setup allows for a clear separation between the thermal only models utilising Simulation X components that can be turned into Functional Mock-Up units (FMU) as discussed in Section 2.4. The hydraulic interfaces for the prosumer, the balancing unit, and the grid model are illustrated in Figure 7.

For the prosumer hydraulic interface unit, the key inputs and outputs from the hydraulic interface are temperature [°C] and flowrate [l/min]. Signals for the set flowrate $\dot{V}_{g,set}$ asked by the prosumer and the output temperature $T_{p,out}$ from the prosumer are sent to a volume representing the prosumer, allowing for thermal inertia to be accounted for, resulting in the temperature the grid actually sees from the prosumer, $T_{g,out}$. Depending on the instantaneous demand mode (heating or cooling), the respective pump from the interface becomes active and flow is thus changing direction respectively. We use a PI controller to

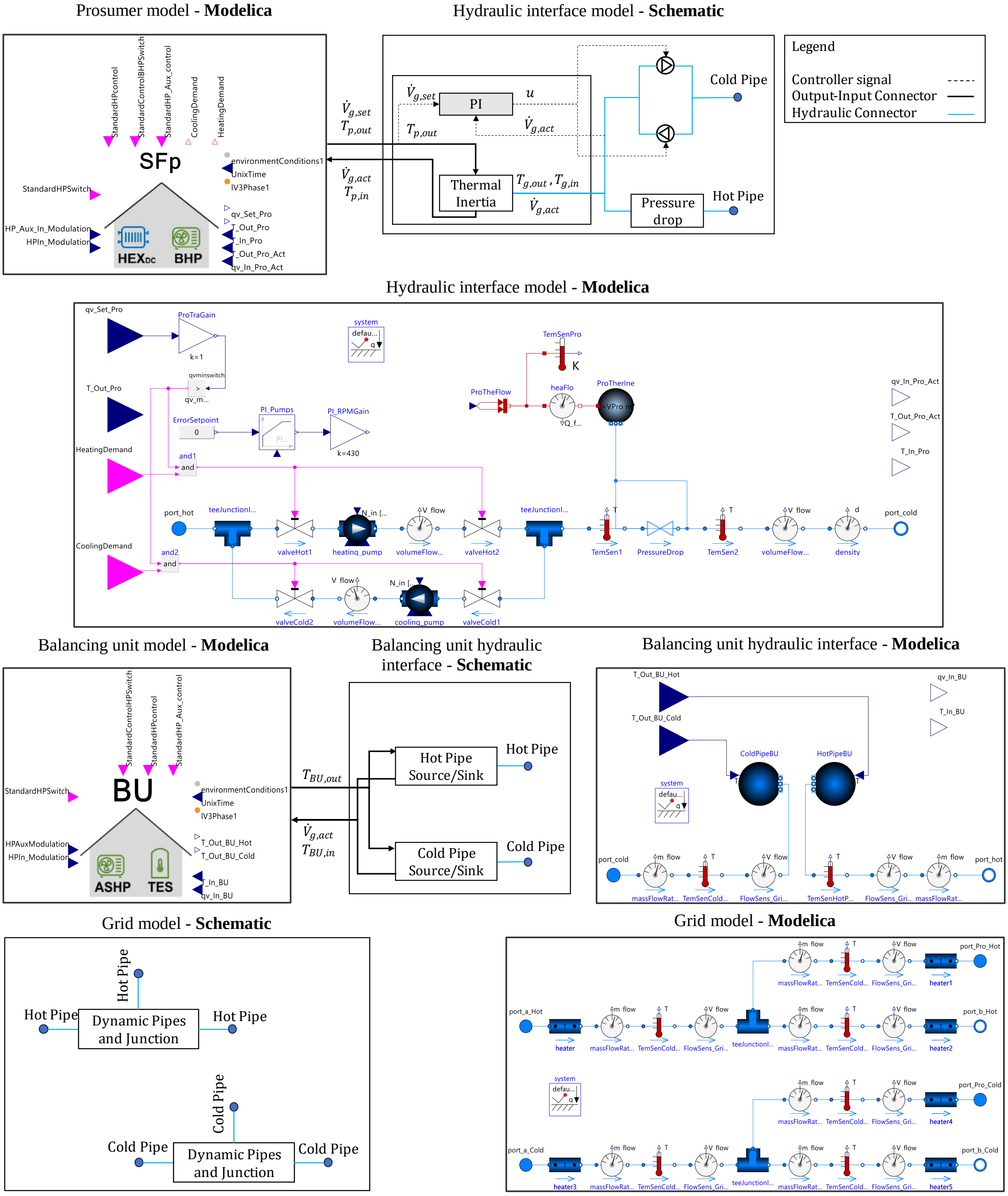


Figure 7. Hydraulic interfaces for prosumer and balancing unit as well as hydraulic model of the grid

give the setpoint u to the respective pump, considering the actual $\dot{V}_{g,act}$ and set flowrate $\dot{V}_{g,set}$. Then, $\dot{V}_{g,act}$ and $T_{g,in}$ are fed back to the prosumer as inputs.

For the balancing unit's hydraulic interface, the key input is the temperature from the balancing unit. The temperature corresponds to the top or bottom of the TES, depending on the flow direction, namely the sign of $\dot{V}_{g,act}$ as described in Section 2.2.2. If $\dot{V}_{g,act}$ is positive, which means there is dominant heating demands in the grid (flow from cold to hot port), then the hot pipe volume acts as a source with $T_{BU,out}$ being equal to the temperature at the top of the TES. $T_{BU,in}$ equal to the temperature of the cold pipe flows at the bottom of the TES. The opposite happens when there is cooling dominant operation and thus a negative $\dot{V}_{g,act}$, with the cold pipe volume becoming a source and the hot pipe volume becoming a sink.

The pipe network, namely the grid model, comprises dynamic pipes, sensors and junctions to allow for the connection of the prosumers and the BU. The grid model allows for parallel connection between loads, and includes ports for both the hot and cold pipes.

2.4 FMUs of Prosumers and Balancing Unit

To further increase the usability of the model, both prosumer and balancing unit models are developed so that they can be exported to FMUs, allowing for their use through the FMI standard for application in all Modelica environments (The Modelica Association 2023). With FMUs for these components, an arbitrary size of network can be built, with varying topologies and design and operational characteristics in any Modelica environment. However, the benefits from using a FMU come at a cost of transparency and editability. The components become "black boxes" that have specific elements that can be edited, significantly limiting the flexibility of the models to change. To maximise their usability, a set of key parameters have been made editable in the FMU. These follow the ProHMo library methodology as described in (Zinsmeister and Perić 2022), and include:

- Inputs for individual control setpoints
- Weather files
- Consumption parameters
- Energy generator unit capacities
- TES dimensions

3 Exemplary Use Case

To showcase the usability of the produced models, a simple system is used. It involves a heating and cooling prosumer as well as a balancing unit connected through a grid element in parallel. This setup is the one shown in Figure 1, Section 2. A constant temperature difference is kept between the cold and the hot pipe, and the grid pipes are modulated based on variable flowrate. HEX_{DC} is used for the cooling prosumer while the BHP for the heating prosumer (connected in series to the TES).

The simulation is performed for one day, with an aim to observe the behaviour of the system and qualitatively verify its operation. Figure 8 displays key outputs, namely the temperature levels of the top and bottom layers of the BU TES, the temperature in the living zones of the prosumers as well as the temperature and flowrate values on the grid's junction.

Plot A indicates the fluctuations of the temperature levels in the TES of the balancing unit, responding to heating and cooling requirements in the grid while keeping the upper (22°C) and lower (13°C) temperature limits. The spikes observed occur during ASHP start-up, with a momentary large intake. Plot B shows that the temperatures in both prosumer's living areas are maintained at the target reference temperatures (21°C for heating and 23°C for cooling). Larger deviations are observed during cooling due to the controller setting, underfloor cooling system behavior and the house pump's flowrate capacity.

Graphs C and D present temperature levels at both the hot and cold pipes. In plots E and F, flow halts for the cooling prosumer after hour 13, causing the respective pipe temperatures to track ambient temperatures and those of the segment preceding it. During the flow interruption

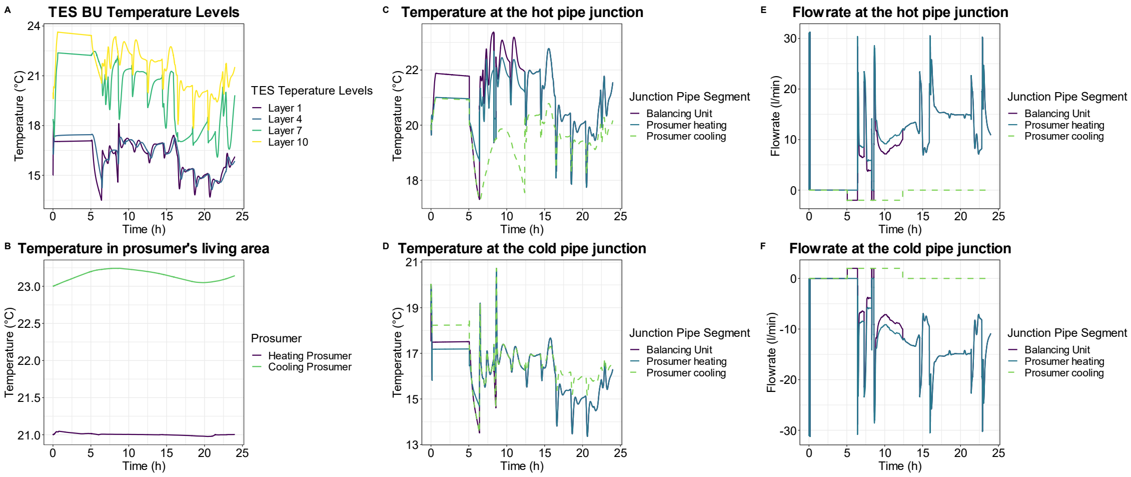


Figure 8. Exemplary case study outputs

until hour 5, the balancing unit remains idle, with the TES temperature slightly decreasing due to energy losses.

Overall, hydraulic and thermodynamic balances are kept in the system. The temperatures are maintained in the prosumer houses and bidirectionality of flow is captured. The balancing unit can operate both in heating and cooling mode ensuring that the top and bottom temperature levels are kept. It is shown that the components provide a working basis for investigations of different design cases and operation strategies. The next section describes how such designs can be validated with minimal hardware utilising PHIL approaches.

4 Power Hardware-in-the-Loop

Prosumer behaviour and interaction under different design conditions and control methodologies is one of the key gaps in research of 5GDHC systems. Experimentally validating models would require multiple BHP and buildings with both heating and cooling demand as well as the ancillary equipment (valves, pipes etc.) for developing a thermal network. To facilitate experimental validation of generated system models or the experimental assessment of prosumer interaction under varying control and design philosophies, the components are designed in such a way as to be able to utilise PHIL with minimal hardware requirements. Figure 9 illustrates how PHIL can be used for experimentally simulating a prosumer with only a HEX.

The HEX is sending metered signals to the prosumer simulation model for the flowrate and temperature present both on the primary and secondary side of the HEX. These are converted to standard unit values via a conversion module and fed to Modelica, which in turn sends back control signals. For the conversion & control modules, various software/hardware interaction methodologies are available. For example, the CoSES lab utilises Industrial Controllers for the hardware, communicating in real time with NI VeriStand for the conversion of logged data and control setpoints, as thoroughly explained in (Zinsmeister et al. 2023). Regarding hardware, other than the HEX, a heating and/or cooling unit are required to raise/drop the temperature for both the prosumer and grid side.

Prosumers' BHP and HEX$_{\mathrm{DC}}$ can be emulated with a PHIL setup. As mentioned in Section 2.1, the prosumer model features a BHP and HEX$_{\mathrm{DC}}$, controlled in either constant flowrate or temperature difference. For the HEX$_{\mathrm{DC}}$ operation, based on the measured flowrate and temperature, the set return temperature of the house $T_{h,set}$ is calculated based on the heating/cooling system of the building and the building and outdoor temperature. The 3WV mixes water from the supply side to reach $T_{h,set}$. A signal is also provided for the grid pump $\dot{V}_{g,set}$, as explained in Figure 7 found in Section 2.3. For the BHP emulation, the grid pump is still operated according to the control signal $\dot{V}_{g,set}$ but the house side operates differently. The 3WV is closed, so that it doesn't mix water from the supply into the return line and the pump on the building side is operated to supply $\dot{V}_{h,set}$ to achieve the outlet temperature of the heat pump on the grid side.

Further implementations are possible that follow the same principles as the ones mentioned above. These could include multiple HEX connected in series or in parallel to study the interaction of various prosumers. In addition, the balancing unit could be connected in a similar approach to study its characteristics. Even an entire network with multiple prosumers and balancing units could be included as a simulation model on the grid side which would allow for investigating the impact of single/multiple prosumers on larger grids.

5 Discussion

This paper presents a set of models for the development of 5GDHC systems. The models have been developed with a focus on usability, scalability, accuracy, flexibility, and validity. The following sections provide some insight on strengths and limitations as well as a discussion on potential applications of the models.

5.1 Strengths

These components utilise validated models from the ProHMo library that are modular and can provide a detailed representation of component operation and building behaviour. They provide a good rule-based control allowing for BHP operation with low number of starts and stops for a longer component lifetime and a stratified TES. Start up and slew times are included as well as solutions for hysteresis. Computational time is kept low since we are using hydraulic equations only for the network, significantly reducing the complexity of the model. The models are made open access and have platform independent

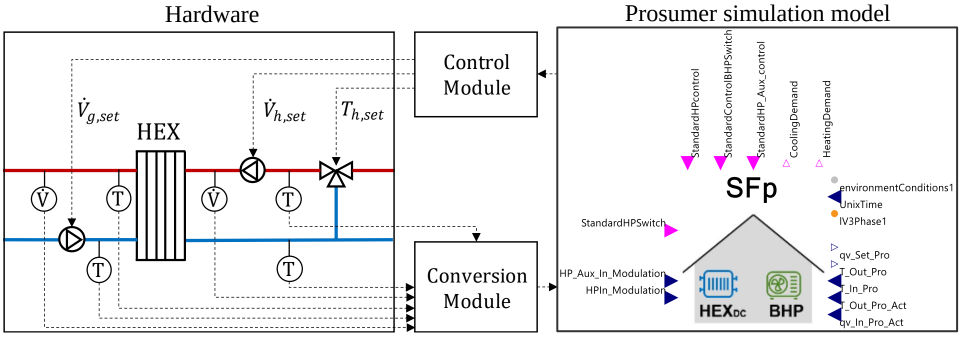


Figure 9. PHIL for a prosumer using a heat exchanger.

FMUs where commercial components are used. They can be coupled with various grid models and elements such as seasonal thermal storage.

Another key benefit that is arising from these models, is the capacity for PHIL experimentations with minimal hardware to study prosumer interaction. The models can be used to emulate both building and BHP/HEX$_{DC}$ behaviour. Different levels of detail for PHIL experiments allow a detailed analysis of grid behaviour and component interaction with low costs, space requirements and overall complexity.

5.2 Limitations

The key limitations of the components come from the use of the ProHMo library. It is built in Simulation X which is not an open access tool. This limits the capacity to freely edit the components. FMU provision has been presented as a workaround, but it does not fully open the “black box” of the component and does not allow for simple drag and drop of the individual components for use on any Modelica environment. The prosumer and balancing unit component models could be integrated into other libraries which are using open access components, while keeping the methodology of their operation intact.

The building models are focused on residential properties and may not accurately represent different consumer classes such as office blocks or retail properties. Moreover, the operational behaviors of the energy transformation components are tied to the units used in the CoSES lab, which are designed for household-scale applications. Consequently, when attempting to model much larger units or units with different technical specifications (e.g., refrigerants), the scalability and accuracy of the models may be compromised.

5.3 Potential Applications

The main benefit of this work is the provision of bespoke models and methodologies that facilitate the studying and analysis of 5GDHC systems. They can act as a basis for the creation of research cases on the impact of several parameters on the overall performance of the system. For example, they could be used to investigate different network topologies and the effect that network behaviour has on the hydraulic operation. The effect of including different consumer classes as prosumers as well as the seasonal co-occurrence of their heating/cooling demands could also be studied. The models could be used to replicate bespoke networks for industrial applications with given building schedules. Detailed operational strategies could also be investigated, identifying the effect of the hydraulic setup on the creation of thermodynamic subcycles and pump hunting phenomena. By developing relevant network and ground models, the effect of the ground type on the network performance can be studied for different insulation levels of the pipework, with a focus on the capacity for thermal losses under different network temperature regimes, insulation series and pipe materials. The impact on the number and location of balancing units as well as the introduction of passive balancing units such as seasonal energy storage (e.g., aquifers) can be quantified. The level of centralisation can also be studied, by changing the consumption parameters, allowing for a deeper investigation of the thermal zoning effect and combination of 4GDH with 4GDC and 5GDHC networks.

6 Conclusions

This paper presents a comprehensive set of Modelica models for the key components of 5GDHC, namely prosumers, balancing units, and hydraulic interfaces.

The component design and assessment, including their interconnections and control strategies, have been discussed and demonstrated through an exemplary use case. The paper has also demonstrated the applicability of PHIL setups for experimental analysis of prosumer interactions with the use of minimal hardware requirements, exemplified through a theoretical case study setup utilizing only a HEX to model a prosumer.

The presented models and methodologies provide an advancement in the understanding and analysis of 5GDHC systems. The provision of FMU models allows for their utilization in various coding environments through FMI, promoting open access as part of the ProHMo library.

Overall, this work contributes to the development of tools and methodologies for the analysis and study of 5GDHC systems, offering potential avenues for future research and application. By further refining and expanding the accessibility of the models, the understanding and adoption of 5GDHC systems can be advanced in a more open and collaborative manner.

Acknowledgments

This work was partially funded by the University of Glasgow, Energy Technology Partnership (ETP) and Ramboll under grant agreement N^{o} ETP 189. The authors would like to thank the Ramboll Foundation for its financial support under grant agreement N^{o} 2020-101 and Ramboll UK Energy Systems business unit for its technical support. The work of Daniel Zinsmeister was funded by the Federal Ministry for Economic Affairs and Climate Action, Germany (FKZ: 03EN3032).

Nomenclature

Abbreviation	*Meaning*
5GDHC	5th Generation District Heating and Cooling
3WV	3-Way Valve
ASHP	Air Source Heat Pump
BHP	Booster Heat Pump
DHW	Domestic Hot Water
EMS	Energy Management System
FMI	Functional Mock-Up Interface

Abbreviation	*Meaning*
FMU	Functional Mock-Up Unit
HEX	Heat Exchanger
HEX_{DC}	Direct Cooling Heat Exchanger
PHIL	Power Hardware In the Loop
SC	Space Cooling
SH	Space Heating
TES	Thermal Energy Storage

References

Abugabbara, Marwan. 2021. *Modelling and Simulation of the Fifth-Generation District Heating and Cooling*. Lunds Universitet. https://portal.research.lu.se/ws/files/98018067/MA_LicDiss.pdf.

Abugabbara, Marwan, Saqib Javed, and Dennis Johansson. 2022. "A Simulation Model for the Design and Analysis of District Systems with Simultaneous Heating and Cooling Demands." *Energy* 261 (December): 125245. https://doi.org/10.1016/j.energy.2022.125245.

Angelidis, Orestis, Anastasia Ioannou, Daniel Friedrich, Alan Thomson, and Gioia Falcone. 2023. "District Heating and Cooling Networks with Decentralised Energy Substations: Opportunities and Barriers for Holistic Energy System Decarbonisation." *Energy* 269 (April): 126740. https://doi.org/10.1016/j.energy.2023.126740.

Blacha, Tobias, Michael Mans, Peter Remmen, and Dirk Mueller. 2019. "Dynamic Simulation Of Bidirectional Low-Temperature Networks - A Case Study To Facilitate The Integration Of Renewable Energies." In , 3491–98. Rome, Italy. https://doi.org/10.26868/25222708.2019.210670.

Buffa, Simone, Marco Cozzini, Matteo D'Antoni, Marco Baratieri, and Roberto Fedrizzi. 2019. "5th Generation District Heating and Cooling Systems: A Review of Existing Cases in Europe." *Renewable and Sustainable Energy Reviews* 104 (April): 504–22. https://doi.org/10.1016/j.rser.2018.12.059.

Chartered Institution of Building Services Engineers (CIBSE). 2020. *CP1 Heat Networks: Code of Practice for the UK (2020)*. 2nd ed. London: Association for Decentralised Energy (ADE). https://www.cibse.org/knowledge/knowledge-items/detail?id=a0q3Y00000IMrmGQAT.

Elizarov, Ilya, and Thomas Licklederer. 2021. "ProsNet – a Modelica Library for Prosumer-Based Heat Networks: Description and Validation." *Journal of Physics: Conference Series* 2042 (1): 012031. https://doi.org/10.1088/1742-6596/2042/1/012031.

Gjoka, Kristian, Behzad Rismanchi, and Robert H. Crawford. 2023. "Fifth-Generation District Heating and Cooling Systems: A Review of Recent Advancements and Implementation Barriers." *Renewable and Sustainable Energy Reviews* 171 (January): 112997. https://doi.org/10.1016/j.rser.2022.112997.

Heijde, Bram van der, M. Fuchs, C. Ribas Tugores, G. Schweiger, K. Sartor, D. Basciotti, D. Müller, C. Nytsch-Geusen, M. Wetter, and L. Helsen. 2017. "Dynamic Equation-Based Thermo-Hydraulic Pipe Model for District Heating and Cooling Systems." *Energy Conversion and Management* 151 (November): 158–69. https://doi.org/10.1016/j.enconman.2017.08.072.

IEA. 2021. "Heating." Tracking Report. Paris. https://www.iea.org/reports/heating.

Mueller, Dirk, Moritz Lauster, Ana Constantin, Marcus Fuchs, and Peter Remmen. 2016. *AixLib – An Open-Source Modelica Library within the IEA-EBC Annex 60 Framework.*

The Modelica Association. 2023. "FMI 3.0 Implementers' Guide." The Modelica Association. https://modelica.github.io/fmi-guides/main/fmi-guide/#_references.

Wetter, Michael, and Christoph van Treeck. 2017. *IEA EBC Annex 60: New Generation Computing Tools for Building and Community Energy Systems*. Annex 60. Lawrence Berkeley National Laboratory and RWTH Aachen University: IEA. https://www.iea-annex60.org/downloads/iea-ebc-annex60-final-report.pdf.

Wetter, Michael, Wangda Zuo, Thierry S. Nouidui, and Xiufeng Pang. 2014. "Modelica Buildings Library." *Journal of Building Performance Simulation* 7 (4): 253–70. https://doi.org/10.1080/19401493.2013.765506.

Wirtz, Marco, Lisa Neumaier, Peter Remmen, and Dirk Müller. 2021. "Temperature Control in 5th Generation District Heating and Cooling Networks: An MILP-Based Operation Optimization." *Applied Energy* 288 (April): 116608. https://doi.org/10.1016/j.apenergy.2021.116608.

Wüllhorst, Fabian, Laura Maier, David Jansen, Larissa Kühn, Dominik Hering, and Dirk Müller. 2022. "BESMod - A Modelica Library Providing Building Energy System Modules." *Modelica Conferences*, 9–18. https://doi.org/10.3384/ECP211869.

Zinsmeister, Daniel, Thomas Licklederer, Stefan Adldinger, Franz Christange, Peter Tzscheutschler, Thomas Hamacher, and Vedran S. Perić. 2023. "A Prosumer-Based Sector-Coupled District Heating and Cooling Laboratory Architecture." *Smart Energy* 9 (February): 100095. https://doi.org/10.1016/j.segy.2023.100095.

Zinsmeister, Daniel, and Vedran Perić. 2022. "Implementation of a Digital Twin of the CoSES District Heating Prosumer Laboratory." In *Energy Proceedings*, 7. Bochum, Germany.

Status of the ClaRa Library: Detailed Transient Simulation of Complex Energy Systems

A. Vojacek[1] J. Brunnemann[1] T. Hanke[1] T. Marx-Schubach[1] J. Eiden[1]

[1]XRG Simulation GmbH, Hamburg, Germany,
{vojacek,brunnemann,hanke,marx_schubach,eiden}@xrg-simulation.de

Abstract

This paper presents the current state of the open-source Modelica library ClaRa, which provides its users with the capability to proficiently tackle tasks in the disciplines of thermal hydraulics, instrumentation and control pertaining to power plants and other kind of energy systems. We provide a comprehensive overview of how the library has successfully broadened its scope over the years of its development, transcending the original focus on conventional power plants to encompass renewable power plants, waste heat utilization, general process plants, refrigeration cycles, heat pumps and beyond. The new version, ClaRa 1.8.1, brings an exciting addition to the already impressive suite of features - support for the utilization of various artificial intelligence models in Modelica simulation tools. Furthermore, the authors unveil ClaRa's ambition to serve as a potential publication platform for third-party models from a steadily growing community of ClaRa users. This is underscored by several application models. Finally, we also describe the funding scheme for maintenance of open source ClaRa by an extended commercial version, ClaRa-Plus.

Keywords: Energy system, ClaRa library, TransiEnt library, Thermal Separation library, Artificial Intelligence, Waste heat, Concentrating solar, Refrigeration cycle, Heat Pump

1 Introduction

1.1 Context of Paper

Climate change is an ongoing and pressing issue, and governments worldwide are taking measures to promote the use of renewable energy sources. However, as the transition to renewable energy takes time, fossil fuels like coal and gas will continue to play a critical role in the world's energy mix for the foreseeable future. Despite this, the proportion of renewable energy sources is growing significantly, underscoring the need for accurate and reliable simulation tools to capture the effects of this transition. Simulation tools enable stakeholders to evaluate the impact of different renewable energy options and assess the feasibility of transitioning to a more sustainable energy system. By leveraging simulation tools capturing the dynamics of the energy system, we can develop efficient and effective strategies that balance the practicality of fossil fuels with the long-term benefits of renewable energy.

Numerous simulation programs are being used in industry today depending on the field, application and desired type of simulation. In the Modelica community, there are several open source libraries available for investigating energy systems in detail, particularly power plant transients such as start-up, shut down, and load change: The ThermoPower (ThermoPower 2023) library, being the first power plant library written in Modelica, the ThermoSysPro (ThermoSysPro 2023) library having a strong industry background, the TRANSFORM (TRANSFORM 2023) focusing on nuclear power plants and the ClaRa (ClaRa 2023) library. ClaRa was developed by a German research collaboration (DYNCAP/DYNSTART 2011-2019)[1] of Hamburg University of Technology, TLK-Thermo GmbH and XRG Simulation GmbH. Its first official release of version 1.0.0 dates from March 2015 (see (Brunnemann, Gottelt, et al. 2012; Gottelt, Wellner, et al. 2012; Gottelt, Hoppe, and L. Nielsen 2017) for an introduction to ClaRa and a control-related application). The aim of ClaRa's Development team was to create a library that is suitable for both beginners and advanced researchers in the field of Modelica simulation. This paper serves as a companion to the ClaRa paper from 2012 (Brunnemann, Gottelt, et al. 2012), introducing several of the most recent enhancements that have been integrated.

1.2 Outline of Paper

This manuscript is organised as follows: Section 2 summarises the scope and structure of the library. In Section 3, we present two novel features of the recent release of ClaRa 1.8.1. The first feature enables the integration of different artificial intelligence models in Modelica simulation tools. The second feature includes new models for investigating generalized thermodynamic cycles, e.g. for refrigeration and heat pumps. The introduction of these new models in ClaRa library expands the range of possible applications, particularly in the area of refrigeration and heat pumps in industries such as automotive, aerospace, and buildings. In Section 4, we introduce the new ClaRa Open Development Repository intended for fast publishing third-party models prior to full ClaRa integration. Two example model, a waste heat in-

[1] funded by the German Ministry for Economic Affairs and Energy under reference number FKZ 03ET2009C/03ET7060D

DOI
10.3384/ecp204617

cinerator and parabolic solar receivers, are shown. Section 5 describes the integration of ClaRa with other Modelica libraries, specifically TransiEnt for simulating coupled energy networks (TransiEnt 2023) and ThermsSeparation (ThermalSeparation 2023) for process engineering. This coupling presents a promising approach to address contemporary energy challenges. Section 6 presents the ClaRa funding scheme of and introduces supplementary content offered in the commercial variant, ClaRaPlus. Finally, the Summary & Outlook can be found section 7.

2 Overview of the Library ClaRa

2.1 Scope of Library

The ClaRa is used for a broad scope of applications and can support all project phases with dynamic simulations: from the evaluation of concept variants to component design, optimization of control technology, virtual commissioning and optimization during operation. The ClaRa is flexible and user friendly and also provides efficient means for simplified steady state analysis (via `ClaRa.StaticCycles` models which one integrates into a dynamic model) for the calculation of consistent initial values for states to smooth the initialisation of complex thermodynamic cycles. The library follows well defined model design principles including level of detail, level of insight, naming conventions, limited inheritance depth in comparison to the Modelica Standard library (MSL 2023), comprehensive documentation, and more (Brunnemann, Gottelt, et al. 2012). For detailed information, please refer to the ClaRa documentation (ClaRa-documentation 2023).

The initial goal of the ClaRa library was to provide models for the analysis of complex conventional power plants with CO_2 capture in both static and dynamic operation mode (DYNCAP/DYNSTART 2011-2019). Consequently, ClaRa has been effectively utilized in multiple projects where a number of digital twin models were created for several lignite-fired power plants, incorporating the complete water-steam cycle, flue gas and air path, coal mills, and a virtual representation of the control technology (H.Prausse et al. 2021), (Marcel Richter 2018), (Brunnemann, Gottelt, et al. 2012), (Gottelt, Wellner, et al. 2012), (Gottelt, Hoppe, and L. Nielsen 2017). In addition to its applications in coal power plants, ClaRa has also been utilized in several commercial projects involving combined-power plants as well as captive power plants with bubbling fluidized bed boilers, and more. Based on the experience gained, it is evident that the established design principles have been effective. As a result, we are now extending the accessibility of ClaRa to the community via (ClaRa-openDevelopment 2023).

The scope of ClaRa's applications extends today far beyond conventional steam power plants towards renewable power plants, waste heat utilization, process plants, generalized thermodynamic cycles, and more. Following the 2012 ClaRa paper (Brunnemann, Gottelt, et al. 2012) that provided an overview of the ClaRa library's status, many new models have been introduced together with improved numerical robustness, initialisation and functionality of the library. For comprehensive details, see the Revisions in ClaRa documentation (ClaRa-documentation 2023).

Regarding the applications from ClaRa community, various references are available. For instance, users have leveraged ClaRa in the design of supercritical CO_2 cycles as a power cycle (Vojacek, Melichar, and al. 2019), heat removal in nuclear power plants (sCO_2-4-NPP 2023), effective bulk energy storage (Kriz, Vlcek, and Frybort 2023), the design of experimental loops for the development of gas-cooled reactors, and small modular reactors cooled with molten salt (Krivsky 2020),(ClaRa User Meeting 2019), and beyond. ClaRa's further noteworthy reference is its role in the Future Energy Solution (FES) storage system, utilizing volcanic rock to store electricity generated from renewable energy, where ClaRa played a crucial role in modelling and analysing its performance, see (Heyde, Schmitz, and Brunnemann 2021). Additional applications of ClaRa are explicated in section 4.

2.2 Structure of Library

ClaRa library comes in a bundle (Table 1) of three libraries, ClaRa (Core), TILMedia (Thermo-physical properties) and SMArtIInt (AI).

Table 2 gives an overview of the top level content of the ClaRa library covering a diverse range of physics with well-structured and user-friendly architecture. The library adopts a functional approach to its structure, organizing components based on their functionality rather than the specific medium being modeled. As an illustration, a pipe model for vapor-liquid equilibrium can be found within the same package, `Components.VolumesValvesFittings.Pipes`, alongside models for gas pipes.

Table 1. ClaRa bundle

TILMedia	Thermo-physical properties of incompressible liquids, ideal gases as well as real fluids containing a vapor liquid equilibrium.
ClaRa	"The core" of the bundle supplying models within the fields of thermal hydraulics and instrumentation and control
SMArtIInt	Support SMArtIInt (SMArtIInt 2023) for usage of (trained) TensorFlow (TensorFlow 2023) models from within Modelica.

Table 2. ClaRa library structure

ClaRa	
UsersGuide	Information on basic modelling concepts, revisions, contact, license
Examples	Introducing examples showing different levels of complexity and library capabilities.
Basics	Basic models providing fundamentals to all models contained in ClaRa.
Components	Models for turbo machines and electrical machines, connection pipings, heat exchanger, mass storage and steam separation, valves, coal grinding, furnace, flue gas cleaning, and sensors, i.e. "the core of the library"
SubSystems	Conceptual package containing models of increasing complexity which are based on each other
Visualisation	Elements for displaying and plotting of dynamic simulation data.
StaticCycles	Static models for the calculation of consistent initial guess values or conceptual design purposes
SimCenter	A top level model which is mandatory for every complex ClaRa simulation. It provides global settings such as the media models etc.

3 Latest Features of ClaRa

3.1 ClaRaDelay now Open Source

Since its first release the ClaRa library comes with ClaRaDelay (ClaRaDelay 2023), an extended version of the Modelica delay operator `ModelicaReference.Operators.delay()`. The latter can be used in order to read past values of an expression from a buffer during simulation. While the original Modelica implementation only allows reading of a single past time value per buffer, ClaRaDelay takes a vector of past times as input and gives a vector of the according past values as output. This is particularly useful for numerical evaluation of convolution integrals during simulation, a feature that is used in ClaRa's implementation of a transmission line pipe `PipeFlowVLE_L1_TML`. In order to foster application and discussion with the Modelica community ClaRaDelay is now available open source.

3.2 Support for Hybrid Modelling

With the ongoing advances of machine learning and the availability of operational data there is a growing user interest in using trained neural networks from within Modelica and ClaRa. Combining data driven and a physics based model parts in a system model results in what is called a 'hybrid model': internally both parts exchange data through an internal communication interface ('com' see figure 1) and share the overall model parameters and outside connectors. This combination offers new perspectives for system simulation that go beyond 'traditional' characteristic fields (feed forward neural networks) or response surfaces (recurrent neural networks): complex correlations inside data can be directly derived from the data and encoded by the network parameters. Especially for high dimensional data fields this can be much more memory efficient than by using conventional interpolation methods (Chahrour and Wells 2022). Moreover neural networks can be applied in order to increase the level of detail offered by system models at low performance costs, e.g. spatially resolved temperature profiles or velocity distributions.

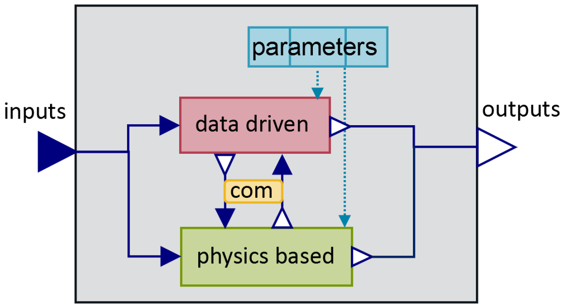


Figure 1. Hybrid Modeling

The Modelica library SMArtIInt (SMArtIInt 2023) addresses this issue and ships with ClaRa from version 1.8.1. Currently it supports tensorFlow (TensorFlow 2023) models for both feed forward and recurrent neural networks (RNN, stateful and stateless), using ClaRaDelay (ClaRaDelay 2023). Being open source the intention is to further develop SMArtIInt to support more network formats, architectures and Modelica tools by means of the user community. The library was created within the DIZPROVI (2021 - 2024) research project [2].

SMArtIInt allows the import of pre-trained neural networks into Modelica using external C-functions inside a Modelica block, similarly to the tables blocks available in the Modelica standard library. Figure 3 shows an example for a super heater boiler section of a conventional coal fired power plant taken from (Brunnemann, Kolter-

[2] funded by the German Federal Ministry of Education and Research under reference number FKZ 03WIR0105E

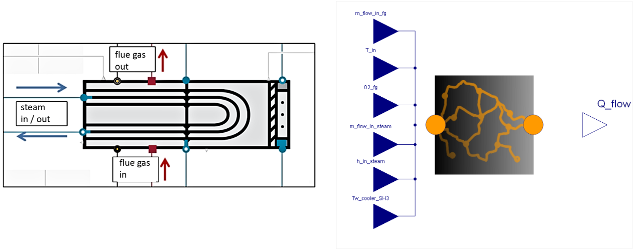


Figure 2. Usage of SMArtIInt: Left: physics based model of a super heater. Right: diagram of the build in data based heat transfer correlation.

mann, et al. 2022). The heat transfer Q_{flow} between hot flue gas and the water steam side is computed by a neural network that was trained on measurement data. It takes inputs from the surrounding physics based system model. Figure 3 shows a comparison of obtained heat flow rates from a physics based correlation ($Q_{flow,sim}$) and the data driven $Q_{flow,sim,AI}$ to the measurement $Q_{flow,meas}$ during a load change of the plant.

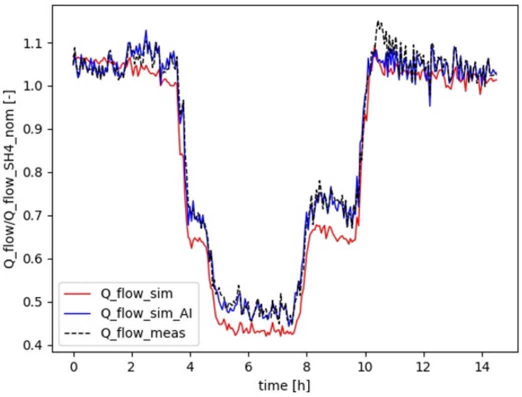


Figure 3. Hybrid Modeling from (Brunnemann, Koltermann, et al. 2022)

Hybrid models certainly offer exciting new perspectives for system simulation. However identifying the optimal combination of physics based and data driven model part (see Figure 1) can be a challenging trade-off between measurement data from the real plant and abstracted model assumptions. Special care has to be taken in consistent validity range of data driven models build into physics based system models: firstly, the inputs may be driven outside the trained ranges by the system dynamics, e.g. when it comes to non-standard operation or control feedback loops. Secondly solvers with variable time steps can may ask for a step that takes the inputs outside their range, possibly causing time steps to be accepted that would be rejected for the true correlation encoded in the data driven model.

Additionally stateful RNNs need to be sampled at the time interval they have bee trained with. This is undesirable, as it would slow down simulations using variable time step solvers. SMArtIInt addresses this issue by evolving the RNN inside a container in the external C-code, hidden from the solver. If the Modelica solver proposes a time step larger than the trained step size of the RNN, then inside the container the RNN is evolved with input values that are equidistantly interpolated between the current time step and the proposed time step with the time step compatible to the RNN.

The functionalities of SMArtIInt are currently under active testing and research and we invite the ClaRa community to experiment with it.

3.3 Generalized Application of Thermodynamic Cycles

The recent release of ClaRa, which includes models for commonly designed condensers and evaporators (a flat tube finned HX and plate HX), enables Clara users to model and analyse different thermodynamic cycles such as refrigeration cycles and heat pumps.

To showcase the capabilities of ClaRa, a simple refrigeration cycle was modelled, consisting of a compressor, a flat tube finned plate air-cooled condenser, separator, expansion valve, and water-heated plate evaporator/chiller [3]. The results are visualized in Figure 4. Regarding numerical performance, the ClaRa model is much similar to other Modelica libraries in the field such as AirConditioning (2023) or TIL (2023). These initial findings and experiences demonstrate that ClaRa is capable of delivering accurate and reliable results. This positions ClaRa as a viable alternative for modelling refrigeration cycles, alongside other Modelica libraries in that field. We invite the community to test such applications.

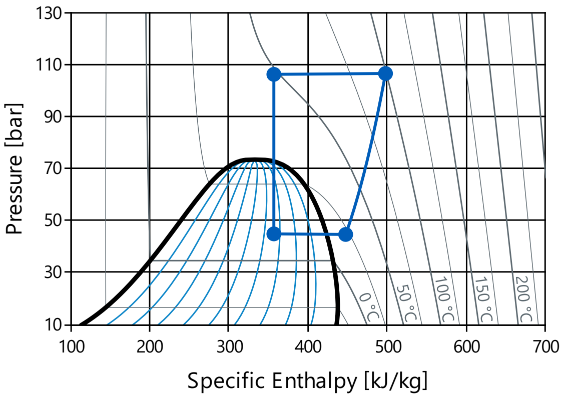


Figure 4. Refrigeration cycle with CO_2 modelled in ClaRa.

4 ClaRa Open Development Repository

The ClaRa Development team has set a new objective to establish the ClaRa library as a potential publication platform for third-party models originating from the

[3] Similar model can be found in `ClaRa.Examples.VapourCycle_01`

ever-expanding ClaRa users' community. To achieve this, there is now a dedicated Git repository (ClaRa-openDevelopment 2023), intended for model publication prior to full integration into the official ClaRa library. To ensure optimal usability, there are some minimum requirements to these new models: they should contain introductory functional examples, declarative comments, well structured parameter dialogues and model diagrams as well as a self-explanatory and consistent nomenclature, good source code transparency and at least basic documentation. Following a comprehensive quality check by the ClaRa development team, models and their corresponding documentation can be considered for integration into the official ClaRa release if there is sufficient user interest.

To make sure everyone is protected and ClaRa can keep growing, all contributors must sign a Contributor License Agreement (CLA 2022). Basically, it means that contributors give the ClaRa Development team (CDT) permission to publish their work in the ClaRa library, currently under 3-clause-BSD License. In return, the CDT assures to always make the contributions available as open source in ClaRa. A few models have already been included in the official release of ClaRa, thanks to contributions from different individuals. Specifically, there are models from (FVTR 2021) sponsored by LEAG (LEAG 2021), as well as from the Future Energy Solution (FES) storage system project (Heyde, Schmitz, and Brunnemann 2021).

In this chapter, we further demonstrate our objective through the presentation of various application models that are vital for sustainable energy solutions. The first model focuses on waste heat incineration, highlighting its effectiveness in harnessing valuable energy resources while minimizing environmental impact. The second model demonstrates the use of renewable sources, such as solar energy, which plays a crucial role in achieving an environmentally friendly energy mix and driving the transition towards a cleaner and more sustainable future. All of the models shown in Section 4, as well as their basic equations, can be found on the ClaRa Open Development Repository (ClaRa-openDevelopment 2023).

4.1 Waste Heat Incinerator

A part of the ClaRa Open Development Repository is the model of a grate combustion system, which was developed as part of a master's thesis (Gulba 2019). Such grate combustion systems find application in the incineration of biomass and waste materials. The combustion process within such a grate combustion system can be conceptually divided into four stages. Initially, the untreated fuel undergoes a heating process that leads to evaporation of its water content during the drying phase. Temperatures exceeding 300 °C result in the release of volatile components and chemical transformations of certain fuel constituents (pyrolysis). The ensuing gases subsequently react with oxygen present in the flue gas. Following the pyrolysis stage, solid carbon (coke) reacts with oxygen, leading to the formation of CO and CO2. The remaining inert components, such as ash and slag undergo cooling through interaction with the flow of cooler incoming air.

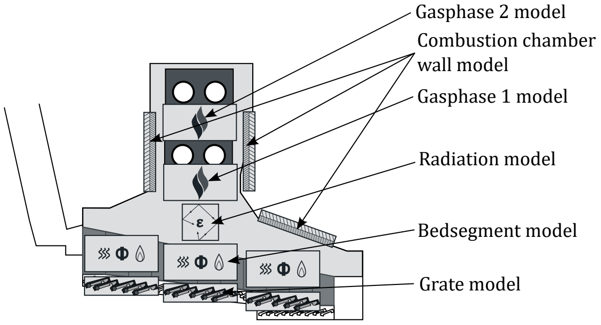


Figure 5. Modeling of the combustion chamber - representation and spatial assignment of all submodels

The in Figure 5 shown combustion chamber is divided into the components of grate, fuel bed, combustion chamber volume, chamber walls, and radiation model. The grate and fuel bed are discretized, consisting of multiple individual elements that are interconnected. Figure 6 provides a schematic representation of the processes occurring within the fuel bed segment. In this illustration, dark red arrows symbolize heat transfer exclusively, while all other arrows represent mass transfer as well as the associated energy transfer.

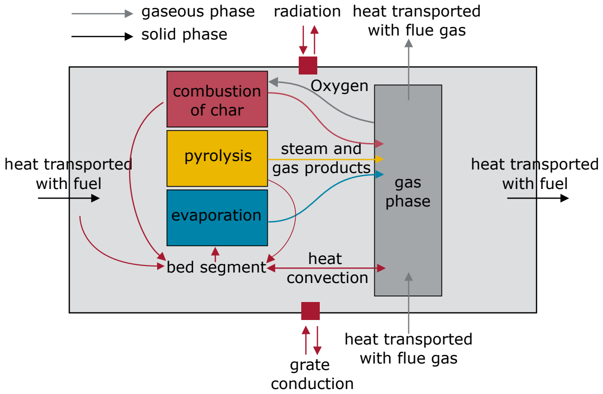


Figure 6. Schematic representation of the bed segment model with heat and material flows

The processes of evaporation and pyrolysis are endothermic processes that require energy to occur. This required energy is extracted from the fuel bed, as indicated by the representation of dark red arrows. On the other hand, residual coke combustion is an exothermic process that releases energy. It takes place in the presence of oxygen, which must be supplied from the gas phase. During evaporation and pyrolysis, there is no requirement for material transport from the gas phase to the solid phase. Instead, gaseous products generated in these processes directly transition into the gas phase. Additionally, the material flows transport energy in the form of heat from one phase to another. The drying and pyrolysis models employed in the fuel bed segment model are based on (R. J.

Nielsen et al. 2018). On the other hand, the coke combustion model and the gas phase model were developed within the scope of the master's thesis (Gulba 2019). Furthermore, the radiation model utilized in the combustion chamber model is based on the approach of (R. J. Nielsen et al. 2018). The fuel model, grate model, and model of the combustion chamber walls all consist of components of the ClaRa library. The combustion chamber model expands Nielsen's model by incorporating temperature-dependent reaction rates for both residual coke combustion and pyrolysis gas combustion. This allows for the consideration of activation energies for the different reactions.

For this combustion chamber model, two different cases are considered below. First, a start-up process and second, a change of state in steady-state operation. For the investigation of these scenarios, both the bed and the grate model are discretized into 10 segments. All of these segments are connected to the gas phase 1, allowing for gas exchange between the fuel bed and the gas phase. The gas phase 1 is also connected to gas phase 2, where additional secondary combustion air is supplied.

The startup process of the system shows that initially, the vaporization of the fuel primarily occurs in the three rear segments (8, 9, 10) (Fig. 7). This is due to the external heat supply from an auxiliary burner in these segments. Despite the externally added heat, temperatures and thus reaction rates remain low due to necessary drying of the initially wet fuel in these segments. The temperature profile in Figure 8 demonstrates that combustion of the dried fuel starts in segment 10. As heat generation increases, the drying of the fuel and therefore the coke combustion shifts to preceding segments. Steady state is reached after more than one hour, with fuel drying in the first four segments and the substantial combustion occurring in segments 5 and 6. In the second scenario, the primary air (grate air) is reduced after 2.5 hours, followed by a reduction in secondary air in the gas phase after 3 hours. These changes initially affect the gas phase 1. Due to the lack of air, there is insufficient oxygen to convert all volatile components (H2, CH4, CO). As a result, less energy is released in gas phase 1. Meanwhile, more energy is released in gas phase 2 as the secondary air reacts with the volatile components (Fig. 9). However, with the reduction of secondary air, the energy released in gas phase 2 also decreases. These changes in reaction energetics affect the gas phase temperatures and the heat transferred to the segments through radiation (Fig. 10). The reduced heat transfer to the front segments causes a shift in the drying and pyrolysis process to later segments. Consequently, the coke combustion in the bed segment is also shifted from segments 5 and 6 to segments 6, 7, and 8. This shift is evident in both temperature profiles and heat radiation in the figures 8 and 10. Prior to the state change, segments 5 and 6 released heat to the gas phase. However, after the adjustment in the air supply, segment 5 absorbs heat from the gas phase, while segment 6 releases significantly less heat. Conversely, segments 7 and 8 initially absorb heat and then release the heat generated during combustion after the change in air supply.

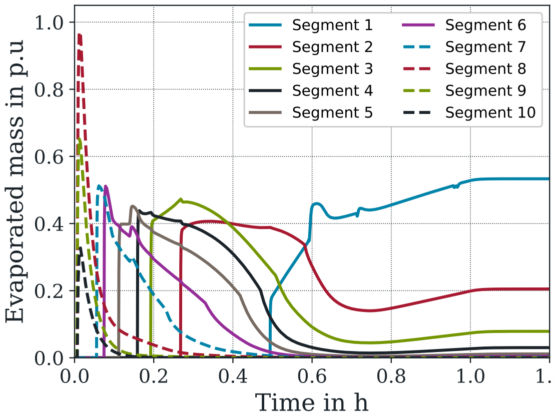


Figure 7. Amount of water evaporated from the fuel in the individual segments

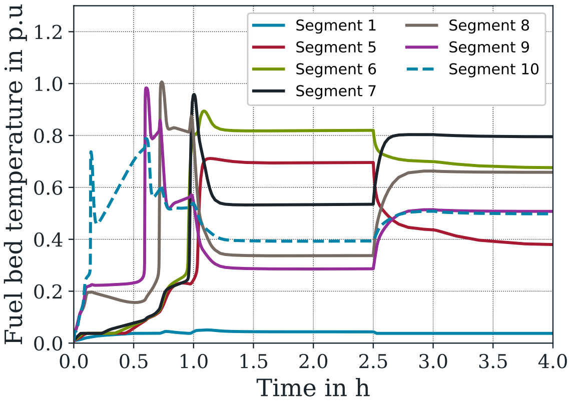


Figure 8. Fuel bed temperature in the individual segments

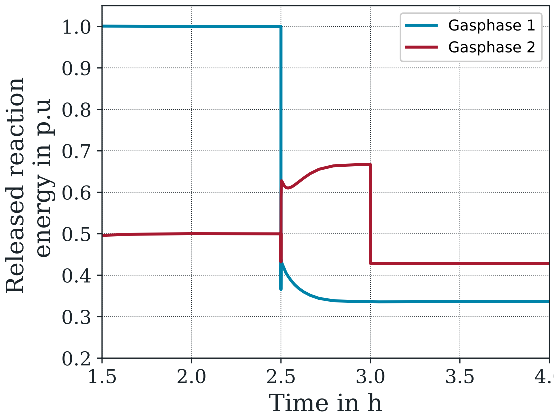


Figure 9. Energy released during the reaction in the gas phase

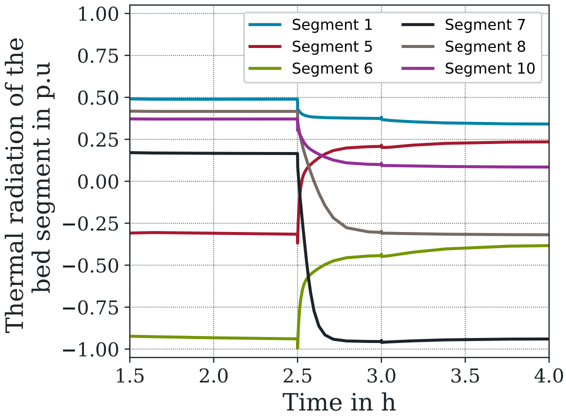


Figure 10. Thermal radiation between the individual segments and the gas phase

4.2 Concentrating Solar Thermal (CST) Power Plants

The Concentrating Solar Thermal (CST) power plants are an important part of the ClaRa Open Development Repository. Concentrated solar power (CSP) technologies utilize concentrated solar radiation to heat a fluid, which drives a heat engine to generate electricity through a generator. The CST power plant models integrated into the ClaRa Open Development Repository are currently based on the principle of the directly evaporating parabolic trough power plant. The collector model combines a tube and a tube wall model from the ClaRa library with a parabolic mirror model that was developed as part of a master's thesis (Hoppe 2013). A collector model realized in this way is shown in Figure 11.

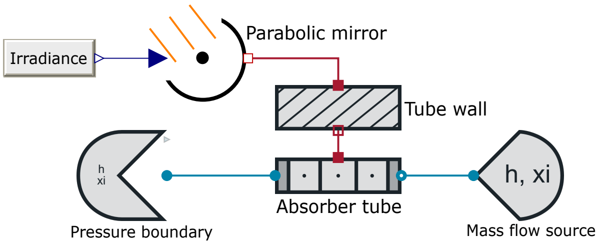


Figure 11. Collector model

The parabolic mirror model calculates a heat flux to be passed to the pipe wall from a given irradiance. Two different parabolic mirror models exist: a simplified and a detailed model. The simplified model disregards the position of the sun and the orientation of the collector. It is used to investigate short-term disturbance situations where no fundamental changes in sun position occur. The detailed parabolic mirror model considers sun position and the orientation of the collector in its heat flux calculation and can be used to investigate whole day operation cycles.

The collector models can be used in combination with ClaRa's water-steam cycle components to simulate complete parabolic trough power plants such as shown in figure 12. The model of this solar thermal power plant operates according to the principle of direct evaporation and uses water as a heat transfer medium. The collector field consists of 6 parallel evaporator rows and 3 parallel superheater rows. Two different operation scenarios are investigated for the model described above.

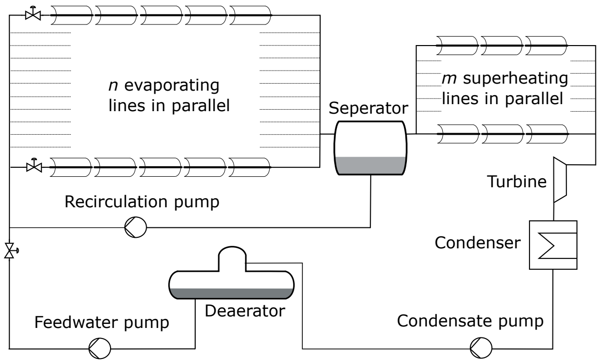


Figure 12. Schematic CST power plant

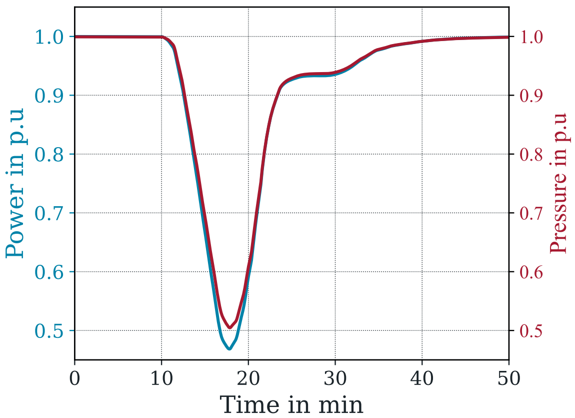


Figure 13. Turbine power and steam pressure at turbine inlet

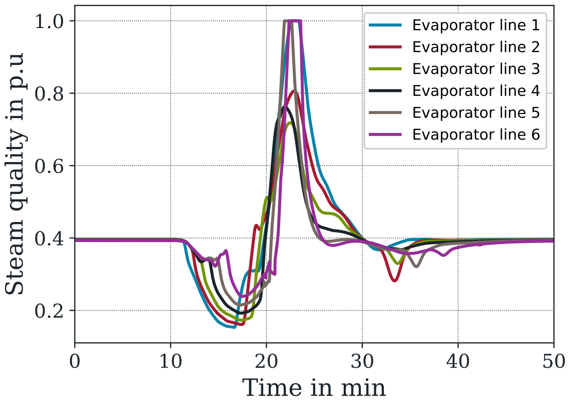


Figure 14. Tube steam quality outlet

In the first scenario, a cloud front passes over the collector field after 10 minutes, resulting in a decrease in solar irradiation on each individual collector. This causes a pressure drop at the turbine inlet, leading to a reduction in the mechanical power output of the turbine (as shown in Figure 13). Figure 14 illustrates the steam quality at the outlet of each evaporator line. Initially, the steam quality decreases due to the reduced irradiation. Consequently, the power plant reduces the mass flow rate through the individual evaporator lines, which causes the steam quality to rise again. Once the disruption ends after 20 minutes, the water-steam cycle's inertia leads to a sharp increase in the steam quality before reaching a steady-state level once again.

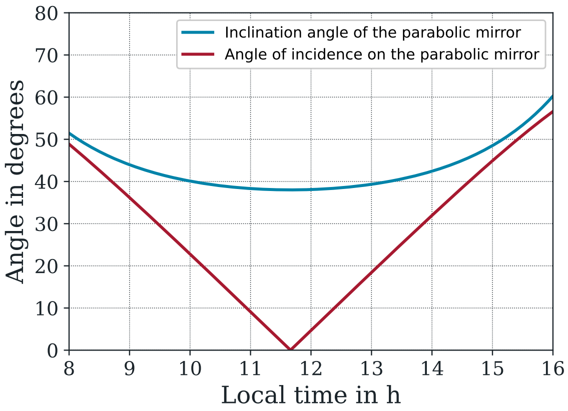


Figure 15. Angle of inclination of the parabolic mirror and angle of incidence on parabolic mirror

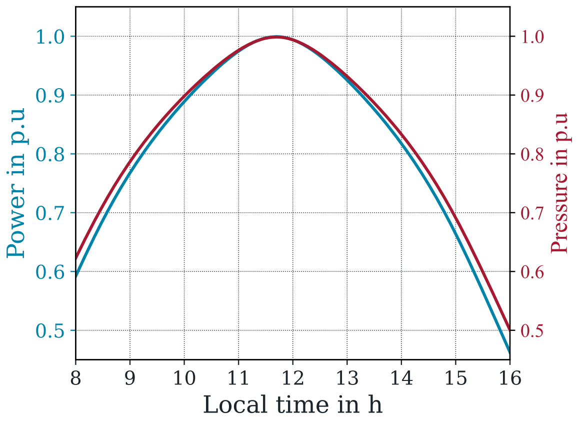


Figure 16. Turbine power and steam pressure at turbine inlet

The second scenario examines how the daily cycle of the sun affects the CST power plant. Figure 15 illustrates the behavior of the parabolic mirrors. The collector field is oriented parallel to the east-west axis, leading to an incidence angle of 0° when the sun is in the south. The parabolic mirrors are then tilted in complement to the sun's elevation angle. Figure 16 displays the resulting inlet pressure and the mechanical power output of the turbine. The detailed model is capable of simulating various locations, seasons, and orientations of the collector field, making it possible to support analyses of expected yield.

5 Combining ClaRa with other Modelica Libraries

Beside adapters to fluid components of the Modelica Standard library (MSL 2023) and the ThermoPower library (ThermoPower 2023) ClaRa's range of applicability is further broadened by its integration with other Modelica libraries.

5.1 TransiEnt Library

The TransiEnt Modelica library (TransiEnt 2023) aims at studying complex sector coupled energy systems on a scale from single settlements (Schindhelm et al. 2021), city districts (Benthin et al. 2019) or entire geographic regions (Senkel et al. 2022) and was awarded as one of the best submitted Modelica libraries at the previous Modelica conference (Senkel et al. 2021).

ClaRa models provide the foundation of the TransiEnt packages for heating and gas grid through code inheritance in basic components such as connectors, control volumes or components such as pipes, fans or sensors. Moreover TransiEnt's global model settings (SimCenter) as well as ModelStatistics inherit code from the according ClaRa componets. Hence TransiEnt user models contain also code from ClaRa and there is a deep interconnection between the two libraries. Consequently TransiEnt ships in a bundle with ClaRa.

5.2 Thermal Separation Library

The Thermal Separation library (ThermalSeparation 2023) is an open-source Modelica library designed for simulating thermal separation processes, including rectification and absorption processes. The library primarily focuses on simulating amine scrubbing processes, which are utilized for capturing CO_2 from flue gases emitted by various sources such as biogas or thermal power plants.

Models of the Thermal Separation library can be coupled to ClaRa-library. Adapters for linking both libraries can be provided upon request. This enables the ClaRa users to simulate coupled combustion + carbon capture processes in order to understand the dynamic behaviour or test suitable control schemes or evaluate the potential of carbon capture and storage (CCS) strategies.

Detailed results of the coupled operation of a coal fired power plant with post-combustion carbon capture can be found in (Wellner, Marx-Schubach, and Schmitz 2016) and (Marx-Schubach and Schmitz 2019).

6 The ClaRa Funding Scheme: Sustaining Open-Source Development

ClaRa has been and will be an open source project with regular updates and releases. In order to make its ongoing development financially sustainable it comes with an an extended commercial companion version called ClaRaPlus (ClaRaPlus 2023) since 2017.

ClaRaPlus offers access to a wider variety of models and components that enable more advanced simulation capabilities, such as instrumentation and control very close to modern power plants, in the ClaRa_DCS module, as well as modelling interactions between power plants and electric grids using the ClaRa_Grid module. Additionally, ClaRaPlus provides advanced pump and fan models (defined in all operational modes - 4 Quadrant), which enable the investigation of detailed start-up and shut-down as well as abnormal operations such as pump trip, pressure shocks, and beyond. ClaRa (ClaRaPlus 2023) provides a detailed comparison between the two libraries and highlights the differences in features and capabilities offered by ClaRa and ClaRaPlus.

Additionally, ClaRaPlus offers technical support and maintenance services, which can be valuable for users who require assistance with the implementation and use of the library. Ultimately, the choice between ClaRa and ClaRaPlus will depend on the specific needs and requirements of each user or organization. The ClaRaPlus is available at (LTX Simulation GmbH 2023) and alongside Dymola via (Dassault Systèmes 2023).

Also sponsorship (models or money) is warmly welcomed and greatly appreciated, as it plays a vital role in ensuring the long-term viability of the open-source ClaRa initiative.

7 Summary & Outlook

We are pleased to announce the latest functionalities in ClaRa, the open-source Modelica library that continues to push the boundaries of simulation technology.

With the recent release, users of ClaRa can now seamlessly integrate AI models which opens up a world of possibilities in the realm of energy system analysis, enabling users to harness the power of machine learning, deep learning, and other AI techniques to gain deeper insights into complex systems. Further, the introduction of new models in the ClaRa Open Development Repository such as waste heat incinerator, parabolic solar receiver, evaporators/condensers for refrigeration cycles and heat pumps, expands the range of possible applications.

We look forward to seeing new community contributions and sponsorships in the third-party ClaRa Open Developemnt Repository that unlocks new possibilities. We are excited to continue supporting our users in their quest for innovation across a wide range of industries, including renewable energy, process engineering, building management, and many others.

Currently, Dymola remains the primary development environment for the ClaRa library. However, we are committed to ensuring its compatibility with OpenModelica (OpenModelica 2023) and have made significant progress in this regard thanks to the OpenModelica development team. More than half of the examples can now run in OpenModelica (OM-ClaRa-Suport 2023). At the moment, OpenModelica and Dymola handle symbolic manipulations differently. The current approach used in OpenModelica leads to numerically unstable models, as seen in cases such as discretized pipe models. We will continue our efforts to expand this compatibility further.

Acknowledgements

This work has been partially supported by the German Federal Ministry of Education and Research within the DIZPROVI research project under grant number 03WIR0105E.

SPONSORED BY THE
Federal Ministry
of Education
and Research

The authors thank the anonymous referees for valuable comments and suggestions.

References

AirConditioning (2023). Ed. by Modelon. version 1.24, fetched May, 2nd 2023. URL: https://modelon.com/library/air-conditioning-library/.

Benthin, J. et al. (2019). " Demand oriented Modelling of coupled Energy Grids". In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany*. URL: https://doi.org/10.3384/ecp19157.

Brunnemann, Johannes, Friedrich Gottelt, et al. (2012). "Status of ClaRaCCS: Modelling and Simulation of Coal-Fired Power Plants with CO_2 Capture". In: *Proceedings of the 9th International Modelica Conference, Munich, Germany*, pp. 609–618.

Brunnemann, Johannes, Jan Koltermann, et al. (2022). "Maschinelles Lernen zur vereinfachten Erstellung und Wartung Hybrider Digitaler Zwillinge (Machine Learning for Simplification of Creation and Maintenance of Hybrid Digital Twin Models)". In: *Proceedings of the 24th Colloquium on Power Plant Technology, Dresden, Germany*. in German.

Chahrour, Ibrahim and James D. Wells (2022). "Comparing machine learning and interpolation methods for loop-level calculations". In: *SciPost Phys.* 12, p. 187. DOI: 10.21468/SciPostPhys.12.6.187. URL: https://scipost.org/10.21468/SciPostPhys.12.6.187.

CLA (2022). Ed. by ClaRa Development Team. version 1.1, fetched May, 2nd 2023. URL: https://claralib.com/pdf/CLA.pdf.

ClaRa (2023). Ed. by ClaRa Development Team. version 1.8.1, fetched June, 9th 2023. URL: https://github.com/xrg-simulation/ClaRa-official.

ClaRa User Meeting (2019). Ed. by ClaRa. fetched May, 2nd 2023. URL: https://www.claralib.com/pdf/ClaRa_UserMeeting_2019.zip.

ClaRa-documentation (2023). Ed. by ClaRa Development Team. version 1.8.1, fetched June, 9th 2023. URL: https://github.com/xrg-simulation/ClaRa-documentation.

ClaRa-openDevelopment (2023). Ed. by XRG Simulation. fetched June, 9th 2023. URL: https://www.github.com/xrg-simulation/ClaRa-openDevelopment.

OM-ClaRa-Suport (2023). Nightly build check of ClaRa compatibility, fetched August, 3rd 2023. URL: https://libraries.openmodelica.org/branches/master/ClaRa_dev/ClaRa_dev.html.

ClaRaDelay (2023). Ed. by XRG Simulation GmbH. fetched June, 6th 2023. URL: https://github.com/xrg-simulation/ClaRaDelay.

ClaRaPlus (2023). Ed. by ClaRa Development Team. version 1.7.1, fetched June, 9th 2023. URL: https://claralib.com.

Dassault Systèmes (2023). Ed. by DS. fetched May, 2nd 2023. URL: https://www.3ds.com/products-services/catia/products/dymola/.

FVTR (2021). Ed. by Forschungszentrum für Verbrennungsmotoren und Thermodynamik Rostock GmbH. fetched May, 2nd 2023. URL: https://fvtr.de/.

Gottelt, Friedrich, Timm Hoppe, and Lasse Nielsen (2017). "Applying the Power Plant Library ClaRa for Control Optimisation". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic*, pp. 867–877.

Gottelt, Friedrich, Kai Wellner, et al. (2012). "A Unifieded Control Scheme for Coal-Fired Power Plants with Integrated Post Combustion CO_2 Capture". In: *Proceedings of the In 8th IFAC Conference on Power Plant & Power System Control, Toulouse, France*.

Gulba, Milan (2019). "Erstellung eines Modells einer Rostfeuerung für die ClaRa-Bibliothek mit Modelica zur Simulation des dynamischen Kraftwerkverhaltens". in German. MA thesis. Hamburg University of Technology, Germany.

H.Prausse, Jens et al. (2021). "Evaluation of the effectsof a load shedding at a lignite power plant". In: *Energy Procedia*.

Heyde, Michael, Gerhard Schmitz, and Johannes Brunnemann (2021). "Abschlussbericht zum Teilvorhaben: Detaillierte Untersuchung der Thermofluiddynamik eines heterogenen thermischen Schüttgutspeichers und dessen Bewertung im Einsatz in Energieversorgungssystemen im Verbundvorhaben: Future Energy Solution (FES) - Entwicklung eines kostengünstigen Massenenergiespeichers für die Erneuerbaren Energien". In: *Leibniz-Informationszentrum Technik und Naturwissenschaften und Universitätsbibliothek*. in German. eprint: https://doi.org/10.2314/KXP:1772161624.

Hoppe, Timm (2013). "Simulative Untersuchung der Dynamik des Parabolrinnenkollektorfeldes eines solarthermischen Kraftwerks mit Direktverdampfung". in German. MA thesis. Hamburg University of Technology, Germany.

Krivsky, Tomas (2020). "Thermohydraulic model of S-ALLEGRO loop". In: *CTU Prague*. eprint: https://dspace.cvut.cz/handle/10467/89749.

Kriz, Daniel, Petr Vlcek, and Otakar Frybort (2023). "System analysis of experimental sCO2 cycle Sofia". In: *5th European Conference on Supercritical CO2 (sCO2) Power Systems, Prague, Czech Republic*.

LEAG (2021). Ed. by Lausitz Energie Kraftwerke AG. fetched May, 2nd 2023. URL: https://www.leag.de/.

LTX Simulation GmbH (2023). Ed. by LTX. fetched May, 2nd 2023. URL: https://www.ltx.de/.

Marcel Richter Gerd Oeljeklaus, Klaus Görner: (2018). "Improving the load flexibility of coal-fired power plants by the integration of a thermal energy storage". In: *Applied Energy*, pp. 607–621.

Marx-Schubach, Thomas and Gerhard Schmitz (2019). "Modeling and simulation of the start-up process of coal fired power plants with post-combustion CO2 capture". In: *International Journal of Greenhouse Gas Control* 87, pp. 44–57. ISSN: 1750-5836. DOI: https://doi.org/10.1016/j.ijggc.2019.05.003. URL: https://www.sciencedirect.com/science/article/pii/S1750583618307357.

MSL (2023). version 4.0, fetched Agust, 3rd 2023. URL: https://github.com/modelica/ModelicaStandardLibrary.

Nielsen, René Just et al. (2018). *Grate Boiler Modeling for Soft Sensor Based Control*. Tech. rep. 2018:4901. Energiforsk AB.

OpenModelica (2023). Ed. by OpenModelica. version 1.21.0, fetched May, 2nd 2023. URL: https://openmodelica.org/.

Schindhelm, L. et al. (2021). "Long Term Technical and Economic Evaluation of Hydrogen Storage Technologies for Energy Autarkic Residential Complexes ". In: *Proceedings of the 14th International Modelica Conference, Linköping, Sweden*. URL: https://doi.org/10.3384/ecp21181.

sCO_2-4-NPP (2023). Ed. by sCO_2-4-NPP. fetched May, 2nd 2023. URL: https://www.sco2-4-npp.eu/.

Senkel, A. et al. (2021). "Status of the TransiEnt Library: Transient Simulation of Complex Integrated Energy Systems". In: *Proceedings of the 14th International Modelica Conference, Linköping, Sweden*. URL: doi:%2010.3384/ecp21181187.

Senkel, A. et al. (2022). "Investigation of Dynamic Interactions in Integrated Energy Systems". In: *Working paper, Hamburg University of Technology, Germany*. URL: https://doi.org/doi:%2010.15480/882.4678.

SMArtIInt (2023). Ed. by XRG Simulation GmbH. version 0.1.0, fetched June, 6th 2023. URL: https://github.com/xrg-simulation/SMArtIInt.

TensorFlow (2023). Ed. by tensorflow. fetched June, 6th 2023. URL: https://github.com/tensorflow.

ThermalSeparation (2023). Ed. by Thomas Marx-Schubach. fetched June, 8th 2023. URL: https://www.github.com/xrg-simulation/ThermalSeparation.

ThermoPower (2023). Ed. by F.Casella. version 3.1, fetched May, 2nd 2023. URL: https://casella.github.io/ThermoPower/.

ThermoSysPro (2023). Ed. by EDF. version 4.0, fetched May, 2nd 2023. URL: https://thermosyspro.com/.

TIL (2023). Ed. by TLK Thermo GmbH. version 3.13.0, fetched May, 2nd 2023. URL: https://www.tlk-thermo.com/index.php/en/software/38-til-suite.

TRANSFORM (2023). Ed. by Greenwood, M. S. fetched June, 9th 2023. URL: https://www.github.com/ORNL-Modelica/TRANSFORM-Library.

TransiEnt (2023). Ed. by TransiEnt Development Team. fetched May, 2nd 2023. URL: https://github.com/TransiEnt-official/transient-lib.

Vojacek, Ales, Tomas Melichar, and Petr Hajek et al. (2019). "Experimental investigations and simulations of the control system in supercritical CO2 loop". In: *3rd European Conference on Supercritical CO2 (sCO2) Power Systems, Paris, France*.

Wellner, Kai, Thomas Marx-Schubach, and Gerhard Schmitz (2016). "Dynamic Behavior of Coal-Fired Power Plants with Postcombustion CO2 Capture". In: *Industrial & Engineering Chemistry Research* 55.46, pp. 12038–12045. DOI: 10.1021/acs.iecr.6b02752. eprint: https://doi.org/10.1021/acs.iecr.6b02752. URL: https://doi.org/10.1021/acs.iecr.6b02752.

Open-Source Models for Sand-Based Thermal Energy Storage in Heating Applications

Kathryn Hinkelman[1] David Milner[2] Wangda Zuo[1]

[1]Architectural Engineering, Pennsylvania State University, USA, {khinkelman,wangda.zuo}@psu.edu
[2]Civil, Environmental and Architectural Engineering, University of Colorado, USA, dmilner@colorado.edu

Abstract

This paper presents a new open-source modeling package in the Modelica language for particle-based silica-sand thermal energy storage (TES) in heating applications, available at https://github.com/sbslab/modelica-sand. Silica sand is an abundant, low-cost, and efficient storage medium for concentrated solar power and electricity generation. Although uncommon today, solid particle TES could benefit building and district heating systems, particularly as building electrification and renewable energy penetration increases. To enable heating system design and evaluation with sand TES, this work developed and open-source released Modelica models from base classes through complete systems with both physical equipment and controls. This paper first presents the new models. Then, we demonstrate their application with a heating plant that supplies steam for district heating, while also providing power-to-heat grid services by storing excesses renewable electricity as thermal energy.

Keywords: District Energy, Load Shift, Modelica, Power-to-X, Renewable Energy, Silica Sand, Thermal Storage

1 Introduction

As renewable energy penetration increases with decarbonization efforts, silica sand has emerged as an effective low-cost, low-toxicity option for thermal storage of excess renewable power (Gifford, Ma, and Davenport 2020). To date, most applications of solid sand particle thermal energy storage (TES) replace molten-salt in concentrated solar power (CSP) systems for long-duration energy storage for electric power (Ma, Glatzmaier, and Mehos 2014; Mahfoudi, Moummi, and Ganaoui 2014; Gomez-Garcia, Gauthier, and Flamant 2017). For heating applications, a test pilot site at Vatajankoski's district heating network in Kankaanpää, Finland adopted sand-based heat storage (Polar Night Energy 2022). However, scientific research on the application of sand-based TES for heating applications is limited.

To enable wide-scale testing and evaluation of sand-based TES for heating applications, this work developed and open-source released equation-based, object-oriented, multi-domain models with the Modelica language at https://github.com/sbslab/modelica-sand (Hinkelman, Milner, and Zuo 2023). Our application domain is thermofluid and electrical simulation of building and district heating systems with silica-sand particle-based TES in support of design, operation, and energy analysis. Water, molten salt, and phase-change materials are typically used for building TES heating applications (Sarbu and Sebarchievici 2018). Packed-bed TES with rocks/pebbles or ceramic bricks also exist where a fluid, typically air, circulates through the static bed (Sarbu and Sebarchievici 2018). In contrast, this work adopts *fluidized* particle-based TES and heat exchangers where the packed sand particles themselves move through the system, which to our knowledge, has not been evaluated in scientific literature for heating purposes.

To address this gap, we develop computationally efficient models for fluidized silica-sand particle-based TES in heating applications. This includes a polynomial medium model for the thermodynamics of silica sand, several equipment, and a novel heating plant as demonstration. Further, this paper focuses on the assembly of multiple heating equipment into *plants*, such that sand TES can be leveraged to provide heating for buildings or districts. The new models introduced with this paper are open-source (Hinkelman, Milner, and Zuo 2023). In the following sections, we present the new models. First, section 2 summarizes the overall package, while section 3 details the primary contents. As an example, section 4 presents the model and simulation results for a silica sand particle-based heating plant under two control scenarios. Lastly, conclusions are in section 5.

2 Package Overview

Following Modelica standards, model packages are assembled hierarchically, as shown in Figure 1. Designed for compatibility with the Modelica Buildings Library (MBL) v9.0.0 (Wetter et al. 2014) and the Modelica Standard Library (MSL) v4.0.0, this small package contains 9 instantiable models – from control base classes to a complete plant – and two medium models. For each of these, there is at least one runnable validation or example model with a pre-programmed *Simulate and Plot* script.

The six main-level packages are as follows. First, *Blocks* contains key performance indicators for building energy analysis, including source/site energy calculations, operational carbon emissions, and thermal discomfort. Then, *Equipment* and *Media* contains physical equipment

DOI
10.3384/ecp204627

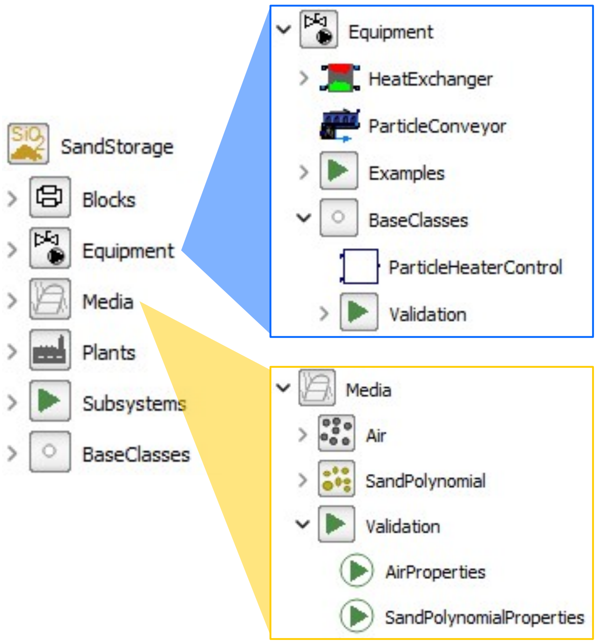


Figure 1. Package structure.

(e.g., heat exchanger) and fluid medium models (e.g., silica sand), respectively. In Figure 1, the internal contents for these two packages are shown. The *Plants* package contains sand-based TES plants, which will be demonstrated in section 4. Lastly, *Subsystems* contains several runnable examples with assemblies of equipment and controls, while *BaseClasses* contains fundamental models that can apply to several sub-packages.

3 Model Contents

While further details for the models included with this open-source release are in the Modelica code itself, here we will present some of the primary contributions. Below, we include the physics and modeling assumptions for the sand medium, particle conveyor, and renewable responsive particle heater control. Validation results for thermodynamic properties of silica sand are also included. For models not described herein, please see the documentation included within the *annotation()* of each respective model (Hinkelman, Milner, and Zuo 2023).

3.1 Sand Medium

The sand medium is modeled as silica (SiO_2, i.e. quartz), which at atmospheric pressure maintains a solid state until 1800°C (Gifford, Ma, and Davenport 2020). Consisting of two of the most common elements in the Earth's crust – silicon and oxygen – silica sand is highly abundant and nontoxic. In nature, solid silica can exist in one of seven unique crystalline structures (Huang and M. Wang 2005), but "the bulk of silica at Earth's surface is stable in the form of low-temperature α-quartz with traces of [other polymorphs]" (Davenport et al. 2020). When heated at atmospheric pressure, quartz undergoes a fast, reversible *displacive transformation*, where around 573°C, the molecular structure of low-temperature α-quartz shifts to high-temperature β-quartz (Davenport et al. 2020). This transition – also known as *quartz inversion* – is visible in silica's specific heat during a heating process (as depicted in Figure 2).

This model covers the solid phase of silica quartz at atmospheric pressure (temperatures from 298–1800K). For heating applications, we assume the sand has constant density and remains tightly packed in all states without mass loss. While the complete thermodynamic formulation is available in the open-source model (Hinkelman, Milner, and Zuo 2023), we will present the details for a critical function: specific heat. Specific heat at constant pressure c_p is formulated as two polynomial functions with a cubic Hermite spline over the quartz inversion temperature as

$$c_p(T) = \begin{cases} a_1T^3 + a_2T^2 + a_3T + a_4 & T \leq 847 - \delta \\ \tilde{c}_p & 847 - \delta < T < 847 + \delta \\ b_1T + b_2 & T \geq 847 + \delta \end{cases} \tag{1}$$

where coefficients a_i, b_i are polynomial fits to the NIST Thermochemical Tables (Chase 1998) with values given in Table 1; T is temperature in units Kelvin; the small temperature transition δ is 10^{-6}; and $\tilde{c}_p$ is a smooth approximation for specific heat defined as a cubic Hermite spline with end positions and first order derivatives that are continuous with the upper and lower polynomials. Other than temperature as a function of specific entropy $T(s)$ – which takes on a similar form as $c_p(T)$ with a Hermite spline – we implemented either pure polynomials (e.g., specific enthalpy $h(T)$ as cubic) or thermodynamic relationships (e.g., internal energy $u = h - p/d$, where pressure p and density d are both constant) for all remaining functions.

Table 1. Coefficients for Equation 1.

i	a_i	b_i
1	$2.799\,140 \times 10^{-6}$	$1.671\,556 \times 10^{-1}$
2	$-5.394\,235 \times 10^{-3}$	$9.804\,303 \times 10^{2}$
3	$4.181\,354 \times 10^{0}$	–
4	$-9.929\,403 \times 10^{1}$	–

The medium implementation is validated with respect to NIST Thermochemical Tables (Chase 1998). Figure 2 depicts the validation results for $c_p(T)$ and $h(T)$. The coefficient of variation of the root mean square error was less than 1% for all thermodynamic functions.

3.2 Particle Conveyor

Sand is moved from low to high elevations by a vertical conveyor. Conveyors have been used in silica-sand CSP systems and have simple physics with large maximum allowable flow rates (Ma, X. Wang, et al. 2021). In this Modelica package, the conveyor is based on a *skip hoist*, which can be used to transport high density particles like

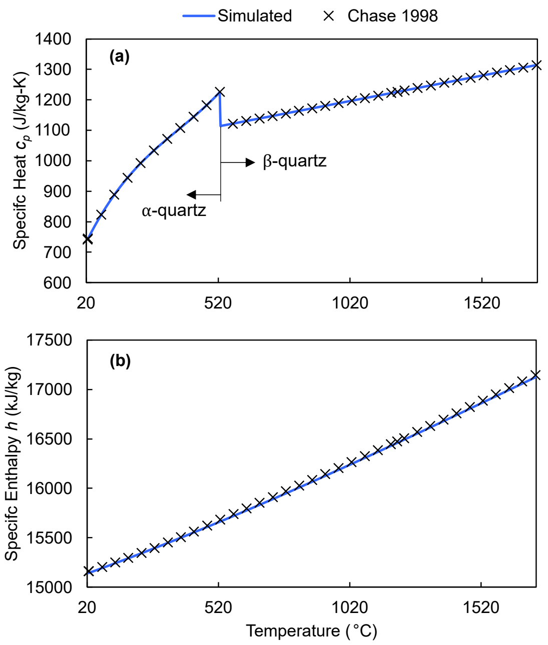


Figure 2. Validation of silica sand medium implementation. The molecular transition from α- to β-quartz is apparent in (a).

concrete or grain over short distances. The power consumed by the hoist P is

$$P = \frac{\dot{m}gh}{\eta}, \tag{2}$$

where $\dot{m}$ is the mass flow rate of the bulk sand, g is the gravitational constant, h is the total lifted height, and η is the conveyor efficiency.

3.3 Particle Heater Control

Typically, the particle heater is responsible for most of the energy consumption of sand-based heating systems. As such, the control design is paramount. Designed with this work, the particle heater control (model diagram in Figure 3) maintains a minimum silica-sand tank temperature, unless renewable energy is present. If the amount of renewable power is greater than the power required by the heater, then the sand is heated above the nominal setpoint up until but not exceeding a maximum value. We refer to this design as *renewable responsive control*.

4 Example

As a demonstration of the new library components, we modeled a novel sand-based heating plant for a steam-service district heating system. In the United States, steam is the most common heat transfer medium for district heating, representing 97% of all installations (ICF LLC and International District Energy Association 2018). The objective of this plant is to meet the steam heating load of the district without on-site fossil fuel consumption and *power-to-heat* thermal storage response when there is a surplus of on-site renewable energy. In addition to the new sand heating models presented in sections 2 and 3, this example also uses two additional open-source libraries: the MBL v9.0.0 (Wetter et al. 2014) and the MSL v4.0.0. The following sections present the system description and models, followed by the simulation results.

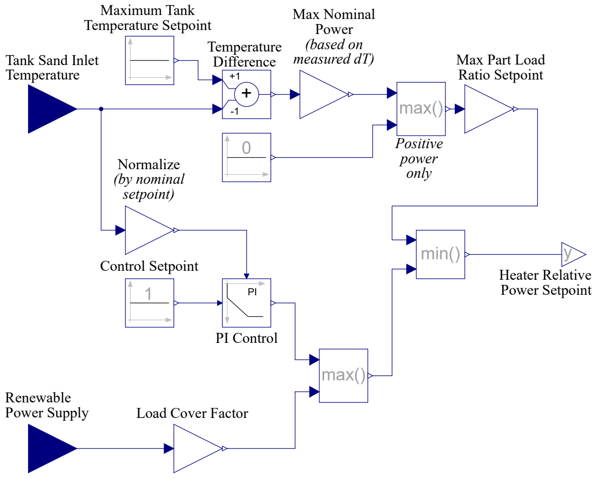


Figure 3. Modelica diagram of the renewable responsive particle heater control.

4.1 System Description and Model

As shown in Figure 4, this 100% electric plant provides steam service to a district network with sand-based TES to convert excess renewable electricity to high-quality thermal energy that can be stored for later use. Inspiration for the heating plant stems from a Brayton combined cycle power system (Gifford, Ma, and Davenport 2020), where silica sand is used for long-duration storage in electrical power generation. However in this case, steam is exported to the district for heating rather than for generating electricity in a turbine.

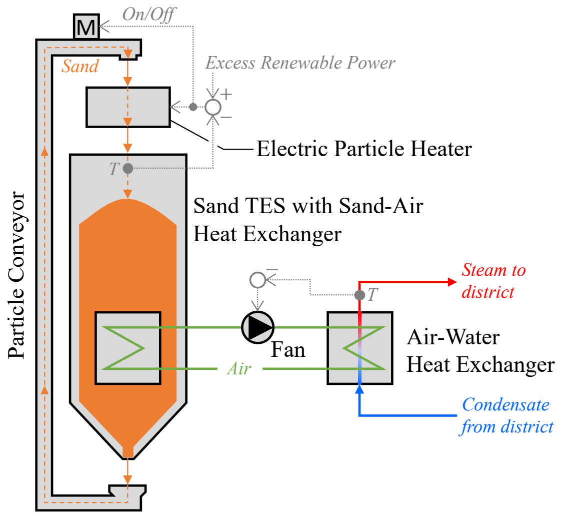


Figure 4. System schematic for the sand TES heating plant.

The plant system design is as follows. This plant contains three fluid loops: (1) a fluidized, packed-bed sand particle loop, (2) an air loop, and (3) a water/steam

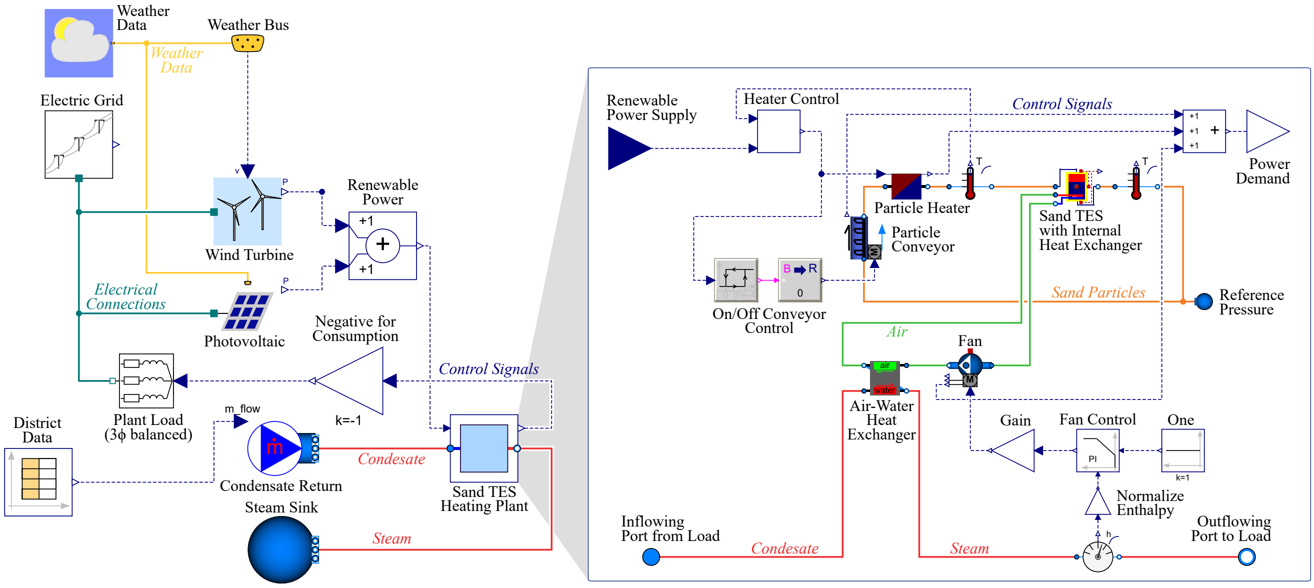


Figure 5. Modelica diagrams of the top-level example (left) and the sand TES heating plant (right).

loop with service to end-use heating loads. On the sand side, sand is heated electrically with a packed-bed particle heater per Ma, Gifford, et al. (2022), which was designed for thermally storing off-peak electricity for later on-peak usage. From the heater, sand is gravity fed into the packed-bed TES, which contains an internal shell-and-plate heat exchanger. This TES/heat exchanger design is similar to Albrecht and Ho (2019), except the heat exchanger is particle-to-air rather than particle-to-sCO$_2$ (supercritical carbon dioxide). From the cold sand hopper at the TES discharge, the particle conveyor (i.e., skip hoist) transports the sand back up to the particle heater. The air loop transfers heat from the sand TES to an air-to-water heat exchanger, where condensate water returned from the district is heated to a superheated state. It is worth noting that direct heat transfer from sand to water is theoretically possible, but previous studies that could validate this design are lacking. Thus, the sand-air-water configuration was selected for this first demonstration case.

The plant controls are as follows. On the sand side, the particle heater is as described in section 3.3 with nominal and maximum temperature setpoints of 1200°C and 1600°C, respectively. In the results, this is called *renewable responsive control*. As a basis for comparison, we also implement a *constant setpoint control* with the heater maintaining the nominal setpoint only. The skip hoist control is on/off (on if the particle heater is on) with a constant mass flow rate of 19.8 kg/s. Lastly, the fan is variable speed (nominal mass flow rate of 14.6 kg/s), and it is modulated with a PI controller to maintain the steam discharge temperature setpoint.

Figure 5 depicts the Modelica model diagrams for the top-level simulation example and the internal blocks for the sand TES heating plant. In the top-level simulation example (left), the plant is tied to the electric grid and contains an on-site photovoltaic (PV) array and wind turbine. At the top-level, this site – located in Denver, Colorado, USA – contains a 2 MW wind turbine, a 2 MW PV array, and a 480V/3ϕ electric service. The heating plant (nominal heating load 3.5 MW) supplies superheated steam to a district network at 180°C and 9.6 bar. The district demand is input as a table-look up based on a typical January heating load profile for a small university campus. Except for the plant, this top-level example uses MBL and MSL models. For example, all electrical components are from *Buildings.Electrical.AC.ThreePhasesBalanced*. Specifically, the PV model is *PVSimpleOriented*.

The Modelica diagram for the sand TES heating plant (Figure 5, right) contains the physical and control systems depicted schematically in Figure 4. In this plant model, the sand medium, air-water heat exchanger, particle conveyor, and heater control are from the new sand heating package. The water/steam medium is modeled with the *StandardWater* model from the MSL, which implements the commonly-adopted IF97 model (Wagner et al. 2000). Air is modeled as *Modelica.Media.Air.ReferenceAir.Air_pT*, which is a detailed dry air model based on the Helmholtz equations of state (Lemmon et al. 2000). For the particle heater, we adopt the MBL model *HeaterCooler_u* from the *Fluid* package, which is an ideal heater that prescribes the heat flow rate and is suitable for energy analysis purposes. For the sand TES with internal heat exchanger, we model this equipment using the *StratifiedEnhancedInternalHex* from the MBL, which represents the desired geometry and general physical principles.

4.2 Results

We simulate the sand plant system for a typical winter day with two control scenarios for the particle heater (1) constant setpoint and (2) renewable responsive. Figure 6 shows how the temperature trajectories at the sand TES with internal heat exchanger responded during periods

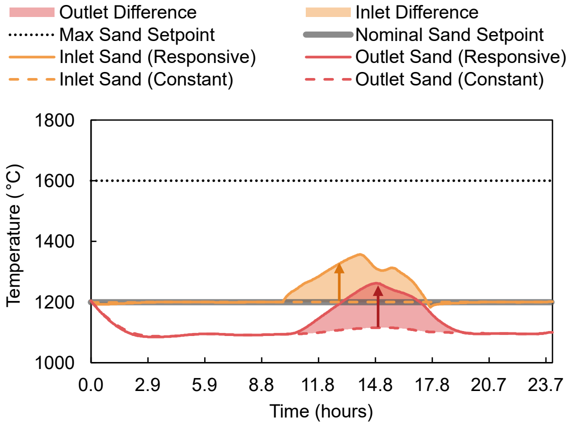


Figure 6. Temperatures at the ports of the sand TES with integrated heat exchanger with both constant setpoint control and renewable responsive control for the particle heater. Arrow direction indicates changes from constant to responsive.

with excess renewable energy. As expected, the constant control scenario maintains the 1200°C inlet sand temperature during the entire simulation period. In contrast, the renewable responsive control uses excess renewable power to increase the sand temperature above the nominal setpoint. This effectively stores the surplus electricity as high-quality heat for later use (i.e., power-to-heat).

In addition to the thermal performance, Figure 7 shows the electrical power results. This includes the power generated by renewable sources (PV and wind turbine), power consumed by the plant, and power provide from the grid (positive if consumed, negative if exported). Before 9:00 and after 18:00, the heating demand is high; because onsite renewable energy production is also low during these times, the electric grid contributes most of the electricity, and there is negligible difference between the two control scenarios.

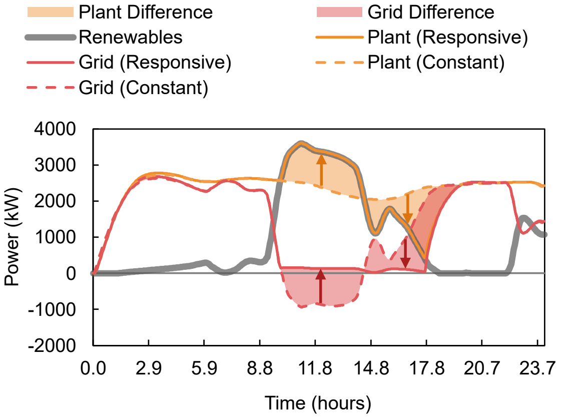


Figure 7. Power generated by renewable sources, consumed by the plant, and provided from the grid (negative if exported), with both constant setpoint control and renewable responsive control for the particle heater. Arrow direction indicates changes from constant to responsive.

In contrast, differences between the constant setpoint and renewable responsive control occur during the middle of the day when renewable generation is high. With the constant control scenario (dashed lines), the site exports surplus renewable energy back to the grid from approximately 10:00 to 14:00. Meanwhile, the plant electrical load is relatively constant. With the renewable responsive control (solid lines), no electricity is exported to the grid, and most of the energy demanded by the plant is from renewable sources (from approximately 9:30 to 17:30). In brief, the renewable responsive control produced a *load shifting response* where excess renewable energy from peak sunlight hours stored thermal energy in the sand for later use. This is evident by the arrows changing direction from up (higher power consumption) two down (lower power consumption) in Figure 7.

5 Conclusion

This paper presented new open-source Modelica models for particle-based silica-sand TES in heating applications. While most of the previous TES with fluidized sand have been developed for CSP and electric power systems, building and district heating applications are also viable use cases that, to date, remain largely unexplored. To enable silica sand as a low-cost, low-toxic storage medium for heating applications with power-to-heat grid service capabilities, we developed several Modelica models and open-source released them in a Modelica package on GitHub (Hinkelman, Milner, and Zuo 2023). As demonstration, we modeled a sand TES heating plant for steam district heating applications with onsite renewable energy (PV and wind turbine). The plant was simulated with two control scenarios for the particle heater: a constant setpoint control and a renewable responsive control.

Results indicated that the renewable responsive control increased the sand tank temperature when excess renewable power was present. This resulted in a load-shift demand response, where excess electricity stored during peak sunlight hours reduced the electricity demand from the grid later in the day. While this paper released the new models and provided a first demonstration case study with promising outcomes, future work is merited before conclusions can be drawn regarding the validity of sand TES for heating applications. Most importantly, this work did not compare the performance of sand TES plants with other designs; thus the impact on financial cost, energy consumption, and carbon emissions remain unexplored. Future work also includes the improvement and thorough validation of equipment models (primarily, the sand TES with internal heat exchanger), detailed design of sand TES for heating purposes, and wide-scale evaluation of silica sand in this new application domain.

Acknowledgements

This research was supported in part by an appointment with the IBUILD Graduate Student Research Program sponsored by the U.S. Department of Energy (DOE), Office of Energy Efficiency and Renewable Energy, and Building Technologies Office. This program is managed by Oak Ridge National Laboratory (ORNL). This program is administered by the Oak Ridge Institute for Science and Education (ORISE) for the DOE. ORISE is managed by ORAU under DOE contract number DESC0014664. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of DOE, ORNL, ORAU, or ORISE. This material is also based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Solar Energy Technologies Office Award Number 38394. Lastly, this work emerged from the IBPSA Project 1, an international project conducted under the umbrella of the International Building Performance Simulation Association (IBPSA). Project 1 will develop and demonstrate a BIM/GIS and Modelica Framework for building and community energy system design and operation.

References

Albrecht, Kevin J. and Clifford K. Ho (2019). "Design and operating considerations for a shell-and-plate, moving packed-bed, particle-to-sCO2 heat exchanger". In: *Solar Energy* 178, pp. 331–340. DOI: 10.1016/j.solener.2018.11.065.

Chase, M.W. Jr. (1998). "NIST-JANAF Themochemical Tables, Fourth Edition". In: *Journal of Physical and Chemical Reference Data Monographs* 9, pp. 1–1951. URL: https://webbook.nist.gov/cgi/cbook.cgi?ID=C14808607&Mask=2.

Davenport, Patrick et al. (2020). *Thermal stability of silica for application in thermal energy storage: Preprint*. Tech. rep. NREL/CP-5700-77426. Golden, CO: National Renewable Energy Laboratory, pp. 1–12. URL: https://www.nrel.gov/docs/fy21osti/77426.pdf.

Gifford, Jeffrey, Zhiwen Ma, and Patrick Davenport (2020). "Thermal Analysis of Insulation Design for a Thermal Energy Storage Silo Containment for Long-Duration Electricity Storage". In: *Frontiers in Energy Research* 8. ISSN: 2296598X. DOI: 10.3389/fenrg.2020.00099.

Gomez-Garcia, Fabrisio, Daniel Gauthier, and Gilles Flamant (2017). "Design and performance of a multistage fluidised bed heat exchanger for particle-receiver solar power plants with storage". In: *Applied Energy* 190, pp. 510–523. DOI: 10.1016/j.apenergy.2016.12.140.

Hinkelman, Kathryn, David Milner, and Wangda Zuo (2023-07). *Modelica Sand Heating Package*. v1.0. URL: https://github.com/sbslab/modelica-sand.

Huang, P.M. and M.K. Wang (2005). *Minerals, Primary*.

ICF LLC and International District Energy Association (2018). *U.S. District Energy Services Market Characterization*. Tech. rep. Washington DC: U.S. Energy Information Administration.

Lemmon, Eric W. et al. (2000). "Thermodynamic properties of air and mixtures of nitrogen, argon, and oxygen from 60 to 2000 K at pressures to 2000 MPa". In: *Journal of Physical and Chemical Reference Data* 29.3, pp. 331–362. DOI: 10.1063/1.1285884.

Ma, Zhiwen, Jeffrey Gifford, et al. (2022). "Electric Charging Particle Heater for Thermal Energy Storage". US 20220021239A1.

Ma, Zhiwen, Greg Glatzmaier, and Mark Mehos (2014). "Fluidized bed technology for concentrating solar power with thermal energy storage". In: *Journal of Solar Energy Engineering* 136.3, p. 031014. DOI: 10.1115/1.4027262.

Ma, Zhiwen, Xingchao Wang, et al. (2021). "Economic analysis of an electric thermal energy storage system using solid particles for grid electricity storage". In: *The ASME 15th International Conference on Energy Sustainability, ES2021*, pp. 1–10. DOI: 10.1115/ES2021-61729.

Mahfoudi, Nadjiba, Abdelhafid Moummi, and Mohammed El Ganaoui (2014). "Sand as a heat storage media for a solar application: Simulation results". In: *Applied Mechanics and Materials* 621, pp. 214–220. DOI: 10.4028/www.scientific.net/AMM.621.214.

Polar Night Energy (2022). "Store Wind and Solar Power as Heat in Sand". In: URL: https://polarnightenergy.fi/technology.

Sarbu, Ioan and Calin Sebarchievici (2018). "A comprehensive review of thermal energy storage". In: *Sustainability* 10.1, p. 191. DOI: 10.3390/su10010191.

Wagner, W. et al. (2000). "The IAPWS industrial formulation 1997 for the thermodynamic properties of water and steam". In: *Journal of Engineering for Gas Turbines and Power* 122.1, pp. 150–180. ISSN: 07424795. DOI: 10.1115/1.483186.

Wetter, Michael et al. (2014). "Modelica Buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

An Open-Source Benchmark of IEEE Test Cases for Easily Testing a New Approach for Steady State Calculations in Power Systems

Joy El Feghali[1] Quentin Cossart[1] Gautier Bureau[1] Baptiste Letellier[1] Ian Menezes[1]
Florentine Rosiere[1] Marco Chiaramello[1]

[1]R&D department, RTE Réseau de transport d'électricité, France
{joy.elfeghali,quentin.cossart,marco.chiaramello}@rte-france.com

Abstract

Power systems modeling and simulation are essential to conduct studies on the electrical transmission system and ensure its security. For this purpose, RTE, the French Transmission System Operator (TSO), is developing Dynaωo, a hybrid Modelica/C++ open-source suite of simulation tools for power systems. Most power systems models for Dynaωo are developed in the Modelica language using the Dynaωo Modelica library. This paper presents a full Modelica standard electrical power system benchmark implemented using the Dynaωo library. The IEEE 14-bus system benchmark is modeled here for steady-state calculation, with an approach that replaces the static load flow. Two test cases are simulated using the OpenModelica environment showing differences in the final steady-state result. We show flexibility in modeling with this library where different system behaviors can be observed and where models with different levels of details can be replaced depending on the application: steady-state calculation, long-term stability, or short-term stability.

Keywords: Modelica, IEEE 14-bus benchmark, power systems simulation, steady-state calculations, Dynawo.

1 Introduction

The electrical transmission system is facing many challenges due to power electronic devices added to the system to connect renewable energy sources and HVDC links. Also, the control of the system is getting more complex due to the increasing use of controllers, with both discrete and continuous behaviors. The transmission system operator has to adapt the system to all these changes and ensure consumers good quality and availability of electricity. The need for a simulation tool that considers all the new dynamics introduced to the electrical system is primordial to conduct studies to face all the challenges. This is true even for the calculation of steady states where the different dynamics of the system can interact and influence the reached steady state. Moreover, this simulation tool should be flexible to adapt to the quick changes in the power system's environment.

RTE, the French electricity Transmission System Operator (TSO), is developing Dynaωo, a hybrid Modelica/C++ open-source suite of simulation tools for power systems (Guironnet, Saugier, et al. 2018). The tools' objective is to conduct various studies like steady-state calculations, long and short-term stability, and short-circuit calculations while considering the different dynamics on the system depending on the study (*Dynawo* 2023). Dynaωo uses the equation-based Modelica language (Fritzson and Engelson 1998) for the modeling part of power systems. Physical and acausal modeling are possible with the Modelica language, and it is an advantage to implement the model in an understandable and usable way. However, the solving part is separated from the modeling part, a choice made to obtain a flexible simulation tool for several applications (Guironnet, Rosière, et al. 2021). Some results of comparison of Dynaωo against other existing tools can be found in (Marin et al. 2022).

In this paper, we show that the Modelica Dynaωo library can be used on its own, to model common electrical transmission system test cases on Modelica modeling environments (here OpenModelica). In particular, the IEEE 14-bus system is modeled with the Dynaωo library, and its different components are presented. The IEEE 14-bus system is successfully implemented in the literature using other Modelica libraries like in (Adib Murad, Gómez, and Vanfretti 2015) and (Fernandes et al. 2018), and results show that it can be simulated in Modelica simulation environments. Also, several open-source electrical power system libraries are available in Modelica language ((Winkler 2017), (Bartolini, Casella, Guironnet, et al. 2019)). In this paper, we focus on the flexibility in modeling offered by the Dynaωo Modelica library, with an extensive choice for modeling depending on the purpose of the study, either long-term or short-term simulation, and on the time constant of the available components. Indeed, short-term simulations use high-detail models.

Steady-state calculations are highlighted in this paper. In particular, the DynaFlow approach that considers the dynamics of the components for the steady-state calculation is shown to give more realistic results than classical approaches like static load flow (Cossart et al. 2021). Several test cases can be easily created using the Dynaωo library to observe the impact that have the different dynamics of the system on the final steady state. In particular, two test cases are performed in this paper to calculate the final steady state after occurred events on the system. The

DOI
10.3384/ecp204633

first test case is a line contingency, and the second is an increase in power consumption that activates current limit automatons depending on their reaction time. Here, components are modeled in a chosen detail level for the calculation of steady states.

This paper is organized as follows: section two presents the Dynaωo Modelica library and focuses on the models for the calculation of steady states and the physical phenomenon studied. The third section focuses on the IEEE 14-bus implementation with the Dynaωo library. Two test cases and their simulation results using the IEEE 14-bus model are presented in the fourth section. Finally, a conclusion and perspectives are given in section five.

2 Dynaωo Modelica Library

The Dynaωo Modelica library comprises several models for the same element, with different levels of details depending on the application.

The main components found in the electrical transmission system are synchronous machines, lines, buses, loads, controllers, transformers, HVDC, photovoltaics, and wind power plants. Mainly models of machines, loads, and controls vary depending on the study we would want to achieve. It is important to note that, at the library level, the models are divided by component type and not by application purpose.

Dynaωo Modelica library (Figure 1) contains a wide range of models to describe power systems. A combination of models with different levels of detail is possible, and it is one of the advantages of this library that creates flexibility to switch between component models. Multiple test cases can thus be modeled, and the user can create models without limiting to any application.

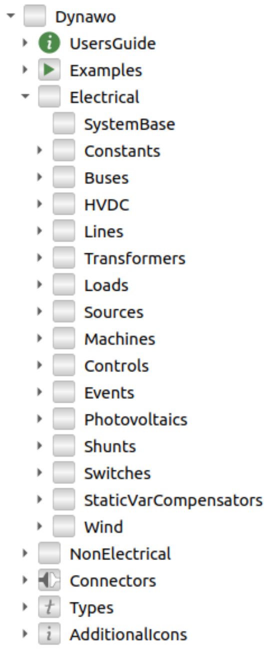


Figure 1. Dynaωo library in OpenModelica

Dynaωo library offers models to calculate steady states while properly taking into account the interactions between continuous and discrete controllers. The approach differs from static load flow calculation by considering the dynamics of systems like controllers and HVDC that impact the final steady state (Cossart et al. 2021). The focus will remain on the final steady-state result for voltage and currents that should not cross threshold limits. But with this approach, steady state is obtained through a time-domain simulation.

Models for steady-state calculations contain the minimum details needed to perform the study. Fast dynamics are neglected, and transitory phenomena are not taken into account in modeling since only the final steady state is important. These models are put in the DynaFlow category. Other phenomena can be observed by replacing components with more detailed models. DynaFlow simulations will thus use a higher time step (e.g. few seconds) compared to simulations with more detailed models considering fast dynamics (e.g. few milliseconds).

The following part presents the IEEE 14-bus system modeled with the Modelica language (using models from the Dynaωo library) and simulated with the OpenModelica environment for the calculation of steady state. The model has been integrated within the package `Examples.DynaFlow` and can be found on `https://github.com/dynawo/dynawo/tree/master/dynawo/sources/Models/Modelica/Dynawo`.

3 Implementation of IEEE 14-bus System using Dynaωo Modelica Library

3.1 Description of the IEEE 14-Bus System

The IEEE 14-bus system (Figure 2) is a standard test case in the power system community. It represents a simple approximation of the American Electric Power system in the early 1960s (Kodsi and Canizares 2003). The IEEE 14-bus test case system comprises 14 buses, 2 generators, 3 synchronous condensers, 1 shunt, 3 transformers, 17 lines, and 11 loads.

The three transformers separate the system into two parts with two voltage levels: 69 kV and 13.8 kV. The lower part of the system presented in Figure 2 corresponds to 69 kV, and the upper part corresponds to 13.8 kV.

The benchmark is modeled using the Dynaωo Modelica library on the OpenModelica tool (Figure 3). Since a steady-state calculation test case is used, the chosen models for components comprise the minimum details for simulation, considering the interactions of the components for steady-state calculations. Models of components that form the IEEE 14-bus system are presented in the following part.

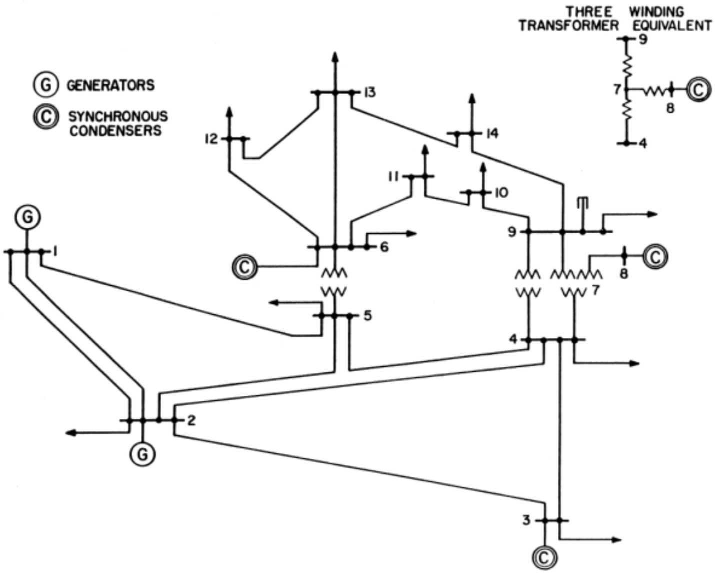


Figure 2. IEEE 14-bus benchmark

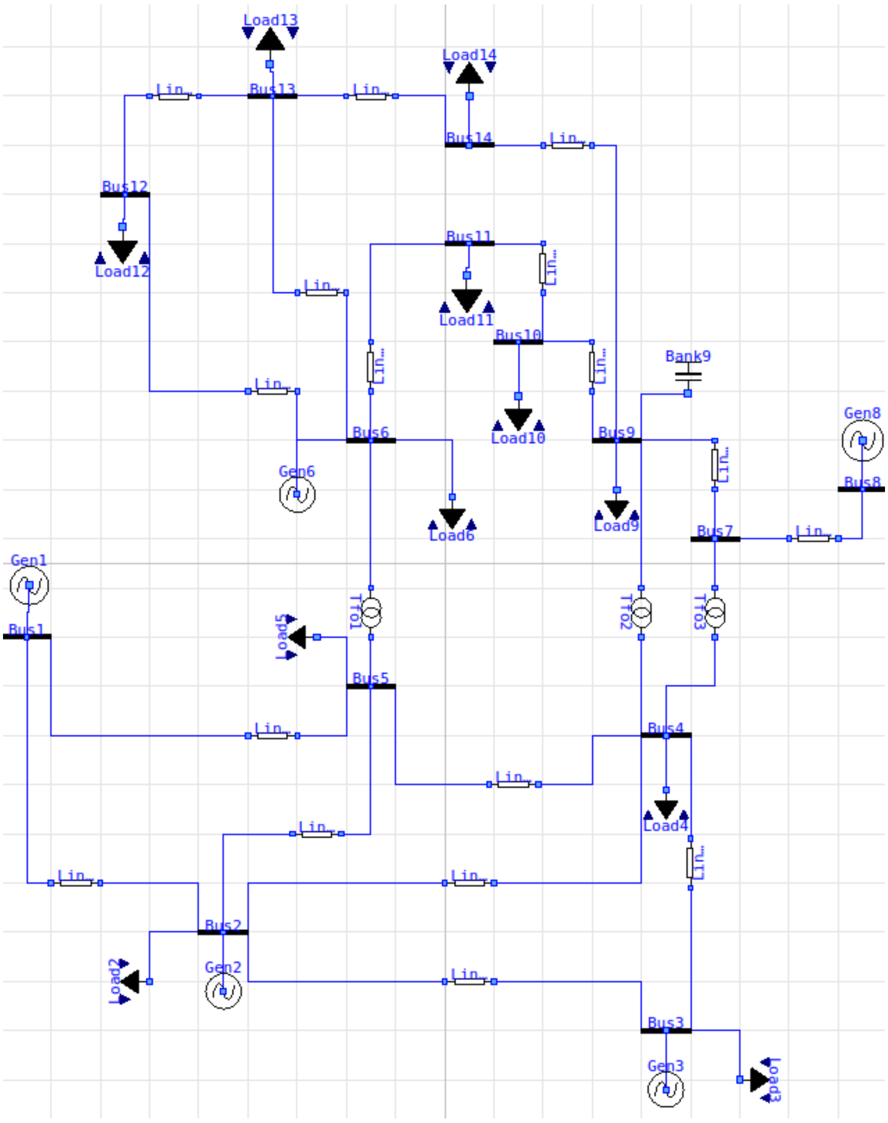


Figure 3. IEEE 14-bus model on OpenModelica

3.2 Models

Power systems models are developed in the Dynaωo Modelica library. The models presented in this section have the lowest level of detail. These models can easily be replaced with other complex models since the connections ports, input, and output variables are common for each component model. These models are used to simulate the test cases presented in section 4. Other models are indeed available in the library to perform other test cases.

3.2.1 Generators

On Figure 3, generator 3, 6 and 8 are synchronous condensers. The same model is used for all the generators and the synchronous condensers, but the produced power of the latter is set to zero. `GeneratorPV`, the model of these generators containing minimum details is found within the package `Dynawo.Electrical.Machines.SignalN`. An input "signal N" described below is used in this generator model for primary frequency regulation.

These components can adjust their reactive power Q_{Gen} to maintain the voltage U at a certain reference level U_{Ref}. This model regulates the voltage unless the reactive power of the generator hits its limits at Q_{Min} or Q_{Max} (Equation 1). The active power P_{Gen} is modified by the frequency regulation model through the signal N variable (Equation 2, Equation 3). P_{Ref} is the set point active power and P_{Nom} is the nominal active power.

$$\begin{cases} Q_{Gen} = Q_{Max} \text{ if maximum generation} \\ Q_{Gen} = Q_{Min} \text{ if maximum absorption} \\ U = U_{Ref} \text{ if not} \end{cases} \tag{1}$$

$$P_{GenRaw} = P_{Ref} + P_{Nom} * K_{Gover} * N \tag{2}$$

$$P_{Gen} = \begin{cases} P_{Max} \text{ if } P_{GenRaw} > P_{max} \\ P_{Min} \text{ if } P_{GenRaw} < P_{Min} \\ P_{GenRaw} \text{ if not} \end{cases} \tag{3}$$

3.2.2 Active Power Control

For steady-state calculations, the active power control of the generators is adjusted with a variable signal N to balance the active power mismatch between generation and consumption. The purpose is to regulate the frequency. All the generators are connected to the `SignalN` model found within the package `Dynawo.Electrical.Controls.Frequency`. When using this model, the frequency is not explicitly modeled. Instead, a voltage angle reference node is set to balance the equations. All generators receive the same signal N control, the generation power depends on the participation percentage of each generator K_{Gover} (Equation 2). This value is set to zero for synchronous condensers.

3.2.3 Current Limit Control

A model `CurrentLimitAutomaton` (CLA) for controlling the current of a component is available in the library within the package `Dynawo.Electrical.Controls.Current`. This controller will open one or several components when the current stays above a predefined threshold I_{Max} during a certain amount of time t_{lag} on a monitored component like a line or a transformer.

3.2.4 Loads

Loads are modeled as $\alpha\beta$ restorative loads. In the model `LoadAlphaBetaRestorative` found within the package `Dynawo.Electrical.Loads`, the load restoration emulates the behavior of a tap changer transformer that connects the load to the system and regulates the voltage at its terminal (Equation 4 – Equation 7). After an event, the load goes back to its initial active power P and reactive power Q respecting the time constant t_{filter} unless the filtered voltage amplitude at the terminal $U_{Filtered}$ is below U_{Min} or above U_{Max}. In these cases, the load behaves as a classical $\alpha\beta$ load. The variation of the load voltage and power is not instantaneous, which impacts the final steady-state value, a phenomenon not considered with static load-flow calculation. P_{Ref} and Q_{Ref} are the set point active and reactive power.

$$t_{filter} * \frac{U_{FilteredRaw}}{dt} = U - U_{FilteredRaw} \tag{4}$$

$$U_{Filtered} = \begin{cases} U_{Max} \text{ if } U_{FilteredRaw} \geq U_{Max} \\ U_{Min} \text{ if } U_{FilteredRaw} \leq U_{Min} \\ U_{FilteredRaw} \text{ if } U_{Min} \leq U_{FilteredRaw} \leq U_{Max} \end{cases} \tag{5}$$

$$P = P_{Ref}\left(\frac{U}{U_{Filtered}}\right)^{\alpha} \tag{6}$$

$$Q = Q_{Ref}\left(\frac{U}{U_{Filtered}}\right)^{\beta} \tag{7}$$

3.2.5 Transformers

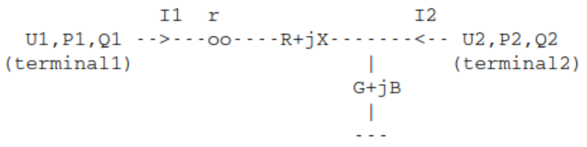
For simplification purposes, the three-winding transformer of Figure 2 is modeled as a two-winding transformer in the IEEE 14-bus test case model as in Figure 3.

The `TransformerFixedRatio` model of the `Dynawo.Electrical.Transformers` package represents a two-winding transformer with a fixed ratio r (Figure 4). This model is used for the three transformers in the IEEE 14-bus model.

Equation 8 and Equation 9 describe the behavior of this component with respect to conventions taken as in Figure 4.

$$r^2\underline{V}_1 = r\underline{V}_2 + \underline{Z}\,\underline{I}_1 \tag{8}$$

$$\underline{I}_1 = r(\underline{Y}\,\underline{V}_2 - \underline{I}_2) \tag{9}$$

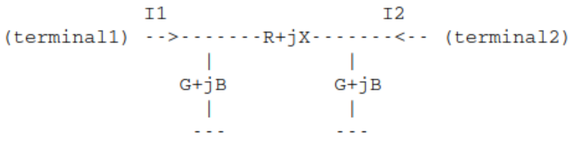


```
                 I1  r                    I2
  U1,P1,Q1 -->---oo----R+jX-------<-- U2,P2,Q2
 (terminal1)                     |       (terminal2)
                                G+jB
                                 |
                                ---
```

Figure 4. Transformer model

3.2.6 Lines

Lines are modeled with the model `Line` of the package `Dynawo.Electrical.Lines.Line`. This model represents a classical π line (Figure 5).



```
                 I1                   I2
 (terminal1) -->-------R+jX-------<-- (terminal2)
                  |           |
                 G+jB        G+jB
                  |           |
                 ---         ---
```

Figure 5. Line model

The model of the line represents the voltage drop between terminal 1 and terminal 2 (Equation 10 and Equation 11).

$$\underline{Z} * (\underline{I}_2 - \underline{Y}\,\underline{V}_2) = \underline{V}_2 - \underline{V}_1 \qquad (10)$$

$$\underline{Z} * (\underline{I}_1 - \underline{Y}\,\underline{V}_1) = \underline{V}_1 - \underline{V}_2 \qquad (11)$$

3.2.7 Shunt

A shunt model is connected to bus 9. The model `ShuntB` of the package `Dynawo.Electrical.Shunts` represents a shunt with constant susceptance and voltage-dependent reactive power Q (Equation 12).

$$Q = BU^2 \qquad (12)$$

3.2.8 Buses

The model `Bus` of the package `Dynawo.Electrical.Buses` is used to model all the buses of the system. The bus does not add any equations to the system. It is used to connect several components into one node.

3.2.9 Switch-Off Equations

Switch-off equations for each component are also included in the IEEE 14-bus model. These equations have the purpose of determining if the component is connected to the electrical transmission system or not. These equations are put in the extended IEEE 14-bus model to enable connecting and disconnecting lines to simulate different test cases.

The number of switch-off signals differs from one component to another. Loads, transformers, lines, and shunts have two switch-off signals. Generators have three switch-off signals. When the switch-off is activated, some values in the disconnected model are set to zero like current I, active power P, and reactive power Q.

4 Test Cases

In this section, two test cases are presented to highlight the use of Dynaωo Modelica library for benchmarks like the IEEE 14-bus system, the advantage of the DynaFlow steady-state approach over a static load flow calculation, and the ease of adding models and creating different test cases to observe multiple phenomena.

Test cases are chosen in the transmission electrical grid context, where studies focus on the state of the system after a loss of a line or a generator that may cause a variation in voltage, current, and frequency, with values that may become critical to the system. Controllers present in the system react to these changes.

The first test case describes the behavior of the system after an occurrence of a line contingency. This simulation highlights the use of the library to observe commonly studied phenomena on the electrical grid. The second test case describes the interaction of multiple current limit automatons after an increase in load consumption. For the second test case, two simulations are done with different parameters, resulting in different final steady states. These simulations highlight the importance of representing the dynamics of the system and performing a time-domain simulation while calculating the final steady state.

The initial data for generators and the simulation solver needed to perform the simulation are given. Then, the results of the two test cases are presented after a DynaFlow time-domain simulation of the model presented in the previous section.

4.1 Data

The data used corresponds to the available online data for IEEE 14-bus system. In Table 1, initial values for the five generators are given.

Table 1. Initial values for generators

Generator	P_{Gen} *(MW)*	Q_{Gen} *(Mvar)*	*U (kV)*	Θ *(°)*
1	232.39	-16.55	73.14	0.00
2	40.00	43.56	72.11	-4.98
3	0.00	25.07	69.69	-12.73
6	0.00	12.73	14.77	-14.22
8	0.00	17.62	15.04	-13.36

4.2 Simulation Solver

The solver used in OpenModelica is Euler with a 10 s step and a 10^{-6} tolerance. An additional translation flag, "–daeMode", is added to the model. klu is chosen as a linear solver and kinsol as a non-linear solver. The simulation time is 200 s.

4.3 Test Case with Line Contingency

The test case simulates a loss of a line. We are interested in studying how the electrical transmission system will behave after a loss of a line and if the model describes the phenomenon well. The line contingency between bus 1 and bus 5 occurs at time $t = 100\ s$. In the model, the value of the switch-off signal of the line becomes `true` which causes the values of active power, reactive power, and current to drop to zero on both line terminals. For example, Figure 6 shows the line's active power on terminal 1, here given in per unit with S_{Ref} base.

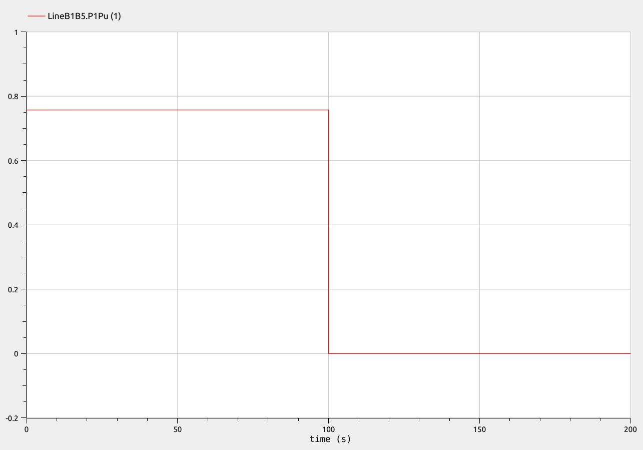


Figure 6. Line contingency test case: Active power of the line Bus1-Bus5 at terminal 1 in per unit (base S_{Ref})

The increase of the active power is immediate for generator 1 when the line is disconnected at time $t = 100\ s$, as seen in Figure 7. Generator 2 has a similar behavior as generator 1. The power of the three other synchronous condensers remains at 0. Since one line is lost, active losses on the other lines will increase, and since the consumption demand remains the same, the active power production of generators will increase to satisfy the consumption demand and thus regulate the frequency. The active power variation is caused by the generator's primary frequency regulation, which adjusts the active power to balance power generation and consumption. The voltage angles have also changed for all the generators except for generator 1, which corresponds to the reference node. The value of the reactive power also changes to maintain the voltage at the same level U_{Ref} when reactive losses increase on the lines. Steady-state values at the end of the simulation are given in Table 2.

The restoration phenomenon of loads modeled in Equation 4 – Equation 7 can be observed, for example, for load 5 in Figure 8. The active power of the load (given in per unit with S_{Ref} base) drops after the loss of line Bus1-Bus5 since the voltage has decreased. The active power of the load goes back to its initial value after the event.

In Table 3 and Table 4, the power flow is observed at all the lines at the initial and final time of the simulation. Since the line between bus 1 and bus 5 is disconnected from the system, the power is distributed to all the other lines. Then, power values of each line differ from Table 3 to Table 4. After the contingency, bus 1 is connected only to bus 2, and all the generated power is transmitted in the line between these two buses.

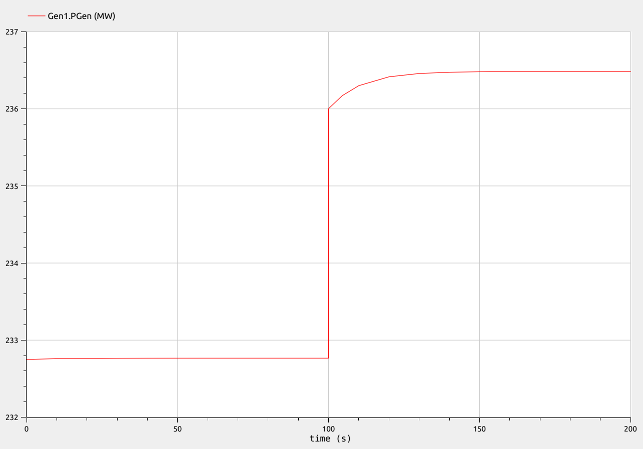


Figure 7. Line contingency test case: Generator 1 active power in MW

Table 2. Final values for generators

Generator	P_{Gen} *(MW)*	Q_{Gen} *(Mvar)*	*U (kV)*	Θ *(°)*
1	236.48	-37.22	73.14	0.00
2	43.79	75.33	72.11	-7.60
3	0.00	30.08	69.69	-17.34
6	0.00	21.26	14.77	-19.43
8	0.00	19.90	15.04	-19.58

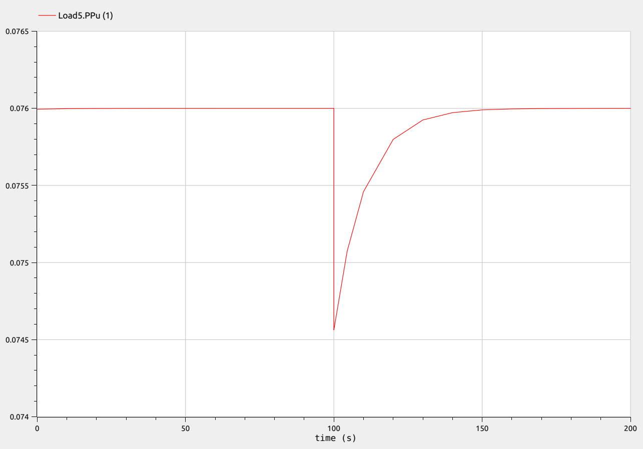


Figure 8. Line contingency test case: Load 5 active power in per unit (base S_{Ref})

The result can be obtained with a static load flow starting with the correct initial conditions: without line Bus1-Bus5. The advantage of the DynaFlow approach is the possibility of observing the system's evolution in time (like the load restoration phenomena) and the occurrence of events that may change the final steady-state result.

Table 3. Initial power flow values for lines

Line	$P_1(MW)$	$P_2(MW)$	$Q_1(Mvar)$	$Q_2(Mvar)$
Bus1-Bus2	157.05	-152.74	-20.57	27.87
Bus1-Bus5	75.71	-72.94	3.86	2.28
Bus2-Bus3	73.34	-71.01	3.59	1.60
Bus2-Bus4	56.33	-54.64	-1.50	3.01
Bus2-Bus5	41.72	-40.81	1.19	2.09
Bus3-Bus4	23.56	-23.19	-4.85	4.48
Bus4-Bus5	61.67	-61.15	-14.13	15.76
Bus6-Bus11	7.36	-7.31	3.70	-3.58
Bus6-Bus12	-7.74	7.81	-2.38	2.53
Bus6-Bus13	-17.55	17.76	-6.86	7.28
Bus7-Bus8	0	0	-17.17	17.63
Bus7-Bus9	-28.25	28.25	-5.01	5.82
Bus9-Bus10	-5.25	5.26	-4.09	4.12
Bus9-Bus14	-9.30	9.42	-3.28	3.53
Bus10-Bus11	3.77	-3.76	1.75	-1.72
Bus12-Bus13	1.59	-1.58	0.76	-0.75
Bus13-Bus14	5.62	-5.57	1.81	-1.71

Table 4. Final power flow values for lines

Line	$P_1(MW)$	$P_2(MW)$	$Q_1(Mvar)$	$Q_2(Mvar)$
Bus1-Bus2	236.48	-226.64	-37.22	61.44
Bus1-Bus5	0	0	0	0
Bus2-Bus3	86.87	-83.61	2.44	6.66
Bus2-Bus4	83.61	-79.89	-1.94	9.64
Bus2-Bus5	78.24	-75.04	0.69	5.42
Bus3-Bus4	10.67	-10.59	-5.50	4.43
Bus4-Bus5	24.95	-24.84	-13.08	13.41
Bus6-Bus11	6.13	-6.08	4.88	-4.77
Bus6-Bus12	-7.62	7.69	-2.56	2.71
Bus6-Bus13	-16.94	17.15	-7.48	7.88
Bus7-Bus8	0	0	-19.32	19.91
Bus7-Bus9	-29.43	29.43	-4.28	5.16
Bus9-Bus10	-6.46	6.48	-2.88	2.92
Bus9-Bus14	-10.04	10.17	-2.48	2.76
Bus10-Bus11	2.55	-2.54	2.95	-2.92
Bus12-Bus13	1.47	-1.46	0.94	-0.93
Bus13-Bus14	4.91	-4.86	2.61	-2.51

In fact, in both approaches, we are interested in the final steady state. However, phenomena observed with DynaFlow, like the interaction of multiple discrete and continuous components, can change the final result, which can not be seen with a static load flow.

4.4 Test Case with Current Limit Automatons

In this part, values are given in per unit (p.u.). For power values, the p.u. base is the reference apparent power S_{Ref}. The p.u. base is the nominal voltage at the line U_{Nom} for voltage values. For current values, the p.u. base is deduced from S_{Ref} and U_{Nom}.

The test case simulates an increase in the current due to increased consumption of load 5 of 0.3 p.u. at $t = 50$ s (Figure 9). A current limit controller is available on each of the lines: CLAB1B2 for line Bus1-Bus2, CLAB2B5 on line Bus2-Bus5, and CLAB1B5 on line Bus1-Bus5. Increasing load consumption can thus cause a line loss if we reach the maximum allowed current I_{Max}. In this test case, we observe how the reaction time t_{lag} of each controller and the line's maximal current I_{Max} are important and impact the final steady-state result. In fact, when one of the controllers reacts to disconnect a component, the current variation impacts other lines.

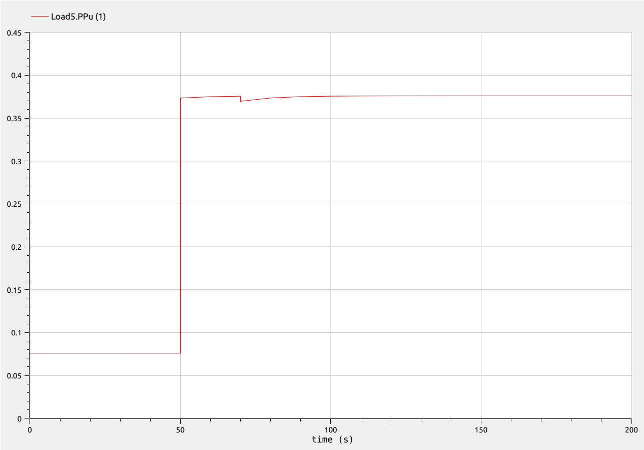


Figure 9. Increased consumption with CLA test case: Load 5 active power in per unit (base S_{Ref})

In the first simulation, we take parameters for each controller as in Table 5.

Table 5. Parameters of the `CurrentLimitAutomaton` for case 1

Controller	I_{Max} *(p.u.)*	t_{lag} *(s)*
CLAB1B2	1.55	30
CLAB1B5	2.00	50
CLAB2B5	0.49	20

After the event at $t = 50$ s, the current on the lines increases as seen in Figure 10. However, for line Bus1-Bus2 and line Bus2-Bus5, the current is now higher than the allowed I_{Max}. The controller CLAB2B5 will react after 20 s to disconnect the line Bus1-Bus2, before the controller CLAB1B2 that can only interfere after 30 s. The disconnection of line Bus2-Bus5 decreases the current of line Bus1-Bus2, which is now below the $I_{Max} = 1.55$ $p.u.$ The current of line Bus1-Bus5 increases but stays below $I_{Max} = 2$ $p.u.$ The final steady state is reached after the restoration of the loads. Here, the system can operate after the loss of the line.

If we change the reaction time of CLAB1B2 to 20 s and the reaction time of CLAB2B5 to 30 s as in Table 6, different results are obtained. In this case, after the increase of the power of load 5 at time $t = 50$ s, all the currents in-

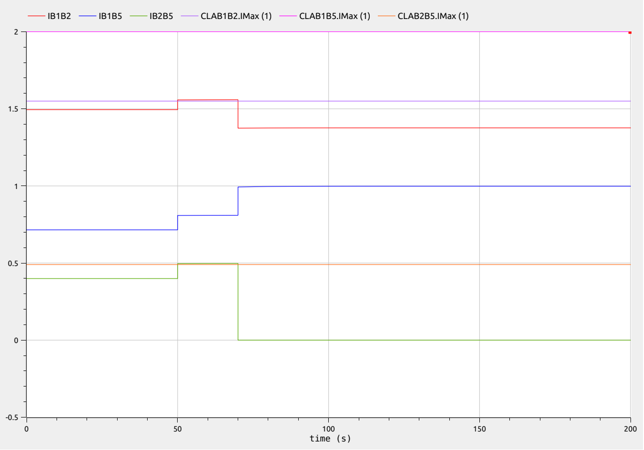


Figure 10. Increased consumption with CLA test case: Current for lines Bus1-Bus2, Bus1-Bus5, and Bus2-Bus5 in per unit

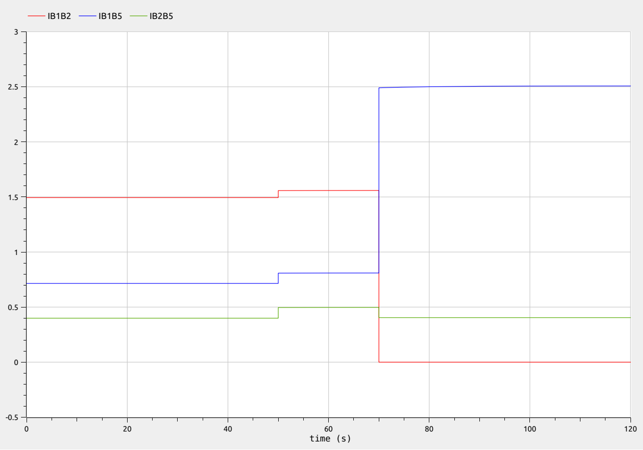


Figure 11. Increased consumption with CLA test case: Current for lines Bus1-Bus2, Bus1-Bus5, and Bus2-Bus5 in per unit

crease. However, the controller CLAB1B2 will react before the controller CLAB2B5, and line Bus1-Bus2 is thus disconnected at $t = 70\ s$. The current of line Bus2-Bus5 decreases below the $I_{Max} = 0.49\ p.u.$, and the current of line Bus1-Bus5 increases above the $I_{Max} = 2\ p.u.$ After 50 s (at $t = 120\ s$), CLAB1B5 will react to decrease the current by disconnecting line Bus1-Bus5. But this event disconnects generator 1 since there are no more lines connected. All the generated power should now come from generator 2. The simulation fails and stops at $t = 120\ s$. Here, the system can not operate after the loss of the two lines Bus1-Bus2 and Bus1-Bus5, and generator 1.

Table 6. Parameters of the `CurrentLimitAutomaton` for case 2

Controller	I_{Max} *(p.u.)*	t_{lag} *(s)*
CLAB1B2	1.55	20
CLAB1B5	2.00	50
CLAB2B5	0.49	30

We conclude that the final steady state result depends on the different controllers with different time constants available in the system. Indeed, other parameters would have given us other final steady states. The final steady state calculation with the DynaFlow approach after time-domain simulation gives realistic results of components' interactions (like causing system failure) that cannot be seen when performing a static load flow. In fact, it is difficult to reproduce the final steady state with one static load flow calculation because values are not calculated in a time domain. For instance, several initial conditions should be taken into account to represent the reaction of each current limit automaton. Also, no flexibility in changing the reaction time is available. A time-domain simulation allows changing parameters and testing different cases for final steady-state calculations.

5 Conclusion and future works

The Dynaωo library is an open-source Modelica library with an extensive choice of models that offers flexibility in modeling depending on the desired application. Steady-state calculations, long-term stability, short-term stability, and short-circuit studies can be done. The library also offers the flexibility of combining different models with different levels of detail.

In this paper, models from the Dynaωo Modelica library are presented for steady-state calculations with a time-domain simulation approach that considers dynamic phenomena not taken into account with static load flow. Two different test cases are developed in this paper using the IEEE 14-bus system benchmark. Different simulations are performed, and we show that final steady-state results depend on the different dynamics present in the system and the parameters that can be modified easily using the library. These test cases show the importance of the approach that considers the system's dynamics. The Dynaωo Modelica library allows modeling several test cases for different system studies and behaviors of the system while mixing different dynamic models.

Other test cases can be developed based on the IEEE 14-bus benchmark test cases presented in this paper by adding other controllers to the system, like phase shifters or tap changers for transformers, with models available in the library. Also, IEEE benchmarks with higher nodes, like IEEE 30-bus and IEEE 57-bus systems, will be added to the library. Benchmarks like the Nordic 32-bus system and RVS are modeled with the Dynaωo Modelica library and are not presented in this paper but are available on the GitHub repository. These models are used for voltage stability studies (long-term stability) where more detailed component models are used. Future works for this library include adding more detailed models and controls, for example, renewable energy control systems, to study their impact on the electrical transmission system.

References

Adib Murad, Ahsan, Francisco José Gómez, and Luigi Vanfretti (2015). “Equation-Based Modeling of Three-Winding and Regulating Transformers using Modelica”. In: *PowerTech Eindhoven 2015*. IEEE conference proceedings.

Bartolini, Andrea, Francesco Casella, Adrien Guironnet, et al. (2019). “Towards pan-european power grid modelling in modelica: Design principles and a prototype for a reference power system library”. In: *LINKÖPING ELECTRONIC CONFERENCE PROCEEDINGS*. Vol. 157, pp. 627–636.

Cossart, Quentin et al. (2021). “A novel approach for the calculation of steady states in transmission systems using simplified time-domain simulation”. In: *2021 IEEE Madrid PowerTech*. IEEE, pp. 1–6.

Dynawo (2023). https://dynawo.github.io/.

Fernandes, Marcelo de C et al. (2018). “Modeling and simulation of a hybrid single-phase/three-phase system in modelica”. In: *2018 Simposio Brasileiro de Sistemas Eletricos (SBSE)*. IEEE, pp. 1–7.

Fritzson, Peter and Vadim Engelson (1998). “Modelica—A unified object-oriented language for system modeling and simulation”. In: *European Conference on Object-Oriented Programming*. Springer, pp. 67–90.

Guironnet, Adrien, Florentine Rosière, et al. (2021). “Speed-up Of Large-Scale Voltage Stability Simulations within a Fully Separated Modeler/Solver Framework”. In: *2021 International Conference on Smart Energy Systems and Technologies (SEST)*. IEEE, pp. 1–6.

Guironnet, Adrien, Marianne Saugier, et al. (2018). “Towards an open-source solution using Modelica for time-domain simulation of power systems”. In: *2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. IEEE, pp. 1–6.

Kodsi, Sameh Kamel Mena and Claudio A Canizares (2003). “Modeling and simulation of IEEE 14-bus system with FACTS controllers”. In: *University of Waterloo, Canada, Tech. Rep*.

Marin, Jose Luis et al. (2022). “An Open Source Tool to Compare Simulators on Large-Scale Cases—Application to Dynawo”. In: *2022 Open Source Modelling and Simulation of Energy Systems (OSMSES)*. IEEE, pp. 1–6.

Winkler, Dietmar (2017). “Electrical power system modelling in modelica–comparing open-source library options”. In: *Proceedings of the 58th Conference on Simulation and Modelling (SIMS 58) Reykjavik, Iceland, September 25th–27th, 2017*. 138. Linköping University Electronic Press, pp. 263–270.

Dropwise Condensation Water Drainage Model

Marcus Richardson[1] Robert Kunz[2]

[1]The Boeing Company, USA, {marcus.k.richardson}@boeing.com
[2]Mechanical Engineering, Pennsylvania State University, USA, {rkf102}@psu.edu

Abstract

Modeling of condensation is important to predicting the amount of residual water in small channels. The residual water that forms becomes a source of humidity for permeable materials such as wooden structure and insulation. A Modelica model has been implemented that predicts the amount of residual moisture after a period of water build up. The model has a very low computational requirement and runs in minutes on a desktop computer. This model includes parameters to relate droplet physics to a control volume. The parameters provide a macroscopic means of varying droplet adhesion force, droplet velocity, and drainage dynamics. Using CFD data as an example of real world data, this model has been correlated to demonstrate the effects of the parameters. This model enables analytical prediction of the amount of time that is needed to dry the internal surfaces of an aircraft after flight and may be connected to a diffusion model for permeable materials.

Keywords: condensation, Droplet Distribution

1 Introduction

Research into mathematical condensation modeling was motivated by the author's experience with aircraft moisture management. Aircraft operators have struggled with managing the effects of condensate since the mid-20th century (Huber, Schuster, and Townsend 1999). Uncontrolled condensation leads to uncomfortable passenger experiences, costly maintenance actions, and extra weight, where every pound counts. Accumulated water in passenger aircraft contributes to moisture related problems, including structure corrosion, uncontrolled water flow, increased fuel consumption, higher maintenance costs and mold growth (Wörner et al. 2002). The insulation systems are heavily impacted by condensation. An aircraft insulation system comprises fiberglass batting and cover films that enclose the insulation. The insulation system is installed around the circumference of the fuselage and extends from the flight deck to the back of the airplane (see figures 1 and 2). To improve and test their designs, aircraft manufacturers invest substantially in designs and test methods (Connell and Richardson 2022; Connell, Carnegie, and Richardson 2020; Richardson, Imada, and Sarinas 2021; Khashayer et al. 2019). When summed for the whole airplane, moisture absorption into the blankets can result in a measurable weight increase. Each pound of moisture can translate to 0.03 pounds or more of extra fuel consumption per flight (Lents 2021).

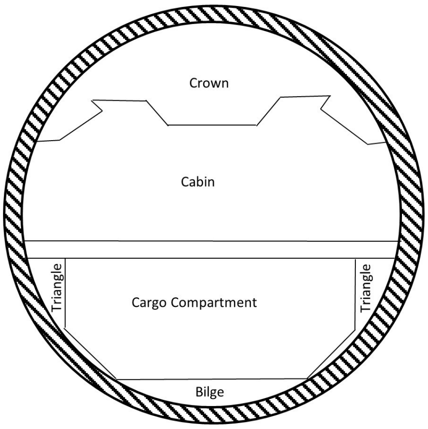


Figure 1. Insulation is typically installed around the circumference of an aircraft monocoque adapted from (Connell and Richardson 2022).

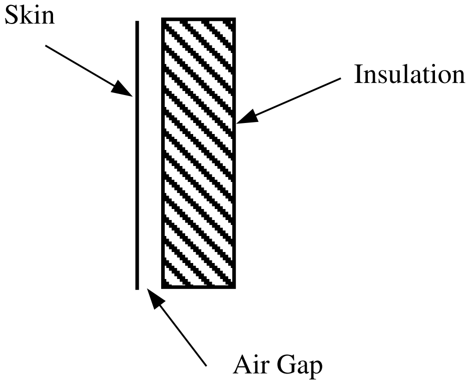


Figure 2. A typical insulation blanket leaves a gap between the skin and the insulation (Wörner et al. 2002).

Droplet motion is a critical aspect of condensation. Droplet motion, particularly dropwise condensation, has been studied since the 1930s because it was reported to have a much higher heat transfer coefficient than film condensation. Although, mass transfer was not the motivation of the studies, the dominant mode of heat transfer was the latent heat of condensation and evaporation (mass

DOI
10.3384/ecp204643

and heat transfer). The mass transfer rate is affected by the density of droplets and type of condensation. Droplet density is defined here as the number of droplets per unit surface area. Rose et al. provide a seminal work on the distribution of droplets on a surface (Rose and Glicksman 1973; Rose 1976), also referred to as the Rose droplet distribution curve. It was extended to low pressure ambient environments (saturation temperature of 31°C) and reproduced later (Graham and Griffith 1973). These researchers studied dropwise condensation on a vertical surface, observing droplets forming, coalescing and falling such that under steady-state conditions a maximum droplet size was noted. Rose and Glicksman broke up the process into generations of droplets. Their examination of the data revealed two key parameters, the fraction of available area and the ratio of the maximum radius of a current generation of droplets to its predecessors. The paper develops a calculation method and presents a comparison with the test data (Rose and Glicksman 1973).

Water droplets on inclined surfaces have an internal flow field that affects the rate at which the droplet moves. The droplet internal flow field, adhesion forces, and surface inclination angle on a hydrophilic surface have been related by CFD simulations and experimental tests (Al-Sharafi et al. 2020). This study determined that the shear force term in the force balance for a water droplet is negligible. The paper provides droplet geometry models and a table of advancing and receding contact angles, each for a variety of droplet volumes and surface inclination angles.

Condensation and management of the residual water is an important topic to the aircraft industry (Wörner et al. 2002; Huber, Schuster, and Townsend 1999; Liu, Aizawa, and Yoshino 2004). The current focus of the aerospace industry on digital twins (Meyer et al. 2020; Arthur et al. 2020), and modern cyber-physical engineering design trends (Sztipanovits et al. 2012; Seshia et al. 2017) present a need for models of varying fidelity with varying computational performance demands.

The topic of condensation includes both the interfacial mass transfer (Gu, Min, and Tang 2018; Steeman et al. 2009) and the motion models for the condensate. The dropwise condensation models typically focus on a constant generation source to enable steady-state estimations of heat transfer rates on various surfaces (Weisensee et al. 2017; Grooten and Van Der Geld 2012). However, there is a gap when it comes to determining the residual moisture, or the droplets that are left after the temperature of the surface has risen above the dew point. Furthermore, many papers provide a good description of modeling methods (Rose and Glicksman 1973; Graham and Griffith 1973) but none have applied them to a Modelica model, and few have integrated the droplet physics models (Al-Sharafi et al. 2020; Pilat et al. 2012; Sun et al. 2020). This paper proposes a hybrid model that uses a detailed droplet force balance model in conjunction with the Rose distribution curve to determine the residual moisture on a surface. This Modelica model complements the existing work (Casella et al. 2006; Norrefeldt, Grün, and Sedlbauer 2012). The paper is organized as follows: section 2 describes the fundamental dropwise condensation equations, section 3 presents the Modelica implementation and its comparison with a Star CCM+ CFD model, section 4 reports the results of an implementation of the Modelica model that exercises all the functions the model, and section 5 is the conclusion.

2 Dropwise Condensation Equations

The conservation equations (1, 2, 3) are applied to a control volume, which represents the total volume of water on the surface. However, A force balance on the largest droplet on the surface is added to the momentum equation, integrating the physics of the control volume with that of the largest droplet on the surface. This allows the model to initiate droplet motion when the largest droplet reaches a critical size. It also simulates the sweeping of other droplets in the path of the largest droplet by integrating the control volume and the droplet physics.

$$\frac{\partial \rho}{\partial t}\forall = \dot{m}_x + \dot{m}_{x+\Delta x} + \Gamma \tag{1}$$

$$\frac{\partial \rho v}{\partial t}\forall = (\dot{m}v)_x + (\dot{m}v)_{x+\Delta x} + \zeta(\eta F_{ad} + F_g) \tag{2}$$

$$\frac{\partial \rho h}{\partial t}\forall = (\dot{m}h)_x + (\dot{m}h)_{x+\Delta x} + Q_{plate} + Q_{latent} \tag{3}$$

Where $\dot{m}$ is the liquid mass flow rate, $\forall$ is the total water volume, $\Gamma$is the interfacial mass transfer rate. The subscripts *ad* and *g* of the force term are adhesion and gravity. The subscripts x and x+δx indicate the upper and lower edge of a control volume. ζ and η are parameters that have been added to calibrate the model. ρ is density, h is specific enthalpy, and Q is heat transfer. Q_{plate} is the conductive and convective heat transfer from the plate to the water assuming the thickness of the largest droplet and Q_{latent} applies the heat of vaporization and condensation to the control volume.

The velocity of the water droplets is determined by the velocity factor (v_f,4).

$$\dot{m}_{x+\Delta x} = m\frac{v}{v_f} \tag{4}$$

The condensation rate is calculated using equations 5, 6, 7, 8, and 9. A, c, p, D, α, cp, subscript *inf*, and subscript s represent area, water vapor concentration, pressure, the diffusion factor, thermal diffusivity, specific heat, fluid free stream properties, and surface. The heat transfer coefficient (h) was set to 10 $\frac{W}{m^2 \deg K}$ and the diffusion factor was set to 2.6 e-5 $\frac{m^2}{s}$.h_m is the mass transfer coefficient.

$$\Gamma = h_m \rho_{H2Ovapor} A_s (c_s - c_{inf}) \tag{5}$$

$$h_m = \frac{h}{\rho_{air} c_{p-air}} (\frac{D}{\alpha_{air}})^{\frac{2}{3}} \tag{6}$$

$$c_s - c_{\text{inf}} \approx \frac{p_{sat} - p_{H2O,\text{inf}}}{p_{\text{inf}}} \tag{7}$$

$$A_{dry} = A_s - A_{wet} \tag{8}$$

The hyberbolic tangent function is used to prevent the evaporation function from producing a negative mass on the surface.

$$A_s(\Gamma, mass) = \begin{cases} A_{dry}, & \text{if } \Gamma > 0 \\ A_{dry}(\frac{1+tanh(\beta(mass - mass_{min}))}{2}), & \text{else} \end{cases} \tag{9}$$

The droplet adhesion force F_{AD} is determined by equations 10 and 11 with advancing and receding contact angles (θ_A and θ_R) for a hydrophilic surface (Al-Sharafi et al. 2020). The variables d, γ_{SL}, γ_{LV} are the droplet diameter, solid-liquid surface tension, and liquid-vapor surface tension. θ_{ave} is the average of the advancing and receding contact angles.

$$F_{AD} \approx \frac{24}{\pi^3} \gamma_{SL} d (cos(\theta_R) - cos(\theta_A)) \tag{10}$$

$$cos(\theta_{ave}) = \frac{\gamma_{SL}}{\gamma_{LV}} \tag{11}$$

The droplet distribution curve (Graham and Griffith 1973) (13) is critical to predicting the diameter of the largest droplet on the surface as a function of total water volume and the water droplet volume model (12).

$$\forall = \frac{\pi}{24} \frac{d^3}{sin^3 \theta_{ave}} [2 + cos\theta_{ave}][1 - cos\theta_{ave}]^2 \tag{12}$$

$$N_o = 0.05 d^{-2} \tag{13}$$

$$\forall_{tot} = B[d_{max} - d_{min}] \tag{14}$$

$$B = \frac{A_s \pi}{160} \frac{[2 + cos\theta_{ave}][1 - cos\theta_{ave}]^2}{sin^3 \theta_{ave}} \tag{15}$$

$$d_{max} = \frac{\forall_{tot}}{B} + d_{min} \tag{16}$$

$$A_{wet} = \frac{A_s \pi}{40} [ln(d_{max}) - ln(d_{min})] \tag{17}$$

3 Modelica Implementation and Verification

The modelica implementation sought to take advantage of existing Modelica Library components. The standard water model, fluid library ports, and thermal libary components were used as shown in figure 4. This model calculates the conservation equations for the water on the surface and the surface temperature. The entire model is displayed in figure 3. The water state was treated as a hyberbolic tangent function (18) to prevent events. The temperature of the water was limited to above freezing to prevent range errors. Due to this limitation, the temperature of the surface was used to determine the solid/liquid state of the water. The Modelica tables contain the advancing and receding contact angles. The thermal mass represents the mass of the plate.

The surface model includes one thermal port for contact with the air in the channel, another for contact with an exterior heat transfer source, a real input to receive the average velocity of the water flowing into the surface, a real output to report the same velocity flowing out of the surface, three fluid ports to transfer water by interfacial mass transfer (portHorizontal) and allow water to flow from the upper surfaces to the lower surfaces. The Modelica Library prescribed heat flow component was used to transfer heat from the surface to the water droplets.

$$WS = \frac{1 + tanh(\alpha(T - 273.15) + 1)}{2} \tag{18}$$

The surface model was connected to a buoyant air volume (blue-green box in figure 3), which calculates the condensation rate and transfers it across a fluid port. The buoyant air volume applies a pressure correction to the upper and lower fluid ports, which acts as a motive force for moist air to be drawn into the volume in the upper port and ejected out the lower port when the air is being cooled. The heat transfer port applies the convection heat transfer of the channel air to the surface and the real input port receives the dry surface area of the plate.

The Star-CCM+ CFD model used a fluid film model that included as inputs the nucleation density (N) and the minimum diameter radius diameter. It distributes the mass of water on the surface into a constant number of droplets, as specified by the nucleation density. A film begins to flow when the calculated droplet radius exceeds the minimum allowed, which is set by the user. A translation was developed to equivocate the film thickness (H), being the fraction of the volume of water to the surface area, to droplet diameter using the same nucleation density as the CFD model (Equation 19). The surface was maintained at 274°K and the air was supplied at at 300°K, with a velocity of 1.0 $\frac{m}{s}$, a moist air mass fraction of 0.023 $\frac{H2O}{totalmass}$ and pressure of 101,326 Pa.

$$d = \left(\frac{12H}{\pi N}\right)^{\frac{1}{3}} \tag{19}$$

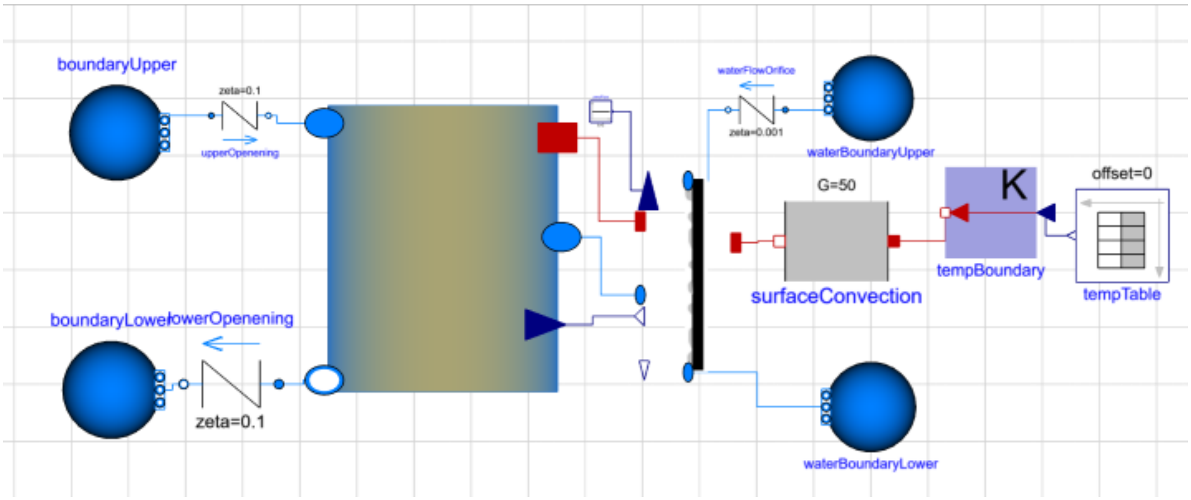


Figure 3. The 1-D model for comparison with the CFD model.

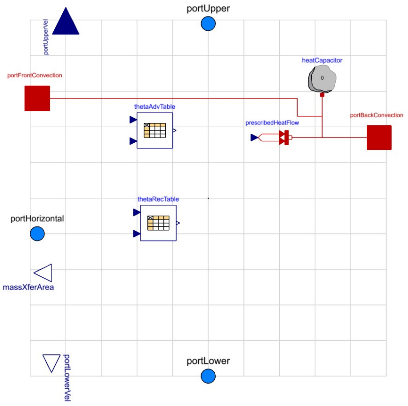


Figure 4. The surface model.

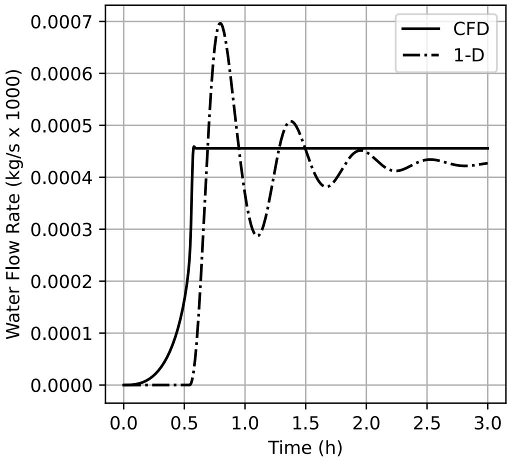


Figure 5. A comparison of the CFD and 1-D Modelica model water flow rates.

Figure 5 verifies that the drainage rates in the CFD and 1-D Modelica Model were close. Figure 6 verifies that the average film thickness was the same in both models.

The parameters ζ and v_f and η were varied to study their effect on the film thickness and water flow rate. ζ scales the value of the force balance on a single droplet. The velocity factor can be tuned to the average velocity of the droplets draining off the surface. The variable η was added to adjust the steady-state film thickness, the steady-state droplet diameter, and the departure point of the droplets (the moment when the water begins to drain). The results are shown in figures 7 to 10, with parameter values given in table 1.

The oscillatory behavior of figures 5 and 6 is caused by the momentum equation. ζ and v_f effect the frequency, amplitude, and decay rate of the water flow rate. v_f can be constrained to an experimentally observed average droplet velocity, leaving variation of ζ for final tuning of the model. It is dampened by decreasing ζ. The oscillatory behavior is a symptom of using a continuous conservation equation to describe a discontinuous process of droplets growing, sliding, and growing again. Once the velocity factor has been tuned ζ should be adjusted to ensure that the average film thickness response approximates the observed values.

Figures 7 and 8 present the effects of varying ζ. Increasing ζ minimizes the amount of initial water buildup on the surface and decreases the stabilization time. Figures 9 and 10 show that an increase in η increases the maximum steady-state volume of water that the surface will hold. η scales the adhesion force of the consevation equation (2).

4 Complete Model Simulation Results

The complete model simulates typical flight conditions by applying the temperature profile of figure 12 to a temper-

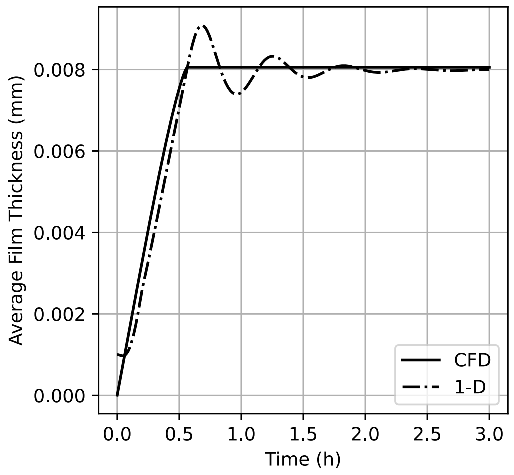


Figure 6. A comparison of the CFD and 1-D Modelica film thickness.

Table 1. Parameter Variations

Case Index	ζ	v_f (m)	η
1	0.5	1000	0.1
2	1.0	1000	0.1
3	1.5	1000	0.1
4	0.5	10	0.1
5	1.0	10	0.1
6	1.5	10	0.1
7*	1.0	10	1.0

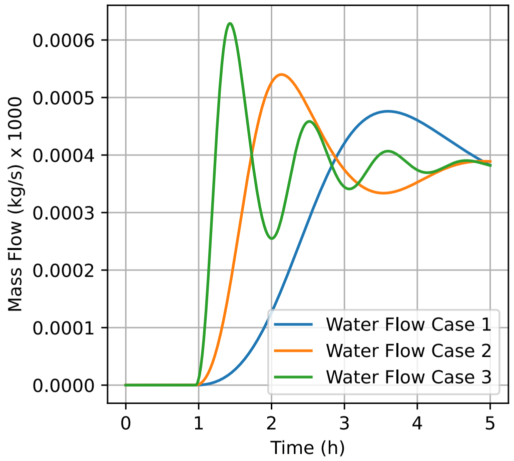


Figure 7. Parametric results water flow rate (cases 1 to 3).

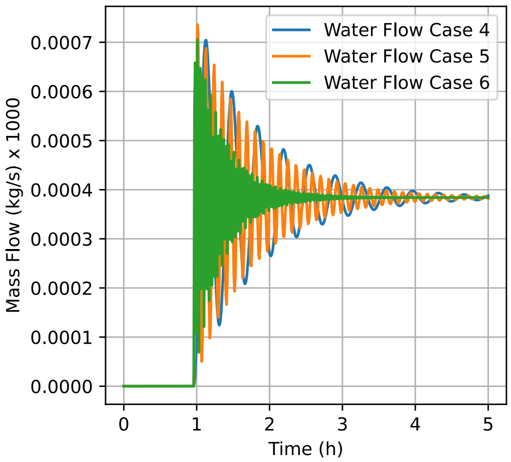


Figure 8. Parametric results water flow rate (cases 4 to 6).

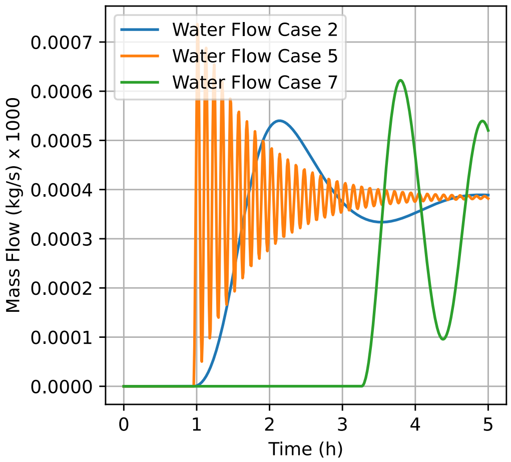


Figure 9. Parametric results water flow rate (cases 5 and 7).

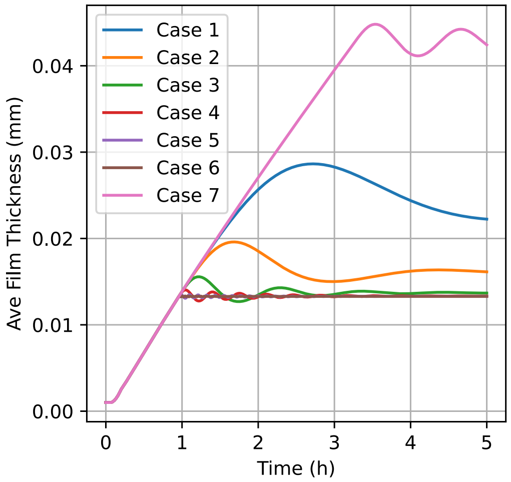


Figure 10. Parametric results average film thickness.

ature boundary condition. It also exercises all the connections of the surface model. The convective heat transfer coefficient being applied on the external side (right side, see figure 11) of the plate is so large that it acts as an infinite sink, nearly reaching the boundary condition temperature. The temperature profile represents a flight from a cold to a hot location. It reflects a winter to summer flight (across the equator). The mass of water on the surface is initially liquid, freezes in flight, and thaws as the flight arrives at the hot location. To further validate the

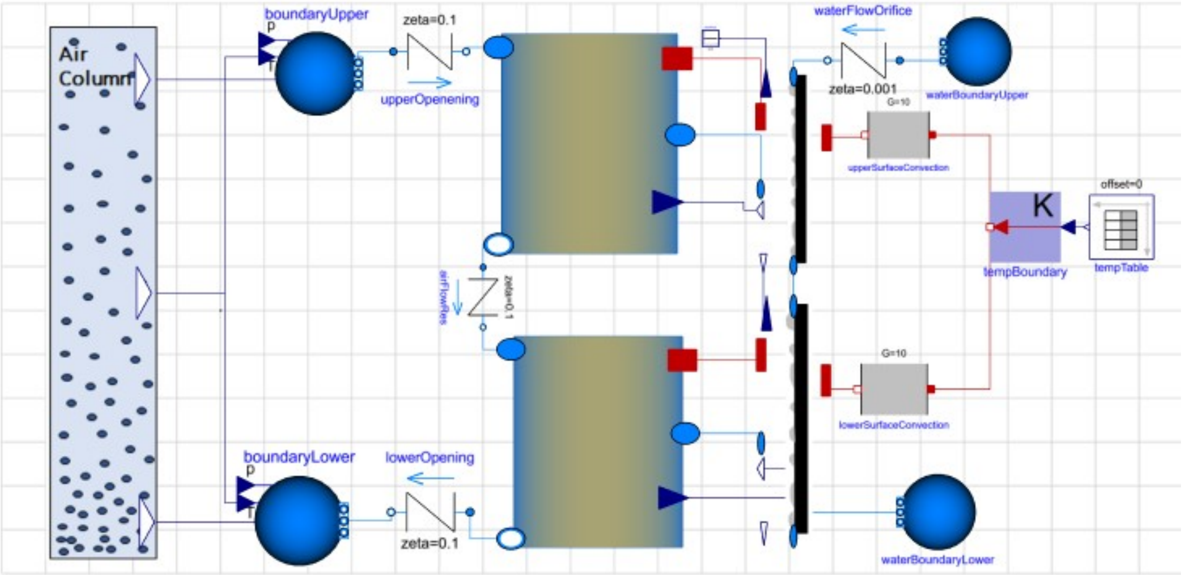


Figure 11. The complete model.

component behavior, this model includes an upper gap, a lower gap, an upper surface, and a lower surface (see figures 11 and 13). The model calculates the upper and lower opening pressure boundary with a buoyant air column of uniform temperature. The upper and lower openings are modeled using a simple generic orifice from the Modelica Fluid library. The fixed boundary fluid source named "waterBoundaryUpper" is only included to close the water flow network, its flow rate is zero.

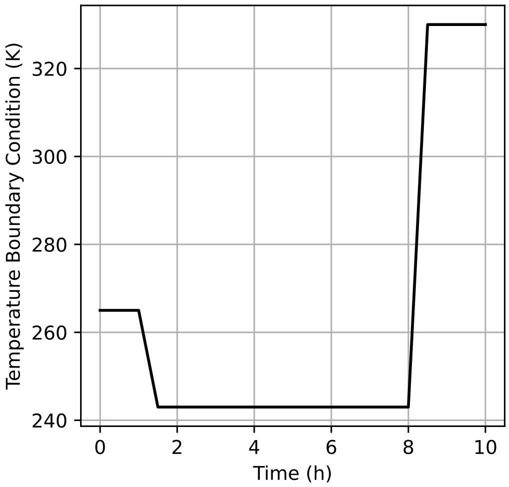


Figure 12. Realistic Temperature Profile.

The model simulates two connected sections of skin and insulation, with moist air and water flowing from the upper skin and air gap to the lower skin and air gap (see figure 13). Figures 14 to 20 show the thermodynamic state of the water, the droplet growth, and the flow rate of the water on the surface. The following observations can be made from figures 15 to 21: 1) the largest droplet diameter will increase when the surface temperature is below the freezing point of water beyond the critical droplet diameter, 2) upon melting gravity will quickly drain the water, 3) eventually evaporation will dominate the mass transfer, and 4) by the end of the flight the largest droplet diameter will be significantly diminished.

Elaborating on observation 1, the critical droplet diameter is reached when the gravity acting on mass of water in droplet is greater than the adhesion force securing the droplet to the surface. However, the model applies a scaling factor to the adhesion force to prevent water from draining when the surface temperature is below the freezing point of water. This allows the largest diameter of the surface to exceed the critical droplet diameter. See figures 15 and 19.

Regarding observation 2, the water drains until the critical diameter is reached. Figure 16 indicates that this occurs when the diameter is 0.82 mm. Figure 20 reveals that this happens 8.25 hours and 8.3 hours into the flight for the upper surface and lower surfaces, respectively.

According to observation 3, evaporation is the primary means of water removal once the critical diameter has been reached. Figure 17 shows that most of the mass flow of the water occurs as evaporation. A comparison of figures 20 and 21 reveals that while the peak mass flow rate of the drainage phase is larger than the evaporation rate, the integral of the evaporation is still larger than the integral of the drainage flow.

Finally, observation 4 exposes the limitations of the Modelica implementation method used for this model. The model required a minimum water mass control parameter to prevent solver instability as the surface dried. The sudden inflection at 9.2 hours (see figures 16 and 21) occurs because the actual mass of water on the surface is nearing the minimum mass.

The results verify that the buoyant air volume and the

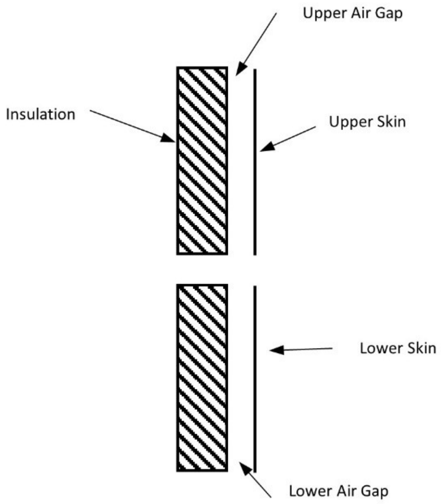


Figure 13. The complete simulation insulation system.

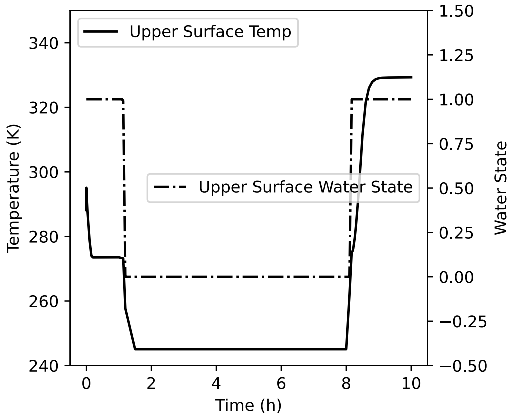


Figure 14. The temperature of the upper surface and the state of its water droplets.

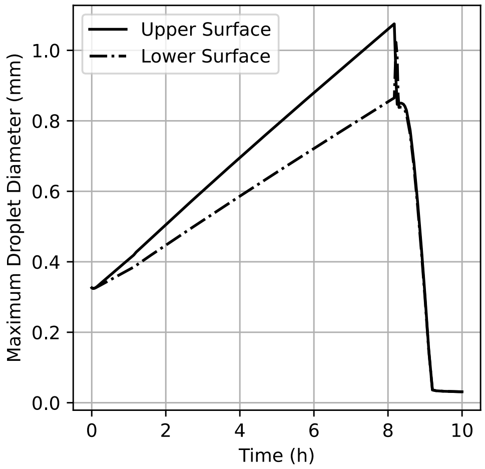


Figure 15. Size of the largest droplet on each surface throughout the flight.

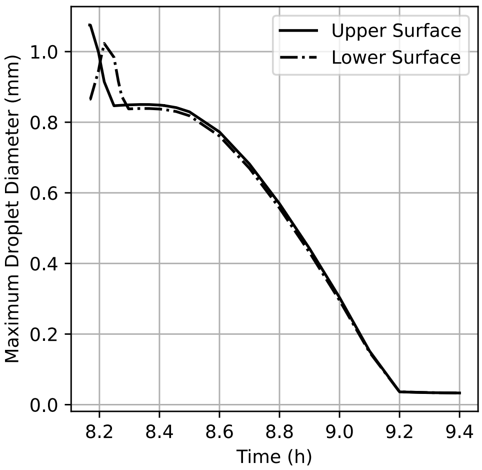


Figure 16. Size of the largest droplet after water has melted.

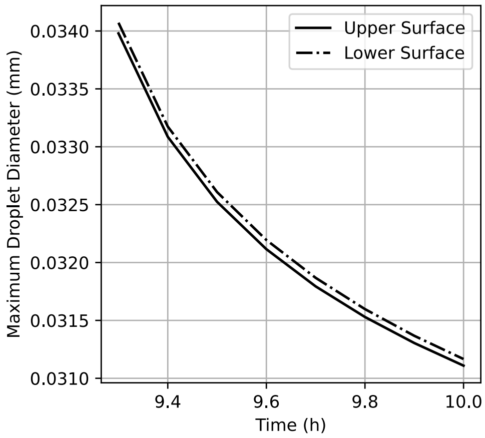


Figure 17. Size of the largest droplet at the end of the flight.

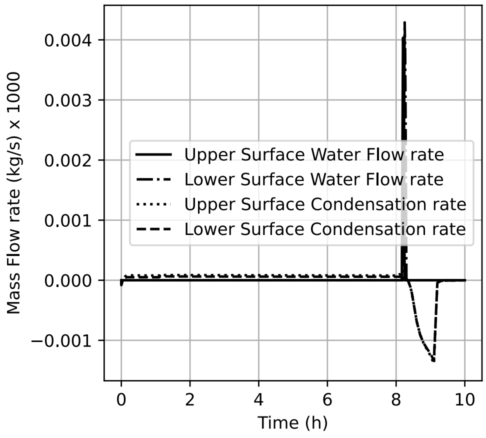


Figure 18. Drainage and condensation rates on each surface throughout the flight.

surface models can be combined to represent a system of connected channels and surfaces. The model runs quickly enough to accommodate any industrially meaningful time scale. Though an actual industrial scale analysis of an aircraft will include orders of magnitude more channels and surfaces. Condensation occurs very slowly in this exam-

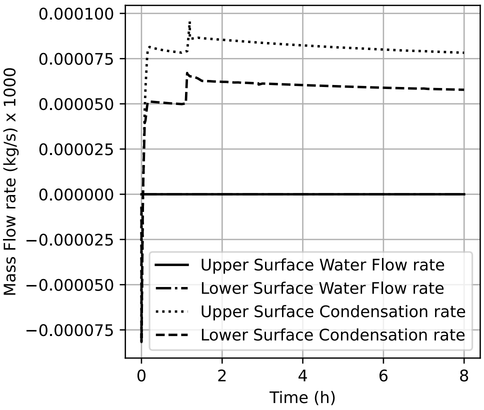


Figure 19. Condensation rates while the surface is frozen.

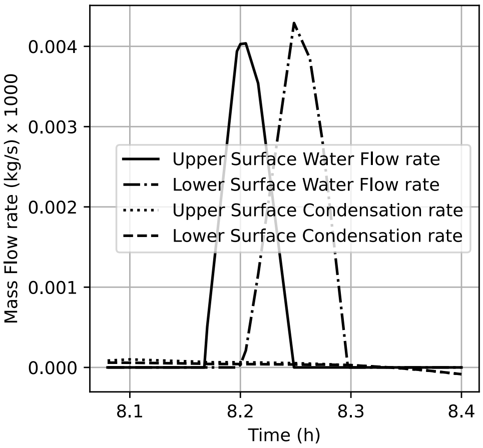


Figure 20. A zoomed in view of the drainage phase.

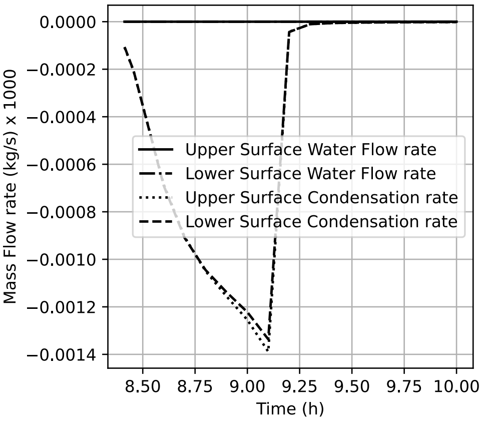


Figure 21. A zoomed in view of the evaporation phase.

ple. This agrees with industry experience, that undesirable conditions must persist over days to present an operational problem. The upper and lower surface temperatures were dropped below the freezing pointss of water in flight and are raised upon landing to exercise the evaporation functionality of the model. In summary, figure 14 shows that when the model transitions from frozen to liquid water without stability issues. Figure 15 demonstrates that the model simulates all the stages of the condensation process as expected. Figures 18 to 21 show how drainage and evaporation relate, that drainage occurs as a short term event while evaporation or condensation is always occurring.

Having simulated the droplet motion, the model may now be used to determine how much time is needed to evaporate the remaining water on the surface. This allows a more accurate prediction of the moisture load to which the insulation is exposed. Furthermore, a diffusion model could be applied to estimate the diffusion across the insulation lining.

The model was run using the desktop version of Modelon Impact (R). The solver selected was the CVode solver. It was run on PC with 2 Intel® Core^{TM} i7-557U CPUs @ 3.1GHz. The model took 3.3 minutes to run.

5 Conclusion

The reduced-order model described in this paper provides a means of estimating the residual moisture on a surface, has a very low computational requirement, and runs in minutes. Furthermore, it has been verified by comparison with a CFD model. It offers an advantage over CFD methods in that it can be easily applied to aircraft skin and insulation moisture modeling with large time scales. The 1-D model can be used to predict the residual water content at the end of a flight using the factors ζ, η, v_f. This model will support integration with a modeling ecosystem for product design, verification analysis, and digital twin methods. It supports moisture management system modeling for the day-to-day operation of an aircraft; i.e., the cycle of wetting, freezing, frost growth, thawing, draining, and drying. A standard water model that is valid below the freezing point would have simplified this drainage model.

Acknowledgements

The authors would like to thank Pieter Dermont of Modelon for his support with implementation of the software.

References

Arthur, Richard et al. (2020). *Digital twin: definition value*. Tech. rep. AIAA, pp. 1–16. URL: https://www.aia-aerospace.org/report/digital-twin-paper/.

Casella, Francesco et al. (2006). “The modelica fluid and media library for modeling of incompressible and compressible thermo-fluid pipe networks”. In: *Proceedings of the 5th International Modelica Conference*, pp. 631–640. URL: https://elib.dlr.de/47217/.

Connell, William J, Cameron L Carnegie, and Marcus K Richardson (2020). *Air drying system and method therefor*.

Connell, William J and Marcus K Richardson (2022). *Test cage for testing a gap in a vehicle*.

Graham, Clark and Peter Griffith (1973). “Drop size distributions and heat transfer in dropwise condensation”. In: *International Journal of Heat and Mass Transfer* 16.2, pp. 337–346. ISSN: 00179310. DOI: 10.1016/0017-9310(73)90062-8.

Grooten, M. H.M. and C. W.M. Van Der Geld (2012-04). “Surface property effects on dropwise condensation heat transfer from flowing air-steam mixtures to promote drainage”. In: *International Journal of Thermal Sciences* 54, pp. 220–229. ISSN: 12900729. DOI: 10.1016/j.ijthermalsci.2011.12.004.

Gu, L. D., J. C. Min, and Y. C. Tang (2018). “Effects of mass transfer on heat and mass transfer characteristics between water surface and airstream”. In: *International Journal of Heat and Mass Transfer* 122, pp. 1093–1102. ISSN: 00179310. DOI: 10.1016/j.ijheatmasstransfer.2018.02.061.

Huber, Paul, Karl Schuster, and Rob Townsend (1999). “Nuisance moisture”. In: *AERO*. URL: https://www.boeing.com/commercial/aeromagazine/aero_05/m/m01/index.html.

Khashayer, Borumand et al. (2019). *Modular environmental control chamber*.

Lents, Charles (2021). “Impact of weight, drag and power demand on aircraft energy consumption”. In: *AIAA Propulsion and Energy Forum, 2021*, pp. 1–6. DOI: 10.2514/6.2021-3322.

Liu, Jing, Hiroyoshi Aizawa, and Hiroshi Yoshino (2004). “CFD prediction of surface condensation on walls and its experimental validation”. In: *Building and Environment* 39.8, pp. 905–911. DOI: 10.1016/j.buildenv.2004.01.015.

Meyer, H et al. (2020). “Development of a digital twin for aviation research”. In: *Deutscher Luft- und Raumfahrt Kongress*, pp. 1–8. URL: https://elib.dlr.de/136848/.

Norrefeldt, Victor, Gunnar Grün, and Klaus Sedlbauer (2012). “VEPZO - velocity propagating zonal model for the estimation of the airflow pattern and temperature distribution in a confined space”. In: *Building and Environment* 48.1, pp. 183–194. ISSN: 03601323. DOI: 10.1016/j.buildenv.2011.09.007. URL: http://dx.doi.org/10.1016/j.buildenv.2011.09.007.

Pilat, D. W. et al. (2012-12). “Dynamic measurement of the force required to move a liquid drop on a solid surface”. In: *Langmuir* 28.49, pp. 16812–16820. ISSN: 07437463. DOI: 10.1021/la3041067.

Richardson, Marcus K, Brian T Imada, and Eric L Sarinas (2021). *Passive moisture management bladder in an aircraft*.

Rose, J. W. (1976). “Further aspects of dropwise condensation theory”. In: *International Journal of Heat and Mass Transfer* 19.12, pp. 1363–1370. ISSN: 00179310. DOI: 10.1016/0017-9310(76)90064-8.

Rose, J. W. and L. R. Glicksman (1973). “Dropwise condensation-The distribution of drop sizes”. In: *International Journal of Heat and Mass Transfer* 16.2, pp. 411–425. ISSN: 00179310. DOI: 10.1016/0017-9310(73)90068-9.

Seshia, Sanjit A. et al. (2017-09). “Design automation of cyber-physical systems: challenges, advances, and opportunities”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.9, pp. 1421–1434. ISSN: 02780070. DOI: 10.1109/TCAD.2016.2633961.

Al-Sharafi, Abdullah et al. (2020-01). “Adhesion of a water droplet on inclined hydrophilic surface and internal fluidity”. In: *International Journal of Adhesion and Adhesives* 96, pp. 1–10. ISSN: 01437496. DOI: 10.1016/j.ijadhadh.2019.102464.

Steeman, H. J. et al. (2009). “Evaluation of the different definitions of the convective mass transfer coefficient for water evaporation into air”. In: *International Journal of Heat and Mass Transfer* 52.15-16, pp. 3757–3766. ISSN: 00179310. DOI: 10.1016/j.ijheatmasstransfer.2009.01.047. URL: http://dx.doi.org/10.1016/j.ijheatmasstransfer.2009.01.047.

Sun, Yujin et al. (2020-04). “Spreading and adhesion forces for water droplets on methylated glass surfaces”. In: *Colloids and Surfaces A: Physicochemical and Engineering Aspects* 591, pp. 1–9. ISSN: 18734359. DOI: 10.1016/j.colsurfa.2020.124562.

Sztipanovits, Janos et al. (2012). “Toward a science of cyber-physical system integration”. In: *Proceedings of the IEEE*. Vol. 100. 1. Institute of Electrical and Electronics Engineers Inc., pp. 29–44. DOI: 10.1109/JPROC.2011.2161529.

Weisensee, Patricia B. et al. (2017). “Condensate droplet size distribution on lubricant-infused surfaces”. In: *International Journal of Heat and Mass Transfer* 109, pp. 187–199. ISSN: 00179310. DOI: 10.1016/j.ijheatmasstransfer.2017.01.119.

Wörner, Mario et al. (2002). “Theoretical and experimental investigation of the humidity transport in aircraft cabins”. In: *8th AIAA/ASME Joint Thermophysics and Heat Transfer Conference*, pp. 1–11. ISBN: 9781624101182. DOI: 10.2514/6.2002-3023.

Modeling Specialized Electric Power Generators, Excitation Systems and Prime Movers used by North American Utilities

Md Shamimul Islam[1] Giuseppe Laera[1] Marcelo de Castro Fernandes[1] Luigi Vanfretti[1]
Chetan Mishra[2] Kevin D. Jones[2]

[1]Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, United States,
{islamm7,vanfrl,laerag,decasm3}@rpi.edu
[2]Dominion Energy, United States, {chetan.mishra,kevin.d.jones}@dominionenergy.com

Abstract

The North American Electric Reliability Corporation (NERC) is expected to mandate model validation of power plant equipment in the near future. This will create a need to validate models for a large number of existing and future power plants. Historically, model validation of synchronous generators, excitation system, turbine governor, and other power system equipment has been conducted using diverse software platforms. As a contribution to the power system model implementation using Modelica language and validation against commercial tools, this work continues to develop power system component models and enriching the Open-Instance Power System Library (OpenIPSL). As a part of the development of OpenIPSL this paper describes the development of models used by North American utilities that follow NERC modeling requirements, including models of a synchronous generator, an excitation system, a turbine and governor using Modelica language in Dymola. The component implementation process is described and the validation of the models implemented in Modelica against PSS®E using both a single machine infinite bus (SMIB) and multi-machine system models is illustrated.

Keywords: Modelica, OpenIPSL, model validation, power systems, power grid.

1 Introduction

Precise mathematical modeling and simulation has become an integral part of different engineering tasks carried out by utilities and grid operators, as it aids in optimization of operations, planning of future expansions, and ensuring regulatory compliance. The ongoing efforts in de-carbonizaiton, rising energy costs, extreme climate events (Franke and Wiesmann 2014), etc., are challenging the resilience of power grids and may result in increased costs and may pose potential risks of disruption if not addressed timely. To address these challenges, engineers rely upon extensive computer simulations to understand, design and analyze the performance of power grids under diverse operating scenarios. In turn, these simulations enable stakeholders to identify vulnerabilities, implement improvements, and minimize risks associated with power delivery. Reliable and accessible modeling solutions of power system components and networks are essential for comprehensive analysis and decision making.

There are many different software tools available for power system modeling and simulation. The *de facto* standard tools used in the industry are domain-specific software, including PSS®E and PowerFactory, to name a few (Laera et al. 2022) concerned with power system dynamics and stability, with time-scales of tens of ms to tens of seconds. There is also software with a specific focus on ultra-fast time-scales, such as EMTP and PSCAD for analysis of electromagnetic transients and power electronic-based components. Despite meeting industry requirements and providing a vast library, there are limitations for closed-source software such as PSS®E. These include a lack of transparency regarding underlying algorithms and models, limited and less flexible modeling capabilities, limited simulation solver options, the requirement of a specific skill set for its use, rigid data format and high software cost, as well as limited community support.

To address these issues, there have been open-source initiatives to democratize research, development and overall access to alternative modeling tools in this field. Vanfretti et al. proposed the use of object-oriented equation-based modeling language Modelica to model power systems (Vanfretti et al. 2013). This started the efforts of the development of an open source power system library for power grid modeling and simulation consistent with the power industry practices and requirements, now called Open-Instance Power System Library (OpenIPSL) (Baudette et al. 2018). It is worth to note that other past efforts in power grid modeling with Modelica have been summarized in (Winkler 2017), with more recent efforts reported in (Adrien Guironnet et al. 2018) and (Bartolini, Casella, and A. Guironnet 2019). Meanwhile, outside the scope of power system dynamic modeling, other open-source initiatives that offer specific solutions to other analysis needs have emerged. These include OpenDSS(Montenegro, Dugan, and Reno 2017) for power distribution analysis, GridCal[1] and PyPSA (Brown, Hörsch, and Schlachtberger 2017; Parzen et al. 2023) etc., that allow

[1] See: https://gridcal.readthedocs.io/en/latest/

DOI
10.3384/ecp204653

to perform studies focused on the solution of steady-state problems, while also establishing a community to support such efforts.

Outside the power grid domain, significant research efforts have taken place to model various energy systems with Modelica such as modeling of building energy system modules (Wüllhorst, Maier, et al. 2022), Heat Ventilation and Air Conditioning systems (Wüllhorst, Storek, et al. 2022), and overall modeling of buildings for energy efficiency (Wetter et al. 2014), to name a few. These efforts show that there is tremendous potential in the Modelica “approach" towards modeling of different facets of energy systems. In this context, OpenIPSL aims to contribute to this growing body of Modelica libraries providing an open source and Modelica-based resource to address energy system needs. OpenIPSL offers different power grid component models with specific characteristics, such as a synchronous generator, excitation system, turbine governor/prime mover, power system stabilizer, load models, transformers, transmission line models, control unit, and so on, enabling modeling of power system dynamic modeling for generation, transmission and distribution systems (Castro et al. 2023).

This work aims to expand the existing capabilities of OpenIPSL by introducing the modeling of three new components using Modelica language and OpenIPSL library that are available in PSS®E and are required by the North American Electric Reliability Corporation (NERC) for the modeling of real-world power plants in the United States. Targeting specific power generation facilities in the Eastern Interconnection of the United States, new models of generators (GENTPJ), excitation systems (ESURRY), and prime movers (WPIDHY) that contain unique characteristics in terms of design, response, and functionality, are presented herein.

The specific contributions of this paper are:

- A Modelica language-based implementation of new and representative power generators, excitation systems, and prime mover models. These models will be included in the future release of the open-source OpenIPSL library.

  Generator (GENTPJ): A recent synchronous machine model (Birchfield et al. 2017) allowing the representation of sub-transient saliency and containing a much more complex saturation representation than existing models. Although being used by utilities and being available in the PSS®E software, this newer model is not currently in the IEEE standards. This model addresses some of the limitations of widely used GENROU and GENSAL models.

  Excitation System (ESURRY): This model is specialized for nuclear power plants in North American Utilities.

  Turbine Governor (WPIDHY):. This is a specialized model called Woodward PID Hydro Governor (WPIDHY) for hydraulic turbines and their speed control system used by utilities in the Eastern Interconnection. It includes a hydro turbine, governor and penstock representation for the modeling of hydro power plants using Woodward governor control systems. The model includes a nonlinear gate to power relationship and a linearized turbine and penstock model.

- Benchmarking the developed models by comparing the Modelica implementation results with the industry *de facto* standard propriety software tool PSS®E.
- A discussion of challenges and a framework for future development.

The reminder of this paper is organized as follows. Section 2 describes the Modelica implementation of the new electrical machine, excitation system, and turbine & governor models. Section 3 discusses the validation approach and explains the test procedure to test each of the components in the SMIB and multi-machine system. Section 4 describes the simulation results and discusses the validation results. Finally, Section 5 concludes the paper.

2 Modelica Implementation

In modeling power system components, it is critical to have a comprehensive understanding of the model's specifications and conceptual framework for its derivation. This understanding is largely domain-specific and gives a guide in the identification of relevant equations and/or block diagrams that accurately describe the system's dynamic behavior. Once these equations and/or block diagrams have been identified, they can be used to construct a model in Modelica.

One important stage in the modeling process is to determine how to initialize the model, which involves specifying how the initial values of the model's variables should be determined. After initialization, the model should be subjected to a software-to-software validation test, in which it is compared to a reference result (Otter et al. 2022) to ensure that the model's inputs and outputs are consistent with its specifications. This test is crucial in verifying the model's accuracy and ensuring it behaves as intended.

In summary, modeling a system requires a clear understanding of its specifications and conceptual framework, identification of relevant equations and/or block diagrams, construction of the model in Modelica, initialization, and validation through software-to-software testing.

In this article, the process summarized above is applied to three components used for the modeling of power generation systems in the Eastern Interconnection of the US. Namely, the components are: a generator GENTPJ (as GENTPJU1 in PSS®E), an excitation system ESURRY, and a turbine governor WPIDHY. The components have been modeled using Modelica language and verified against PSS®E. The model is presented as block diagrams in PSS®E's manual, which is used along with other literature to start the implementation in Modelica using developed components available in Modelica Standard Li-

DOI
10.3384/ecp204653

brary and OpenIPSL. These components were already developed and modeled in PSS®E.

2.1 Synchronous Machine (GENTPJ)

The Western Electricity Coordinating Council (WECC) has historically used the GENROU model to simulate round-rotor synchronous generators (used in steam- and gas-turbine power plants) and the GENSAL model to simulate salient-pole synchronous generators (used in hydropower plants). In recent years, WECC has begun to use the GENTPF model for round-rotor generators and a modified version of the GENTPF model called GENTPJ for salient-pole generators (Pourbeik et al. 2016). The GENTPJ model was first developed by (J. Undrill 2012) by introducing a new parameter K_{is}, which is a scalar multiplier of total stator current. Finally, NERC issued a "Modeling Notification(North American Electric Reliability Corporation 2018)," in November 18, 2016, recommending *to use the GENTPJ model for new modeling of salient pole generators and future (re)verification of salient pole generator models*.

Using GENTPJ, each synchronous machine is modeled in its rotor reference frame, i.e., rotating at the speed of the rotor. The electric source is represented by equations of the flux behavior in orthogonal dq-axes. When the system containing the generator model is subjected to a disturbance this results in an imbalance between the power generated and consumed, which in turn results in a speed deviation from its synchronous reference at the machine. Therefore, all of the machine variables are transferred to the synchronous reference frame.

The synchronous machine is implemented using Modelica in Dymola software by using the equations listed in (Pourbeik et al. 2016; Olive 1968; J. M. Undrill 1969). The GENTPJ model is similar to the GENROU, GENSAL, and to GENTPF, except the saturation function uses the K_{is} term. More details regarding the saturation function in the original 2007 and current 2012 specification are discussed in a presentation by BC Hydro (Cui 2022). The reader is refereed to (Zhang et al. 2015) for a discussion on the Modelica implementation of GENROU and GENSAL included in OpenIPSL. The equations that are used to implement the GENTPJ model using Modelica and the OpenIPSL library are listed in the Appendix. The meaning of the symbols are specified to (Schulz 1975; Kundur 1994), and they are discussed in (Pourbeik et al. 2016) and in the Modelica implementation within the annotations and comments in the Appendix.

Finally, it should be noted that when $K_{is} = 0$, GENTPJ can be used to represent the WECC Type F generator model, GENTPF.

2.2 Excitation System (ESURRY)

The excitation system is an essential component of a power generator, providing the necessary voltage/current to excite the generator's field winding. The ESURRY model can be seen as a modified version of the IEEE Type AC1A ("IEEE Recommended Practice for Excitation System Models for Power System Stability Studies" 2016) excitation system that was developed in PSS®E, and it is used to model the specialized excitation system in synchronous machines at nuclear power plants.

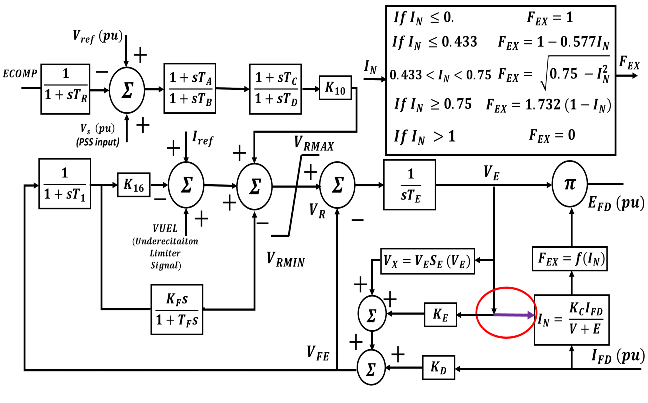


Figure 1. Block diagram of ESURRY from PSS®E (PTI 2017), corrected by adding the purple arrow circled in red.

The ESURRY model, as shown in Figure 1, consists of a non-controlled rectifier and an alternator. The model has three inputs: the generator terminal voltage, $ECOMP$, generator field current, I_{FD}, and an input for power system stabilizer (PSS) signal, V_S. The model's output is the exciter field voltage, E_{FD} , which is connected to the synchronous machine.

The non-controlled rectifier in the ESURRY model is responsible for rectifying the generator's output voltage and producing a DC voltage that is applied to the exciter's field winding. The alternator generates a small AC voltage that controls the rectifier's firing angle.

The ESURRY model also includes an input signal to couple a power system stabilizer (PSS), which dampens the generator's response to disturbances. The input signal provided by the PSS modulates the exciter's output voltage in response to system frequency or electrical load changes. This exciter model is implemented in Modelica using a block diagram, however, the implementation was challenging as detailed information or documentation to the different functions within the blocks were not available. During this study, a connection error was identified in the PSS®E reference manual that is corrected in this article with its location marked in Figure 1 in red[2], and the corrected diagram is shown in Figure 2. Note that the initialization equations are implemented in the text layer within the `initial equation` section of the model and therefore are omitted in the Figure.

2.3 Turbine Governor (WPIDHY)

The primary frequency control of synchronous machines is a critical function of a power plant. The governor achieves this by adjusting the mechanical power output

[2] The error is a missing input signal V_E used in the computation of I_N in the lower right corner of the diagram. The correction is shown as a purple arrow, added to the diagram taken from the manual.

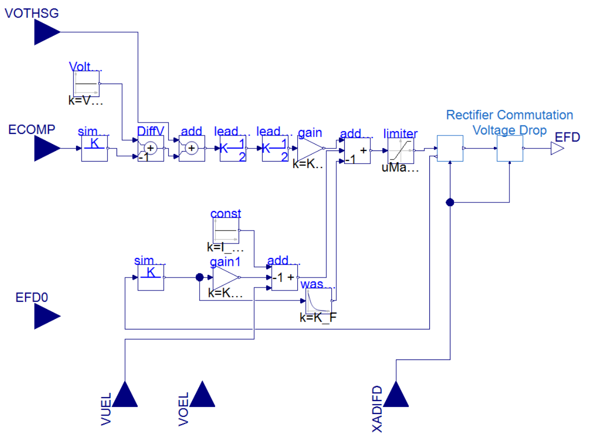


Figure 2. Implementation of ESURRY in Modelica using the OpenIPSL.

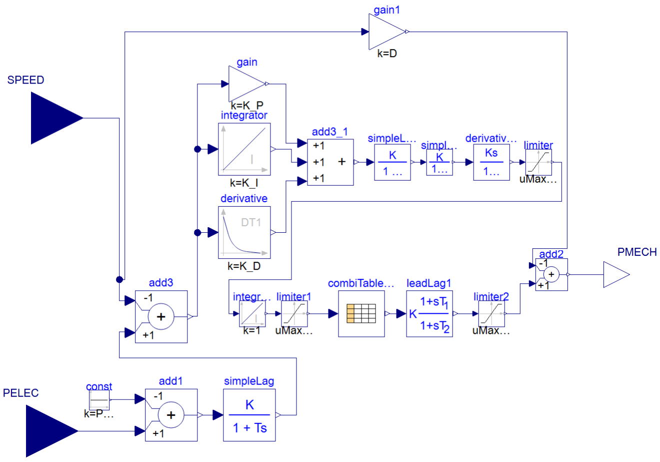


Figure 4. Implementation of WPIDHY in Modelica using the OpenIPSL

of the turbine in response to changes in electrical load or disturbances on the system.

The WPIDHY model, as shown in Figure 3, considers the rotor speed deviation $\Delta\omega$, electric power P_{ELEC}, and reference power P_{REF} as input signals. These signals are used to determine the mechanical power $PMECH$ to be applied to the generator. The model uses a proportional-integral-derivative (PID) control scheme. The PID controller compares the generator's speed with a predefined reference and adjusts the mechanical power output to maintain the desired frequency. Similar to ESURRY, the Modelica implementation was carried out using a block diagram, as shown in Figure 4. Note that the initialization equations were implemented in the text layer within the `initial equation` section of the model and therefore are omitted in Figure 4.

In Figure 3 after the integrator, a graph shows the relationship between gate position $(X - axis)$ and the turbine governor's corresponding output per unit $(Y - axis)$. Each data point signifies a unique mapping between the gate's position and power output. In the graph, the turbine generates no power starting at the zero gate position $(MBASE = 0)$. As the gate opens towards its maximum extent (one), the turbine reaches its peak capacity and produces the maximum possible per unit output $(MBASE = P_3)$. Between these two points lies a range of gate positions and their accompanying power outputs.

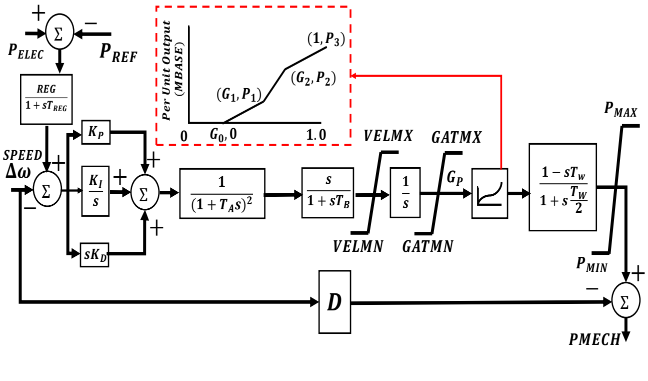


Figure 3. Block diagram of WPIDHY from PSS®E (PTI 2017).

The curve shows an exponential rise in power output between the zero gate position and G_1 gate position. At Gate Position G_1, the power output rises sharply before leveling off around Gate Position G_2. Beyond Gate Position G_2, minor variations in gate position yield only minimal gains in power output. Finally, the curve approaches the maximum power output $(MBASE = P_3)$ as the gate approaches its whole opening (one).

This graph serves multiple analysis purposes:

- It enables analysis of the minimum gate position needed to generate electricity G_1.
- It identifies the optimal gate position for maximum power output G_2.
- It suggests appropriate gate settings based on desired power output levels between G_1 and G_2.

Overall, understanding this graph is necessary for the effective modeling of hydro turbines with non-linear gate-to-power relationships, as in the case of WPIDHY.

3 Model Validation

3.1 Model Validation Procedure

In this paper, we followed the model validation approach according to (Laera et al. 2022), which is summarized as follows:

- Obtain the steady state computation results of a power flow solution in PSS®E.
- Export the results and provide them as initial guess values to solve the initialization problem of the corresponding SMIB in the Modelica-compliant software tool.
- Define the scenario for the dynamic simulation in both tools and run a dynamic simulation of the SMIB or other test system in both software.

DOI
10.3384/ecp204653

- Choose the quantities to compare and export them in the appropriate format to be used in another tool, for example, CSV Compare (`https://github.com/modelica-tools/csv-compare`) or funnel (`https://github.com/lbl-srg/funnel`).
- Use a tool (e.g. CSV Compare or funnel) to quantify the discrepancies between the simulation software tools after defining an acceptable tolerance level.
- The validation is complete if the errors between the quantities to compare are within the tolerance band.
- If the errors are more significant than the defined tolerance, then more model debugging is required.
- Compare the implemented model's sub-component inputs, outputs, and states with the analogous signals from the SMIB in PSS®E.
- Continue the iterative process until the signal difference is lower than the tolerance.

In the sequel, the model validation procedure described above is applied to perform a software-to-software validation of the models implemented in Modelica against PSS®E using two types of test systems.

3.2 Test in SMIB Models

The Single-Machine Infinite-Bus (SMIB) model is a simplified representation of a power network typically used to analyze the interconnection of generation facilities to the rest of the grid. The SMIB model includes one generator, one infinite bus, several transmission lines and/or transformers. Figures 5, 6 and 7 show the implementation of the test system, one for each of the components under test, all of which are located on the left side of the Figures, close to bus GEN1.

In Figures 5, 6 and 7, the infinite bus is modeled with a GENCLS machine with constant voltage and high inertia. It is connected to bus GEN2 to represent the interconnection to a stiff network. In contrast, the generator connected at bus GEN1 is composed of synchronous machine: GENTPJ in Figure 5, GENROU in Figure and GENSAL in Figure 7. When validating ESURRY, in Figure 6, this fast static excitation system model is added to the GENROU model. Meanwhile, to validate WPIDHY, this turbine and governor model is added to a GENSAL model.

To verify the simulation, the power network is perturbed during the simulation. To this end a three-phase-to-ground fault is applied on bus FAULT to the models in Figures 5, 6 and 7.

3.3 Test using a Real Power Plant Model

To further validate the developed models (GENTPJ, ESURRY, and WPIDHY), we perform the validation test of a power plant composed by two generators, connected to a similar network as used before, as shown in Figure 8. This simple model is representative of a specific generation station in the Eastern Interconnection of the US. According to Figure, this test system has two generation units, each consisting of a machine, an excitation system, a turbine governor, and a PSS. In Figure 8, the generators are connected to two buses, and then through a transformer, they are connected to the SMIB network we discussed earlier.

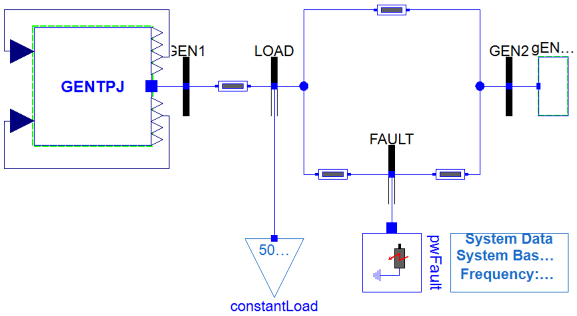


Figure 5. Implementation of the SMIB system to test GENTPJ in Modelica using the OpenIPSL.

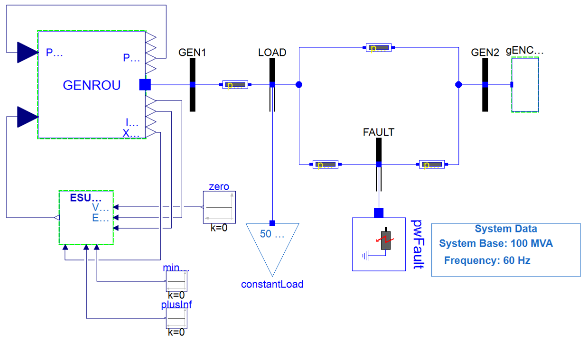


Figure 6. Implementation of the SMIB system to test ESURRY in Modelica using the OpenIPSL.

4 Results

To perform software-to-software validation of the newly implemented components in Modelica, we performed simulations in a simple SMIB network and multi-machine test system in both Dymola and PSS®E. In all simulation scenarios, a three-phase bus fault is applied to the FAULT bus at $t = 2\,s$ and cleared at $t = 2.15\,s$.

In the case of the unit test models in Figures 5, 6 and 7, simulation results are shown in Figures 9, 10, and 11, respectively. These results show the successful validation of individual components that are implemented for this work, i.e., GENTPJ, ESURRY and WPIDHY. These figures show that the Modelica implementation can produce the same results as PSS®E. Here we can see the generator terminal voltage, exciter field voltage, and mechanical power of the turbine governor for the models GENTPJ, ESURRY, and WPIDHY perfectly match the PSS®E for both steady state and dynamic response.

In the case of the real power plant model, Figures 12, 13, and 14 depict the Dymola vs. PSS®E validation results through the steady state and dynamic response of generator terminal voltage, active, reactive power, speed deviation, and exciter field voltage for the system in Figure. 8. Similar to the SMIB test system, the response

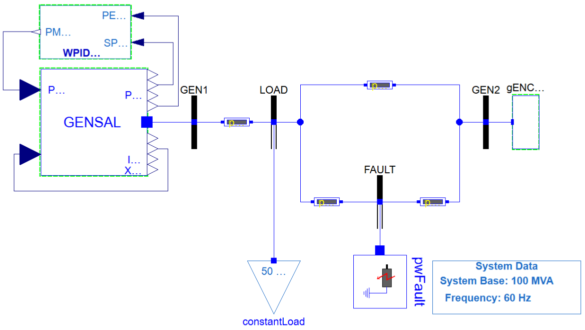


Figure 7. Implementation of the SMIB system to test WPIDHY in Modelica using the OpenIPSL.

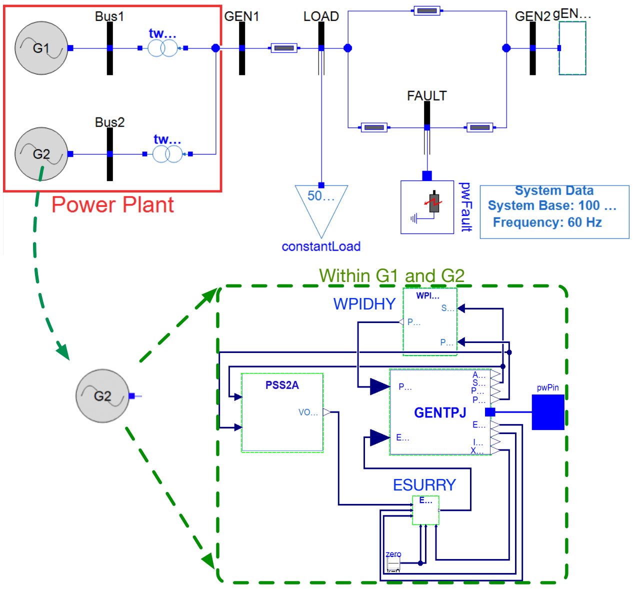


Figure 8. Implementation of the real world power plant model in Modelica using the OpenIPSL.

match that of PSS®E response; however, a small mismatch starts during the fault period. From these figures, we can see the mismatch starts disappearing at $t = 4.5s$, and after that, the dynamic response is the same in both Dymola and PSS®E. This difference is attributed to the differences on how the PSS®E handles the equations during the fault event, which are unknown to the authors.

5 Conclusion and Future Work

The detailed implementation of new power system component models used by North American utilities using the Modelica language, the OpenIPSL, and Dymola has been documented in this article. This paper summarizes the Modelica implementation of three power system components: a round rotor synchronous machine, an excitation system, and a turbine & governor system. Moreover, the implemented components were tested through using both the Dymola software and the PSS®E software for three simple test unit power system models (i.e., the SMIB) and multi-machine test system model representative of a power plant in the Eastern Interconnection of the US. Finally, the simulation results obtained using Dymola were compared against PSS®E. According to the results discussed in this paper, the Modelica implementation of power systems components performs similar to those of PSS®E. These models need to be tested in various test model setups to solve any possible issues that have not yet appeared during this work. Future work includes performing further simulation experiments to detect any remaining issues, performing validation and finally integrating the newly developed models into a future release of OpenIPSL.

Acknowledgements

This material is based upon work supported in part by Dominion Energy, in part by the National Science Foundation Award No. 2231677, and in part by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office, Award Number DE-EE0009139.

References

Bartolini, A., F. Casella, and A. Guironnet (2019-09). "Towards Pan-European Power Grid Modelling in Modelica: Design Principles and a Prototype for a Reference Power System Library". In: *Proceedings of the 13th International Modelica Conference*. Linköping Electronic Conference Proceedings 157:64. Regensburg, Germany: Modelica Association and Linköping University Electronic Press, pp. 628–636. ISBN: 978-91-7929-027-6. DOI: 10.3384/ecp19157627.

Baudette, Maxime et al. (2018). "OpenIPSL: Open-instance power system libraryupdate 1.5 to iTesla power systems library (iPSL): A modelica library for phasor time-domain simulations". In: *SoftwareX* 7, pp. 34–36.

Birchfield, Adam B et al. (2017). *Impact of Synchronous Generator Model GENTPJ on System Dynamics*. URL: https://adambirchfield.com/cv/gm2017_paper.pdf.

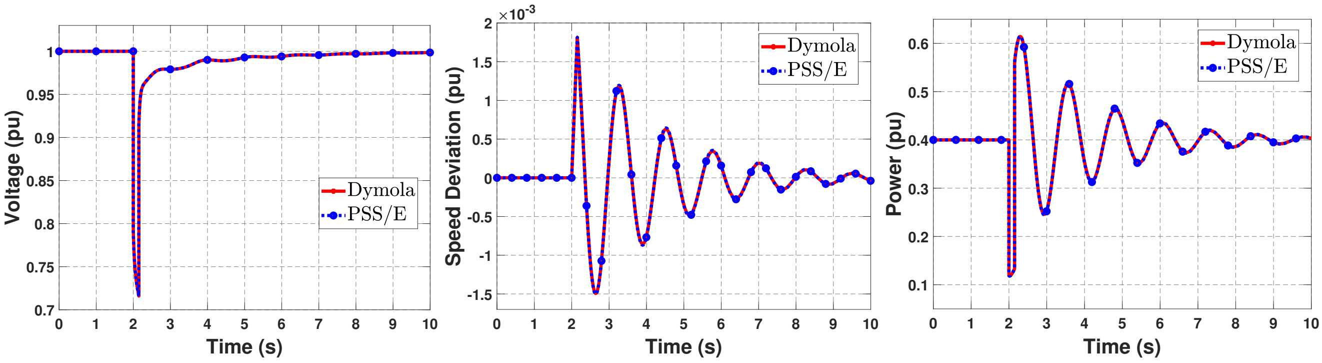
Brown, Tom, Jonas Hörsch, and David Schlachtberger (2017). "PyPSA: Python for power system analysis". In: *arXiv preprint arXiv:1707.09913*.

Castro, Marcelo de et al. (2023). "Version [OpenIPSL 2.0.0] - [iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations]". In: *SoftwareX* 21. DOI: https://doi.org/10.1016/j.softx.2022.101277. URL: https://www.sciencedirect.com/science/article/pii/S2352711022001959.

Cui, Philip (2022). *GENTPJ Model Saturation Function. WECC MVS Meeting*. URL: https://nerc.com/comm/pc/nercmodelingnotifications/use%20of%20gentpj%20generator%20model.pdf.

Franke, Rüdiger and Hansjürg Wiesmann (2014). "Flexible modeling of electrical power systems–the Modelica PowerSystems library". In: *Proceedings of the 10th International Modelica Conference*. Linköping Electronic Conference Proceedings 96:54. Lund, Sweden: Linköping University Electronic Press, pp. 515–522. ISBN: 978-91-7519-380-9. DOI: 10.3384/ecp14096515.

Guironnet, Adrien et al. (2018-10). "Towards an Open-Source Solution using Modelica for Time-Domain Simulation of Power Systems". In: *2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. DOI: 10.

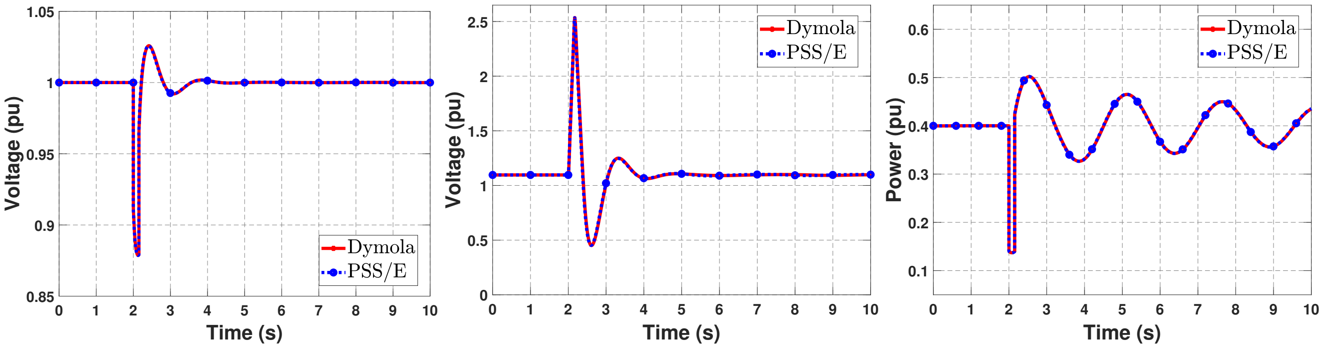


(a) Terminal voltage at bus GEN1.

(b) Speed deviation at bus GEN1.

(c) Active power at bus GEN1.

Figure 9. Generator terminal voltage, speed deviation, and active power of system in Figure 5.



(a) Terminal voltage at bus GEN1.

(b) Exciter field voltage at bus GEN1.

(c) Active power at bus GEN1.

Figure 10. Generator terminal voltage, exciter field voltage, and active power of system in Figure 6.

1109/isgteurope.2018.8571872. URL: https://doi.org/10.1109%2Fisgteurope.2018.8571872.

"IEEE Recommended Practice for Excitation System Models for Power System Stability Studies" (2016). In: *IEEE Std 421.5-2016 (Revision of IEEE Std 421.5-2005)*, pp. 1–207. DOI: 10.1109/IEEESTD.2016.7553421.

Kundur, Prabha (1994). *Power System Stability and Control*. McGraw-Hill.

Laera, Giuseppe et al. (2022). "Guidelines and Use Cases for Power Systems Dynamic Modeling and Model Verification using Modelica and OpenIPSL". In: *Proceedings of the American Modelica Conference 2022*. Dallas, Texas, USA, pp. 146–157. DOI: 10.3384/ECP21186146.

Montenegro, Davis, Roger C Dugan, and Matthew J Reno (2017). "Open source tools for high performance quasi-static-time-series simulation using parallel processing". In: *2017 IEEE 44th Photovoltaic Specialist Conference (PVSC)*. IEEE, pp. 3055–3060.

North American Electric Reliability Corporation (2018). *Modeling Notification. Use of GENTPJ Generator Model*. URL: https://nerc.com/comm/pc/nercmodelingnotifications/use%20of%20gentpj%20generator%20model.pdf.

Olive, David W. (1968). "Digital Simulation of Synchronous Machine Transients". In: *IEEE Transactions on Power Apparatus and Systems* PAS-87.8, pp. 1669–1675. DOI: 10.1109/TPAS.1968.292127.

Otter, Martin et al. (2022). "Towards Modelica Models with Credibility Information". In: *Electronics* 11.17. ISSN: 2079-9292. DOI: 10.3390/electronics11172728. URL: https://www.mdpi.com/2079-9292/11/17/2728.

Parzen, Maximilian et al. (2023). "PyPSA-Earth. A new global open energy system optimization model demonstrated in Africa". In: *Applied Energy* 341, p. 121096.

Pourbeik, Pouyan et al. (2016-10). "Modeling of synchronous generators in power system studies". In: *CIGRE Science & Engineering* 6, pp. 21–32.

PTI, Siemens (2017). *PSS/E 34.2.0 Model Library*. Siemens Power Technologies International. Schenectady, NY, USA.

Schulz, Richard P (1975). "Synchronous machine modeling". In: *IEEE Symposium on Adequacy and Philosophy of Modeling: Dynamic System Performance*, pp. 24–28.

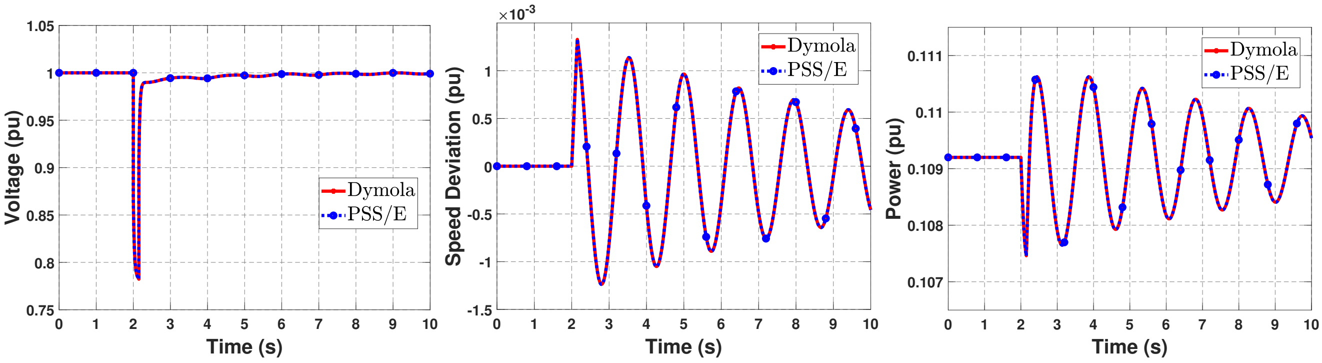
Undrill, John (2012). *The Gentpj Model*. Western Electricity Coordinating Council. URL: https://www.wecc.org/Reliability/gentpj-typej-model-specification.pdf.

Undrill, John M. (1969). "Structure in the Computation of Power-System Nonlinear Dynamical Response". In: *IEEE Transactions on Power Apparatus and Systems* PAS-88.1, pp. 1–6. DOI: 10.1109/TPAS.1969.292330.

Vanfretti, Luigi et al. (2013). "Unambiguous power system dynamic modeling and simulation using Modelica tools". In: *2013 IEEE Power & Energy Society General Meeting*. IEEE, pp. 1–5. DOI: 10.1109/PESMG.2013.6672476.

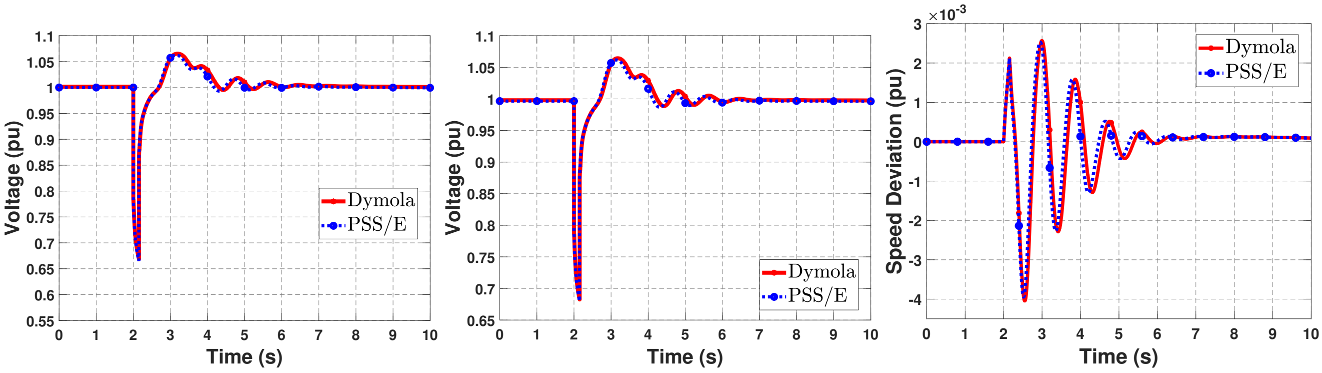
Wetter, Michael et al. (2014). "Modelica Buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506. URL: https://doi.org/10.1080/19401493.2013.765506.

Winkler, Dietmar (2017). "Electrical Power System Modelling in Modelica - Comparing Open-source Library Options". In: *Proceedings of the 58th Conference on Simulation and Modelling (SIMS 58)*, pp. 263–270.



(a) Terminal voltage at bus GEN1. **(b)** Speed deviation at bus GEN1. **(c)** Governor's mech. power at bus GEN1.

Figure 11. Generator terminal voltage, speed deviation, and mechanical power of system in Figure 7.



(a) Terminal voltage at Bus01. **(b)** Terminal voltage at Bus02. **(c)** Speed deviation at Bus01.

Figure 12. Terminal voltage and speed deviation system in Figure 8.

Wüllhorst, Fabian, Laura Maier, et al. (2022). "BESMod-A Modelica Library providing Building Energy System Modules". In: *Modelica Conferences*, pp. 9–18.

Wüllhorst, Fabian, Thomas Storek, et al. (2022). "AixCaliBuHA: Automated calibration of building and HVAC systems". In: *Journal of Open Source Software* 7.72, p. 3861.

Zhang, Mengjia et al. (2015-10). "Modelica Implementation and Software-to-Software Validation of Power System Component Models Commonly used by Nordic TSOs for Dynamic Simulations". In: *Proceedings of the 56th Conference on Simulation and Modelling (SIMS 56)*. Linköping Electronic Conference Proceedings 119 (2015). Linköping University, Sweden: Linköping University Electronic Press, pp. 105–112. ISBN: 978-91-7685-900-1. DOI: 10.3384/ecp15119105.

Appendix

GENTPJ Mathematical Model

The derivation of the dynamic equations of GENTPJ are described in detail in (Pourbeik et al. 2016), they are summarized as follows.

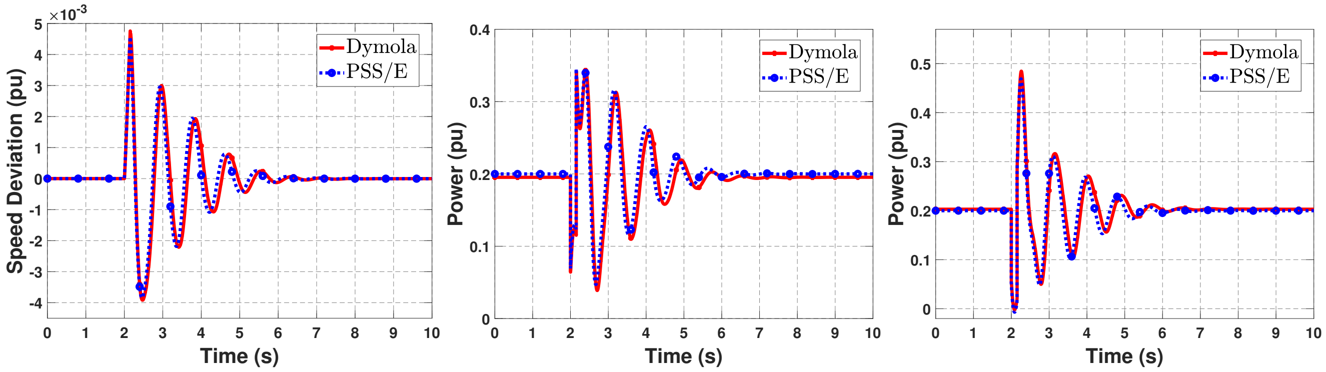
The differential equations are,

$$
\begin{aligned}
\frac{dE'_q}{dt} &= \left[E_{fd} - (1+S_d)E_{q1}\right]\frac{1}{T'_{do}},\\
\frac{dE'_d}{dt} &= -(1+S_q)\frac{E_{d1}}{T'_{qo}},\\
\frac{dE''_q}{dt} &= -(1+S_d)\left(\frac{X'_d - X''_d}{X_d - X''_d}\right)\frac{E_{q2}}{T''_{do}},\\
\frac{dE''_d}{dt} &= -(1+S_q)\left(\frac{X'_q - X''_q}{X_q - X''_q}\right)\frac{E_{d2}}{T''_{qo}}.
\end{aligned}
$$

The algebraic equations for the terminal voltage in *dq*-axis are given by

$$
\begin{aligned}
V_q &= E_{q1} + E_{q2} - I_q r_a - I_d\left(\frac{X_d - X_l}{1+s_d} + X_l\right),\\
V_d &= E_{d1} + E_{d2} - I_d r_a + I_q\left(\frac{X_q - X_l}{1+S_q} + X_l\right).
\end{aligned}
$$

Meanwhile the auxialiry equations for the *dq*-axis voltage behind the transient and sub-transient impedances are de-

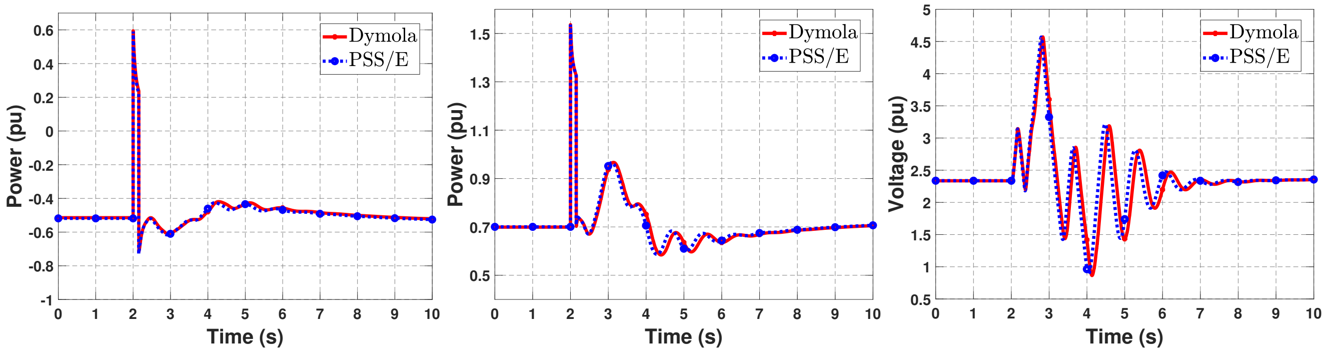


(a) Speed deviation at Bus02.

(b) Active power at Bus01.

(c) Active power at Bus02.

Figure 13. The system's speed deviation and active power in Figure 8.



(a) Reactive power at Bus01.

(b) Reactive power at Bus02.

(c) Field voltage at Bus02.

Figure 14. Reactive power and field voltage of system in Figure 8.

fine, for the q-axis, as

$$E_{q1} = E''_q - E_{q2} + I_d \left(\frac{X_d - X''_d}{1 + S_d} \right),$$

$$E_{q2} = \left[E''_q - E'_q + I_d \left(\frac{X'_d - X''_d}{1 + S_d} \right) \right] \left(\frac{X_d - X''_d}{X'_d - X''_d} \right),$$

and for the d-axis:

$$E_{d1} = E''_d - E_{d2} - I_q \left(\frac{X_q - X''_q}{1 + S_q} \right),$$

$$E_{d2} = \left[E''_d - E'_d - I_q \left(\frac{X'_q - X''_q}{1 + S_q} \right) \right] \left(\frac{X_q - X''_q}{X'_q - X''_q} \right).$$

Finally, the terminal voltage magnitude with leakage and the saturation is defined as:

$$E_l = \sqrt{(V_q + I_q r_a + I_d X_l)^2 + (V_d + I_d r_a - I_q X_l)^2}$$

$$S_d = f_S(E_l) \text{ and } S_q = \frac{X_q}{X_d} f_s(E_l).$$

where $f_s(E_l)$ saturation function introduced by quadratic open-circuit.

GENTPJ Modelica Implementation

Listing 1 presents the Modelica implementation of GENTPJ. Observe that in addition to dependencies to the Modelica Standard Library, there are important dependencies to the OpenIPSL. It is worth to note that the swing equations for the model are inherited from the base machine class `OpenIPSL.Machines.PSSE.baseMachine`, and therefore are not displayed within the listing below.

Listing 1. GENTPJ Partial Modelica Implementation

```
model GENTPJ "WECC Type J GENERATOR: ROUND
    ROTOR WITH SATURATION ON BOTH AXES."
  extends Icons.VerifiedModel;
  //Import of dependencies
  import Complex;
  import Modelica.ComplexMath.arg;
  import Modelica.ComplexMath.real;
  import Modelica.ComplexMath.imag;
  import Modelica.ComplexMath.abs;
  import Modelica.ComplexMath.conj;
  import Modelica.ComplexMath.fromPolar;
  import Modelica.ComplexMath.j;
  import OpenIPSL.NonElectrical.Functions.
      SE;

  extends BaseClasses.baseMachine(
    XADIFD(start=efd0),
    delta(start=delta0, fixed=true),
    id(start=id0),
    iq(start=iq0),
    Te(start=pm0),
    ud(start=ud0),
    uq(start=uq0));
  //Machine parameters
  parameter Types.PerUnit Xpq "q-axis
      transient reactance ";
  parameter Types.Time Tpq0 "q-axis
      transient open-circuit time constant"
      ;
  parameter Types.PerUnit Kis "Current
      multiplier for saturation calculation
      ";
  Types.PerUnit Epd(start=Epd0) "d-axis
      voltage behind transient reactance ";
  Types.PerUnit Epq(start=Epq0) "q-axis
      voltage behind transient reactance ";

  // Machine variable start values
  Types.PerUnit Eq1(start=Eq10);
  Types.PerUnit Eq2(start=Eq20);
  Types.PerUnit Ed1(start=Ed10);
  Types.PerUnit Ed2(start=Ed20);
  Types.PerUnit Xppdsat(start=Xppdsat0);
  Types.PerUnit Xppqsat(start=Xppqsat0);
  Types.PerUnit dsat(start=dsat0);
  Types.PerUnit qsat(start=qsat0);

  //State variables
  Types.PerUnit PSId(start=PSId0) "d-axis
      flux linkage ";
  Types.PerUnit PSIq(start=PSIq0) "q-axis
      flux linkage ";
  Types.PerUnit PSIppd(start=PSIppd0) "d-
      axis subtransient flux linkage ";
  Types.PerUnit PSIppq(start=PSIppq0) "q-
      axis subtransient flux linkage ";
  Types.PerUnit PSIpp "Air-gap flux ";
  Types.PerUnit XadIfd(start=efd0) "d-axis
      machine field current ";

protected
  parameter Complex Zs=R_a + j*Xppqsat0 "
      Equivalent impedance";
  parameter Complex VT=v_0*cos(angle_0) + j
      *v_0*sin(angle_0) "Complex terminal
      voltage";
  parameter Complex S=p0 + j*q0 "Complex
      power on machine base";
  parameter Complex It=real(S/VT) - j*imag(
      S/VT) "Complex current, machine base"
      ;
  parameter Complex Is=real(It + VT/Zs) + j
      *imag(It + VT/Zs) "Equivalent
      internal current source";
  parameter Complex PSIpp0=real(Zs*Is) + j
      *(imag(Zs*Is) - id0*(Xppqsat0-
      Xppdsat0))  "Sub-transient flux
      linkage in stator reference frame";
  parameter Types.Angle ang_PSIpp0=arg(
      PSIpp0) "flux angle";
  parameter Types.Angle ang_It=arg(It) "
      current angle";
  parameter Types.Angle ang_PSIpp0andIt=
      ang_PSIpp0 - ang_It "angle difference
      ";
  parameter Types.PerUnit abs_PSIpp0=abs(
      PSIpp0) "magnitude of sub-transient
      flux linkage";
```

```
parameter Complex Z = R_a+j*Xl;
parameter Complex PSIag= real(VT+Z*It) +
    j*(imag(VT+Z*It));
parameter Real dsat0=1+SE(
    (abs(PSIag)+Kis*sqrt(id0*id0+iq0*iq0)
        ),
    S10,
    S12,
    1,
    1.2) "To include saturation during
        initialization";
parameter Real qsat0=1+(Xq/Xd)*SE(
    (abs(PSIag)+Kis*sqrt(id0*id0+iq0*iq0)
        ),
    S10,
    S12,
    1,
    1.2) "To include saturation during
        initialization";
parameter Real a=(abs(PSIag))*dsat0;
parameter Real b=(It.re^2 + It.im^2)
    ^0.5*(Xppdsat0-Xq);
//Initializion rotor angle position
parameter Types.Angle delta0 = ang_PSIpp0
     + atan(b*cos(ang_PSIpp0andIt)/(b*sin
    (ang_PSIpp0andIt) - a))  "initial
    rotor angle in radians";
parameter Complex DQ_dq=cos(delta0) - j*
    sin(delta0)  "Parks transformation,
    from stator to rotor reference frame"
    ;
parameter Complex PSIpp0_dq=PSIpp0*DQ_dq
    "Flux linkage in rotor reference
    frame";
parameter Complex I_dq=conj(It*DQ_dq); //
    "The terminal current in rotor
    reference frame"
parameter Types.PerUnit PSIppq0=imag(
    PSIpp0_dq) "q-axis component of the
    sub-transient flux linkage";
parameter Types.PerUnit PSIppd0=real(
    PSIpp0_dq)  "d-axis component of the
    sub-transient flux linkage";
//Initialization of current and voltage
    components in rotor reference frame (
    dq-axes).
parameter Types.PerUnit iq0=real(I_dq) "q
    -axis component of initial current";
parameter Types.PerUnit id0=imag(I_dq) "d
    -axis component of initial current";
parameter Types.PerUnit ud0=(-PSIq0) -
    R_a*id0 "d-axis component of initial
    voltage";
parameter Types.PerUnit uq0=PSId0 - R_a*
    iq0 "q-axis component of initial
    voltage";
//Initialization current and voltage
    components in synchronous reference
    frame.
parameter Types.PerUnit vr0=v_0*cos(
    angle_0) "Real component of initial
    terminal voltage";
parameter Types.PerUnit vi0=v_0*sin(
    angle_0) "Imaginary component of
    initial terminal voltage";
```

```
  parameter Types.PerUnit ir0=-CoB*(p0*vr0
      + q0*vi0)/(vr0^2 + vi0^2) "Real
      component of initial armature current
       (system base)";
  parameter Types.PerUnit ii0=-CoB*(p0*vi0
      - q0*vr0)/(vr0^2 + vi0^2) "Imaginary
      component of initial armature current
       (system base)";
  //Initialization mechanical power and
      field voltage.
  parameter Types.PerUnit pm0=p0 + R_a*iq0*
      iq0 + R_a*id0*id0 "Initial mechanical
       power (machine base)";
  parameter Types.PerUnit efd0= dsat0*Eq10
      "Initial field voltage magnitude";
  parameter Types.PerUnit Epq0= PSIppd0 +
      id0*(Xpd-Xppd)/dsat0;
  parameter Types.PerUnit Epd0= -PSIppq0 -
      iq0*(Xpq-Xppq)/qsat0;
  parameter Types.PerUnit Eq10= ((-1)*
      PSIppd0*(Xd-Xpd) + Epq0*(Xd-Xppd))/(
      Xpd-Xppd);
  parameter Types.PerUnit Ed10= (PSIppq0*(
      Xq-Xpq)+Epd0*(Xq-Xppq))/(Xpq-Xppq);
  parameter Types.PerUnit Eq20= (PSIppd0-
      Epq0+id0*((Xpd-Xppd)/dsat0))*((Xd-
      Xppd)/(Xpd-Xppd));
  parameter Types.PerUnit Ed20= (-PSIppq0-
      Epd0-iq0*((Xpq-Xppq)/qsat0))*((Xq-
      Xppq)/(Xpq-Xppq));
  //Initialize remaining variables:
  parameter Types.PerUnit Xppdsat0=((Xppd-
      Xl)/dsat0)+Xl;
  parameter Types.PerUnit Xppqsat0=((Xppq-
      Xl)/qsat0)+Xl;
  parameter Types.PerUnit PSId0=PSIppd0 -
      Xppdsat0*id0;
  parameter Types.PerUnit PSIq0=PSIppq0 -
      Xppqsat0*iq0;
  // Constants
  parameter Real CoB=M_b/S_b  "Constant to
      change from system base to machine
      base";

initial equation
  der(Epd) = 0;
  der(Epq) = 0;
  der(PSIppd) = 0;
  der(PSIppq) = 0;

equation
  //Interfacing outputs with the internal
      variables
  XADIFD = XadIfd;
  ISORCE = XadIfd;
  EFD0 = efd0;
  PMECH0 = pm0;
  // Differential equations
  der(Epq) = (1/Tpd0)*(EFD - XadIfd);
  der(Epd) = (1/Tpq0)*(-1)*qsat*Ed1;
  der(PSIppd) = -(dsat)*((Xpd-Xppd)/(Xd-
      Xppd))*(Eq2/Tppd0);
  der(PSIppq) = (qsat)*((Xpq-Xppq)/(Xq-Xppq
      ))*(Ed2/Tppq0);
  Te = PSId*iq - PSIq*id;
  // Unsaturated air-gap flux
  PSIpp = sqrt((uq+iq*R_a+id*Xl)*(uq+iq*R_a
      +id*Xl)+(ud+id*R_a-iq*Xl)*(ud+id*R_a-
      iq*Xl));
  // Saturation on d-axis
  dsat=1+SE(
      ((PSIpp+Kis*sqrt(id*id+iq*iq))),
      S10,
      S12,
      1,
      1.2);
  // Saturation on q-axis
  qsat=1+(Xq/Xd)*SE(
      ((PSIpp+Kis*sqrt(id*id+iq*iq))),
      S10,
      S12,
      1,
      1.2);
  // Auxiliary Equations
  Eq1= ((-1)*PSIppd*(Xd-Xpd) + Epq*(Xd-Xppd
      ))/(Xpd-Xppd);
  Ed1= (PSIppq*(Xq-Xpq)+Epd*(Xq-Xppq))/(Xpq
      -Xppq);
  Eq2= (PSIppd-Epq+id*((Xpd-Xppd)/dsat))*((
      Xd-Xppd)/(Xpd-Xppd));
  Ed2=-(Epd+PSIppq)*((Xq-Xppq)/(Xpq-Xppq))-
      iq*((Xq-Xppq)/qsat);
  // Field Current
  XadIfd = dsat*Eq1;
  // Flux and saturated inductances
  Xppdsat=((Xppd-Xl)/dsat)+Xl;
  Xppqsat=((Xppq-Xl)/qsat)+Xl;
  PSId=PSIppd - Xppdsat*id;
  PSIq=PSIppq - Xppqsat*iq;
  // Terminal voltage
  ud = (-PSIq) - R_a*id;
  uq = PSId - R_a*iq;
end GENTPJ;
```

Numerically Efficient Degradation Model of Catalyst Layers in PEM Fuel Cells using Modelica

Jakob Trägner[1] Steffen Heinke[1] Wilhelm Tegethoff[1] Jürgen Köhler[1]

[1]Institut für Thermodynamik, TU Braunschweig, Germany, {j.traegner,s.heinke, w.tegethoff, j.koehler}@tu-braunschweig.de

Abstract

Degradation of the catalyst layer is a major challenge for the commercialization of polymer electrolyte membrane fuel cells (PEMFCs). Numerical modeling helps to understand and analyze the degradation phenomena, to transfer results from accelerated stress tests (ASTs) to real applications and to optimize operating conditions regarding degradation. We implemented a typical catalyst degradation model for platinum used in literature in Modelica. A numerical analysis shows the problem of "stiffness" for these models, meaning the tremendous difference in time constants. Assuming the platinum ion concentration in the ionomer to be in quasi-equilibrium helps to reduce the "stiffness", increases simulation speed and numerical robustness without any relevant inaccuracy. For a typical AST, the simulation speed can be more than doubled ending in a real-time factor of over 1,000. Thus, 500 hours of AST can be simulated within less than 30 minutes, which gives room for extensive analysis with the model.

Keywords: PEM Fuel Cells, Catalyst Degradation, Stiff System, Time Constants, quasi-equilibrium

1 Introduction

Polymer electrolyte membrane fuel cells (PEMFCs) are a promising technology which provides locally CO_2-free electrical energy. Their usage, e.g. in electric aircraft or fuel cell electric vehicles (FCEVs) can contribute to the announced aim of climate neutrality (European Union 2021).

PEMFCs use hydrogen at the anode and oxygen at the cathode to produce water, electrical energy and heat through the hydrogen oxidation reaction (HOR) and oxygen reduction reaction (ORR):

$$H_2 \xrightarrow{\text{HOR}} 2\,H^+ + 2\,e^-, \tag{1}$$

$$0.5\,O_2 + 2H^+ + 2\,e^- \xrightarrow{\text{ORR}} H_2O, \tag{2}$$

$$H_2 + 0.5\,O_2 \longrightarrow H_2O. \tag{3}$$

Figure 1 shows a schematic PEMFC. The HOR and ORR take place at the catalyst layers (CLs), which are placed on the membrane. In commercial PEMFCs, platinum or platinum alloys are used as catalyst material. Besides the catalyst particles, the CL consists of a porous support material, typically carbon, and the ionomer. The latter allows the transport of protons.

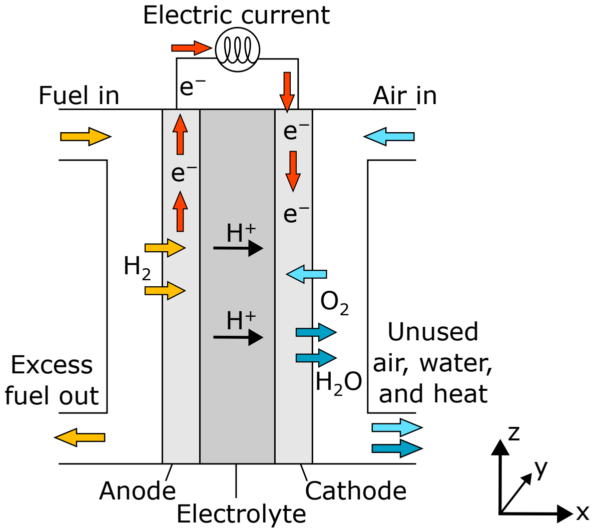


Figure 1. Schematic illustration of a PEMFC (Figure based on *Proton Exchange Fuel Cell Diagram* by Mattuci licensed under CC0 1.0 Universal Public Domain Dedication)

Platinum is costly and its degradation is a main contributor to PEMFC performance loss (Borup et al. 2020). To reduce costs and nevertheless keep the efficiency high, very small platinum particles in the range of nanometres are used, which have a high surface area to mass ratio. Those small particles are known to be less stable than bulk material and, hence, more prone to electrochemical platinum dissolution:

$$Pt \longleftrightarrow Pt^{2+} + 2\,e^-. \tag{4}$$

Platinum ions (Pt^{2+}) migrate through the ionomer to larger particles. That is why smaller particles are getting smaller and finally completely dissolute, while larger particles are growing. The size dependency of the platinum dissolution (Gibbs-Thomson effect) leads to the so-called electrochemical Ostwald-Ripening (Wagner 1961; Shao-Horn et al. 2007). The platinum surface decreases for a constant platinum mass in the catalyst layer, since larger particles have a lower surface to mass ratio. Figure 2 shows a schematic representation of this phenomena. The reduced catalyst surface area leads to increased activation losses (Zihrul et al. 2016; Bernhard et al. 2023), increased

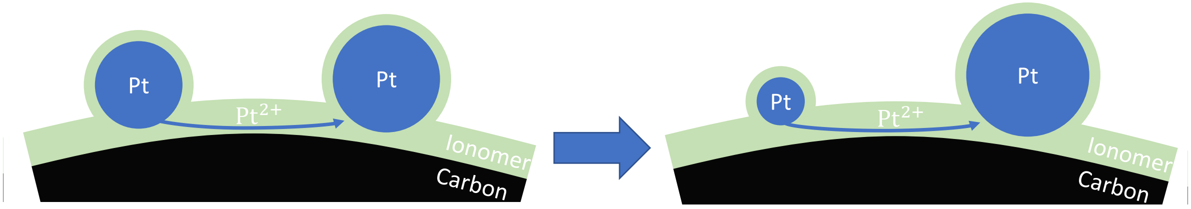


Figure 2. Schematic visualization of electrochemical Ostwald-Ripening. Smaller platinum particles are more prone to electrochemical dissolution, while platinum ions tend to re-deposit on larger particles. The increase of average particle size leads to a decreased platinum surface and accordingly lower cell voltage and fuel cell efficiency.

oxygen transport resistances in the catalyst layer (Greszler, Caulk, and Sinha 2012) and, hence, reduced cell voltage and efficiency.

Different groups have implemented catalyst degradation models for PEMFCs. Most of them use the kinetic model for platinum oxidation and dissolution initially proposed by Darling and Meyers (2003). Bi and Fuller (2008) and Darling and Meyers (2005) calculated the platinum dissolution not only for one but two particle groups with different radii which allowed to describe the reduction of electrochemical surface area (ECSA) for the first time. Later, Holby and Morgan (2012) calculated the platinum dissolution for a particle size distribution (PSD) approximated by several particle groups which allowed to describe the loss of ECSA more precisely. Li et al. (2015) used a similar model approach with a 1D through-plane (x-axis direction in figure 1) discretized CL. Schneider et al. (2019) added other degradation mechanisms like carbon corrosion and cathodic dissolution, i.e. dissolution during a reduction of potential, to the model. Jahnke et al. (2020) coupled the degradation model with a 2D along-the-channel (x and z-axis direction in figure 1) fuel cell model. Other contributions came from, among others, Rinaldo, Stumper, and Eikerling (2010), Zhang et al. (2013), Ahluwalia, Arisetty, Peng, et al. (2014), Kregar et al. (2019) and Prokop et al. (2019).

A lot of work was done with catalyst degradation models of the type based on Darling and Meyers (2003). However, to the best of the authors' knowledge, we present for the first time a numerical analysis for these kind of models and an implementation in the multi-physics modeling language Modelica. Assuming the platinum ion concentration to be in quasi-equilibrium, we propose a possibility to increase simulation speed and numerical robustness without relevant inaccuracies.

2 Modeling of Catalyst Degradation in PEM Fuel Cells

In the following section, the catalyst degradation model is described briefly to allow the reader to understand the differential equations. For the sake of simplicity, the following assumptions were made:

- Anodic platinum dissolution and, thus, Ostwald ripening is the main irreversible degradation mechanism. Neither coalescence nor chemical dissolution of platinum oxide is part of the model.
- Platinum ion diffusion in the membrane can be neglected. Thus, no Platinum band is forming.
- Platinum oxidation can be described with a simple one-step reaction mechanism without size effect on platinum oxidation. Sub-surface oxide is not taken into account. Thus, no cathodic dissolution takes place.
- The catalyst is pure platinum and the geometric surface area of the spherical particles is equal to the ECSA.
- The CL can be described with a 0D model with a uniform platinum ion concentration in the ionomer. This assumption is justified due to high electric conductivity (electric potential as the main stressor) and the high ratio of catalyst surface to ionomer volume.

Platinum particles in the catalyst layers exist with different sizes forming a PSD, which can typically be approximated by using a log-normal distribution. Figure 3 shows the used PSD. It is approximated using 20 equidistant distributed particle groups between $r_1 = 0.75\,\text{nm}$ and $r_{20} = 3.5\,\text{nm}$, where 20 is an arbitrary compromise between accuracy and simulation speed. The average diameter, standard deviation and platinum loading is based on the data from Jahnke et al. (2020), see appendix A. All particles in one group have the same radius and they shrink or grow due to electrochemical dissolution or re-deposition, which is called the radial evolution approach (Holby and Morgan 2012). The surface area of all particles can be described with the roughness factor rf (catalyst surface divided by geometric surface of the fuel cell) using the platinum loading L^i_{Pt} (platinum mass divided by geometric surface of the fuel cell) of each particle group i:

$$\text{rf} = \sum_{i=1} \left(\frac{3}{\rho_{\text{Pt}} r^i} L^i_{\text{Pt}} \right), \tag{5}$$

$$L^i_{\text{Pt}} = \frac{4}{3} \pi \rho_{\text{Pt}} t_{\text{CL}} (r^i)^3 n^i. \tag{6}$$

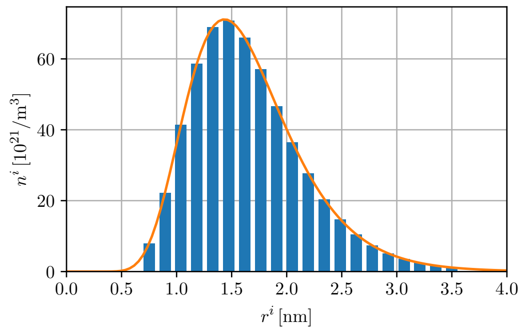


Figure 3. Initial PSD with continuous log-normal distribution and discrete particle groups

The radius r^i of a particle group changes due to Ostwald ripening. The volumetric specific number of particles n^i, the platinum density ρ_{Pt} and the catalyst thickness t_{CL} stay constant.

Platinum oxidation is assumed to follow the simple one step reaction proposed by Darling and Meyers (2003). Oxide coverage might dependent on the particle size. Ahluwalia, Arisetty, Wang, et al. (2013) measured an increasing coverage with particle size, while most models, including Darling and Meyers (2003), assume the opposite. That is why no dependency of the oxide coverage θ_{PtO} on the particle size is considered and the platinum oxide coverage can be described with one differential equation:

$$\frac{\mathrm{d}\theta_{\mathrm{PtO}}}{\mathrm{d}t} = \frac{R_{\mathrm{PtO}}}{\Gamma}, \tag{9}$$

where Γ is the number of active sites on a platinum surface and R_{PtO} is the reaction rate of platinum oxidation calculated according to equation 7. All parameters are listed in table 2.

The proton concentration c_{H^+} used in equation 7 is calculated, according to Darling and Meyers (2003), using the equivalent weight of the membrane EW, the density of the dry ionomer $\rho_{\mathrm{i,dry}}$, the molar weight of water M_{H2O} and its density $\rho_{\mathrm{H_2O}}$:

$$c_{\mathrm{H}^+} = \frac{1}{\frac{\mathrm{EW}}{\rho_{\mathrm{i,dry}}} + \frac{\lambda M_{\mathrm{H2O}}}{\rho_{\mathrm{H2O}}}}. \tag{10}$$

The water content λ is calculated according to Springer, Zawodzinski, and Gottesfeld (1991) and is only a function of relative humidity.

At high electric potential E, platinum tends to dissolve in the ionomer. The reaction rate for platinum dissolution R^i_{diss} is calculated according to equation 8. The higher the oxide coverage, the lower the platinum dissolution since the oxide protects the platinum. In the model, the surface fraction available for platinum dissolution θ_{av} is calculated by the simple relationship $\theta_{\mathrm{av}} = \max(0, 1 - \theta_{\mathrm{PtO}})$. The dissolution leads to an increase of platinum ion concentration $c_{\mathrm{Pt}^{2+}}$ in the ionomer of the catalyst layer, which is described by the differential equation:

$$\varepsilon_{\mathrm{i,CL}} \frac{\mathrm{d}c_{\mathrm{Pt}^{2+}}}{\mathrm{d}t} = 4\pi \sum \left((r^i)^2 n^i R^i_{\mathrm{diss}} \right), \tag{11}$$

where $\varepsilon_{\mathrm{i,CL}}$ is the ionomer volume fraction in the CL.

Small particles are more prone to dissolution, which is described using a constant surface energy σ_{Pt}:

$$E^{\mathrm{eq},i}_{\mathrm{diss}} = E^{\mathrm{eq,bulk}}_{\mathrm{diss}} - \frac{\sigma_{\mathrm{Pt}} M_{\mathrm{Pt}}}{2 r^i \rho_{\mathrm{Pt}} F}. \tag{12}$$

There seems to be a confusion about the radius dependency of the equilibrium potential for platinum dissolution in literature. Since the value for σ_{Pt} is taken from Darling and Meyers (2003), their formulation is used, too. However, one can find a factor of 2 or 3 in the numerator in different publications (Bi and Fuller 2008; Holby and Morgan 2012; Kaptay 2017; Jahnke et al. 2020).

The radius of the particle group i either shrinks or grows due to the dissolution rate R_{diss}, which adds another differential equation per particle group to the system of ordinary differential equations (ODEs):

$$\frac{\mathrm{d}r^i}{\mathrm{d}t} = \frac{-M_{\mathrm{Pt}}}{\rho_{\mathrm{Pt}}} R^i_{\mathrm{diss}}. \tag{13}$$

All in all, the model has initially $n+2$ differential states where $n = 20$ is the chosen number of initial particle groups (see above). The translated model has no nonlinear system to solve. Table 3 lists all differential states including the chosen nominal value within a typical range for that state.

Using the described radial evolution approach, smaller particles are getting smaller until they disappear. Since the equilibrium potential for platinum dissolution (equation 12) goes to negative infinity for a radius of zero, a minimum valid radius r_{min} is defined and set to 0.45 nm. If the radius of a particle group reaches this value, an event is triggered and the particles “disappear”. Thus, the radius is set to zero and an integer in the trigger vector is set to zero which deactivates the corresponding equations, i.e. equation 12 and 13 for the that particle group.

Listing 1. Event indicating and handling

```
for i in 1:n_groups loop
  when r[i] < r_min then
    reinit(r[i], 0);
    trigger[i] = 0;
  end when;
end for;
```

Degradation phenomena are typically measured in accelerated stress tests (ASTs). Such an AST is simulated for a temperature of 80 °C, a relative humidity of 80 % and an electric potential symmetrically changing within 0.5 s between a lower potential limit (LPL) of 0.6 V and an upper potential limit (UPL) of 0.95 V in a period of 5 s. The simulation was performed with the *DASSL* solver and a tolerance of 10^{-4}. Figure 4 shows the result for a

Table 1. Equations used to describe the reaction rates of platinum oxidation and dissolution

$$R_{\mathrm{PtO}} = k_{\mathrm{PtO}} \left[\exp\left(\frac{-\omega_{\mathrm{PtO}} \theta_{\mathrm{PtO}}}{RT} \right) \exp\left(\frac{\alpha_{\mathrm{PtO,ox}} z_{\mathrm{PtO}} F}{RT} (E - E^{\mathrm{eq}}_{\mathrm{PtO}}) \right) - \theta_{\mathrm{PtO}} \left(\frac{c_{\mathrm{H}^+}}{c^{\mathrm{ref}}} \right)^2 \exp\left(\frac{-\alpha_{\mathrm{PtO,red}} z_{\mathrm{PtO}} F}{RT} (E - E^{\mathrm{eq}}_{\mathrm{PtO}}) \right) \right] \tag{7}$$

$$R^i_{\mathrm{diss}} = k_{\mathrm{diss}} \theta_{\mathrm{av}} \left[\exp\left(\frac{\alpha_{\mathrm{diss,ox}} z_{\mathrm{diss}} F}{RT} \left(E - E^{\mathrm{eq},i}_{\mathrm{diss}} \right) \right) - \left(\frac{c_{\mathrm{Pt}^{2+}}}{c^{\mathrm{ref}}} \right) \exp\left(- \frac{\alpha_{\mathrm{diss,red}} z_{\mathrm{diss}} F}{RT} \left(E - E^{\mathrm{eq},i}_{\mathrm{diss}} \right) \right) \right] \tag{8}$$

Table 2. Parameter of the catalyst degradation model

Parameter	*Description*	*Value*	*Unit*	*Literature*
c^{ref}	Reference concentration	1×10^{-3}	$\mathrm{mol\,m^{-3}}$	Darling and Meyers (2003)
$E^{\mathrm{eq,bulk}}_{\mathrm{diss}}$	Equilibrium potential	1.188	V	Darling and Meyers (2003)
$E^{\mathrm{eq}}_{\mathrm{PtO}}$	Equilibrium potential	0.765 †	V	Darling and Meyers (2003)
EW	Equivalent weight of the ionomer	1.1	$\mathrm{kg\,mol^{-1}}$	
F	Faraday constant	96,485.33	$\mathrm{A\,s\,mol^{-1}}$	
k_{diss}	Reaction constant	3×10^{-6}	$\mathrm{mol\,m^{-2}\,s^{-1}}$	Bi and Fuller (2008)
k_{PtO}	Reaction constant	7×10^{-6}	$\mathrm{mol\,m^{-2}\,s^{-1}}$	Bi and Fuller (2008)
M_{Pt}	Molar mass	195×10^{-3}	$\mathrm{kg\,mol^{-1}}$	Darling and Meyers (2003)
$M_{\mathrm{H_2O}}$	Molar mass	18.02×10^{-3}	$\mathrm{kg\,mol^{-1}}$	
R	Molar gas constant	8.314	$\mathrm{J\,mol^{-1}\,K^{-1}}$	
z_{diss}	Number of electrons	2	1	Darling and Meyers (2003)
z_{PtO}	Number of electrons	2	1	Darling and Meyers (2003)
$\alpha_{\mathrm{diss,ox}}$	Transfer coefficient	0.5	1	Darling and Meyers (2003)
$\alpha_{\mathrm{diss,red}}$	Transfer coefficient	0.5	1	Darling and Meyers (2003)
$\alpha_{\mathrm{PtO,ox}}$	Transfer coefficient	0.4	1	Bi and Fuller (2008)
$\alpha_{\mathrm{PtO,red}}$	Transfer coefficient	0.1	1	Bi and Fuller (2008)
Γ	Number of active sites	2.18×10^{-5}	$\mathrm{mol\,m^{-2}}$	Darling and Meyers (2003)
$\varepsilon_{\mathrm{i,CL}}$	Ionomer volume fraction	0.3	1	Bi and Fuller (2008)
$\rho_{\mathrm{i,dry}}$	Density	2×10^3	$\mathrm{kg\,m^{-3}}$	
ρ_{Pt}	Density	21.45×10^3	$\mathrm{kg\,m^{-3}}$	Darling and Meyers (2003)
σ_{Pt}	Surface energy	2.37	$\mathrm{J\,mol^{-1}}$	Darling and Meyers (2003)
ω_{PtO}	Interaction parameter	30×10^3	$\mathrm{J\,mol^{-1}}$	Darling and Meyers (2003)

† Takes into account particle size effect according to Darling and Meyers (2003) for a constant particle radius of 2 nm.

Table 3. Differential states

State	*Unit*	*Nom.*	*Description*
θ_{PtO}	1	1	Platinum oxide coverage
$c_{\mathrm{Pt}^{2+}}$	$\mathrm{mol\,m^{-3}}$	10^{-3}	Platinum ion concentration
r^i	m	10^{-9}	Radius of particles in group *i*

simulation over 500 operating hours. In figure 4 (a), the evolution of the radii can be seen. Smaller particles are getting smaller and finally disappear, while larger particles are getting larger. The color bar indicates the initial radii of the particle groups. The electrochemical Ostwald ripening leads to a reduced catalyst surface, which is shown in figure 4 (b) expressed as the roughness factor rf. The kinks are due to the discretization of the PSD. After 500 h of AST, over 70 % of the initial catalyst surface is lost. This leads to a decrease of cell voltage and, thus, efficiency of the fuel cell (both not part of the model).

The loss of electrochemical surface area shown in figure 4 (b) has the typical characteristic of an high initial loss followed by a slower degradation, since the platinum particles radii has increased. Both, qualitative curvature and quantitative loss is comparable to the available literature mentioned in section 1. Nevertheless, for quantitative statements the model parameters in table 2 needs to be fitted to measurement data.

3 Numerical Analysis

The described irreversible catalyst degradation occurs over several hundreds or thousand of hours. Thus, the model should be much faster than real-time for its usage, e.g. in prognostic and health management (PHM) or for the optimization of operating conditions regarding

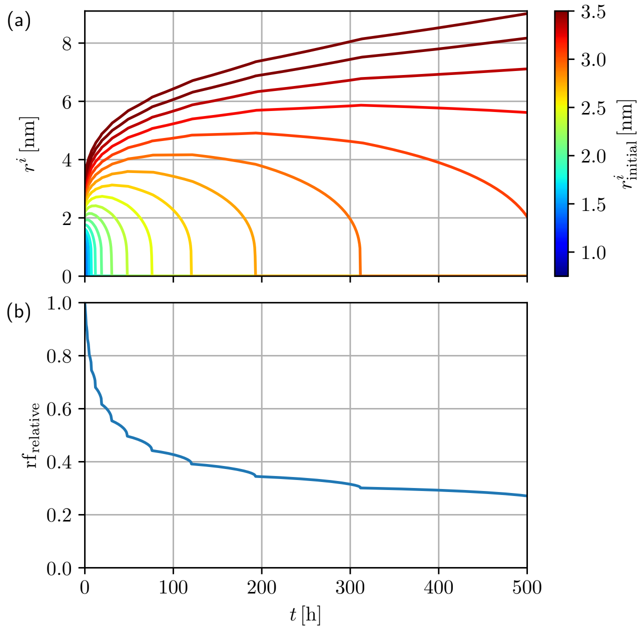


Figure 4. Simulation result for an AST over 500 hours. (a) Radial evolution. (b) Loss of electrochemical surface area expressed as roughness factor.

lifetime. A numerical analysis of the model may help to identify differential equations that slow down the simulation and find more efficient numerical formulations.

For that purpose, the linearized state space formulation is used:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \tag{14}$$

where **A** is the state or system matrix, **B** is the input matrix, **x** is called state vector and **u** is called input vector (Brenan, La Campbell, and Linda Ruth Petzold 1996).

For the described model using SI-units, the typical values of the differential states differ by several orders of magnitude. Visualizing **A** could be misleading since high absolute values are only due to the dimensions. E.g., a change in particle radius would have a massive impact on platinum ion concentration, since the particle radii are in the range of 1×10^{-9} (m) and concentration in the range of 1×10^{-3} (mol m^{-3}). That is why the state space system is normalized with respect to the nominal values n_i of the states (see table 3) and a typical height for a changing input h_i:

$$\begin{aligned}\begin{pmatrix} \frac{\dot{x}_1}{n_1} \\ \vdots \\ \frac{\dot{x}_n}{n_n} \end{pmatrix} &= \begin{pmatrix} a_{11} & \dots & a_{1n}\frac{n_n}{n_1} \\ \vdots & \ddots & \vdots \\ a_{n1}\frac{n_1}{n_n} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} \frac{x_1}{n_1} \\ \vdots \\ \frac{x_n}{n_n} \end{pmatrix} \\ &+ \begin{pmatrix} b_{11}\frac{h_1}{n_1} & \dots & b_{1n}\frac{h_n}{n_1} \\ \vdots & \ddots & \vdots \\ b_{n1}\frac{h_1}{n_n} & \dots & b_{nn}\frac{h_n}{n_n} \end{pmatrix} \begin{pmatrix} \frac{u_1}{h_1} \\ \vdots \\ \frac{u_n}{h_n} \end{pmatrix}. \end{aligned} \tag{15}$$

The input heights were chosen to $h_E = 0.1\,\mathrm{V}$ and $h_T = 10\,\mathrm{K}$ for the electrical potential and temperature at the CL, respectively. Neither the diagonal entries of matrix **A** nor the eigenvalues change due to this normalization.

The model was linearized and analyzed at typical conditions, i.e. an electric potential of 0.85 V, a temperature of 80 °C and a relative humidity of 80 % after reaching quasi-equilibrium using the "full linear analysis" method in *Dymola 2023x* and the *Modelica_LinearSystems2* package (DLR Institute of System Dynamics and Control 2020). Note, that due to degradation and the changing PSD no true equilibrium is reached.

Figure 5 shows a graphic visualization of the system matrix **A** and input matrix **B**. Dark gray is associated with a high absolute value, white are values close to zero or zero. Positive values are additionally marked as blue. Both matrices can be interpreted as follows: The column marks the changing variable (differential state in **A** or input in **B**) and the row marks the normalized change of derivative. The higher the value (dark gray), the higher the absolute impact of the changing variable on the state derivative. Exemplary, all radii have a relatively high impact on the derivation of the platinum ion concentration but no impact on the oxide coverage, since the oxide coverage does not depend on the particle radii (see section 2). However, the oxide coverage has an impact on the derivation of the radii. The impact is positive for small radii (marked as blue), since an increased oxide coverage leads to a reduced electrochemical dissolution and, thus, to a less negative change of particle radii (see equation 8). The largest absolute value on the main diagonal corresponds to the platinum ion concentration, indicating that this state has the smallest time constant.

In matrix **B**, the big impact of the electrical potential on all differential states can be seen. As expected and later further discussed, the impact on platinum ion concentration and oxide coverage is much higher than on the particle radii, since latter are changing much slower due to degradation.

The matrix **A** can be used to determine the eigenvalues λ of the system, since they are the root of

$$[\lambda\mathbf{I} - \mathbf{A}]\,\mathbf{x} = 0 \tag{16}$$

with the identity matrix **I**. The *eigvals-method* from *Scipy* in python is used to determine λ.

Table 4 lists the extracted eigenvalues and time constants τ. They are sorted and numbered from smallest to largest time constant. The eigenvalues can be interpreted as follows: Number one is associated with the platinum ion concentration in the ionomer. Number two is associated with the platinum oxide coverage. All others can be interpreted as the particle radii changing due to degradation. This irreversible degradation phenomena is very slow compared to the fast changing platinum ion concentration and the platinum oxide coverage. Hence, the time constants are much higher. However, for a system matrix **A** that is no triangular matrix, the contribution of the

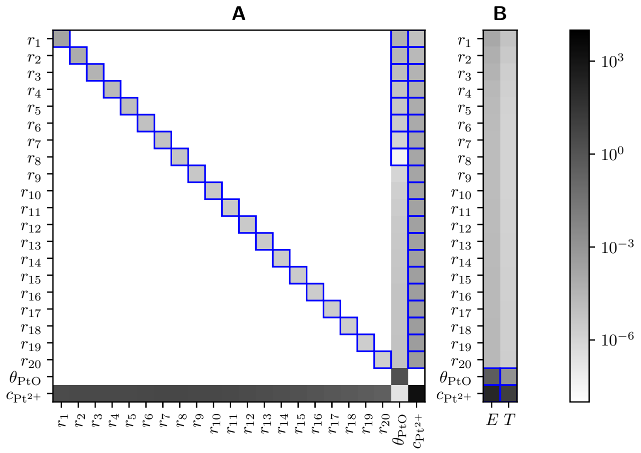


Figure 5. Visualization of the normalized matrices **A** and **B**. Absolute values of matrix entries are used for color bar. Positive values are marked blue.

eigenvalue to the continuous states is not unambiguously. Since 19 out of 22 eigenvalues are positive, the linearized system is not stable.

Table 4. Eigenvalues and time constants τ of the degradation model with dynamic platinum ion concentration

#	*associated*	*Eigenvalue*	*τ [s]*
1	$c_{\mathrm{Pt}^{2+}}$	-1.5×10^{3}	6.7×10^{-4}
2	θ_{PtO}	-1.7	6.0×10^{-1}
3	r^i	2.5×10^{-4}	4.0×10^{3}
…	r^i	…	…
21	r^i	2.1×10^{-6}	4.7×10^{5}
22	r^i	-1.6×10^{-6}	6.3×10^{5}

The ratio from largest to smallest eigenvalue is around 10^9. This large ratio indicates that the system of ODEs is a so called "stiff system" (T. D. Bui and T. R. Bui 1979). The problem of "stiffness" is discussed in literature for several decades. Nevertheless, there is no clear mathematical definition of "stiffness" (Hairer and Wanner 1991). It is typically defined as a system where explicit methods do not work or implicit methods are tremendously better (Curtiss and Hirschfelder 1952; Ascher and Linda R. Petzold 1998). In this work, we use the ratio from largest to smallest eigenvalue to quantify the "stiffness".

Another way to quantify the contribution of the different states to the numerical effort is the analysis of the *states which dominate error* or *limits step size* during integration. Values extracted from *Dymola 2023x* for the previously described simulation (figure 4) are listed in table 5. It can be seen that the platinum ion concentration in the ionomer $c_{\mathrm{Pt}^{2+}}$ dominates the error and is limiting the step size.

Table 5. Contribution of the different states to the numerical effort

state	*limits step size [%]*	*dominates error [%]*
θ_{PtO}	0.03	9.07
$c_{\mathrm{Pt}^{2+}}$	99.97	90.93
$\sum r^i$	0.00	0.00

4 Increasing Simulation Speed

The platinum ion concentration in the ionomer limits the step size and, thus, slows down the simulation. This can be explained with the very small time constant of the corresponding state, c.f. table 4. It is much smaller than the typical excitation signal, i.e. the change of temperature, electrical potential or relative humidity in the CL. Hence, we propose that it should be treated to be in quasi-equilibrium.

The left side of equation 11 is set to zero to calculate the platinum ion concentration in quasi-equilibrium:

$$0 = 4\pi \sum \left((r^i)^2 n^i R^i_{\mathrm{diss}} \right). \tag{17}$$

The explicit formulation

$$c_{\mathrm{Pt}^{2+}} = \frac{\sum \left[(r^i)^2 n^i \exp\left(\frac{\alpha_{\mathrm{diss,ox}} z_{\mathrm{diss}} F}{RT} \left(E - E^{\mathrm{eq},i}_{\mathrm{diss}} \right) \right) \right]}{\sum \left[\frac{(r^i)^2 n^i}{c^{\mathrm{ref}}} \exp\left(-\frac{\alpha_{\mathrm{diss,red}} z_{\mathrm{diss}} F}{RT} \left(E - E^{\mathrm{eq},i}_{\mathrm{diss}} \right) \right) \right]} \tag{18}$$

does not need to be implemented since Modelica can handle implicit formulations but might help others to implement it in programming languages that need explicit formulations. Note, that diffusion of platinum ions into the membrane (Pt-band) using Fick's law of diffusion can be easily integrated in equation 18. The new model uses equation 17 instead of equation 11. Thus, the model has $n+1$ differential states and still no nonlinear system of equations.

Figure 6 shows a comparison between the "classic" formulation where the platinum ion concentration is a differential state and the new formulation where the concentration is analytically calculated in quasi-equilibrium. At $t = 2\,\mathrm{s}$, the electric potential is jumping from 0.6 V to 0.95 V and at $t = 4\,\mathrm{s}$ back to 0.6 V, c.f. figure 6 (a). Figure 6 (b) shows the increasing platinum oxide coverage. The time constant in the sub-second range can be seen, c.f. eigenvalue 2 in table 4. Due to platinum dissolution, the ion concentration suddenly increases (figure 6 (c)). Only a small deviation between the two variants can be seen, visualizing the very small time constant for the platinum ion concentration. The slight increase in concentration between $t = 2\,\mathrm{s}$ and $t = 4\,\mathrm{s}$ is due to irreversible degradation, i.e. the changing PSD.

Note, that a potential jump is the scenario with the highest deviation between both variants. Typically, the potential is changing ramp-like within a second or more. Nevertheless, even for the case with potential jumping

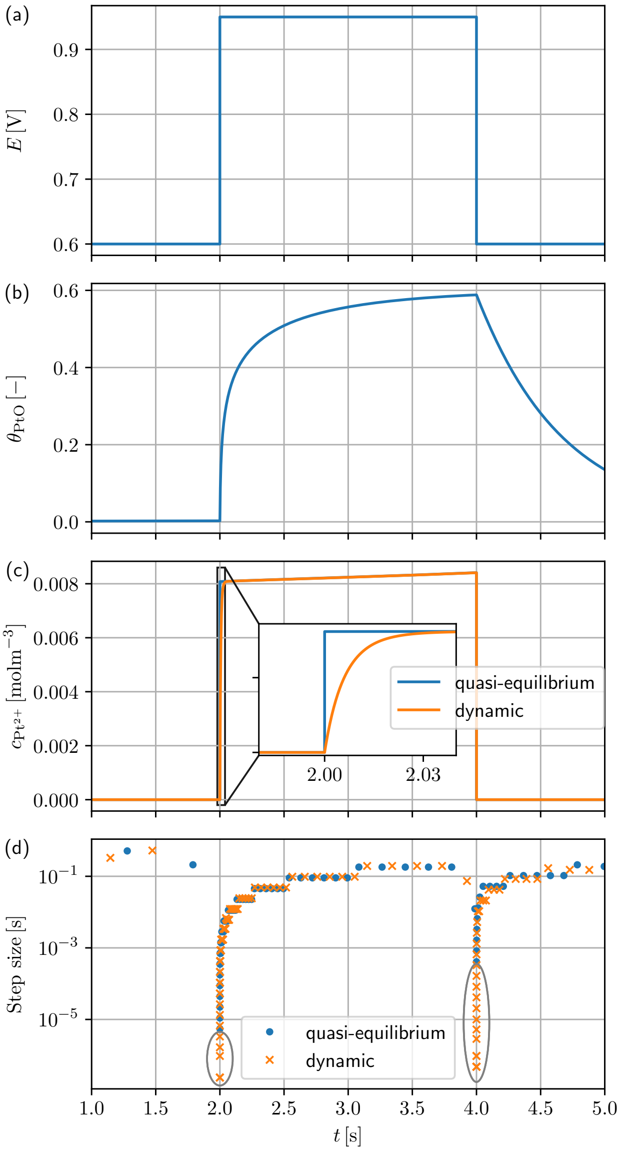


Figure 6. Comparison between the variant with platinum ion concentration in the ionomer as a differential state (orange) and in quasi-equilibrium (blue). (a) shows the electric potential, (b) shows the platinum oxide coverage (same for both variants), (c) shows the platinum ion concentration and (d) shows the steps sizes.

from 0.6 V to 0.95 V and vice versa every 2 s, the rf-loss (loss of electrochemical surface area) after 500 hours differs less than 0.1 % between both variants (not shown).

Figure 6 (d) shows the step sizes used for integration by *DASSL*. It can be seen that the step size is much smaller for the variant where platinum ion concentration is a differential state, especially after the decreasing potential at $t = 4$ s. Fast and large changes in electric potential with the “classic” formulation were also leading to situations where the platinum ion concentration did not converge for the minimum allowed step size. Those problems did not occur with the new formulation.

By calculating the platinum ion concentration in the ionomer in quasi-equilibrium it is possible to get rid of the smallest time constant and decrease the systems “stiffness”. The ratio from largest to smallest time constant can be reduced from 9.4×10^8 to 1.1×10^6. This helps to increase simulation speed and numerical robustness meaning less problems with convergence.

To quantify the increase of simulation speed, a real-time factor is introduced as the ratio of simulation time and CPU-time. The higher the real-time factor, the faster the simulation. All simulations were performed on a personal laptop computer with an AMD Ryzen 7 PRO 4750U (Base Clock 1.7 GHz). The CPU-time and, thus, the real-time factor, varies due to other processes on the computer and, since a solver with variable step size is used, on the chosen step sizes. The chosen steps are the same for each repeated identical simulation but can vary dramatically for different parameters or inputs. Therefore, 1,000 Monte-Carlo simulations are performed with different AST-profiles. A shape-factor is randomly chosen between 0 (square wave signal) and 1 (triangular wave). LPL is varied between 0.4 V and 0.7 V, UPL between 0.8 V and 1.2 V. The period of 5 s is kept constant for a simulation time of 1 h. Again, *DASSL* with a tolerance of 10^{-4} is used.

Figure 7 shows a comparison of the simulation speed for both variants. The thickness of the violin plots indicates the density of occurrence. The differences in simulation speed within one variant is mainly due to the variation of the potential profile. The variant with platinum ion concentration in quasi-equilibrium is much faster. The simulation time is nearly proportional to the number of F-evaluations (not shown), i.e. evaluations of the right hand sight (RHS) of the hybrid ODE. This indicates, again, that larger step sizes are possible due to the quasi-equilibrium formulation. In all 1,000 simulated cases, the variant with platinum ion concentration in quasi-equilibrium is faster than the dynamic variant (not shown). The average real-time factor was more than doubled ending in a factor of approximately 1,260. Thus, 500 hours of AST can be simulated within 23 minutes.

5 Summary and Discussion

A fast and efficient catalyst degradation model for PEMFC was introduced using the multi-physics modeling lan-

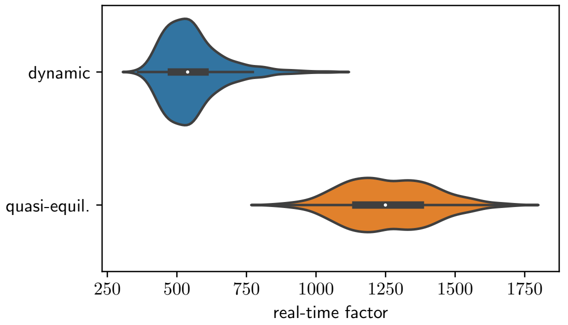


Figure 7. Real-time factor, i.e. the ratio of simulated time and CPU-time, for the variants with dynamic platinum ion concentration and concentration in quasi-equilibrium. The thickness of the violin plots indicates the density of occurrence.

guage Modelica. The model, predominantly based on Darling and Meyers (2003), can describe platinum oxide formation, platinum dissolution and, thus, electrochemical Ostwald ripening and the reduction of catalyst surface area.

Using the state space formulation, the system matrix **A** and the input matrix **B** were discussed and the eigenvalues respectively time constants were extracted. The problem of “stiffness” for this type of degradation model was discussed, meaning a tremendous difference in time constants. It was shown, that the time constant for the platinum ion concentration in the ionomer is much lower than the typical excitation signal, i.e. the change of the inputs temperature and electrical potential. Calculating the platinum ion concentration explicit in quasi-equilibrium removes the differential state with the smallest time constant, reduces “stiffness” and increases simulation speed without creating a nonlinear system of equations or relevant inaccuracies. Using the new formulation, the simulation speed could be more than doubled ending in an average real-time factor for a typical AST of over 1,000. Thus, 500 hours of AST can be simulated within less than 30 minutes which allows the usage of the model for extensive parameter studies, PHM and optimization, e.g. regarding the operating conditions.

References

Ahluwalia, Rajesh K., Srikanth Arisetty, Jui-Kun Peng, et al. (2014). “Dynamics of Particle Growth and Electrochemical Surface Area Loss due to Platinum Dissolution”. In: *Journal of The Electrochemical Society* 161.3, F291–F304. ISSN: 0013-4651. DOI: 10.1149/2.051403jes.

Ahluwalia, Rajesh K., Srikanth Arisetty, Xiaoping Wang, et al. (2013). “Thermodynamics and Kinetics of Platinum Dissolution from Carbon-Supported Electrocatalysts in Aqueous Media under Potentiostatic and Potentiodynamic Conditions”. In: *Journal of The Electrochemical Society* 160.4, F447–F455. ISSN: 0013-4651. DOI: 10.1149/2.018306jes.

Ascher, Uri M. and Linda R. Petzold (1998). *Computer methods for ordinary differential equations and differential-algebraic equations*. Philadelphia: Society for Industrial and Applied Mathematics. ISBN: 9780898714128.

Bernhard, David et al. (2023). “Model-assisted analysis and prediction of activity degradation in PEM-fuel cell cathodes”. In: *Journal of Power Sources* 562, p. 232771. DOI: 10.1016/j.jpowsour.2023.232771.

Bi, Wu and Thomas F. Fuller (2008). “Modeling of PEM fuel cell Pt/C catalyst degradation”. In: *Journal of Power Sources* 178.1, pp. 188–196. DOI: 10.1016/j.jpowsour.2007.12.007.

Borup, Rodney L. et al. (2020). “Recent developments in catalyst-related PEM fuel cell durability”. In: *Current Opinion in Electrochemistry* 21, pp. 192–200. ISSN: 24519103. DOI: 10.1016/j.coelec.2020.02.007.

Brenan, Kathryn Eleda, Stephen Vern La Campbell, and Linda Ruth Petzold (1996). *Numerical solution of initial-value problems in differential-algebraic equations*. Vol. 14. Classics in applied mathematics. Philadelphia: Society for Industrial and Applied Mathematics. ISBN: 0898713536.

Bui, T. D. and T. R. Bui (1979). “Numerical methods for extremely stiff systems of ordinary differential equations”. In: *Applied Mathematical Modelling* 3.5, pp. 355–358. ISSN: 0307-904X. DOI: 10 . 1016 / S0307 - 904X(79) 80042 - 6. URL: https://www.sciencedirect.com/science/article/pii/S0307904X79800426.

Curtiss, C. F. and J. O. Hirschfelder (1952). “Integration of Stiff Equations”. In: *Proceedings of the National Academy of Sciences of the United States of America* 38.3, pp. 235–243. ISSN: 0027-8424. DOI: 10.1073/pnas.38.3.235.

Darling, Robert M. and Jeremy P. Meyers (2003). “Kinetic Model of Platinum Dissolution in PEMFCs”. In: *Journal of The Electrochemical Society* 150.11, A1523. ISSN: 0013-4651. DOI: 10.1149/1.1613669.

Darling, Robert M. and Jeremy P. Meyers (2005). “Mathematical Model of Platinum Movement in PEM Fuel Cells”. In: *Journal of The Electrochemical Society* 152.1, A242. ISSN: 0013-4651. DOI: 10.1149/1.1836156.

DLR Institute of System Dynamics and Control (2020). *Modelica Linear Systems 2*.

European Union (2021). *Regulation (EU) 2021/1119 of the European Parliament and of the Council of 30 June 2021 establishing the framework for achieving climate neutrality and amending Regulations (EC) No 401/2009 and (EU) 2018/1999 (‘European Climate Law’): PE/27/2021/REV/1*. URL: http://data.europa.eu/eli/reg/2021/1119/oj.

Greszler, Thomas A., David Caulk, and Puneet Sinha (2012). “The Impact of Platinum Loading on Oxygen Transport Resistance”. In: *Journal of The Electrochemical Society* 159.12, F831–F840. ISSN: 0013-4651. DOI: 10.1149/2.061212jes.

Hairer, Ernst and Gerhard Wanner (1991). *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Vol. 14. Springer eBook Collection Mathematics and Statistics. Berlin, Heidelberg: Springer. ISBN: 978-3-662-09949-0. DOI: 10.1007/978-3-662-09947-6.

Holby, Edward F. and Dane Morgan (2012). “Application of Pt Nanoparticle Dissolution and Oxidation Modeling to Understanding Degradation in PEM Fuel Cells”. In: *Journal of The Electrochemical Society* 159.5, B578–B591. ISSN: 0013-4651. DOI: 10.1149/2.011204jes.

Jahnke, Thomas et al. (2020). “Physical Modeling of Catalyst Degradation in Low Temperature Fuel Cells: Platinum Oxidation, Dissolution, Particle Growth and Platinum Band Formation”. In: *Journal of The Electrochemical Society* 167.1, p. 013523. ISSN: 0013-4651. DOI: 10.1149/2.0232001JES.

Kaptay, George (2017). “A new paradigm on the chemical potentials of components in multi-component nanophases within multi-phase systems”. In: *RSC Advances* 7.65, pp. 41241–41253. DOI: 10.1039/C7RA07911G.

Kregar, Ambrož et al. (2019). “Predictive virtual modelling framework for performance and platinum degradation modelling of high temperature PEM fuel cells”. In: *Energy Procedia* 158, pp. 1817–1822. ISSN: 18766102. DOI: 10.1016/j.egypro.2019.01.426.

Li, Yubai et al. (2015). “A One-Dimensional Pt Degradation Model for Polymer Electrolyte Fuel Cells”. In: *Journal of The Electrochemical Society* 162.8, F834–F842. ISSN: 0013-4651. DOI: 10.1149/2.0101508jes.

Prokop, M. et al. (2019). “Degradation kinetics of Pt during high-temperature PEM fuel cell operation part I: Kinetics of Pt surface oxidation and dissolution in concentrated H3PO4 electrolyte at elevated temperatures”. In: *Electrochimica Acta* 313, pp. 352–366. ISSN: 00134686. DOI: 10.1016/j.electacta.2019.04.144.

Rinaldo, Steven G., Jürgen Stumper, and Michael Eikerling (2010). “Physical Theory of Platinum Nanoparticle Dissolution in Polymer Electrolyte Fuel Cells”. In: *The Journal of Physical Chemistry C* 114.13, pp. 5773–5785. ISSN: 1932-7447. DOI: 10.1021/jp9101509.

Schneider, Patrick et al. (2019). “Fast and Reliable State-of-Health Model of a PEM Cathode Catalyst Layer”. In: *Journal of The Electrochemical Society* 166.4, F322–F333. ISSN: 0013-4651. DOI: 10.1149/2.0881904jes.

Shao-Horn, Y. et al. (2007). “Instability of Supported Platinum Nanoparticles in Low-Temperature Fuel Cells”. In: *Topics in Catalysis* 46.3-4, pp. 285–305. ISSN: 1022-5528. DOI: 10.1007/s11244-007-9000-0.

Springer, T. E., T. A. Zawodzinski, and S. Gottesfeld (1991). “Polymer Electrolyte Fuel Cell Model”. In: *Journal of The Electrochemical Society* 138.8, pp. 2334–2342. DOI: 10.1149/1.2085971.

Wagner, Carl (1961). “Theorie der Alterung von Niederschlägen durch Umlösen (Ostwald–Reifung)”. In: *Zeitschrift für Elektrochemie, Berichte der Bunsengesellschaft für physikalische Chemie* 65.7–8, pp. 581–591. ISSN: 0005-9021. DOI: 10.1002/bbpc.19610650704.

Zhang, Hao et al. (2013). “The Impact of Potential Cycling on PEMFC Durability”. In: *Journal of The Electrochemical Society* 160.8, F840–F847. ISSN: 0013-4651. DOI: 10.1149/2.083308jes.

Zihrul, Patrick et al. (2016). “Voltage Cycling Induced Losses in Electrochemically Active Surface Area and in H 2 /Air-Performance of PEM Fuel Cells”. In: *Journal of The Electrochemical Society* 163.6, F492–F498. ISSN: 0013-4651. DOI: 10.1149/2.0561606jes.

A Initial Particle Size Distribution

The initial PSD (figure 3) is approximated using 20 particle groups with equidistant distributed radii between 0.75 nm and 3.5 nm and a log-normal distribution

$$n^i = \frac{n_0}{r^i\sigma\sqrt{2\pi}} \exp\left(\frac{(\ln(r_i)-\mu)^2}{2\sigma^2}\right), \qquad (19)$$

where the parameter n_0 describes the absolute amount of platinum particles

$$n_0 = \frac{L_{\mathrm{Pt}}}{\sum_{i=1} \frac{4}{3}\pi(r^i)^3 n^i t_{\mathrm{cl}} \rho_{\mathrm{Pt}}}. \qquad (20)$$

The parameter for the distribution $\ln(\mu) = 1.58\times10^{-9}\,\mathrm{m}$, $\sigma = 0.31$, $t_{\mathrm{cl}} = 20\,\mu\mathrm{m}$ and $L_{\mathrm{Pt}} = 0.6\,\mathrm{mg\,cm^{-2}}$ are taken from Jahnke et al. (2020). The resulting geometric surface area is $71.2\,\mathrm{m^2\,g^{-1}}$.

Race Car Cooling System Model for Real Time use in a Driving Simulator

Massimo Stellato[1] Luca Bergianti[2] Alessandro Picarelli[3]

[1]Dallara Automobili, Italy, m.stellato@dallara.it
[2]Dallara Automobili, Italy, l.bergianti@dallara.it
[3]Claytex, United Kingdom, alessandro.picarelli@claytex.com

Abstract

Powertrain performance optimization is one of main targets in racecar and road hypercar development. A key activity needed for both endothermal and electric powertrains is the cooling system sizing through simulation to make sure that the temperature limits are not exceeded in the most aggressive conditions minimizing or avoiding power derating. This article describes the implementation of a 1D cooling system simulation model integrated with a vehicle multibody model to be used in real time in the Dallara dynamic driving simulator with human driver. This activity is the result of a collaboration between Dallara which uses the model implemented to develop and optimize the cooling system architecture of its vehicles, and Claytex who develop the libraries used to generate these simulation models. The model has been validated through comparison with real data of an existing vehicle yielding a RMSE of 1.0 °C.

Keywords: cooling system, fluids temperatures, powertrain, derating, real time, simulator.

1 Introduction

Considering the complexity of current vehicles, a holistic approach to analyse the interactions of vehicle dynamics, cooling system dynamics and human drivers in the same simulation can be the key to maximize the overall performance of a high performance car as demonstrated already for many years (Bouvy et al, 2012).

These needs have led to the development of a 1D modular cooling system simulation model which has two main targets:

- To develop for each vehicle project the cooling system solution which has the best trade off among aerodynamics, packaging, weight and motor power (ICE or electric) to reach the max vehicle performance.
- To support the detailed vehicle performance analysis on the Dallara driving simulator (Figure 1) with different boundary conditions such as ambient temperatures, initial oil and coolant temperatures, human drivers.

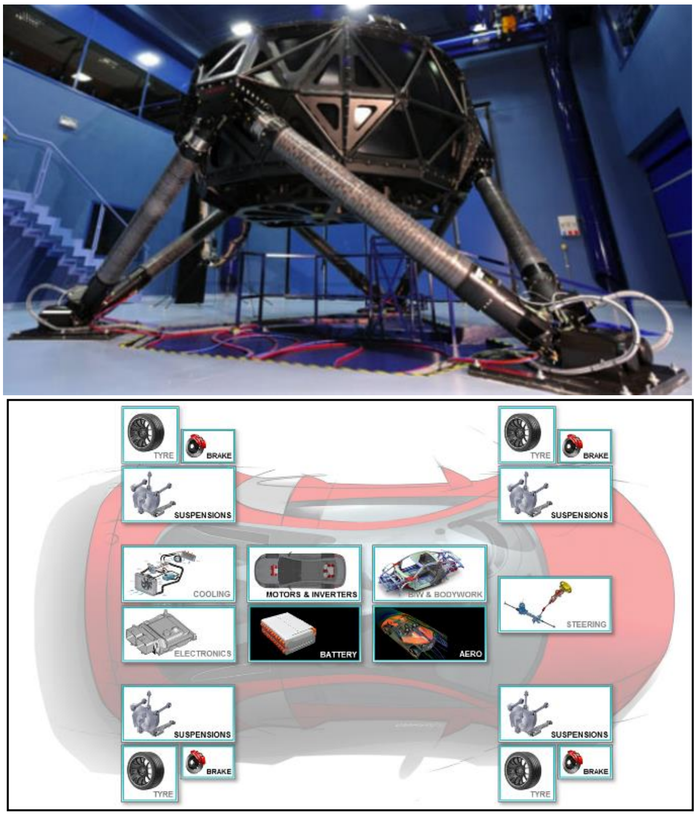


Figure 1. Dallara dynamic driving simulator (top), vehicle model diagram (bottom)

The case study illustrated in this paper refers to an ICE vehicle. Starting from the described architecture it can be modified in a user-friendly way, to develop a cooling system layout for electric and hybrid vehicles.

2 The Library

The Modelica libraries developed at Claytex are designed to be able to include significant levels of model details whilst retaining very robust and efficient simulation performance characteristics. This is done via careful and meticulous development strategies where it's ensured that the models achieve the best possible simulation efficiency from full scale system models right down to component test level scenarios. The Claytex library contains common models that are used in the wide range of libraries developed over the past 25 years.

2.1 Thermofluids

The Claytex.Fluid library (figure 2) is suited to real time performance for HiL (hardware in the loop) and DiL (driver in the loop) applications. It was first applied to driver in the loop models in a demonstrator in 2013 where the whole F1 vehicle according to the new 2014 specs was developed and simulated to test and address a broad range of questions.

The Claytex.Fluid library is based on Modelica.Fluid and the Claytex.Media library is based on Modelica.Media with some streamlining of the models for specific applications and fluid types found in cooling and lubrication. A large part of its development is driven by customer requirements, especially in terms of data input requirements for model parameterisation. This is a fundamental advantage as it is therefore tailored to take in the datasets that are available without requiring extensive calibration of unknown parameters to match test data.

Because Claytex.Fluid is compatible with Modelica.Fluid, this broadens the range of components the user has access to for building the fluids system models. Below is a snapshot of the top level packages within Claytex, Claytex.Fluid and Claytex.Media.

The modelica libraries used to represent the vehicle model are also developed by Claytex. The VeSyMA – Motorsports library was used (Claytex). This library provides all the required suspension and chassis related components to allow the user to build a full multi-body suspension vehicle model based on the customisable VeSyMA model vehicle architecture templates (Claytex; Hammond-Scot et al, 2018). In a similar way to the fluids and media libraries which have been described, all models have been designed with mathematical and numerical efficiency in mind whilst still capturing all the required details. The VeSyMA - Motorsports library is used within motorsports series such as Formula 1, NASCAR and IndyCar including driver in the loop applications and lends itself very well to the study presented in this paper (Dempsey, 2016).

One of the key components in the modelling were the heat exchangers. The Heat Exchanger models take in tabular data from the user table of Gc values (Heat transfer area * heat transfer coefficient) in W/K as a function of the actual mass flow rates (kg/s) on the primary side and secondary side of the heat exchanger. This value is then multiplied by dT, the temperature difference between the primary and secondary side inlets to yield the heat flow between the primary and secondary fluid.

$$HD_{rad} = A \cdot h \cdot dT$$

HD_{rad} = radiator heat dissipation [W]
A = heat transfer area [m^2]
h = heat transfer coefficient [W/(m^2.K)]
dT = temperature difference of inlet primary and inlet secondary [K]
A·h = f(m_flow_primary, m_flow_secondary)

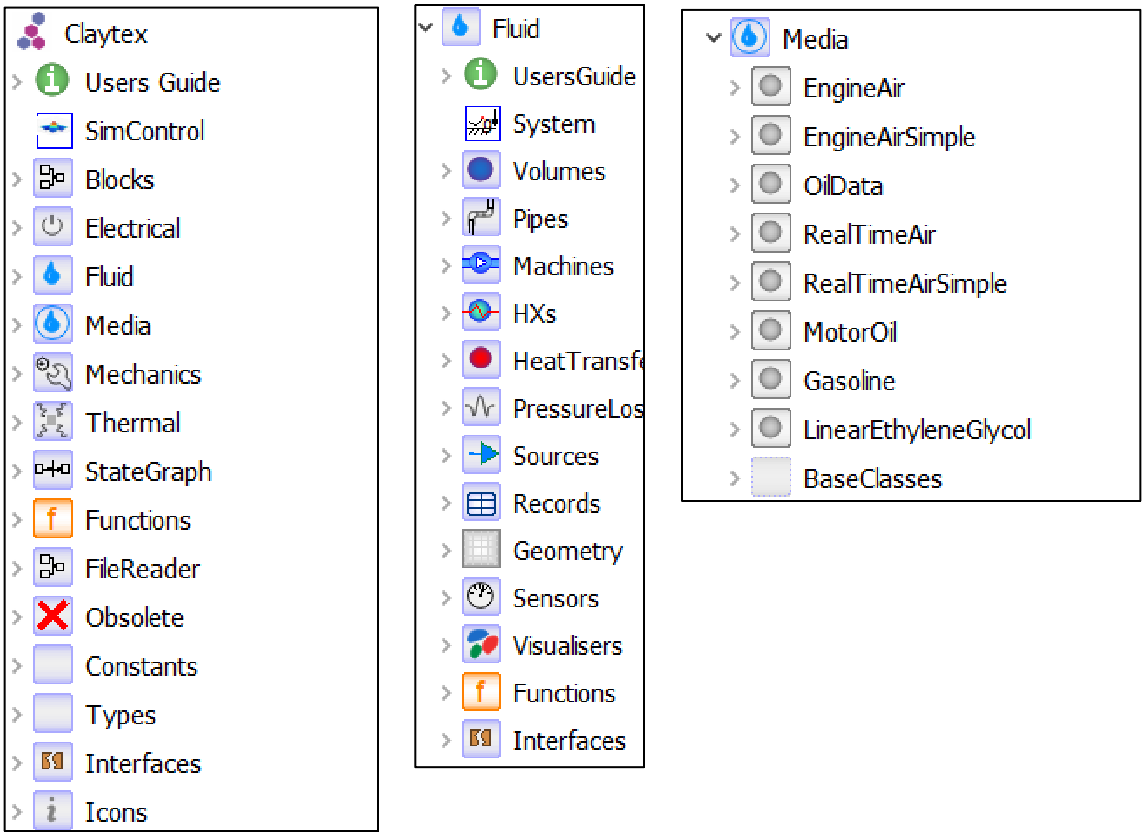
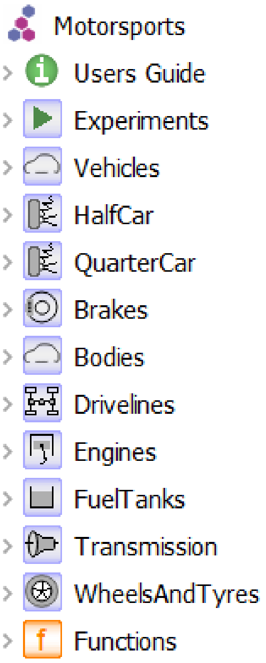


Figure 2. Claytex (left), Claytex.Fluid (middle) and Claytex.Media (right) library structure.

DOI
10.3384/ecp204675



Motorsports
- Users Guide
- Experiments
- Vehicles
- HalfCar
- QuarterCar
- Brakes
- Bodies
- Drivelines
- Engines
- FuelTanks
- Transmission
- WheelsAndTyres
- Functions

Figure 3. VeSyMA – Motorsports library top level package structure

3 The Model

Figure 4 shows the cooling system model connected to the vehicle model and human driver. The ICE power curve of the vehicle case study in this paper is affected by coolant and oil temperatures which are output of the cooling system model according to the graph reported in figure 5. The heat rejections and airflow across the radiators are function of vehicle speed, throttle and engine speed which are output of vehicle model.

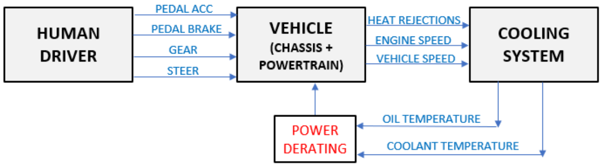


Figure 4. cooling system interactions with vehicle model and human driver

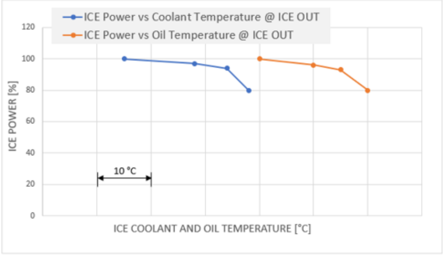


Figure 5. ICE power derating

Coolant and oil flow rates are consequences of pumps head curves, instantaneous engine speed and pressure drops for all components (ICE, pipes, heat exchangers). The fluids (coolant and oil) heat transfer and hence temperatures are calculated by the following equations at each time step.

$$HF_n = HR_{ICE_n} - HD_{rad_n}$$

$$T_{coolant_n} = T_{coolant_{n-1}} + \frac{HF_n \cdot (t_n - t_{n-1})}{TC}$$

HF = Heat Flow [W]
$\mathrm{HR}_{\mathrm{ICE}}$ = ICE Heat Rejection [W]
TC = Thermal capacity [J/K]
t = time [s]

HF is the instantaneous difference between the heat rejection produced by the engine and the heat dissipation of the radiators. The thermal capacity [J/K] takes into account the total coolant volume, components (ICE, radiators, pipes) materials and weight.

3.1 The modular architecture

The model is composed of multiple air/coolant radiators, air/oil coolers and coolant/oil heat exchangers in a modular way allowing to study different cooling system configurations where coolant and oil are cooled together in a single loop or separately in more loops. In each loop air/fluid radiators and coolant/oil heat exchanger can be arranged in series or in parallel connection.

In figure 6 is reported the cooling scheme of the Dallara ICE vehicle case study of this paper, which is composed by two air/coolant radiators and one coolant/oil heat exchanger.

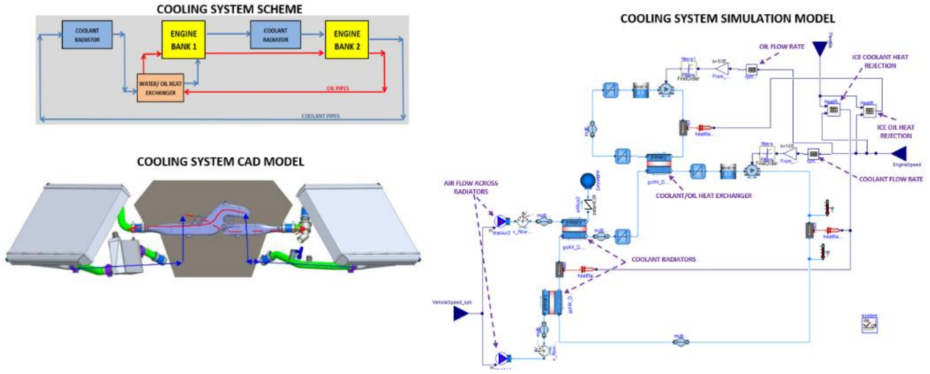


Figure 6. Cooling system of a Dallara vehicle

3.2 Parameters

The model parameters are obtained from CFD simulations and experimental tests:

- The air flow ratio across the radiators is the output of dedicated CFD simulations which take into account all the vehicle geometry or wind tunnel test (Figure 7).
- The ICE heat rejection is provided by the engine manufacturer as function of throttle and engine speed (Figure 8).
- Air/coolant radiator and coolant/oil heat exchanger efficiency has been measured by experimental tests on the Dallara cooling test rig (Figure 9).

Many coolant radiator specifications have been analysed. The chosen radiator specification is the result of the best trade off among air pressure drops, coolant pressure drops and heat dissipation which ensure the optimal cooling operating point for the case study.

The thermal behaviour of an ICE is defined by its capacities, heat transfer and thermal conductivities as well as its surrounding conditions (Morawietz at Al, 2005). The ICE heat capacity is modeled with a lumped thermal element storing heat. This parameter together with the amounts of coolant and oil volumes plays an important role as it affects the thermal inertia and therefore the time before reaching the maximum temperature values (Stellato et al, 2017).

The more coolant in the system and the higher heat capacity thus allowing higher vehicle accelerations for longer periods of time, before the available ICE power decreases for the derating.

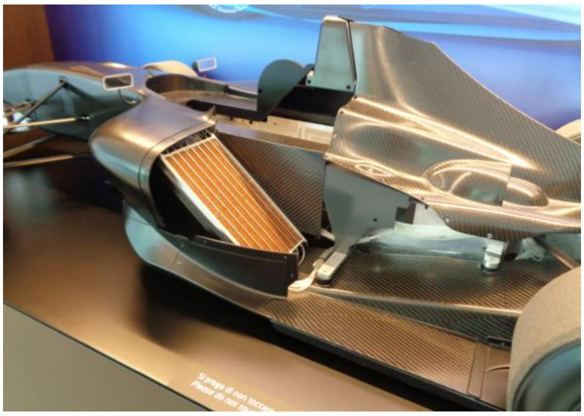


Figure 7. Typical wind tunnel vehicle test physical model

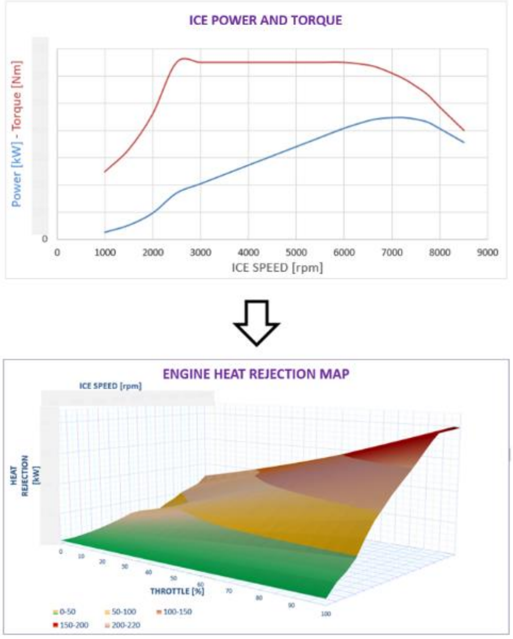
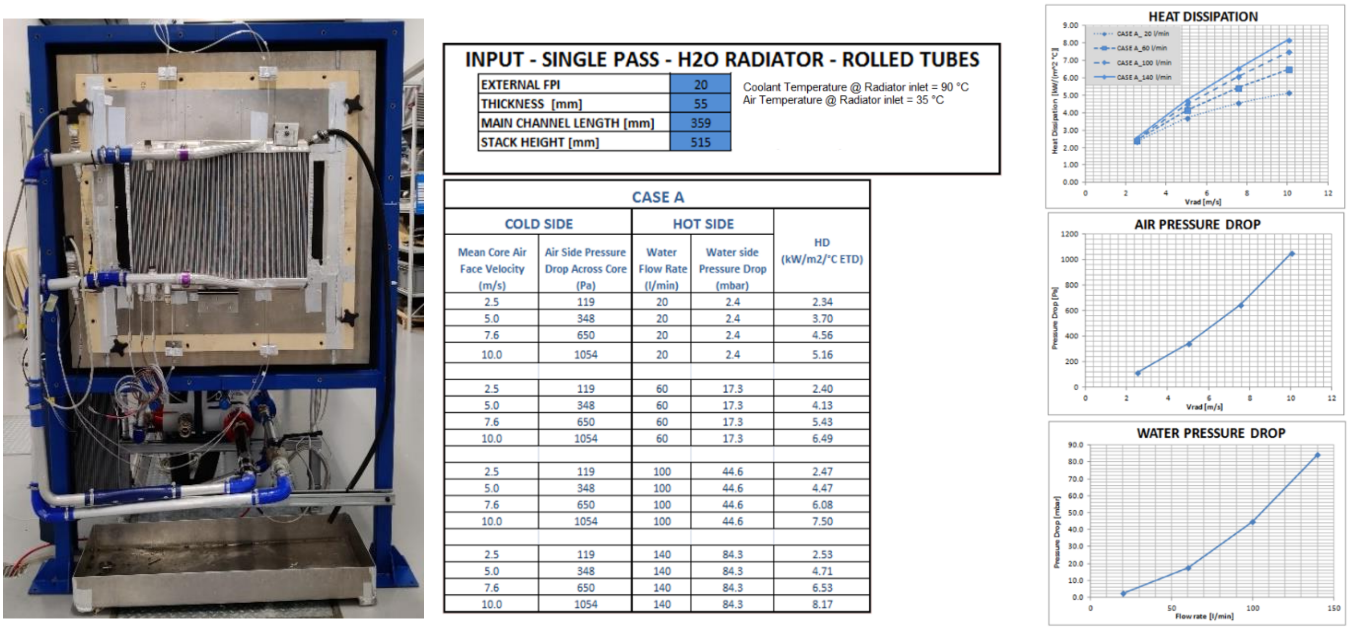


Figure 8. ICE power and heat rejection maps



INPUT - SINGLE PASS - H2O RADIATOR - ROLLED TUBES

EXTERNAL FPI	20
THICKNESS [mm]	55
MAIN CHANNEL LENGTH [mm]	359
STACK HEIGHT [mm]	515

Coolant Temperature @ Radiator inlet = 90 °C
Air Temperature @ Radiator inlet = 35 °C

CASE A

COLD SIDE		HOT SIDE		
Mean Core Air Face Velocity (m/s)	Air Side Pressure Drop Across Core (Pa)	Water Flow Rate (l/min)	Water side Pressure Drop (mbar)	HD (kW/m2/°C ETD)
2.5	119	20	2.4	2.34
5.0	348	20	2.4	3.70
7.6	650	20	2.4	4.56
10.0	1054	20	2.4	5.16
2.5	119	60	17.3	2.40
5.0	348	60	17.3	4.13
7.6	650	60	17.3	5.43
10.0	1054	60	17.3	6.49
2.5	119	100	44.6	2.47
5.0	348	100	44.6	4.47
7.6	650	100	44.6	6.08
10.0	1054	100	44.6	7.50
2.5	119	140	84.3	2.53
5.0	348	140	84.3	4.71
7.6	650	140	84.3	6.53
10.0	1054	140	84.3	8.17

Figure 9. Dallara cooling test bench setup (left) and typical results (right)

DOI
10.3384/ecp204675

3.3 Realtime Features

The fluids models within the cooling system were set to run with fixed step Euler solver at a rate of 1kHz. With the exception of some initialization spikes, the turnaround time was approximately 0.1ms throughout the desktop test runs (figure 10).

The desktop runs were performed on a 2.8GHz Core I7 processor with SSD giving the confidence of being able to run on the hardware used in the PCs in the DiL setup at Dallara simulator which has superior characteristics.

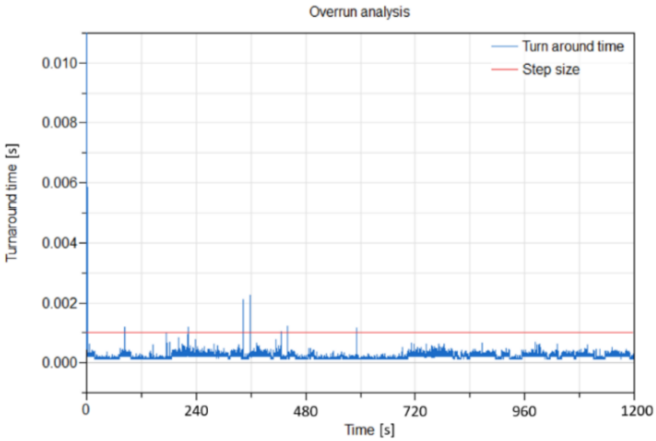


Figure 10. image showing turnaround time of a lap simulation of Monza

The cooling models were compiled with a mixed explicit/implicit Euler method with advanced inline integration settings using the Claytex library Real-time configuration functions to further enhance real time perfomance and model robustness.

Such real time configuration functions also make use of flags described in Section 11.3.3 of the Dymola Full User Manual.

It was not necessary to resort to multicore simulation for the entire fluids system. In fact there would be enough processing power capacity with the aforementioned processor spec to also include a full multibody suspension model of the vehicle and still run in real time at 1kHz and have a physical connector based coupling between the cooling system and the vehicle and powertrain systems.

4 Virtual Validation

The validation was a very important phase of this activity, it consisted of the following steps:

- Implementation of a cooling system architecture relative to an existing vehicle.
- Offline simulation using as inputs the measured vehicle speed, throttle, engine speed and ambient temperature recorded on the track.
- Comparison between the simulated coolant temperature profile and measured coolant temperature profile recorded on the track.

Figure 11 shows the comparison between the coolant temperature profile simulated vs real logged data. The maximum CoolTempDiff, defined as CoolantTempSimulated - CoolantTempMeasured, is 2.7 °C, the average is 0.2 °C, RMSE is 1.0 °C.

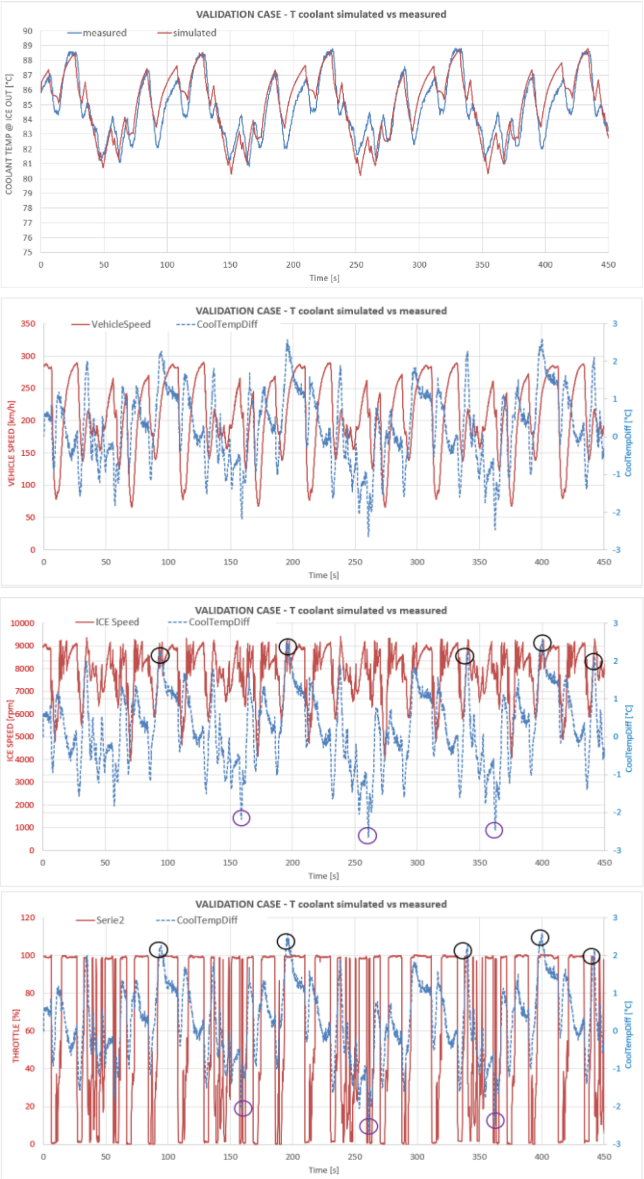


Figure 11. Simulated coolant temperature vs real logged data, vehicle speed, throttle and ICE speed for the validation case

The results accuracy is considered acceptable to size a racecar cooling systems and also for a refined assessment of the global vehicle performance on the driving simulator.

In an attempt to further explain the reasons for the discrepancy with the real data, a possible explanation lies in the fact that the ICE heat rejection computed in the simulation transients is the result of an ICE measurement at test bench in fully stabilized conditions. Consequently, the model doesn't consider any delay between the throttle pedal and the heat rejection produced by the engine.

This hypothesis appears confirmed by acknowledging that the points with higher CoolTempDiff (+2.2/+2.7 °C → simulated temperature being higher than real) occur when ICE is in high speed (>8500) and max throttle, as shown in the black circles at time 95s, 190s, 340s, 400s, 440s.

This hypothesis appears also validated in the opposite direction (low ICE speed and throttle close to zero), when the throttle pedal releases and the ICE heat rejection dramatically reduces. In this second situation CoolTempDiff reaches the minimum values (-2.2/-2.7 °C → measured being higher than simulated) exactly in those working points characterized by low ICE speed (<7000) and zero or low throttle (<10%), as shown in the purple circles at time 160s, 260s, 360s.

In order to introduce the effect described above, as a further step of refinement, a first order filter calibrated according to transient test data, could be applied to the Heat Rejection maps.

5 Results

The simulation model, as developed and validated, is currently used at Dallara to size the cooling systems and to analyse the vehicle performance on the driving simulator for different boundary conditions, human drivers and tracks.

The case study under discussion concerns the global vehicle performance of a Dallara car simulated in the Driving Simulator facility, driven by a professional driver in Monza track (Figure 12). Three different boundary conditions, hereinafter referred to as “OUTINGS” (Figure 13 and 14), are considered and below described:

OUTING 1

Air ambient temperature 25 °C, coolant and oil initial temperature 89 °C (no precooling), 10 laps.

OUTING 2

Air ambient temperature 35 °C, coolant and oil initial temperature 80 °C (precooling), 10 laps.

OUTING 3

Air ambient temperature 35 °C, coolant and oil initial temperature 89 °C (no precooling), 10 laps.

In all the three outings the vehicle is able to run without derating in the first lap.

OUTING 1 shows that when the ambient temperature is 25 °C the vehicle runs all the 10 laps at max ICE power without derating.

OUTING 3 is the most critical case due to the higher ambient temperature and the initial fluids (coolant, oil) temperature.

OUTING 2 shows that a precooling phase, which implies lower initial coolant and oil temperatures, allows the vehicle to run more time with higher power at high ambient temperature until it reaches the same stabilized conditions of Outing 3.

The power derating in the stabilized laps of Outing 3 (ambient temperature 35 °C), due the high coolant and oil temperatures, affects the lap time by 2.3 seconds with respect to a not de-rated lap.

All the results reported are marginally affected by the human driver who is the same in all 3 outings, but he doesn’t drive always exactly in the same way. This can be noted in the lap 2 and 3 of outing 1, where despite the power derating doesn’t occour the laptime is higher than lap 1 and lap 10.

The precooling considered on the initial fluids temperature (coolant and oil) is only 9 °C with respect to the case without precooling, so the consequent effect on vehicle performance is minimal, it can be noted analyzing the power profile in the lap 2.

As a next step, two configurations (with and without precooling) with a larger temperature difference will be tested also taking into account the higher friction losses.

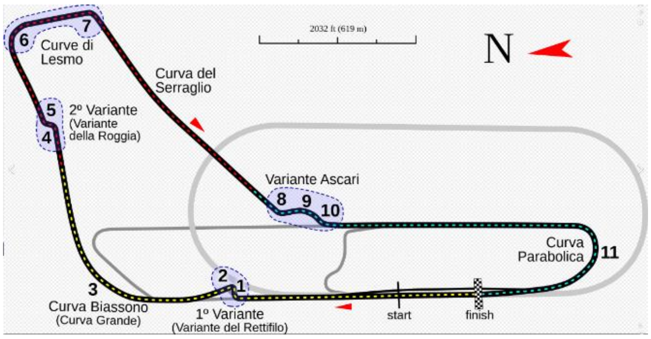
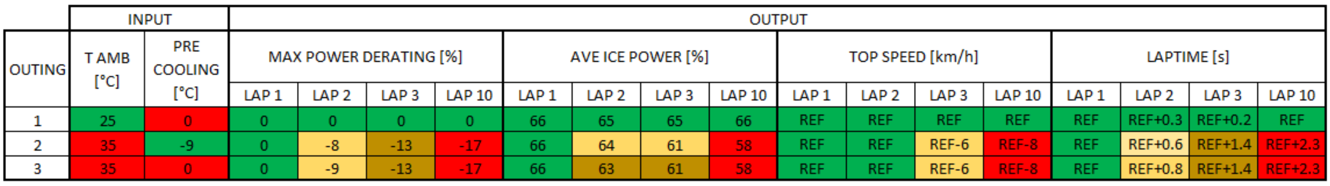


Figure 12. Monza track layout and table results



	INPUT		OUTPUT															
OUTING	T AMB [°C]	PRE COOLING [°C]	MAX POWER DERATING [%]				AVE ICE POWER [%]				TOP SPEED [km/h]				LAPTIME [s]			
			LAP 1	LAP 2	LAP 3	LAP 10	LAP 1	LAP 2	LAP 3	LAP 10	LAP 1	LAP 2	LAP 3	LAP 10	LAP 1	LAP 2	LAP 3	LAP 10
1	25	0	0	0	0	0	66	65	65	66	REF	REF	REF	REF	REF	REF+0.3	REF+0.2	REF
2	35	-9	0	-8	-13	-17	66	64	61	58	REF	REF	REF-6	REF-8	REF	REF+0.6	REF+1.4	REF+2.3
3	35	0	0	-9	-13	-17	66	63	61	58	REF	REF	REF-6	REF-8	REF	REF+0.8	REF+1.4	REF+2.3

Figure 13. Main results at Monza

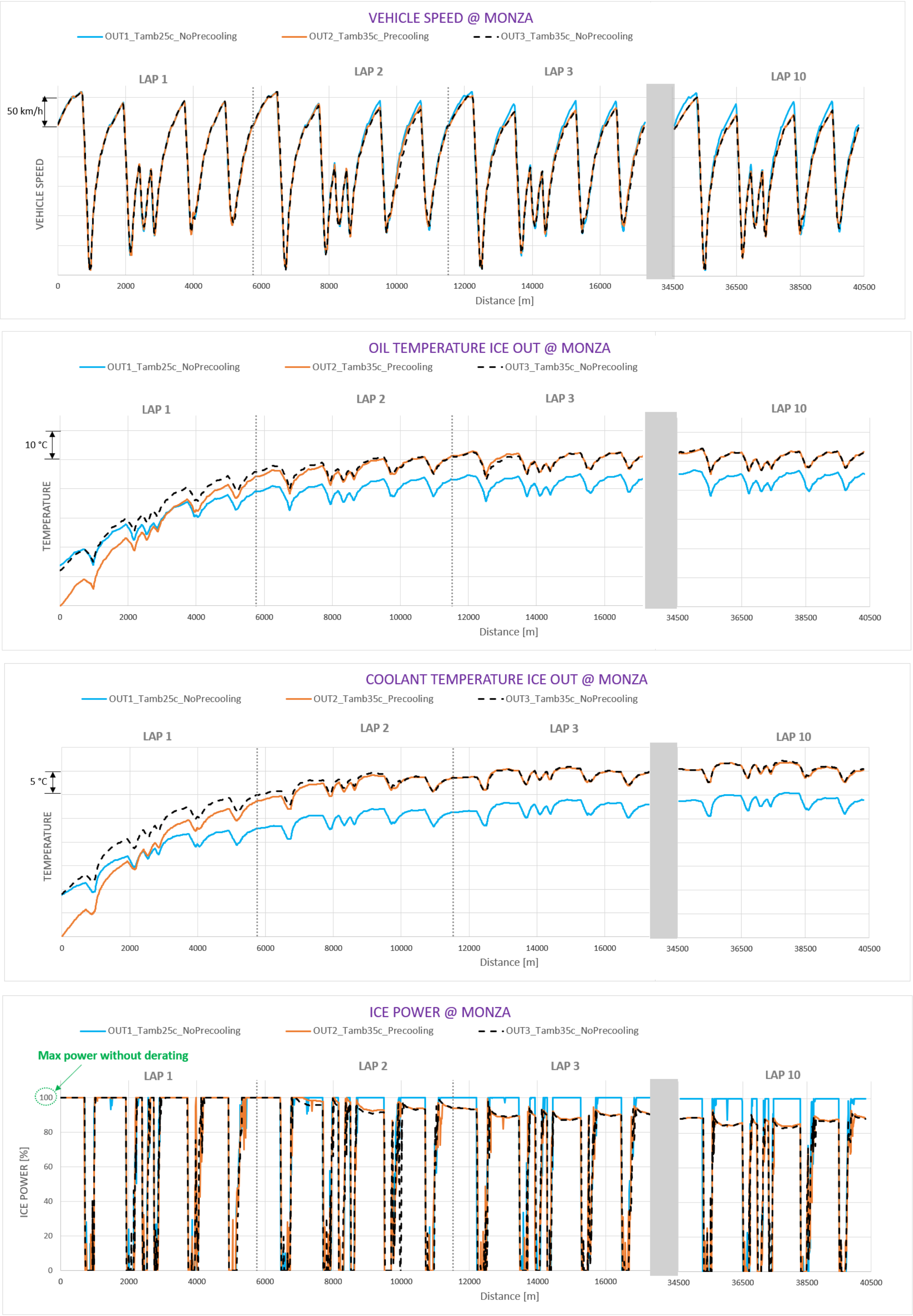


Figure 14. Speed, temperature and power profiles on the Dallara driving simulator

The simulation model, as it is described on this paper, allows to put into the equations different sizes of radiators, considering not only the thermal effect but also their different weight and vehicle aero efficiency.

For example: a configuration with bigger radiator core areas (+15%) is able to improve the laptime by 0.5sec at an ambient temperature of 35 °C (predominant thermal effect because it massively reduces de-rating), but resulted in a 0.3sec slower lap time at an ambient temperature of 25 °C (predominant weight increases and aero efficiency reduction effects). While a configuration with smaller radiator core areas (-15%) resulted slower laps at both ambient temperatures of 25 °C (+0.4sec) and 35 °C (+0.6 sec).

6 Conclusions

The fluids temperature management in high performance vehicles has a crucial impact not only on the reliability but also on the performance and drivability. The obvious answer to that would land to bigger heat exchangers and/or higher air flows, needed to achieve lower fluids (coolant, oil) temperatures. But this implies more weight, drag and a negative impact on packaging constraints. The aforementioned input conditions return a complex and often over constrained or multivariable problem and trade off. Moreover, depending on the mission of the vehicle and on the peculiar driving style it could be more interesting to emphasize the “on power” behavior (typically more favorite for the “track day” drivers) or the “handling” behavior (more important for “professional” drivers). The first case requiring bigger radiators targeting no derating, the second case requiring an overall weight reduction and aero efficiency optimization.

The implemented and here described simulation model is useful to evaluate all these effects together to develop the cooling system architecture for every project targeting the best trade off to maximize the vehicle performance also accounting the driving style and drivability on a professional driving simulator with a real driver.

Powertrain cooling performance is affected by many parameters, this simulation model is useful to analyse the vehicle performance in each condition of ambient temperature, precooling and different human drivers.

The duty cycle analyzed for the case study is a track lap at maximum vehicle performance, but additional critical driving cycles can be studied on the simulator with the model developed, for example, safety car conditions, pit lane or race traffic and grid formation.

The described case study focuses on the coolant and oil cooling system of a high performance naturally aspirated ICE vehicle, clearly similar or greater problems and trade off occur also for charge air temperature in the case of a turbo engine, or in the optimization of an electric powertrain performance (Stellato et al, 2017).

For these reasons, the approach described in this paper, can be considered a good method in the optimization of different vehicle propulsions (ICE, HEV, BEV, FCV, …).

References

C. Bouvy, P. Jeck, J. Gissing, T. Lichius, L. Ecksterin. Holistic Vehicle Simulation using Modelica – An Application on Thermal Management and Operation Strategy for Electrified Vehicles. *Proceedings of 9th International Modelica Conference*, pp. 263-270, 2012.

M. Dempsey. Simulation of the complete race car to improve efficiency. *Autosport International Show*, 2016.

H. Hammond-Scot, M. Dempsey. Vehicle Systens Modelling and Analysis (VeSyMA) Platform. *Proceedings of 2nd Japanese Modelica Conference*, pp. 61-65, 2018.

L. Morawietz, S. Risse, H. Zellbeck, H. Reuss, T. Christ. Modeling an automotive power train and electrical power supply for HiL applications using Modelica. *Proceedings of 4th International Modelica Conference*, pp. 301-307, 2005.

M. Stellato, L. Bergianti, J. Batteh. Powertrain and thermal system simulation model of a high performance electric road vehicle. *Proceeding of 12th International Modelica Conference*, pp. 171-180, 2017.

Claytex. VeSyMA Motorsports library info. https://www.claytex.com/products/dymola/model-libraries/vesyma/motorsports/

Claytex. VeSyMA library info. https://www.claytex.com/products/dymola/model-libraries/vesyma/

Wikipedia. Monza Circuit. https://en.wikipedia.org/wiki/Monza_Circuit

DOI
10.3384/ecp204675

Switching and Averaging Models of a Bidirectional, Half-Bridge Based DC-DC converter with Load Distribution

Andrea Reindl[1]   Andreas Lang[1]   Michael Niemetz[1]   Hans Meier[1]

[1]Ostbayerische Technische Hochschule Regensburg, Germany, andreareindl@ieee.org,
andreas2.lang@gmx.de, {hans.meier,michael.niemetz}@othr.de

Abstract

Bidirectional DC-DC converters are vital for the integration of batteries, for the power conversion during (dis)charge and the battery management. Modeling of these is helpful, especially for the design of larger, more complex systems consisting of multiple DC-DC converters in parallel. Due to the high switching frequencies, the simulation of DC-DC converters is associated with increased computational time and effort. In this paper, three models of different complexity and accuracy are proposed for a bidirectional DC-DC converter consisting of two phase-shifted half-bridges. Two switching models, which differ mainly in the way the MOSFETs are driven, account for the individual switching operations and exhibit high accuracy. An averaging model replaces the switching elements with current and voltage sources providing the mean values. The dynamic behavior of the models is analyzed using the step responses of the load current. For validation, these are compared with the theoretical transfer function. The three models are analyzed comparatively in terms of computational time and effort. The calculation time of the averaging model has been reduced by two thirds compared to the strictly complementary switching model and by 96% relative to the model with diode emulation mode. Recommendations for the use of the models are given and a possible use case is shown. Two parallel connected DC-DC converters with load current sharing between them are simulated using the averaging model.

Keywords: Bidirectional DC-DC Converter, Averaging Model, Switching Model, Computational Effort, Modelica, Half Bridge, Circuit Averaging

1 Introduction

Heterogeneous battery systems combine batteries with differences in cell chemistry, nominal capacity, state of health, state of charge, safe operating area and terminal voltage. They offer advantages such as increased energy density, improved efficiency, enhanced safety and flexibility compared to homogeneous systems. Furthermore, they provide second life batteries a further application with lower requirements regarding dynamics, remaining useful capacity and internal resistance. Due to the heterogeneity, it is challenging to ensure reliability, robustness and safety of the system. Specific requirements arise for the control of the DC-DC converters: Adjustable current limits are required and the different input voltages have to be converted to a common DC output voltage and vice versa. The (dis)charge current has to be limited according to the battery state and the safe operating area in order to realize reliable simultaneous operation of varying batteries. The total current is divided among the batteries depending on their state, which reduces the burden and enables additional possibilities such as state of charge balancing in active operation without recharging or state of health balancing with the goal of achieving a common end of life.

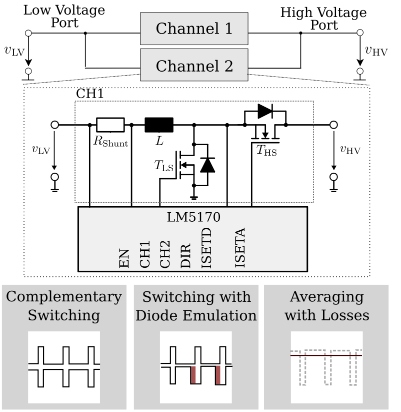


Figure 1. The bidirectional, half-bridge-based DC-DC converter is emulated by three different models of varying accuracy.

The model of the bidirectional DC-DC converter is valuable for the analysis of its behavior and performance. It is the basis for the development of suitable control strategies and enables initial tests of these in parallel operation of several DC-DC converters. The robustness, safety and reliability of the system can be investigated. Robustness is defined as the stability in the presence of disturbances, safety as the behavior in the event of a fault and reliability as the system availability in the event of breakdown of individual components.

In order to consider varying time scales, different abstraction levels are required. Three models of a bidirectional, multiphase DC-DC converter based on two half-bridges are presented (Fig. 1). The behavior of the DC-DC converter is thereby primarily determined by the power semiconductors and the passive components. Two switching models of the DC-DC converter are proposed for de-

DOI
10.3384/ecp204683

tailed analysis of the switching behavior and are suitable for relatively short simulation duration. For the system simulation, an averaging model is proposed which neglects the individual switching processes. The different models are described and compared with the calculated transfer function of the bidirectional DC-DC converter. They are compared in terms of computational effort and time. Two averaging models are subsequently connected in parallel as an exemplary use case and the load current sharing between two DC-DC converters is simulated.

1.1 Related Work

Existing works on the simulation of DC-DC converters also propose averaging models, while concentrating on other aspects, i. e. parameterization of the electrical components, reduction of the simulation time, temperature dependency and losses or testing of control strategies. The paper (Baumann, Weissinger, and Herzog 2019) focuses on the system identification of inverters and proposes a novel model of a bidirectional DC-DC converter which is valid for different frequency ranges. Special focus is given to the identification and parameterization of the electrical components. In article (Navarro et al. 2020), a continuous-time linearized model of a non-isolated bidirectional half-bridge DC-DC converter corresponding to the state-space averaging method is proposed. The key contribution hereby is the reduction of the time complexity. In the paper (Spiliotis et al. 2019) an electrical-thermal model of a DC-DC boost converter is developed. It focuses on the analysis of the temperature dependence on the losses of the converter. In contribution (Winter et al. 2015), a simplified averaging model of a synchronous half-bridge converter is presented, which shows the losses and the dynamic behavior without the need to simulate every switching process of the power semiconductors. An averaging model of a half-bridge converter, more precisely a two-stage DC-AC voltage source converter, is proposed in (Laera et al. 2020). The key topic is the test of different control strategies and their effects on the model behavior. It demonstrates that averaging models can lead to control parameters which cannot be successfully transferred to real power electronic components.

1.2 Main contributions

Two switching models with different accuracy levels and varying control of the MOSFETs and an averaging model of a half-bridge based, multiphase DC-DC converter are proposed. A novel model with diode emulation is developed. The averaging model is based on the circuit averaging technique and uses equivalent circuits. Another novelty is the possibility of efficient switching between the three proposed models. Thus, the diverse behavior of the models according to the simulation objective, e.g. testing of control strategies, can be taken into account. The simulation especially focuses on testing the control strategies and load sharing between DC-DC converters connected in parallel.

2 Bidirectional DC-DC Converter

A bidirectional, multiphase DC-DC converter is used for the implementation of the power flow in (dis)charge direction. Depending on the direction of the current flow, it operates in boost or buck mode. The utilized DC-DC converter consists of two half-bridges (Fig. 2) connected in parallel, between which the current is symmetrically divided. This decreases conduction losses, which in turn has a positive effect on the thermal behavior of the components. Additionally, it allows a larger power range with significantly smaller devices. The two signals for driving the half-bridges are phase-shifted by 180°. This considerably lowers the current ripple that has to be smoothed by the output capacitance (Alharbi et al. 2019; Schuck and Pilawa-Podgurski 2013). The LM5170 module is used for current control, which implements the gate drivers, operational amplifiers for current control, current measurement, and a sawtooth generator for average current mode control. The frequency response of the current controller can be determined by the external circuitry. The magnitude of the setpoint current value is specified for digital setting via a pulse width modulated signal to the pin ISETD or for analog adjustment via a reference voltage at the pin ISETA. The direction of the current flow is defined by a voltage reference at the direction pin DIR. At voltages above 2 V at the DIR pin, the converter operates in buck mode and the current flows from the High Voltage (HV) port to the Low Voltage (LV) side. At voltages below 1 V, the converter operates in boost mode and the current flows in the opposite direction. In case of any other voltage level, the LM5170 detects an invalid command and switches off the gate drivers for both channels.

Table 1. Component dimensioning of the two half bridges (Texas Instruments Incorporated 2016a)

L: 4.7 µH	R: 1 mΩ
C_{HV}: 100 µF	C_{LV}: 470 µF

According to the two function modes buck and boost, one of the two MOSFETs works as the main and one as the sync MOSFET. In boost mode, the main one is the High Side (HS) MOSFET T_{HS}, whereas in buck mode the main one is the Low Side (LS) MOSFET T_{LS}. The other one, in each case, is the sync MOSFET.

They are switched complementary: When the main MOSFET is on, the sync one is off and vice versa. While the

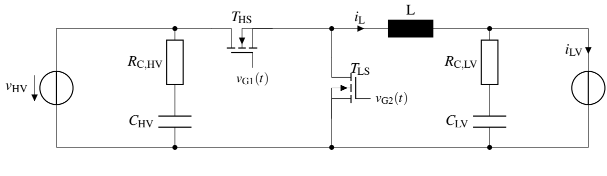


Figure 2. One of the two half-bridges of the multiphase DC-DC converter under investigation. Table 1 lists the properties of the components.

DOI 10.3384/ecp204683

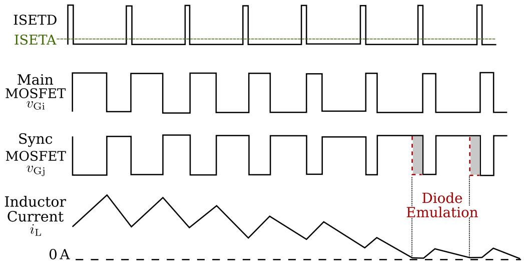


Figure 3. Switching signals of the MOSFET in diode emulation mode: If the current through the inductor reaches the value zero, the sync MOSFET is switched off and negative currents are avoided. (Texas Instruments Incorporated 2016a)

main MOSFET is driven, the current across the inductor increases. The instantaneous value of the inductor current is measured by a shunt resistor. Each channel has a real-time zero-crossing detector to monitor the instantaneous shunt voltage V_{CS}. When a zero crossing of V_{CS} is detected, the gate drive of the sync MOSFET is turned off. In this way, negative currents are prevented and the efficiency is improved at low load. Figure 3 shows the main waveforms of the described switching behavior as a function of the inductor current. The red dashed curve shows the diode emulated mode.

$$G_{\text{theoret.}} = \frac{1.0715\text{e}15 + 1.141\text{e}11 \cdot s + 5.584\text{e}5 \cdot s^2}{2.697\text{e}13 + 7.061\text{e}4 \cdot s + s^2} \quad (1)$$

The models given in the following are compared with the theoretically determined s-transfer function (Eq. (1)) according to the data sheet (Texas Instruments Incorporated 2016b). This is a generalized, approximated transfer function which neglects the high frequency behavior. For a first validation, this transfer function is sufficient. For future comparisons, the defined transfer functions for the different operating modes from a previous work (Reindl et al. 2023 - under review[a]; Reindl et al. 2023 - under review[b]) will be used. This previous work compared the theoretical transfer functions with hardware measurements and showed that, except for the neglection of the high-frequency behavior, the theoretical approximations agree with the measurements.

3 Strictly Complementary Switching Model

The strictly complementary model reproduces the switching behavior of the MOSFET without diode emulation. It allows a detailed analysis of the operating principle of the circuit, taking into account the individual switching steps. Figure 4 shows an overview of the hierarchically structured classes. The `ControlledBuckBoost` model forms the top level and unites all submodels to simulate the current-controlled, bidirectional DC-DC converter with strictly complementary switching of the MOSFETs. Furthermore, the overall model can be subdivided into the physical, electrical simulation of the circuit (`TwoCHBuckBoost`) and its control (`TwoCHController`).

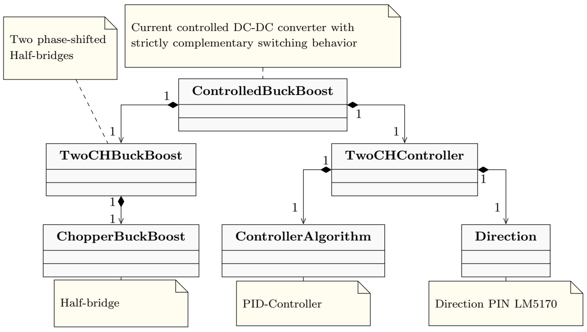


Figure 4. UML class diagram and overview of the composition of the various subclasses of the strictly complementary switching model `ControlledBuckBoost`.

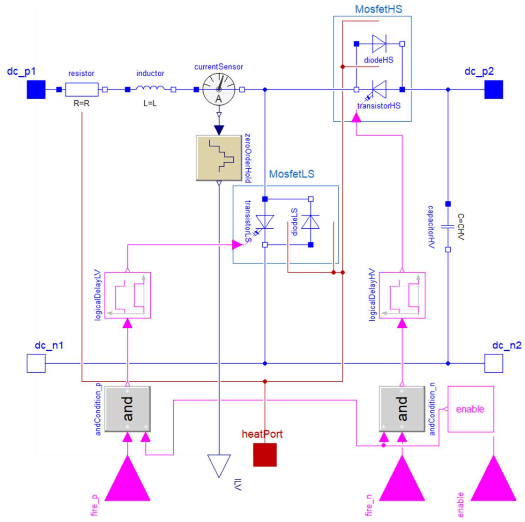


Figure 5. The `ChopperBuckBoost` model forms the switching level of the physical model and represents one of the two half-bridges.

3.1 Electrical Simulation

There are used two models `ChopperBuckBoost` and `TwoCHBuckBoost` to represent the behavior of the circuit of the bidirectional DC-DC converter. The innermost level is the `ChopperBuckBoost` model for the simulation of one half-bridge (Fig. 5). The HS and LS MOSFET are substituted by a combination of a transistor and a diode to simulate the switching behavior.

3.1.1 `ChopperBuckBoost`: Description of the Half-Bridge Model

The model has nine interfaces: The electrical connections for the power flow are realized by the four interfaces `dc_p1, dc_p2, dc_n1` and `dc_n2`. With the `heatPort` the thermal behavior can be observed and first conclusions about losses can be drawn. The three logic ports are used to control the transistors. With `fire_p` and `fire_n` the transistors are switched. The interfaces are logically and connected with the parameter `enable`, so that the DC-DC converter can be switched on and off. The blocks `logicDelayLV` and `logicDelayHV` are cus-

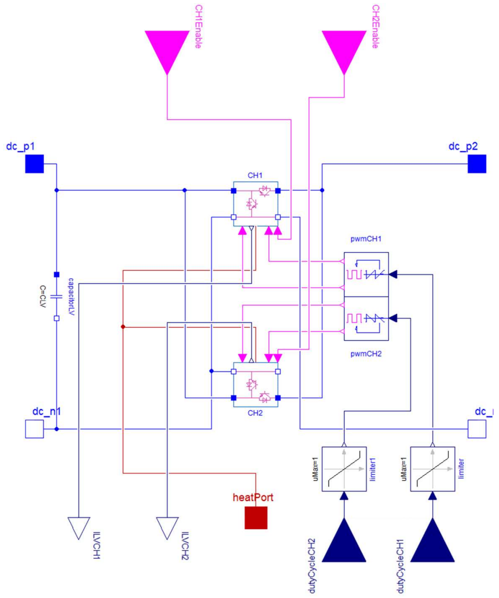


Figure 6. Overview of the model for the bidirectional DC-DC converter together with the interfaces.

tom developed. The block is used to delay a signal by a selectable time by setting the parameter `delayTime`. When an event occurs at the input port `u`, the variable `tSwitch` is set to the actual time. The switching edge is transmitted to the output `y` as soon as the actual time is greater than `tSwitch` + `delayTime`. The parameter `enable` allows (de)activation of the half bridge. External control of the model is possible with the parameter `extenable`. It overwrites the local settings for `enable`. The current through the inductor, dimensioned according to Table 1, is measured by the current sensor `currentSensor`. Using the `zeroOrderHold` block, the signal is sampled with half of the switching frequency. Thus only the average value is obtained. The measured current value is passed on to the real interface `ILV`.

Between the inductor and the electrical interface `dc_p1` there is a resistor. It combines the shunt resistance and the ohmic resistance of the inductor. The thermal connection of the resistor is connected together with the other ones to the `heatPort`. The capacitor `CHV` connecting the HV side to ground is used for voltage smoothing.

3.1.2 `TwoCHBuckBoost`: Electrical Model of the bidirectional DC-DC Converter

The model `TwoCHBuckBoost` on the next higher hierarchy level simulates the bidirectional DC-DC converter and combines two half bridges, i.e. two instances of the model class `ChopperBuckBoost`, which replicate the two channels and are labeled as CH1 and CH2 (Fig. 6).

The electrical connection is analogous to the one of the half-bridge and consists of four interfaces `dc_p1`, `dc_p2`, `dc_n1` and `dc_n2`. At Channel 2 (`CH2`) only the pin `dc_n1` is connected to the other n-pins. A connection of `dc_n2` would form a loop of the ground which in turn cannot be calculated by the simulator and would lead to an abort of the simulation. On the LV side there is used a capacitor `CLV` for voltage smoothing. The interfaces `ILV` of the two channels which contain the averaged current measurement values of the inductance current are passed to `ILVCH1` and `ILVCH2`.

The two half bridges are controlled by phase-shifted PWM signals: The duty cycle of the Channel 1 is defined by the interface `dutyCycleCH1`. The real value is limited between the values 0 and 1 and the signal is transferred to the block `pwmCH1`. This block compares a voltage level with a sawtooth signal or a triangle signal and generates a complementary PWM signal from it according to Table 2. The same model blocks are used for Channel 2. These are identically constructed, with the only difference that the switching signal is shifted by half a period.

3.2 Simulation of the Control

The control of the bidirectional DC-DC converter is simulated by the model `TwoCHController` (Fig. 7). The model consists of the controller implemented by the model `controllerAlgorithm` and the specification of the operating mode (buck or boost) simulated by the model `Direction`. In this case a PID controller with a constant gain setting of 40 and an output limiter is used (Fig. 8). The parameter `DirectedISETA` is controlled considering the feedback of the actual current measurements provided via `measuredCurrentCHx`, $x \in 1 \ldots n$. If the parameter `Enable` is false, the voltage output VLV/VHV is set to a fixed value. The fixed value is only required using the averaging model, but does not interfere with the switching model as the MOSFETs only switch if `Enable` equals true. The direction pin of the LM5170 which determines if the DC-DC converter operates in buck or boost mode is simulated by the model `Direction`. The model checks if `Direction` is either less than the value 1 or greater than 2 and adjusts the ISETA signal accordingly. If the value of `Direction` is less than 1, ISETA is not changed. If `Direction` is greater than 2, ISETA is multiplied by -1 in order to change its sign. The corrected value is set in the output parameter `DirectedISETA` which is further processed in the controller. If one of the two valid conditions (`Direction` <1 or >2) is fulfilled the parameter `Enable` is set to the value true.

3.3 Validation of the Strictly Complementary Switching Model

The model `ControlledBuckBoost` simulates the behavior of the current-controlled DC-DC converter with

Table 2. Relation between duty cycle and the control signals of the transistors

Duty Cycle	Control `fire_n`	Control `fire_p`
0	100 %	0 %
0.5	50 %	50 %
1	0 %	100 %

strictly complementary switching behavior (Fig. 9). The aim of this model is to achieve a high degree of accuracy in simulating the functionality, taking into account the individual switching processes of the MOSFETs. The channels of the bidirectional DC-DC converter can be activated individually. For an accurate current control with two active channels, the required setpoint current is halved (Fig. 9).

Figure 10 shows the test setup for the validation of the models. For validation, the DC-DC converter model is connected via a $0.01\,\Omega$ resistor to an ideal voltage source of $12\,V$ at the LV side and via a $0.02\,\Omega$ resistor to an ideal voltage source of $24\,V$ at the HV side.

Initially, a voltage of $2\,V$ is supplied to the ISETA pin. Between 0.01 s and 0.02 s the DC-DC converter operates in boost mode and between 0.03 s and 0.04 s in buck mode. The curve of the current through the inductor and the averaging of it over one period (mean PWM) are compared with the theoretical transfer function (Eq. (1)) (Fig. 11). Only small deviations between the modeled and the theoretical transfer function occur. Therefore, the first jump is analyzed in more detail (Fig. 12). The theoretical transfer function exhibits an overshoot, but it is below the targeted 80 A and corresponds to a second order behavior. The transfer function of the model also has an overshoot and reaches the 80 A faster than the theoretical one. The averaged transfer function over two periods is similar to the original data. Only minor deviations occur and they are within an acceptable range. The model has been successfully validated.

4 Switching Model with Diode Emulated Mode

The switching model with diode emulation mode differs from the strictly complementary model only slightly in the control mode of the MOSFETs (Fig. 3). Figure 13 shows a hierarchical overview of the submodels and can also be divided into the simulation of the physical components and the control.

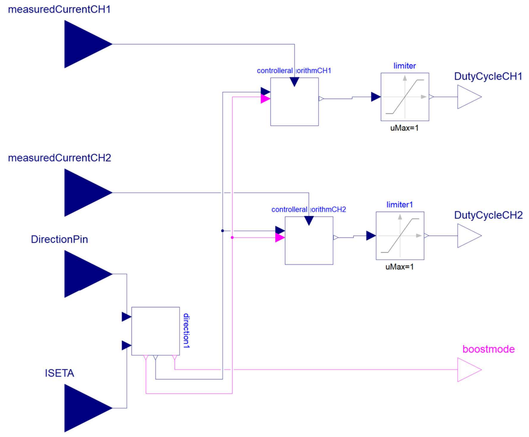


Figure 7. Overview of the model `TwoCHController` for modeling the entire control of the DC-DC converter.

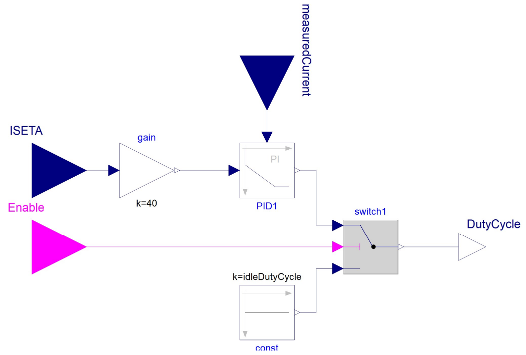


Figure 8. The model `controllerAlgorithm` simulates the PID-controller.

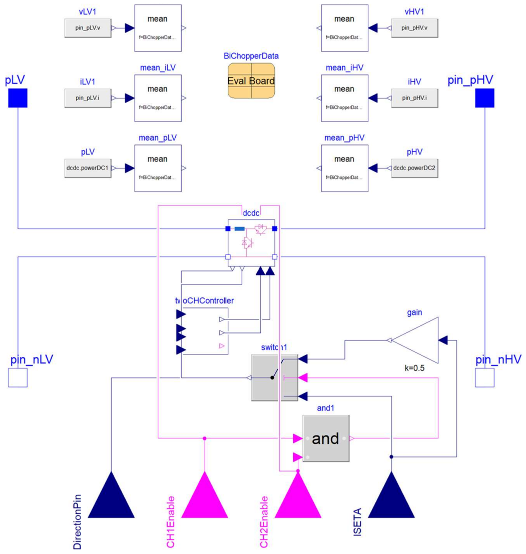


Figure 9. The model `ControlledBuckBoost` simulates the current controlled DC-DC converter with strictly complementary switching behavior.

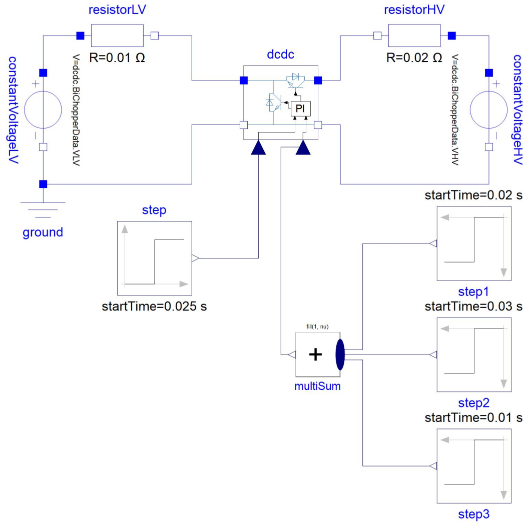


Figure 10. To verify the model, in this case `ControlledBuckBoost`, the step response is simulated and compared with the theoretical transfer function.

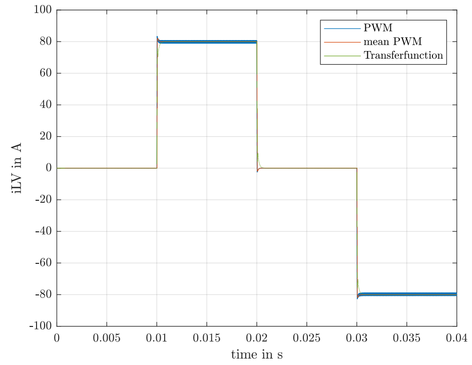


Figure 11. Comparison of the inductor current over time of the strictly complementary switching model and the theoretical transfer function.

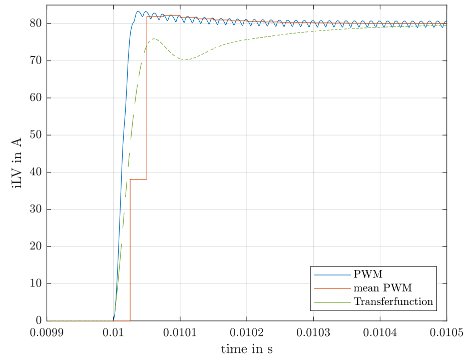
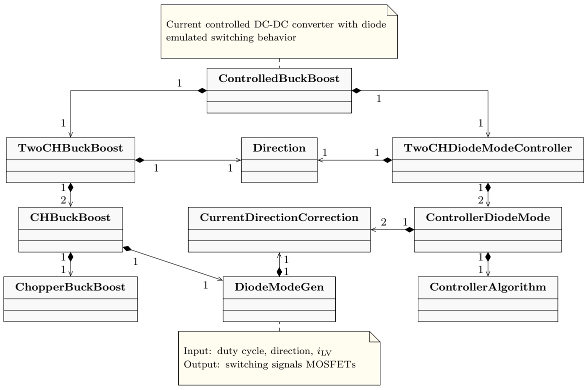


Figure 12. Step reponse of the strictly complementary switching model and the theoretical transfer function,


Current controlled DC-DC converter with diode emulated switching behavior
ControlledBuckBoost
TwoCHBuckBoost
Direction
TwoCHDiodeModeController
CHBuckBoost
CurrentDirectionCorrection
ControllerDiodeMode
ChopperBuckBoost
DiodeModeGen
ControllerAlgorithm
Input: duty cycle, direction, i_{LV}
Output: switching signals MOSFETs
1
2

[b!]

Figure 13. UML class diagram and overview of the composition of the various subclasses of the switching model with diode emulated mode `ControlledBuckBoost`.

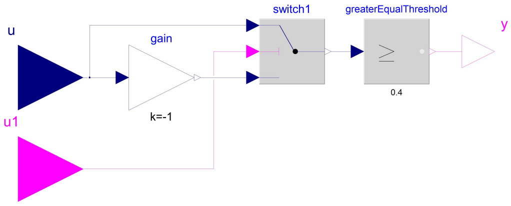


Figure 14. Overview of the model `CurrentDirection-Correction` for the diode emulated mode. In this case, the switching threshold for diode emulation is 0.4 A instead of 0 A in order to reduce the deviation.

For the diode emulation, an additional model for changing the current direction (`CurrentDirection-Correction`) and one for generating the control signals of the MOSFETs (`DiodeModeGen`) are added.

4.1 Additional Control Models

In `CurrentDirectionCorrection`, the current through the inductor is compared with the set direction. If the current is positive or zero in relation to the mode of the DC-DC converter, the output value is one, otherwise it is zero. The output is further processed in `DiodeModeGen`.

The model `DiodeModeGen` uses the duty cycle, the direction and the measured current through the inductor as inputs to define the switching signals for the two MOSFETs. The real signal `dutyCycle` is sampled over one period in `zeroOrderHold` and compared to a triangle signal. The compared signal is passed to the switches `logicSwitch` and `logicSwitch1` as an input. The inverse signal from the comparison between the PWM and the rectangular signal is passed to a logical-and block. The output signal from `currentDirectionCorrection` is the second input. The output signal of the logical-and block is one of the inputs of the switches. The direction is a further input of the switches in order to change between primary and secondary switching behavior of the respective MOSFET. The leading signal in `DiodeModeGen` corresponds to the hundredfold of the switching frequency and starts at the beginning of the simulation.

4.2 Validation of the Switching Model with Diode Emulation

The comparison with the theoretical transfer function shows only minor deviations (Fig. 16). A more detailed analysis of the step response shows that the model responds faster and with a higher accuracy compared to the transfer function (Fig. 17). Figure 18 shows a permanent control

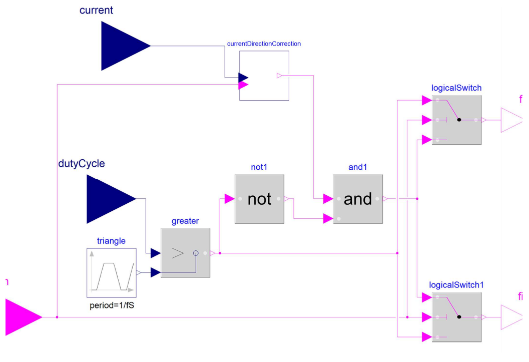


Figure 15. The model `DiodeModeGen` generates the switching signals for the MOSFETs according to the diode emulated mode (Fig. 3).

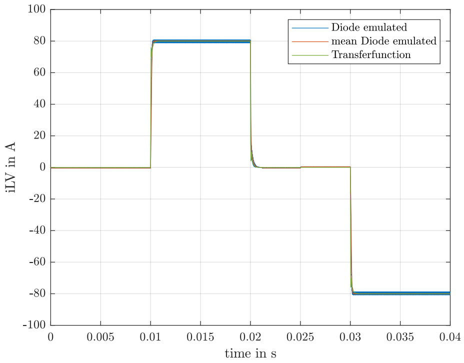


Figure 16. Comparison of the inductor current over time of the switching model with diode emulated mode and the theoretical transfer function.

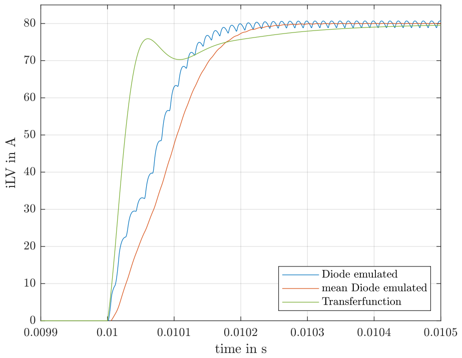


Figure 17. Step response of the switching model with diode emulated mode and the theoretical transfer function.

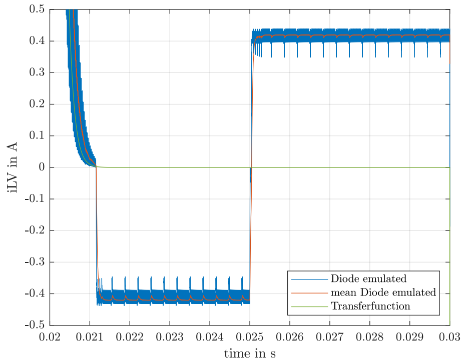


Figure 18. Detailed analysis of the step response showing significant deviation of the model using diode emulated mode. There is a deviation of approximately 0.4 V with the sign depending on the PWM phase.

deviation caused by the `zeroOrderHold` element. This delays the measured current by a hundredth of the period of the switching frequency. Due to this delay, a zero crossing is detected too late and consequently the Sync-MOSFET is also switched off late. The deviation can be reduced by increasing the sampling frequency. This, however, increases the computational effort significantly. The jump at 0.025 s (Fig. 18) is caused by a change at the direction pin. As the measured current is multiplied by -1 due to the differing polarity of the direction, the sign of the deviation changes.

Changing the switching threshold of the Sync-MOSFET from 0 A to 0.4 A shows only a minor deviation and the desired value of 0 A is maintained over the entire range (Fig. 19).

5 Averaging Model

Time-invariant devices, such as switching transistors, are very complex to simulate. They lead to a significant increase in computational effort and thus limit the number of components to be simulated simultaneously as well as the performance. To realize, e.g. a simulation of the behavior of a battery connected to the bidirectional DC-DC converter over several charging processes in order to make statements about aging processes, a suitable model is required (Surewaard, Karden, and Tiller 2003). In the following, a model is designed which neglects the single switching processes but still reproduces the behavior of the switching model as accurately as possible.

The accuracy is lower compared to the switching models, but allows longer simulation duration with several components to be simulated simultaneously.

Without taking losses into account, the averaged output voltage in buck mode is given by:

$$V_{\text{av,ideal}} = (1-D) \cdot V_{\text{HV}} \tag{2}$$

D describes the duty cycle. The following applies to the output current:

$$I_{\text{av,ideal}} = (1-D) \cdot I_{\text{L}} \tag{3}$$

I_{L} is the current through the inductor. In order to consider the ohmic losses of the MOSFETs, the voltage required by them is subtracted from the generated voltage. This results in:

$$V_{\text{av,real}} = (1-D) \cdot V_{\text{HV}} + \frac{P_{\text{V,Rds(on)LV}} + P_{\text{V,Rds(on)HV}}}{I_{\text{L}}} \tag{4}$$

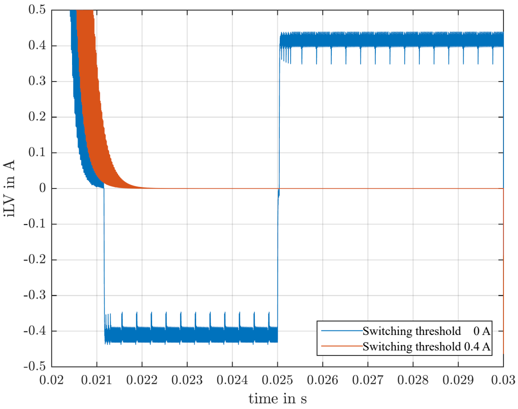


Figure 19. Reduction of the deviation by adjusting the current thresholds.

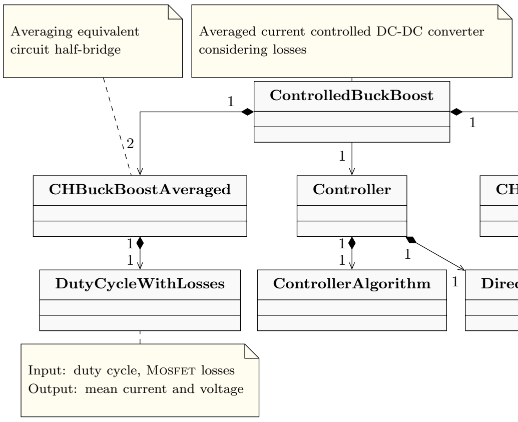

Averaging equivalent circuit half-bridge
Averaged current controlled DC-DC converter considering losses
ControlledBuckBoost
1
1
2
CHBuckBoostAveraged
1
Controller
1
CHCompensation
1
1
DutyCycleWithLosses
1
1
ControllerAlgorithm
1
1
Direction
Input: duty cycle, MOSFET losses
Output: mean current and voltage

Figure 20. Hierarchical overview of the classes of the averaging model.

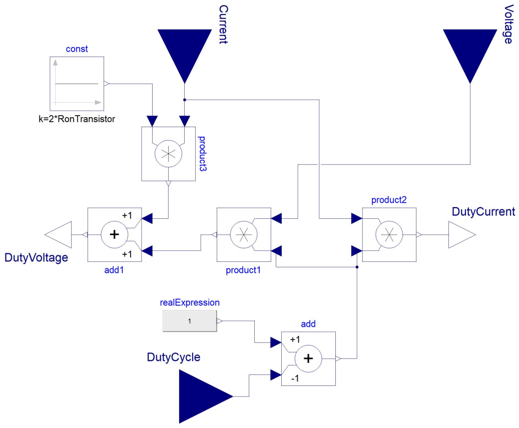


Figure 21. The model DutyCycleWithLosses determines the averaged output current and voltage according to the equations (3) and (4). DutyVoltage corresponds to $V_{\mathrm{av,real}}$ and DutyCurrent to $I_{\mathrm{av,ideal}}$

Assuming that the two MOSFETs are identical, the equation can be simplified to:

$$V_{\mathrm{av,real}} = (2-D)\cdot V_{\mathrm{HV}} + 2\cdot I_{\mathrm{L}}\cdot R_{\mathrm{ds,on}} \tag{5}$$

The averaging model is also hierarchical and is divided accordingly into physical and control simulation (Fig. 20).

5.1 Physical Simulation

At the lowest level of the physical simulation, the averaged output currents and voltages are determined in the model DutyCycleWithLosses as a function of the input voltage or inductor current, the set duty cycle and the power losses of the MOSFETs (Fig. 21). The block add exchanges the DutyCycle so that the behavior of the averaging model corresponds to the one of the switching model. The corrected duty cycle is used together with the measured current and voltage to determine the averaged values, which are passed by the pins DutyVoltage and DutyCurrent to the model CHBuckBoostAveraged.

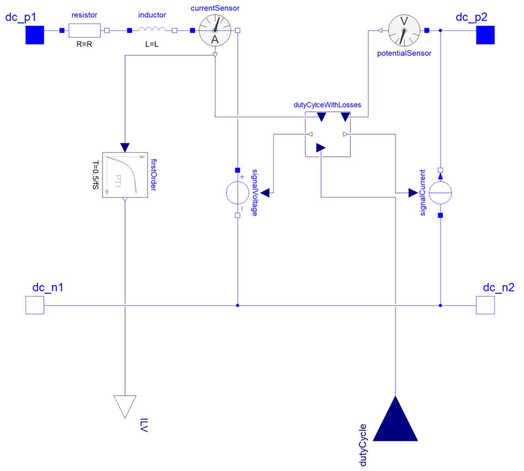


Figure 22. The model CHBuckBoostAveraged is the averaging model of one of the two half bridges. The MOSFETs are replaced by current and voltage sources which provide the average values.

In this model, the MOSFETs are replaced by current and voltage sources which provide the average values (outputs DutyCycleWithLosses). Consequently, the inductor is charged by the voltage source signalVoltage and discharged via the current source signalCurrent. Since there are no ripples in the circuit, no smoothing capacitors are required. Another significant difference to the switching models (Fig. 5) is that the measured current is delayed via a first order proportional time element instead of a zeroOrderHold element. The reason for this is that it is interpreted as a switching element and thus delays the measured value used in the control algorithm.

5.2 Simulation of the Control

The controller blocks are almost identical to those of the switching variants, only the operating mode of the DC-DC converter via the Direction pin is not determined. The additional model CHCompensation adjust the ISETA signal depending of the activated channels. The parameters EnableCH1 and EnableCH2 are the boolean inputs. If both channels are activated, the signal ISETA is halved other it is passed on without change.

5.3 Validation of the Averaging Model

A first comparison between the transfer function and the averaging model shows barely any deviations (Fig. 23). A more detailed comparison shows that the averaging model reacts faster to the jump, but takes longer to compensate the control deviation (Fig. 24). After 0.013 s (Fig. 24) the averaging model reaches the set point of 80 A, while the transfer function has a permanent control deviation of 0.0075 A. One possible reason for the remaining deviation could be that the theoretical transfer function is also an approximation.

6 Comparison of the Models

For the comparison between the models, the simulation processes (Fig. 10) are performed and the results are compared in terms of the computational time and effort. The simulations are executed on an Intel i5-4690K with 3.5 GHz. The experiment is performed at the same interval length of 2E-7 s and the tolerance of 1E-7 s in each case. The high resolution is required for the accuracy of the model with diode emulation mode. As expected, the averaging model is the one with the shortest computation time and only requires 3.83 s (Fig. 25). The strictly complementary switching model takes 13.71 s and the one with diode emulation mode 107.91 s. For the latter, a difference can be seen between the test conditions. If no zero crossings occur (Fig. 25: between the simulation time 0.01 s and app. 0.02 s) , the performance of the model is significantly better compared to the segments with zero crossings and diode emulation (Fig. 25: between the simulation time 0 and 0.01 s). The averaging model requires only 3.5% computation time of the duration of the model with diode emulation and less than a third (27.9%) of the computation time of the strictly complementary switching model. The computation time depends on the equations to be calculated. The diode emulated model requires the most equations with 503370 in total. The strictly complementary switching model uses 25963 equations and the averaging one only 7 equations.

This comparison shows, that the averaging model significantly reduces the computational effort and time. As long as individual switching processes are not relevant, the averaging model is the preferred choice. The strictly complementary switching model can be used if only the rippling signal is necessary, If information about the exact switching processes of the MOSFETs is required, the diode emulated model has to be selected.

7 Parallel Connection of two DC-DC Converters with Load Distribution

As an exemplary use case, the load sharing between two DC-DC converters connected in parallel is simulated. The averaging model with a PI controller is used (Fig. 26). The load sharing is realized in this case for a first test via the two gain blocks. These divide the drive signal between the two DC-DC converters by 70% and 30%.

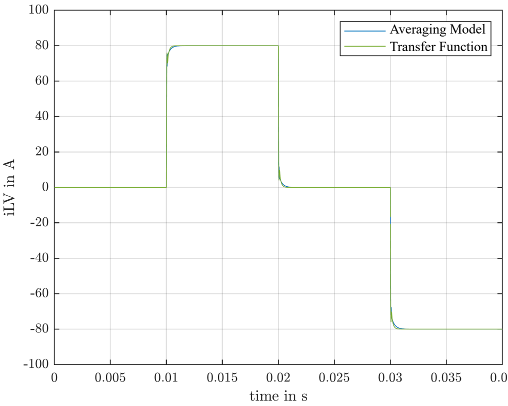


Figure 23. Comparison of the inductor current over time of the averaging model and the theoretical transfer function.

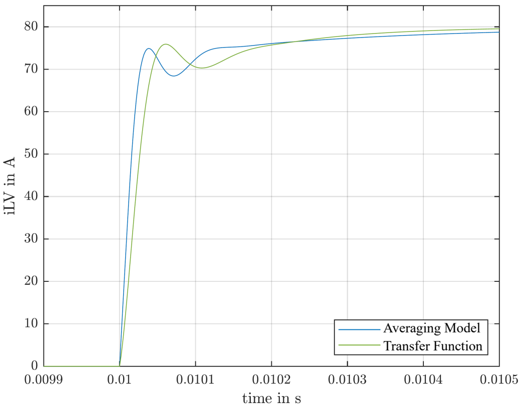


Figure 24. Step response of the averaging model and the theoretical transfer function.

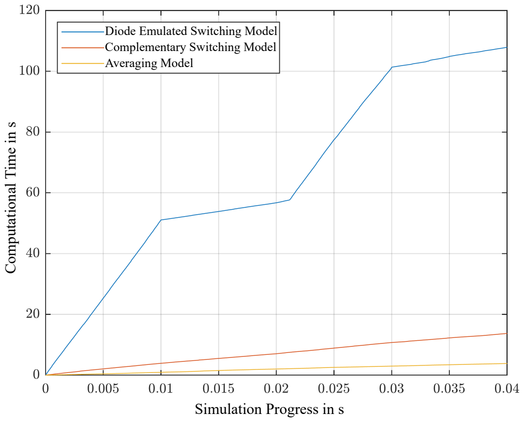


Figure 25. Comparison of the required computation time in relation to the simulation progress of the three proposed models.

The result in Figure 27 shows that the load current is distributed as demanded. The set point of 80 A is reached 1.51 ms later compared to Figure 24. The reason for this is the used PI controller. Parallel connections of the switching models show the same behavior and therefore are not shown.

8 Conclusion and Outlook

Three different models of varying complexity have been proposed to simulate bidirectional DC-DC converters. Two switching models reproduce the individual switching operations and thus provides high accuracy. They differ essentially in driving mode, one of which switches the MOSFETs in a strictly complementary manner while the other one uses diode mode emulation. The strictly complementary switching model emulates current ripples and is suitable for simulation durations up to a few seconds. The model with diode mode emulation switches off the MOSFETs at currents lower than zero. This model is only suitable for

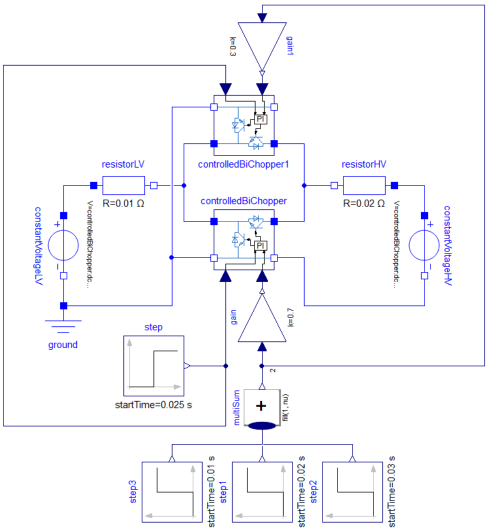


Figure 26. Possible use case of the averaging model: simulation of load sharing between two DC-DC converters connected in parallel, where one (`controlledBiChopper1`) consumes 30 percent of the charging current and the other one (`controlledBiChopper`) 70 percent.

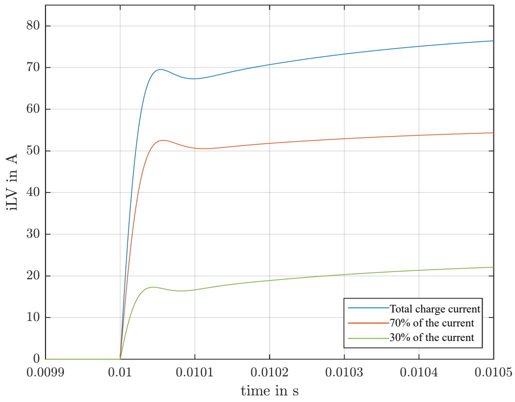


Figure 27. Step response of the parallel connection of two DC-DC converters with load current sharing using the averaging models.

simulations for less than one second, as the model requires high accuracy to provide precise values. For longer simulation durations, an averaging model was proposed. Here, the switching components are replaced with the average values. The model shows clear advantages in terms of computational effort and calculation time, but offers lower accuracy. All three models were successfully validated by comparisons with the theoretical transfer function. The proposed models are the basis for more extensive system simulations. Especially the averaging model offers the possibility to realize longer simulation durations.

In future work, a more detailed consideration of the losses is planned. Separate heat ports for the individual components will be added to analyze the individual losses. Another extension is to make the load sharing between the DC-DC converters changeable during the simulation. The transfer functions of the DC-DC converter were defined in previous work. These will also be integrated into the simulation environment and compared with the averaging models in terms of computational effort and accuracy.

Acknowledgements

The authors thank A. Haumer for guidance in developing the models and C. Schimpfle and M. Farmbauer for helpful questions and insightful discussions. This work was supported by Hanns-Seidel-Foundation grant funded by the German Federal Ministry of Education and Research.

References

Alharbi, Mohammed A. et al. (2019). “Current Ripple Minimisation Based on Phase-Shedding of DC-DC Interleaved Converters for EV Charging System”. In: *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*. Vol. 1, pp. 3456–3462. DOI: 10.1109/IECON.2019.8926959.

Baumann, Martin, Christoph Weissinger, and Hans-Georg Herzog (2019). “System identification and modeling of an automotive bidirectional dc/dc converter”. In: *2019 IEEE Vehicle Power and Propulsion Conference (VPPC)*. IEEE, pp. 1–5.

Laera, Giuseppe et al. (2020). “Object Oriented Modeling and Control Design for Power Electronics Half-Bridge Converter using Modelica”. In.

Navarro, Francisco J Gómez et al. (2020). “DC-DC Linearized Converter Model for Faster Simulation of Lightweight Urban Electric Vehicles”. In: *IEEE Access* 8, pp. 85380–85394.

Reindl, A. et al. (2023 - under review[a]). “Mathematical Modeling of a Bidirectional DC-DC Converter - Part 1: Buck Mode”. In: *AE 2022: 27th International Conference on Applied Electronics (IEEE), Pilzen, Czech Republic*.

Reindl, A. et al. (2023 - under review[b]). “Mathematical Modeling of a Bidirectional DC-DC Converter - Part 2: Boost Mode”. In: *AE 2022: 27th International Conference on Applied Electronics (IEEE), Pilzen, Czech Republic*.

Schuck, Marcel and Robert C. N. Pilawa-Podgurski (2013). “Ripple minimization in asymmetric multiphase interleaved DC-DC switching converters”. In: *2013 IEEE Energy Conversion Congress and Exposition*, pp. 133–139. DOI: 10.1109/ECCE.2013.6646691.

Spiliotis, Konstantinos et al. (2019). “Modeling and validation of a DC/DC power converter for building energy simulations: Application to BIPV systems”. In: *Applied energy* 240, pp. 646–665.

Surewaard, Erik, Eckhard Karden, and Michael Tiller (2003). “Advanced electric storage system modeling in modelica”. In: *Paper presented at the 3rd International Modelica Conference*.

Texas Instruments Incorporated (2016a). *LM5170-Q1 EVM User Guide*. Ed. by Texas Instruments Incorporated. URL: %5Curl%7Bhttps://www.ti.com/tool/LM5170EVM-BIDIR%7D.

Texas Instruments Incorporated (2016b). *LM5170-Q1 Multiphase Bidirectional Current Controller*. Ed. by Texas Instruments Incorporated. URL: %5Curl%7Bhttps://www.ti.com/product/LM5170-Q1?qgpn=lm5170-q1%7D.

Winter, Michael et al. (2015). “Average model of a synchronous half-bridge DC/DC converter considering losses and dynamics”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. 118. Linköping University Electronic Press, pp. 479–484.

Mass Conservation in Vapor Compression Cycles: A Method for Ensuring Consistency with Redundant Dynamic States

Daniel Andersson[1] John Batteh[2]
Matthis Thorade[3] Lixiang Li[4]

[1]Modelon AB, Sweden, {daniel.andersson}@modelon.com
[2,4] Modelon Inc., USA, {john.batteh, lixiang.li}@modelon.com
[3] Modelon Deutschland GmbH, Germany, {matthis.thorade}@modelon.com

Abstract

This paper describes a method to resolve a potential inconsistency when employing redundant dynamic thermofluid states for modeling of vapor compression cycles. Following a brief introduction regarding the motivation and use of redundant thermofluid states, a series of test models ranging from simple component models to complex system models are developed to illustrate the potential inconsistency with Air Conditioning Library. Based on observations of the simulation results from these test models, a method for ensuring consistency is proposed and implemented. The method is then demonstrated on the test suite and evaluated for effectiveness, robustness, and computational efficiency.

Keywords: thermofluid modeling, vapor compression cycles, two phase systems, thermodynamics, state selection

1 Introduction

State selection is a key element of thermofluid modeling. Selecting appropriate dynamic states, or independent thermodynamic variables, is critical for ensuring robust, accurate, and computationally efficient models (Tummescheit 2002). In particular, the following considerations are often made when considering state choices for thermofluid models:

- Thermodynamic considerations to allow full phase identification over the operating regime for the fluid (i.e. single phase, multi-phase, etc.)
- State compatibility with medium model implementation (i.e. choosing states such that medium model can be evaluated explicitly from states as much as possible to avoid unnecessary nonlinear systems)
- Formulation for conservation laws explicitly in state variables (either manually or by Modelica compiler)
- Initialization structure to avoid unnecessary nonlinear systems
- Initialization ease (i.e. typical values specified by the user to describe initial state of system including the use of intensive or extensive variables)

Given the importance of state selection and the tight integration with the medium model implementation, there has been significant focus on state selection and on the development of compatible media packages in the design of the Modelica.Fluid package (Franke 2009) and in various commercial thermofluid libraries. For example, Modelica.Fluid indicates that it is compatible with medium models that have T, (p,T), (p,h), (T,X), (p,T,X) or (p,h,X) as independent variables while other state variables are possible but would require initialization changes [T is temperature, p is pressure, h is specific enthalpy, and X is a mass fraction vector]. For two-phase systems, the standard state choice is (p,h) or (p,h,X) since temperature cannot always be used to uniquely identify the fluid state in the two-phase region.

Mass conservation as denoted by the system charge in vapor compression cycles is a challenge in modeling two-phase systems due to the sharp nonlinearities in thermodynamic properties, especially at the saturated liquid phase boundary. While charge conservation can prove to be numerically challenging, it is an important part of the modeling since variations in charge can produce unphysical responses from the system model, both in steady state and dynamically. Variations in pressures, flowrates, temperatures, heat flow, and key control values such as superheat and subcool due to modeling-induced charge variations can cause not only unphysical results but also computational degradation due to the additional unphysical dynamics. Since a key use case for dynamic models of vapor compression cycles is to develop the system and optimize system charge for operation, these sorts of charge issues are especially problematic from an engineering standpoint.

DOI
10.3384/ecp204693

Excellent work in this area has shown that the use of redundant thermofluid states can greatly improve mass conservation in vapor compression cycle modeling (Laughman 2015). This paper is a follow-up to that work and describes a method to resolve a potential inconsistency that can result when employing the redundant dynamic state approach. Following a brief introduction of the redundant state approach, the results of the potential inconsistency are shown via a series of test models ranging from simple component tests to complex system models using Air Conditioning Library. Based on observations of the simulation results from these test models, a method for ensuring consistency is proposed and implemented. The method is then demonstrated on the test suite and evaluated for effectiveness, robustness, and computational efficiency.

2 Redundant Thermofluid States

This section provides a high-level overview of the motivation for the use of redundant thermofluid states and also describes the potential inconsistency. For a full treatment of the problem, including the derivation of the systems of equations, the interested reader is referred to previous work (Laughman 2015).

Using the standard assumptions for two phase thermofluid systems (one dimensional flow, thermodynamic equilibrium, homogeneous two-phase flow), conservation of mass and energy for a fixed control volume result in ODEs in terms of the following derivatives:

$$\frac{dM}{dt} = V\frac{d\rho}{dt} \tag{1}$$

$$\frac{dU}{dt} = V\frac{d(\rho u)}{dt} \tag{2}$$

where V is the volume, M is the volume mass, ρ is the density, U is the total internal energy, and u is the specific internal energy. These thermodynamic derivatives are then related to the mass, enthalpy, and heat flow into and out of the thermodynamic volume to provide the complete set of ODEs that must be integrated. Using different thermodynamic relationships (Thorade 2013), these derivatives can be written explicitly in different state variables. As discussed previously, the standard choice for two phase thermofluid systems is (p, h). With Equations (1) and (2) rewritten as ODEs in terms of p and h, p and h are integrated via the solver and thus are error controlled by the solver tolerance. Thermodynamic relationships can then be used to calculate other variables such as density ρ(p, h).

In the standard approach with p and h as state variables, there is no explicit error control on the density ρ(p, h). Figure 1 shows the p-h diagram for the common refrigerant R1234yf with isotherms for temperature and lines of constant specific volume 1/ρ. Note that the sharp change in density at the liquid phase boundary. The density isochor increases in slope at the liquid phase boundary, becoming nearly vertical as the pressures are reduced below approximately 6 bar. This nonlinear behavior is especially challenging for numerical integrators and necessitates high precision on the integration of p and h to ensure that the calculated density and thus the mass are accurate. The reasons for this behavior are both that in the low-pressure liquid region, density has a weak dependency on pressure, and therefore there is large sensitivity in the solving of pressure time-derivative from the density and internal energy derivatives, and secondly the density partial derivatives with respect to p and h involved in that solution are discontinuous at the saturated liquid line. In general, precision can be increased by tightening integrator tolerances. In heat pump operation at cold ambients, it is not uncommon for the vapor cycle to operate with the high-pressure side in the range of 4-8 bar. Depending on the level of subcooling in the system, the condenser outlet condition and the high-pressure part of the system up to the expansion valve could have state points that reside just at the liquid phase boundary. These volumes are critical for charge conservation, as the accumulated error in the integration of p and h can lead to large errors in density and thus changes in system charge.

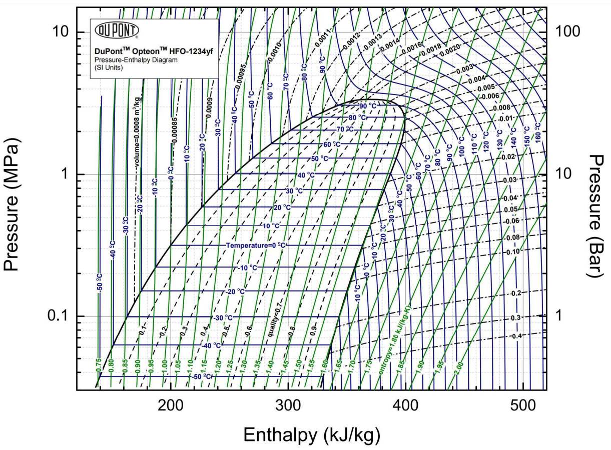


Figure 1. p-h diagram for R1234yf

As proposed in previous work (Laughman 2015), adding density as a redundant state can greatly reduce or even eliminate charge conservation issues. With density as a state, density is also error-controlled by the integrator. The authors propose the following additional equation:

$$\frac{dh}{dt} = \left.\frac{\partial h}{\partial P}\right|_{\rho} \frac{dP}{dt} + \left.\frac{\partial h}{\partial \rho}\right|_{P} \frac{d\rho}{dt} \tag{3}$$

Equation (3) can be combined with the formulated equations from Equations (1) and (2) to provide the set of ODEs that can be integrated for the state variables p, h, and ρ. This approach has the benefit of keeping the standard state variables p and h and thus allowing medium models explicit in p and h (such as the Spline-Based Table Look-Up (SBTL) medium implementations in Air Conditioning Library which have demonstrated significant improvements in computational speed, especially when combined with analytic Jacobians (Li 2018; Li 2020)) to continue to be used even with the redundant density state.

In a subsequent publication (Laughman 2017), the authors acknowledge the potential inconsistency in the redundant state approach:

> *While this approach in theory relaxes the constraint forcing the state variable being integrated to be equal to the computation of the specific enthalpy as a function of the other state variables, for example, h(P, ρ), simulations presented later in the paper provide evidence that these deviations are small in practice.*

Though clearly true in the simulations presented in the cited publication (Laughman 2017), the authors of this work have observed these inconsistencies and their manifestation in system charge issues as described in the following section.

3 Simulations with Inconsistency

As outlined in Section 2, there is a potential inconsistency that can result from the redundant state approach. This section describes a series of models that demonstrate the impact of this inconsistency between the dynamic states. These models are built using Air Conditioning Library (Modelon 2023) and simulated with Dymola (Dassault Systemes 2023).

3.1 Single Volume Test

Since it can be difficult to put models in the state to demonstrate the inconsistency, it is valuable to have simple models that can be easily manipulated, either via initialization and/or via dynamics, and whose correct results are clearly understood. Figure 2 shows a model of a fixed volume with a trapezoidal heat input. The model is initialized such that the volume state is right at the liquid phase boundary (denoted by the blue circle on the p-h diagram) as this state is extremely sensitive due to the sharp changes in density at the liquid phase boundary. The heat input is manipulated such that the volume state repeatedly just enters the two-phase region and then leaves again (into the liquid region) as heat is added and the pressure increases. Since there is no mass flow into or out of the volume, the correct physical result is that the mass is constant, and the fluid state should move along a constant density line. Since we repeatedly add and remove the same amount of heat, all thermodynamic states should also return to their initial values between each full heating and cooling cycle.

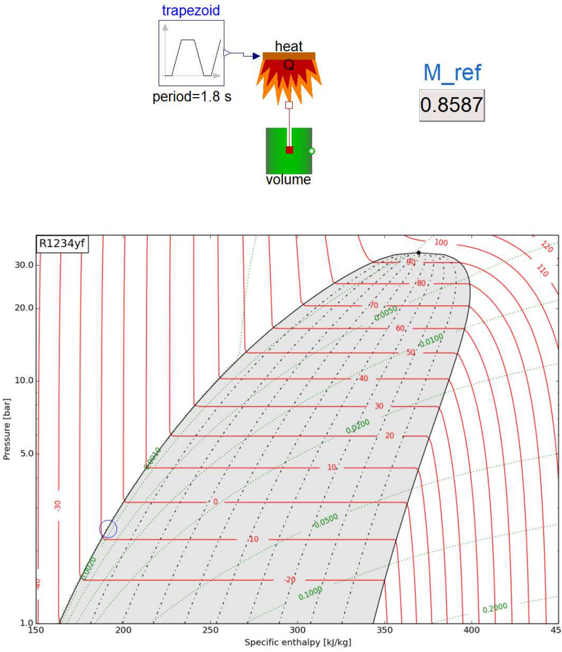


Figure 2. Fixed volume test with heat input

In Air Conditioning Library, users can select different state choices at the top level of the model via the systemACL record. The option for the redundant (p, ρ, h) states is enabled for these simulations. Simulations are run with the SBTL 1234yf medium model in Air Conditioning Library with a relative tolerance of 1e-7. Figure 3 shows simulation results from the fixed volume test. The top plot shows the refrigerant mass in the control volume which is constant as expected when the density is an integrated state. However, the second plot shows that the density state in the volume is not equal to the density ρ(p, h). Thus, there is an inconsistency between the integrated states ρ, p, and h. The third and fourth plots show the pressure and specific enthalpy, indicating that they do not return to their initial values after many heat injections and extraction cycles as they should.

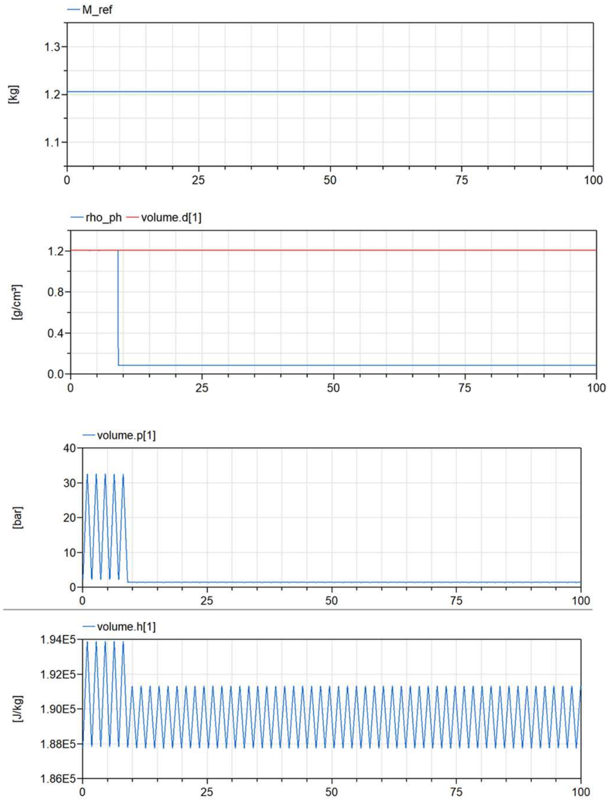


Figure 3. Simulation results from fixed volume test. System mass (top), inconsistent values for density from integrated density state and ρ(p, h) (second figure), volume pressure (third figure), and volume enthalpy (bottom).

3.2 Two Volumes and Flow Model Test

One issue that was reported a few times by users of Air Conditioning Library when using redundant states (p, d, h) is the appearance of negative density values. Negative density is clearly an inaccurate result and often followed by simulation crashes. A hypothesis is that these negative values are caused by inconsistent redundant states along these lines:

1. Consider a simple system of two control volumes connected by an orifice model.
2. An inconsistency in pressure in one of the volumes will induce a mass flow rate from one volume to the other.
3. In a standard (p, h) state model a numerical inaccuracy in pressure would cause a change in density too, as density is just an algebraic function of p & h, and the mass flow would proceed until the pressures of the two volumes are equal again.
4. In a redundant state model though, it can be imagined that a low-density volume has a sudden inaccurate increase in pressure, driving a mass flow out of the volume. In the redundant state model, that inaccuracy does not cause the density to increase, but the mass flow out of the volume will cause the density state to decrease from the accurate value. Therefore, it is postulated that inconsistent redundant states can indirectly cause (locally) incorrect densities via the flow models (momentum balances) of the system although density is under solver error control, and in some situations reach negative values if sufficiently high flow rates out from an already low-density volume is induced by the inconsistency.

To verify the above idea and potential solutions, the authors attempted, and successfully reproduced negative density in a relatively simple model, shown in Figure 4. This model includes the same fixed volume with repeated forced heat injection and rejection and is further connected via an orifice model to another adiabatic, fixed volume, and that is in turn also connected via another orifice model to a fixed pressure reservoir. The two connected volumes allow representation of the effect of mass flow rate induced by inaccurate pressure appearing in the system, and the fixed pressure reservoir prevents the pressure level from going far too high or low. Additionally, the heat flow rate to the first volume is controlled, such that in the event of numerical inaccuracy causing the volume to deviate from the vicinity of the sensitive saturated liquid line, heating will be adjusted to bring the volume state back there again. The idea is to trigger as much state inconsistency as possible and let it accumulate over many heat cycles.

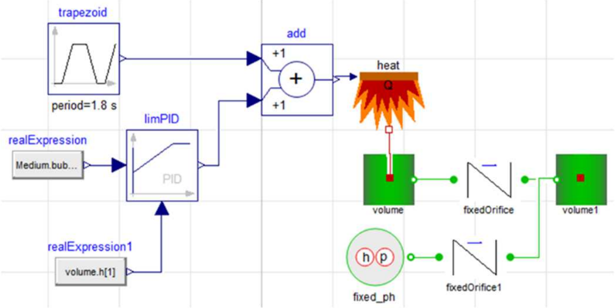


Figure 4. Negative density reproducer model

Simulating this model indeed resulted in build-up of state inconsistency, eventually reaching negative density in the adiabatic volume, followed by a simulation crash.

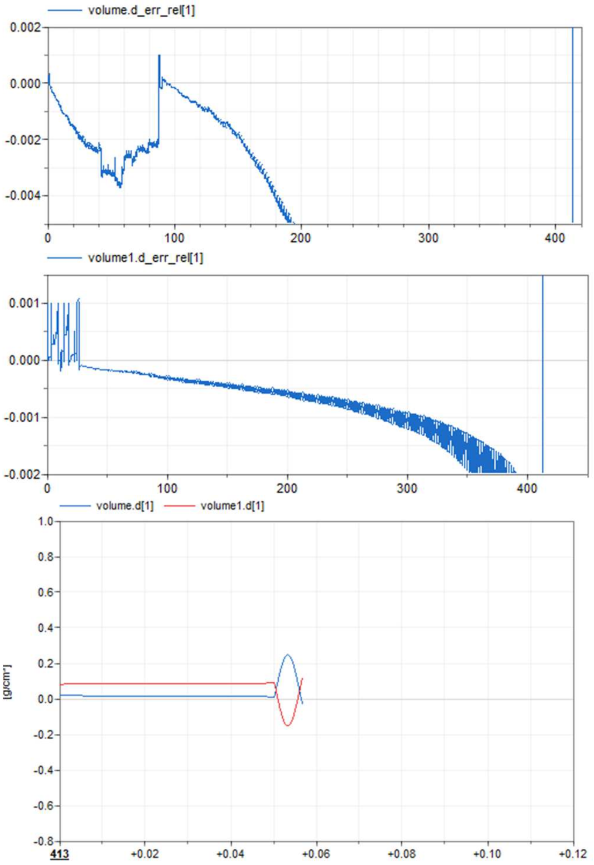


Figure 5. Negative density reproducer results. Relative error in density state and density from p & h states in non-adiabatic volume (top), adiabatic volume (middle). Densities in both volumes at the end of simulation (bottom).

Figure 5 show the gradual build-up of state inconsistency, illustrated by the relative difference between the density state variable and density computed as a medium property function evaluation from pressure and enthalpy dynamic states in each volume. The total system mass cannot be used to evaluate consistency here because the model is not a closed system. Toward the end of the simulation, the density dynamic state in the adiabatic volume also shows a negative value for some time followed by sign changes in both density states and then simulation crash.

4 Method for Consistent Redundant Dynamic States

4.1 Method Overview

Given the possibility for inconsistency between the redundant states and the observation of this inconsistency in test models, a method for ensuring consistency between the redundant states is proposed. This method attempts to solve the redundant state inconsistency problem via computation of the density from the (p, h) dynamic states and comparison with the density dynamic state. This comparison is done for every control volume in the system and continuously throughout the transient simulation. If the deviation exceeds a certain allowed relative error (which can be set via a model parameter), the pressure state is corrected such that the three states are consistent again. This correction is done using the Modelica `reinit` operator and thus happens momentarily in an event which is triggered when the inconsistency is large enough. For simplicity, the new pressure value is taken as a medium property function of pressure from density and temperature, where temperature is as function of pressure and enthalpy. This approach is a simple alternative because the medium package already has equation of state implemented as a function with these inputs.

4.2 Variants Tested

There are multiple possible variants to formulate the inconsistency criterion. The criterion can be specified as tolerance on any of the three state variables and compared to the same property computed from the other two. The authors currently have only tested a consistency criterion for density.

There are also different possible choices of what or which state(s) to re-initialize and how to compute the state to re-initialize to. It was tested to reinitialize enthalpy instead of pressure, but this approach was not at all equally successful. Pressure seems to be the state with most inaccuracy in the low-pressure liquid region and thus could explain why it works better to initialize it.

Lastly, a good value must be found for the maximum allowed relative error of checked state (density in this case). Too tight tolerance may give very frequent re-initializations, reducing simulation performance due to large number of events, and too loose tolerance may fail to keep the states sufficiently consistent to mitigate the original issues.

4.3 Further Improvement

In the initial testing of the reinit approach, the authors observed promising results for maximum relative error in the range of 1e-3 – 1e-4, but for tighter tolerances, observed unexpected results trending towards those of the original redundant states model without consistency check and correction. After some investigation the authors found that this behavior was a result of the simple calculation of the corrected pressure as a function of ρ and T. Since the consistency check uses the medium density from pressure and enthalpy function and reinit used the pressure from density and temperature function, a small inconsistency between those two functions could cause the re-initialized pressure not to fulfill the consistency criterion. In this case the simulation proceeds with no further correction, unless by chance the states become consistent again. Accurate equation of state for two-phase fluid properties are only explicit for one causality, meaning that one of the two functions used in the detection and correction of inconsistent states will include iteration. Therefore, they will only be consistent to a certain numerical tolerance that is part of the medium property model (and not related to ODE solver tolerance or the newly introduced maximum allowed density error parameter), and the reinit method needed to be improved to guarantee that after the reinit the three dynamic states are consistent according to the criterion introduced.

To achieve this response, the authors compute the potential new pressure as before but then the convergence criterion is evaluated. If the convergence criterion is fulfilled, the corrected pressure is taken as the new value for pressure state as before. If the criterion is not fulfilled, Brent iteration of pressure is employed until a new pressure is found such that density computed from p and h is within allowed tolerance. With this improvement to the method, the authors observed the expected trend of ever better consistency for smaller maximum allowed inconsistency, and computed absolute relative density error was never greater than the given allowed tolerance.

5 Simulations with Consistent Redundant States

This section provides results from the test suite from Section 3 using the method described in Section 4 to ensure consistency between the redundant dynamic states.

5.1 Single Volume Test

The single volume test shown in Figure 2 was evaluated to understand the impact of the density relative error tolerance. Pressure correction at density relative error of 1e-3, 1e-4 and 1e-5 was compared to the original results with uncorrected redundant states. Figure 6 shows huge error without any correction while the results are clearly staying within allowed error at different settings. Smaller allowed error can be seen increasing the frequency of corrections. Figure 7 shows density from (p, h) states. For the fixed volume in this test, a constant density is the correct result here. All corrected variants are much better than the uncorrected. For this particular model, CPU time does not vary much depending on tolerance here, and uncorrected is faster, likely because in such a simple model, inconsistency doesn't lead to expensive artificial transients.

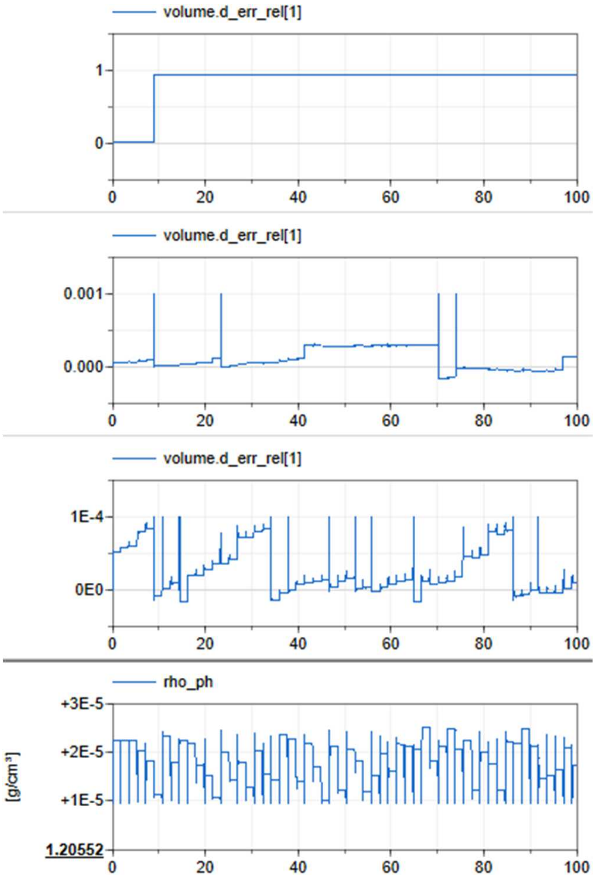


Figure 6. Density relative error for uncorrected redundant states (top) and with maximum allowed error 1e-3 (2nd plot), 1e-4 (3rd plot) and densities at 1e-5 (4th plot)

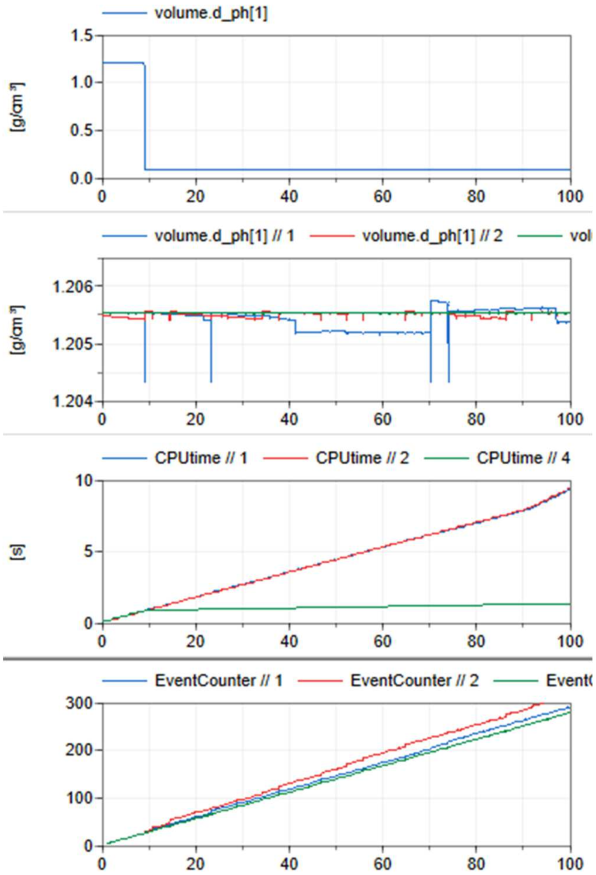


Figure 7. Density for uncorrected redundant states (top) and with maximum allowed error 1e-3 (2nd plot blue), 1e-4 (2nd plot red) and 1e-5 (2nd plot green). CPU times 3rd plot, event count 4th plot.

5.2 Two Volumes and Flow Model Test

The test shown in Figure 4 for negative density was evaluated to assess the original method and the improved method discussed in Section 4.3. Figure 8 shows the original method for state correction without Brent fallback. The relative error goes outside the maximum allowed error of 1e-3 at green circles.

Figure 9 shows results from the same model with the improved reinit and Brent iteration fallback. With this improved approach, the density error always stays within the tolerance. Figure 10 shows the volume densities. With the improved correction, volume densities are never close to zero. As can be observed, the corrections get very frequent toward the end. This result is likely due to unreasonable experiment setup with a high amount of heat injected.

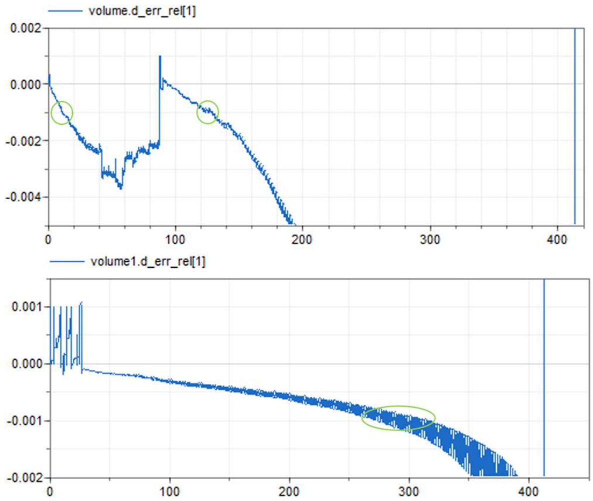


Figure 8. State correction without Brent fallback. Density relative error in first heated volume (top) and second volume (bottom).

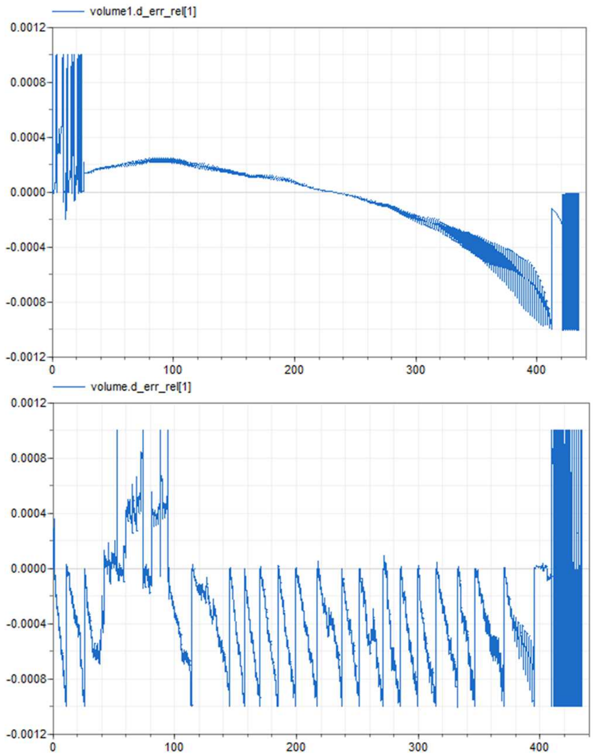


Figure 9. State correction with Brent fallback, Density relative error in first heated volume (top) and second volume (bottom).

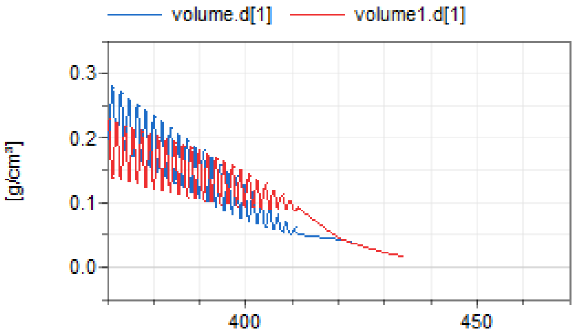


Figure 10. Volume densities

5.3 Complex System Models

This section illustrates results from complex vapor cycle system models. Figure 11 is an example system model from Air Conditioning Library using (p, ρ, h) states (AirConditioning.Examples.TXVCycleOnOff_pdh). Figure 12 compares results from the uncorrected redundant states with the method for ensuring state consistency. Note that this model does not have a large issue with mass conservation. However, employing the method for state consistency even with a 1e-3 tolerance for density inconsistency improves mass conservation significantly. The CPU time cost is low. The computed total system mass shown here is based on density from p & h function calculations. Observing the charge as per density states shows nearly perfect preservation.

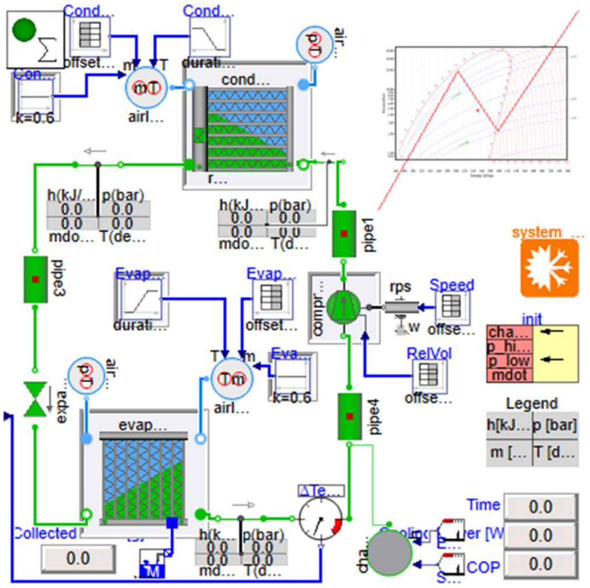


Figure 11. Air Conditioning Library example using (p, ρ, h) states

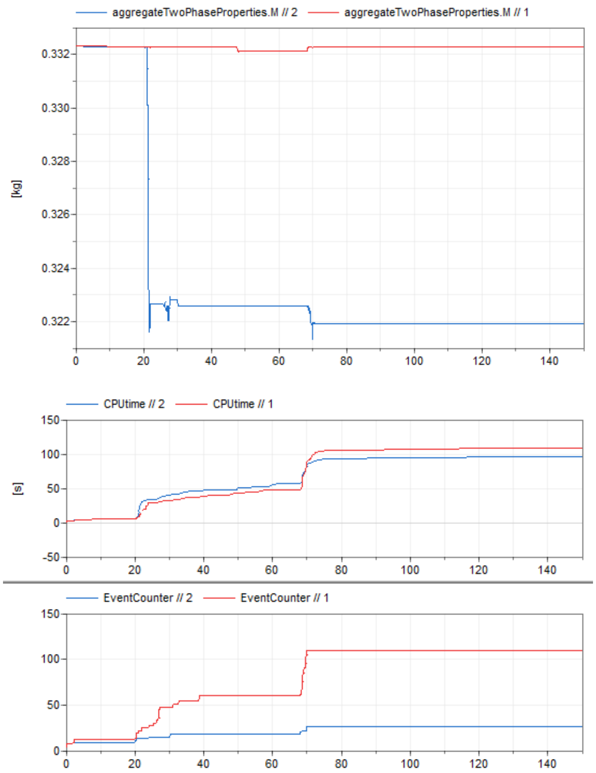


Figure 12. Air Conditioning Library example model with uncorrected redundant states (blue) and with maximum allowed error 1e-3 (red). System mass 1st plot, CPU times 2nd plot, event count 3rd plot.

The method for state consistency was also evaluated on a complex customer model (note that this model cannot be shown graphically as it is customer proprietary). This model uses R1234yf SBTL and has 505 (p, h) states. It was run using CVODE at a tolerance of 1e-7. The original results with the uncorrected redundant states are shown in Figure 13. The results show a significant mass defect and are thus problematic.

Figure 14 show results from the same model now set to use (p, ρ, h) states (591 states in total) and employing the method for correction of the redundant states with a tolerance for density of 1e-3. The figure shows results from the initial method for correction (blue) and improved with Brent (red). Both methods preserve mass much better than p, h (note reported total mass is computed from the p & h states). Note that the improved correction shows no CPU time penalty when compared with uncorrected (p, h) states.

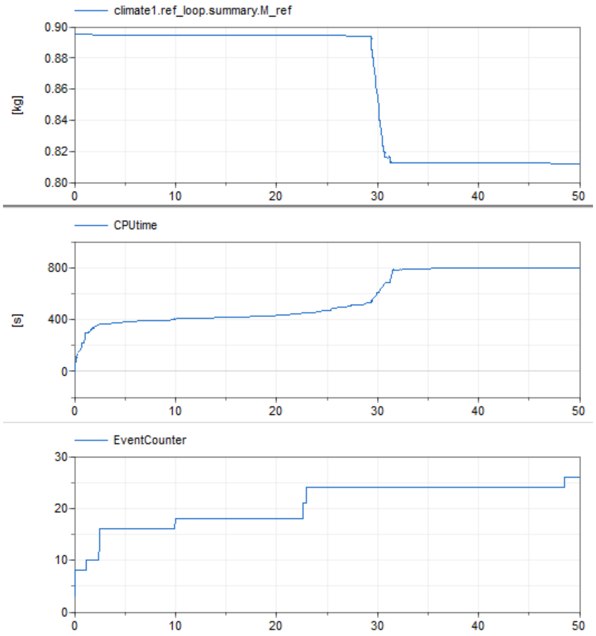


Figure 13. Customer vapor cycle system model with uncorrected redundant states. System mass 1st plot, CPU time 2nd plot, event count 3rd plot.

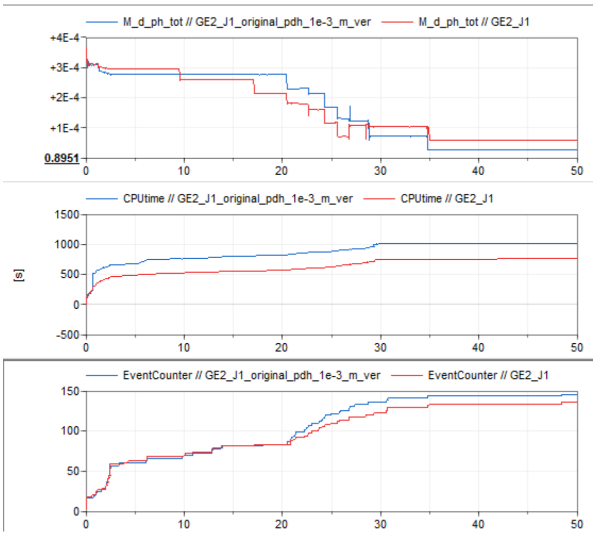


Figure 14. Customer vapor cycle system model with maximum allowed error 1e-3. System mass 1st plot, CPU time 2nd plot, event count 3rd plot.

Figure 15 shows results with a tightened density error tolerance of 1e-4. The initial reinit approach (blue) is compared with the improved approach with Brent (red). Here it can be observed that mass defects from correction are not successfully corrected to a consistent state with the initial method. The improved method is faster and more accurate. The CPU time is still roughly on par with uncorrected (p, h) states for the improved correction.

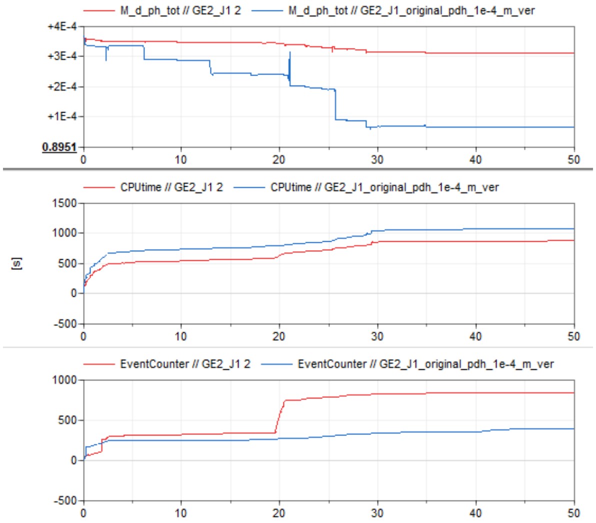


Figure 15. Customer vapor cycle system model with maximum allowed error 1e-4. System mass 1st plot, CPU time 2nd plot, event count 3rd plot.

Figure 16 shows results with a further tightened density error tolerance of 1e-5. The initial reinit approach (blue) is compared with the improved approach with Brent (red). The initial method at this setting demonstrates worse preservation than only (p, h) as states. This response is probably because the correction is mostly dysfunctional due to running in an inconsistent state, and somehow the added density states introduce some additional inaccuracy when exerted to mass flows caused by pressure state inaccuracy. The improved correction performs well. Mass is very well preserved, and relative density error never exceeds 1e-5. There is a CPU time penalty versus only (p, h) as states of approximately 25%.

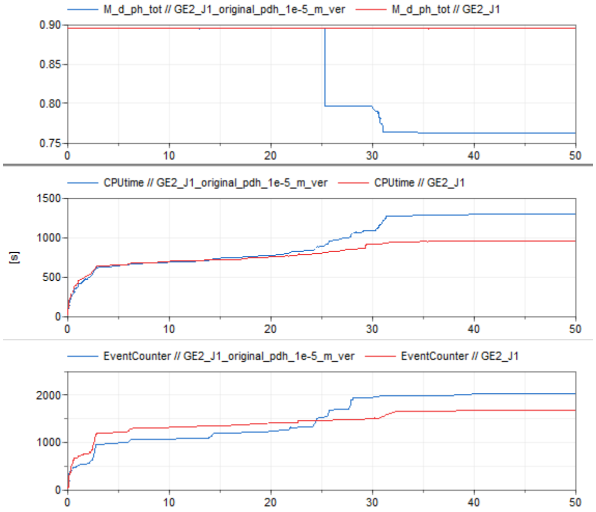


Figure 16. Customer vapor cycle system model with maximum allowed error 1e-5. System mass 1st plot, CPU time 2nd plot, event count 3rd plot.

6 Conclusions

This paper describes a method to resolve a potential inconsistency when employing redundant dynamic thermofluid states for modeling of vapor compression systems. While the impact of redundant thermofluid states is generally positive in terms of mass conservation and even CPU time, it is possible to observe inconsistencies in the separately integrated states as shown in this work. These inconsistencies could manifest not only as mass loss but also as incorrect pressure and enthalpy in the system as shown in the simulation results from both simple and complex models. These inconsistencies can certainly affect the engineering utility of these models if not resolved. A method is proposed and implemented to resolve any inconsistencies as detected during the transient simulation. This method has been tested on models that demonstrated inconsistencies and shows promising results in terms of consistent results and computational impact. These changes have been implemented in Air Conditioning Library 1.26 to ensure that users who employ the redundant state option will not only preserve charge but also will see consistency between the density, pressure, and enthalpy states.

References

Dassault Systemes (2023), Dymola, https://www.3ds.com/products-services/catia/products/dymola.

Franke, R., F. Casella, M. Sielemann, K. Proelss, M. Otter, and M. Wetter. (2009). Standardization of thermo-fluid modeling in Modelica.Fluid. In *Proceedings of the 7th Modelica Conference*, Como.

Laughman, C. and H. Qiao. (2015). Mass Conserving Models of Vapor Compression Cycles. In *Proceedings of the 11th International Modelica Conference*, Versailles, France. doi: 10.3384/ecp15118759.

Laughman, C. and H. Qiao. (2017). On the influence of state selection on mass conservation in dynamic vapour compression cycle models. *Mathematical and Computer Modeling of Dynamical Systems*, Volume 23, Issue 3. https://doi.org/10.1080/13873954.2017.1298625

Li, L., J. Gohl, J. Batteh, C. Greiner, and K. Wang. (2018). Fast Calculation of Refrigerant Properties in Vapor Compression Cycles Using Spline -Based Table Look -Up Method (SBTL). *Proceedings of the 1st American Modelica Conference*, p. 77 -84. DOI: http://dx.doi.org/10.3384/ecp1815477

Li, L., J. Gohl, J. Batteh, C. Greiner, and K. Wang. (2020). Fast Simulations of Air Conditioning Systems Using Spline-Based Table Look-Up Method (SBTL) with Analytic Jacobians. *Proceedings of the American Modelica Conference 2020*, p. 64 -72. DOI: https://doi.org/10.3384/ecp2016964

Modelon (2023). *Air Conditioning Library*. https://modelon.com/library/air-conditioning-library/.

Thorade, M. and A. Saadat. (2013). Partial derivatives of thermodynamic state properties for dynamic simulation. *Environmental earth sciences*, 70(8), pp.3497 -3503. DOI: 10.1007/s12665-013-2394-z

Tummescheit, H. (2002). *Design and Implementation of Object-Oriented Model Libraries using Modelica.* PhD thesis, Lund Institute of Technology.

Calibration Workflow for Mechanical and Thermal Applications

Tim Willert[1] Peter Sundström[2]

[1]Modelon K.K., Japan, tim.willert@modelon.com
[2]Modelon AB, Sweden, peter.sundstrom@modelon.com

Abstract

The calibration of models against measurement data is important to ensure model dynamics that are close to its real-world system. Derivative-free minimizing methods can be used for any model calibration regardless of continuous differentiability requirements, and find a (local) minimum in a reasonable number of iteration steps.

A user-friendly, python-based calibration Dash app to use with the cloud-based Modelica platform Modelon Impact is introduced. Basic calibration setup is done through the GUI of the app and graphical feedback (i.e. plots) is provided.

Two example calibrations are shown: A mechanical Furuta pendulum that only uses Modelica Standard Library components is calibrated against real-world measurement data, and a low-fidelity heat exchanger testbench model that uses Modelon's Air Conditioning Library is calibrated against a corresponding high-fidelity model.

Keywords: calibration, Modelon Impact, Dash, Nelder-Mead, Modelica, Air Conditioning Library, optimization

1 Motivation

Physical modeling can be used to optimize existing systems, predict the behaviour of a system under different boundary conditions, or design new systems.

When working with a system that exists in the real world, it is important that the system model shows the same dynamic behavior as the existing, real-world system with reasonable accuracy. This requires model calibration of relevant system parameters using real-world measurement data.

When working on the detailed design of a complex component, it is common to use testbenches with a high-fidelity model. The finalized model is then integrated in a larger system model. Often, this requires to switch from a high-fidelity to a low-fidelity model to allow for reasonable computation times. The uncalibrated low-fidelity model can behave differently from the high-fidelity model. Therefore, it is required to calibrate the low-fidelity model using high-fidelity model simulation data.

2 Optimization

Following Olsson, Mattsson, and Elmqvist (2006), we can mathematically describe the general problem with a number of measured/simulated inputs v_i, outputs w_i, and a relation between them

$$w_i = m(p, v_i) \tag{1}$$

where $m()$ describes our model function, and p describes system parameters. The goal of the optimization problem is to minimize the residuals by finding the optimal set of parameters p:

$$r_i(p) = m(p, v_i) - w_i \tag{2}$$

Typically, we are interested in more than one residual. Therefore, we can weight and combine all of the residuals into one scalar to be minimized:

$$f(p) = \sum_i k_i r_i^2(p) \tag{3}$$

where k_i is a weighting factor. Note that here the least squares formulation is used. The choice of good weighting factors is difficult. Finding good weighting factors has to be done with the problem formulation in mind and a good understanding of the underlying equations. It is also possible to formulate it using the square root or even custom error functions. Our optimization problem can then be represented as

$$f(p_{\text{opt}}) \leq f(p) \tag{4}$$

with f as our objective function, and p_{opt} as the optimal set of parameters. In this case, p can be the set of all possible design parameters and therefore the equation describes the global optimum. This is typically unfeasible to solve for. In the following, it is assumed to solve for a local optimum.

There are different methods and approaches to find the optimal solution for the problem at hand.

Normally, a sensitvity analysis is carried out preliminary to a calibration. This will not be discussed here.

In the following subsections, three optimization methods are described.

2.1 Parameter Sweep

A parameter sweep is a simple method where simulations are run repeatedly with a range of different parameter values. The batch of simulation results (outputs) can then be compared to the target data (inputs). Using a compare algorithm or visualization tools, parameters can be picked.

DOI
10.3384/ecp204703

Since each simulation run with a set of parameters is independent from the others, each can be run completely in parallel. This method does not pose any model restrictions in terms of continuity, differentiability and/or events.

As picking the set of parameters is not based on an optimization algorithm, it is unlikely to find an optimal solution within a reasonable number of iteration steps.

2.2 Advanced Methods

For the advanced methods, picking a set of parameters follows a certain pattern. Typically, after a few initializing simulation runs, the next set of parameters is picked based on previous results and/or on (local) derivatives. Whether derivatives are used or not has different implications for the model architecture and end applications. This iterative process continues until a termination condition is met, e.g. the difference of the objective function results from consecutive simulation runs falls below a threshold.

In general, it is recommended to give bounds to the system parameters. Otherwise, it is possible to find a mathematically optimal solution which is unphyscial (e.g. negative joint friction) or economically not feasible (e.g. heat exchanger with the size of a space ship). Since the minimization can be sensitive to the choice of initial parameter sets, it is common to run the optimization method of choice several times with different initial parameters.

2.2.1 Derivative-free Methods

Derivative-free methods involve algorithms that work without using derivatives. The methods can be categorized depending on the type of method and shape of objective function. For an extensive overview see Larson, Menickelly, and Wild (2019).

A prominent example is the Nelder-Mead method (Nelder and Mead 1965) which uses $n+1$ test points arranged as a simplex for an optimization problem with n parameters. By reflection, expansion, inner contraction and shrinkage of the simplex (see Figure 1), a local minimum can be found. For a description with an example implementation in C see Press et al. (1992).

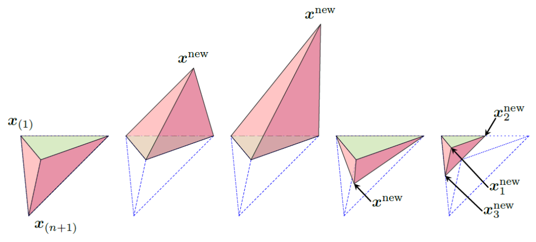


Figure 1. Nelder-Mead simplex operations: original simplex, reflection, expansion, inner contraction, and shrinkage (Larson, Menickelly, and Wild 2019).

Other examples of derivative-free optimization methods are the Powell method (see Powell (1964)) and the further developed COBYLA method (see Powell (1994)).

Since the objective function is not required to be continuous and differentiable, derivative-free optimization methods can be applied to any function. This is especially useful for complex models in Modelica that involve model discontinuities (such as media phase changes) or events. As opposed to the parameter sweep, with these methods it is more likely to find a (local) optimal solution with a limited number of iteration steps. Depending on the used algorithm, parts of the implementation can be parallelized, e.g. the calls to the objective function within one iteration of the Nelder-Mead algorithm.

For (two times) continuously differentiable objective functions, these methods can be inefficient to find the (local) minimum.

2.2.2 Derivative-based Methods

Derivative-based methods involve algorithms that use derivatives of the objective function. This means our underlying system model needs to be continuously differentiable or have even stricter requirements. Depending on the used method it might also be necessary to explicitly give the derivative in form of the Jacobian as an extra input.

This sub-category of optimization methods can for example be used for techno-economic assessments of well formulated system models (Köppen et al. 2022). Since our focus is the calibration of general mechanical and thermal applications, regardless of differentiability and continuity, further explanation of derivative-based methods is left out.

2.3 Existing Frameworks

In this subsection, some existing frameworks are introduced. One of them has been tested with the thermal application described below. The others serve as a reference for an interested reader to test and compare.

All of the introduced frameworks use functional mock-up units (FMU) and work in a python environment. This makes the user independent from Modelica tools, but is as a stand-alone workflow less integrated in the modeling process.

ModestPy

ModestPy (see Arendt et al. (2018)) is a discontinued open-source python package for parameter estimation in FMUs. Available algorithms are genetic algorithm, (legacy) single-process genetic algorithm, pattern search and some SciPy solvers. These methods can be used in a sequence within the framework, which makes it a powerful framework. It outputs several plots that help to analyze the results and parameter interdependencies. The whole package is command line based and changes to the setup are done in the python code. There is no direct graphical interface.

For an inexperienced user, the framework can be difficult to use even for simple applications. Without the knowledge of the different algorithms and how to effectively use them sequentially, this framework can be too complicated.

EstimationPy

Similarly to ModestPy, EstimationPy (see *EstimationPy* (2023)) is a Python package for the estimation of state and parameters of dynamic systems that conform to FMI standard. It uses the packages NumPy and SciPy for the computation , is compatible with Pandas DataFrames and DataSeries and relies on PyFMI and Assimulo to run the model simulations. It goes beyond typical calibration problems, such as model-based fault detection. It also gives the user the option to use Kalman filter to solve state estimation problems. The whole package is command line based and changes to the setup are done in the python code. There is no direct graphical interface.

For an inexperienced user, the package can be difficult to use even for simple applications.

AixCaliBuHA

AixCaliBuHA (see Wüllhorst et al. (2022)) is a framework that aims at automatizing the process of calibrating models used in Building and HVAC simulations. As opposed to the aforementioned python packages, this framework has a focus on calibrating models. It includes the capability to perform a sensitivity analysis and several visualization options that help to analyze the results. For the optimization method, it is possible to choose between SciPy's differential evolution method, SciPy's minimize methods, and dlib's minimize (implemented in C++). The whole package is command line based and changes to the setup are done in the python code. There is no direct graphical interface.

For an inexperienced user, the package can be difficult to use even for simple applications.

This framework was tested with the thermal application described here and delivered reasonable results.

3 Technical setup

In the following, the general calibration setup for two different models - one mechanical and one thermal - is outlined. The model specific setup is described in section section 4.

On the modeling side, the cloud-based platform Modelon Impact is used. It is not necessary to manually convert the model into a FMU. There are no specific requirements to the model itself. A python-based dashboard app created in Dash (https://dash.plotly.com) connects to Modelon Impact using the Impact Python Client (https://github.com/modelon-community/impact-client-python). The dashboard uses the client to set up the workspace, get and set parameter values and compile and run the model. Dash is low-code frameowrk to build data apps in Python. The used optimization algorithm is the Nelder-Mead algorithm that is one method included in `scipy.optimize.minimize`. Note that it is also possible to use the Powell or Cobyla method from the same python module or `differential_evolution`, as these are derivative-free methods as well. The underlying objective function is constructed as the sum of the squared error over the time window for the calibration. A simplified graph of the base setup can be seen in Figure 2.

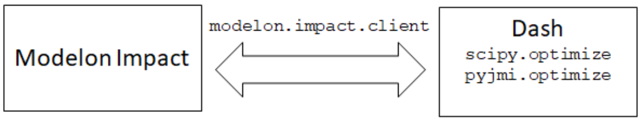


Figure 2. Simplified graph of software setup.

4 Applications

The calibration app is structured with three tabs: Model, Measurement, and Calibration.

The Model tab is used to pick a workspace that is stored in the user's Modelon Impact account, and the model of interest inside that workspace. The picked model is then to be compiled and simulated for an adjustable stop time. For verification, model variables can be plotted.

The Measurement tab is used to load in the measurement data. Allowed formats are `.mat` and `.csv`. For verification, the data can be plotted.

The Calibration tab is used for the actual calibration process. Model variables of interest (output w_i as in section 2) are assigned manually to the corresponding measurement variable (input v_i as in section 2). The variables can be weighted (weight k_i in section 2). The relevant system parameters (p in section 2) are picked and the bounds can be assigned as well as a nominal value which serves as start value. It is possible to pick multiple system parameters for a single calibration. In the following examples, only two parameters are picked for each application. Note that with each extra parameter, the number of iteration steps will increase and thus can become unfeasible to solve the calibration within a reasonable amount of time. The calibration time interval can be picked and finally the calibration algorithm is started. When the calibration is finished, a plot shows the nominal model variables, measurement variables and the calibrated model variables. The nominal and calibrated simulation run results are stored in the model in Modelon Impact.

4.1 Mechanical

A real-world furuta pendulum is modeled in Modelica. For the model to show the same behavior as the real-world system a calibration is necessary.

The Furuta pendulum consists of an arm rotating in the horizontal plane and a pendulum which is free to rotate in the vertical plane. The construction has two degrees of freedom, the angle of the arm, φ, and the angle of the pendulum, θ. The real-world system is shown in Figure 3.

The corresponding Modelica model is modeled using Modelica Standard Library components only. The revolute joint `armJoint` is connected to the world component and the horizontal arm, and allows rotational

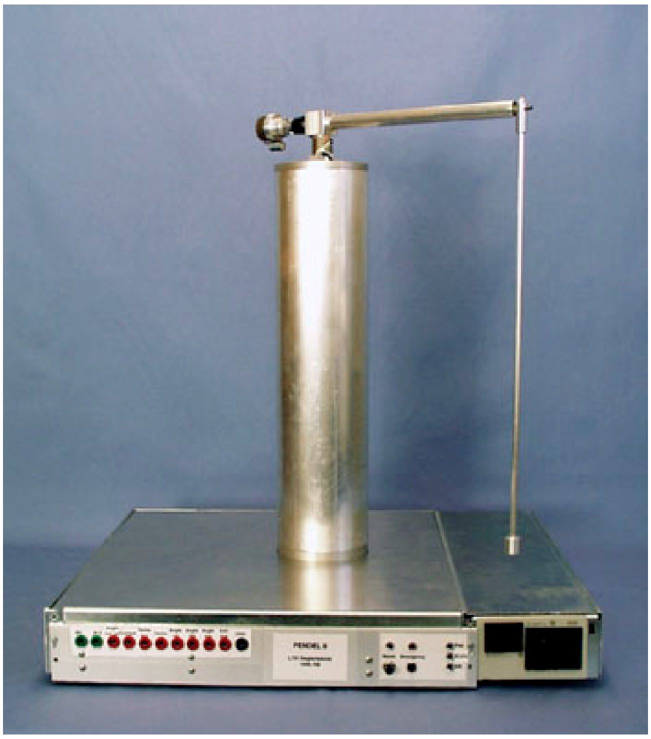


Figure 3. Photograph of Furuta pendulum.

movement in the global y-axis. Another revolute joint, `pendulumJoint`, connects the horizontal arm with the vertical pendulum, and allows rotational movement in the local x-axis. Both revolute joints are connected through their axis flange to a bearing friction component to allow friction modeling. The model can be seen in Figure 4.

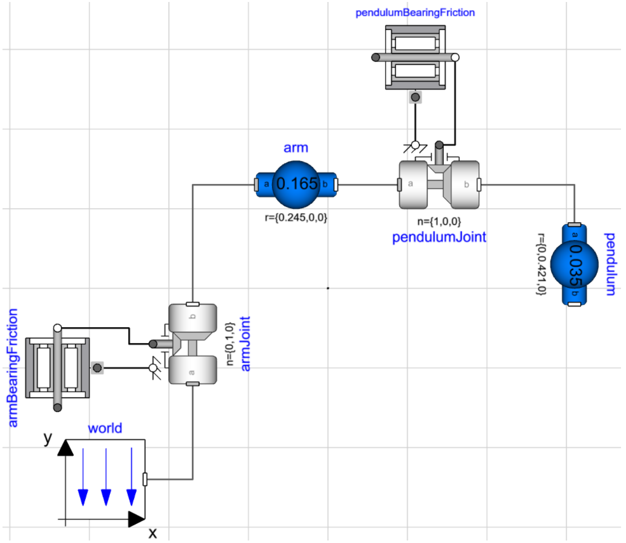
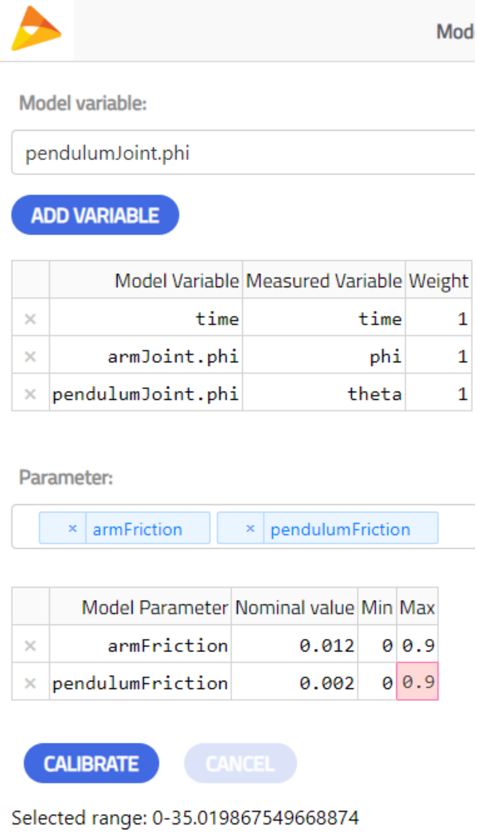


Figure 4. Modelica model of Furuta pendulum.

Experimental data of the real-world pendulum is used for the calibration (input v_i as in section 2). Variables of interest are arm angle φ, and the pendulum angle θ.

Correspondingly, model variables of interest (output w_i as in section 2) are the angle of the arm joint `armJoint.phi`, and the angle of the pendulum joint `pendulumJoint.phi`. Both variables are weighted equally with factor 1.



Mod

Model variable:

pendulumJoint.phi

ADD VARIABLE

	Model Variable	Measured Variable	Weight
×	time	time	1
×	armJoint.phi	phi	1
×	pendulumJoint.phi	theta	1

Parameter:

× armFriction × pendulumFriction

	Model Parameter	Nominal value	Min	Max
×	armFriction	0.012	0	0.9
×	pendulumFriction	0.002	0	0.9

CALIBRATE CANCEL

Selected range: 0-35.019867549668874

Figure 5. Furuta calibration setup in the calibration app's GUI.

The parameters of interest for the calibration process are the bearing friction coefficient for the arm joint `armFriction`, and the bearing friction coefficient for the pendulum joint `pendulumFriction`. For `armFriction`, we use 0.012 as nominal value, and for `pendulumFriction` we use 0.002 as nominal value. For both parameters, we set the bounds as 0 and 0.9.

Calibration time interval is set from approximately 0 to 35 seconds.

An overview of all the settings in the calibration app can be seen in Figure 5.

It took 27 iteration steps to find a local minimum at

$$\texttt{armFriction} = 0.010125 \tag{5}$$

$$\texttt{pendulumFriction} = 0.00117 \tag{6}$$

Figure 6 and Figure 7 respectively show the the arm angle and the pendulum angle over time for the experimental data, and the nominal and calibrated case of the model. In comparison to the uncalibrated model, we can see that the result of the calibrated model is in good agreement with the experimental data. The value of the objective function has decreased from $1.645 \cdot 10^2$ to 2.729.

DOI
10.3384/ecp204703

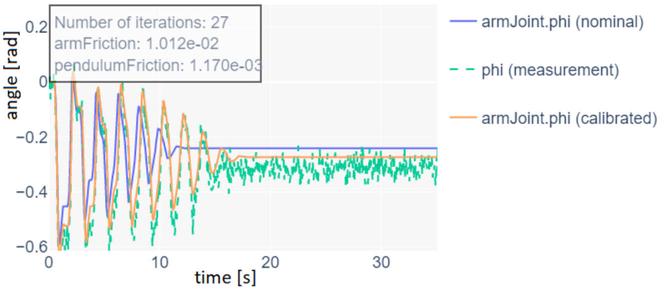


Figure 6. Arm angle over time curves of experimental data (green), and nominal (blue) and calibrated case (orange) for system model.

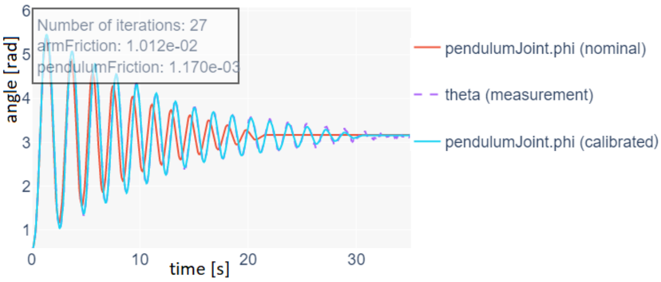


Figure 7. Pendulum angle over time curves of experimental data (purple), and nominal (red) and calibrated case (light blue) for system model.

4.2 Thermal

A heat exchanger was designed as a high-fidelity model. This heat exchanger is now to be used in a complex system model requiring fast respond times. This requires a computationally less expensive model, a low-fidelity model. For the low-fidelity model to show the same behavior as the high-fidelity model, a calibration is necessary.

In our example, we are using a testbench model for microtube heat exchanger models from Modelon's Air Conditioning Library (see Figure 8). A testbench is a type of model that focuses on a single complex component. Adequate boundary conditions in the form of sources and sinks are connected to complex model that is to be tested. In this case, a heat exchanger is connected to a source and sink for the air and refrigerant side respectively. Except for the air sink, each source and sink is connected to a ramp signal for each variable. Since we are interested in the dynamic behavior, the boundary conditions (i.e. all ramp signals) are configured to change within 170 seconds starting with the first boundary condition after 60 seconds.

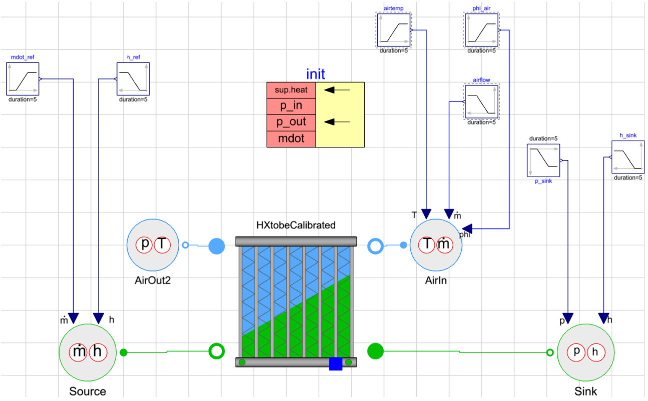


Figure 8. Testbench model for a microtube heat exchanger model using Modelon's Air Conditioning Library.

The low-fidelity heat exchanger testbench model extends the testbench model for the high-fidelity heat exchanger. The difference between the two testbench models is the pressure correlation of the refrigerant side in the heat exchanger model. The high-fidelity model uses `PlossCommon`, which is a Reynolds-based two-phase pressure loss model for laminar and turbulent flow based on Friedel (1979). The low-fidelity model uses `DensityProfilePressureLossHX`, which is a distributed pressure loss model for two-phase fluids using a density profile.

Measurement data (input v_i in section 2) is generated from the high-fidelity testbench model. Variables of interest are the pressure drop on the refrigerant side `dp_ref`, and the total heat flow on the refrigerant side `Qdot_refTotal`. Correspondingly, low-fidelity model variables of interest (output w_i in section 2) are the same: Pressure drop on the refrigerant side `dp_ref`, and the total heat flow on the refrigerant side `Qdot_refTotal`. Both variables are weighted equally with factor 1. Note that with the given problem, we could also give the pressure drop a higher weighting factor as the heat flow.

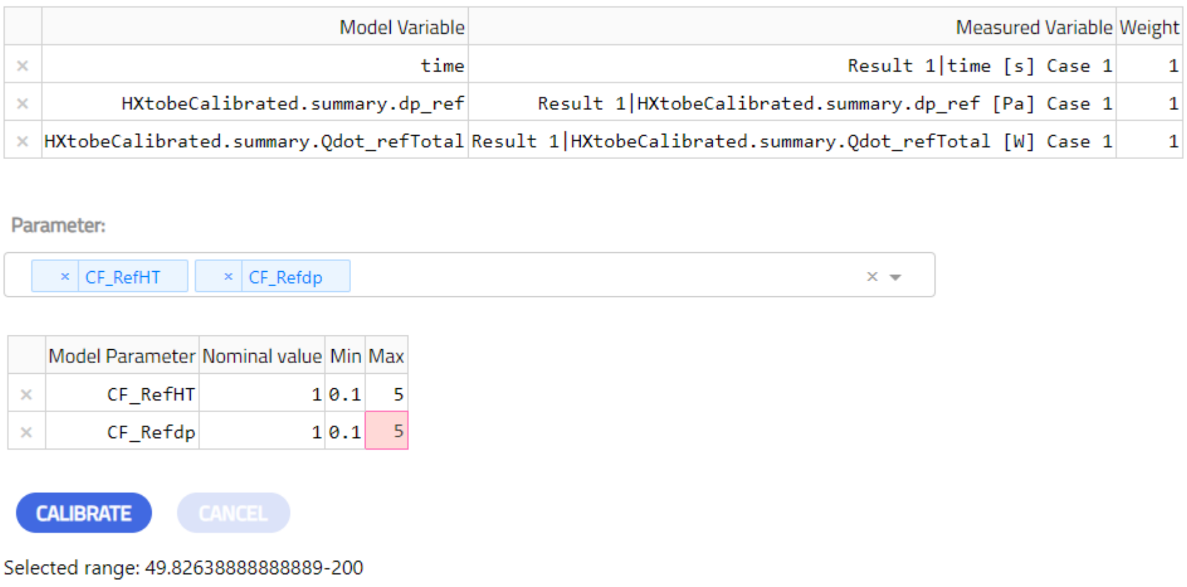
The parameters of interest for the calibration process are four calibration factors for the corresponding correlation equations:

1. `CF_RefHT`, for heat transfer correlation on refrigerant side.
2. `CF_AirHT`, for heat transfer correlation on air side.
3. `CF_Refdp`, for pressure loss correlation on refrigerant side.
4. `CF_Airdp`, for pressure loss correlation on air side.

A preliminary investigation showed that only the calibration factors for the refrigerant side are relevant for this particular case. This makes sense, as we are changing the pressure loss correlation only on the refrigerant side, and our variables of interest are on the refrigerant side. For both parameters, we use 1.0 as nominal value. We set the bounds to 0.1 and 5.

Calibration time interval is set from approximately 50 to 200 seconds. This is to not capture the initial transient, and to allow enough time for the system to reach a steady-state before the ramp signal steps are activated.

An overview of all the settings in the calibration app can be seen in Figure 9.



	Model Variable	Measured Variable	Weight
×	time	Result 1|time [s] Case 1	1
×	HXtobeCalibrated.summary.dp_ref	Result 1|HXtobeCalibrated.summary.dp_ref [Pa] Case 1	1
×	HXtobeCalibrated.summary.Qdot_refTotal	Result 1|HXtobeCalibrated.summary.Qdot_refTotal [W] Case 1	1

Parameter:

× CF_RefHT × CF_Refdp

	Model Parameter	Nominal value	Min	Max
×	CF_RefHT	1	0.1	5
×	CF_Refdp	1	0.1	5

Selected range: 49.82638888888889-200

Figure 9. Heat exchanger calibration setup in the calibration app's GUI.

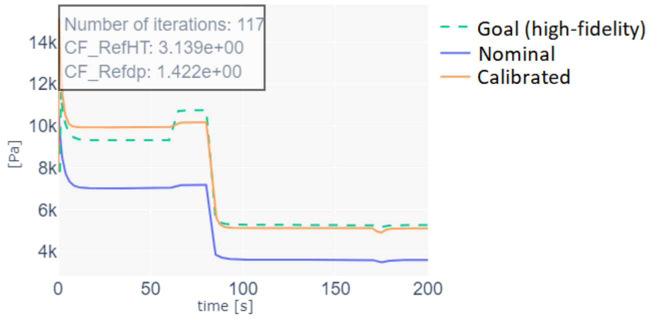


Figure 10. Pressure drop over time curves for high-fidelity model (green), and nominal (blue) and calibrated case (orange) for low-fidelity model.

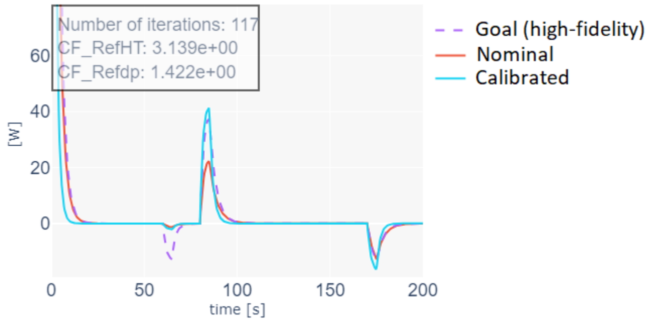


Figure 11. Heat transfer over time curves for high-fidelity model (purple), and nominal (red) and calibrated case (light blue) for low-fidelity model.

It took 117 iteration steps to find a local minimum at

$$\texttt{CF\_RefHT} = 3.139 \tag{7}$$

$$\texttt{CF\_Refdp} = 1.422 \tag{8}$$

Figure 10 and Figure 11 respectively show the pressure drop and total heat flow over time for the high-fidelity model, and the nominal and calibrated case of the low-fidelity model.

In comparison to the uncalibrated model, we can see that the result of the calibrated low-fidelity model is closer to the results of the high-fidelity model. The value of the objective function has decreased from $1.495 \cdot 10^9$ to $4.029 \cdot 10^7$.

However, it is also visible that the calibrated low-fidelity model parameterization requires more optimization. Including other system parameters can help to find a parameterization that follows the results of the high-fidelity model more closely. The underlying correlation of the low-fidelity model uses 4 extra parameters (pressure, enthalpy, mass flow, pressure drop) that can be adjusted to match the steady-state solution of the high-fidelity model. This way, the nominal solution is closer to the high-fidelity solution and potentially less iteration steps are required to get a more optimized solution.

In a further step, different start values could be picked and/or different minimizing methods (i.e. Cobyla or Powell) could be picked, to verify whether a global minimum within the bounds was found. These next steps are not discussed here.

5 Conclusion

Derivative-free minimization methods are a good way to calibrate system models, as these methods do not set strict requirements on the model architecture and find a (local) optimal solution in a reasonable number of iteration steps. To verify that an optimal solution is found with little computational effort, it is recommended to do a sensitivity analysis, run several calibration runs with different start values and use different minimization methods, such as Nelder-Mead, Powell, and Cobyla.

Using a Dash app and `scipy.optimize.minimize`, it is possible for a user with limited knowledge to calibrate Modelica models against measured or simulation data with the Nelder-Mead method. This can be used independently from the physics domain. The reliability of the results can be improved by implementing an automization and parallelization of the calibration runs with different start values and minimization methods.

References

Arendt, K. et al. (2018). *ModestPy: An Open-Source Python Tool for Parameter Estimation in Functional Mock-up Units*. Tech. rep. Modelica Association. URL: https://github.com/sdu-cfei/modest-py.

EstimationPy (2023). URL: https://github.com/lbl-srg/EstimationPy (visited on 2023-08-14).

Friedel, L. (1979). "Improved Friction Pressure Drop Correlation for Horizontal and Vertical Two-Phase Pipe Flow". In: *Proc. of European Two-Phase Flow Group Meet., Ispra, Italy, 1979*. URL: https://cir.nii.ac.jp/crid/1570572701507538176.

Köppen, Arne et al. (2022-11). *Techno-economic assessment of an industrial project towards carbon neutrality*. DOI: 10.3384/ecp19325.

Larson, Jeffrey, Matt Menickelly, and Stefan M. Wild (2019). *Derivative-free optimization methods*.

Nelder, J. A. and R. Mead (1965-01). "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4, pp. 308–313. DOI: 10.1093/comjnl/7.4.308.

Olsson Hans, Jonas Eborn, Sven Erik Mattsson, and Hilding Elmqvist (2006-09). "Calibration of Static Models using Dymola". In: *Modelica 2006*. The Modelica Association, pp. 615–620.

Powell, M. J. D. (1964-01). "An efficient method for finding the minimum of a function of several variables without calculating derivatives". In: *The Computer Journal* 7.2, pp. 155–162. DOI: 10.1093/comjnl/7.2.155.

Powell, M. J. D. (1994). "A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation". In: *Advances in Optimization and Numerical Analysis*. Ed. by Susana Gomez and Jean-Pierre Hennart. Dordrecht: Springer Netherlands, pp. 51–67. DOI: 10.1007/978-94-015-8330-5_4.

Press, William H. et al. (1992). *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. USA: Cambridge University Press. ISBN: 0521431085.

Wüllhorst, Fabian et al. (2022). "AixCaliBuHA: Automated calibration of building and HVAC systems". In: *Journal of Open Source Software* 7.72, p. 3861. DOI: 10.21105/joss.03861. URL: https://doi.org/10.21105/joss.03861.

SSP in a Modelica Environment

Dag Brück

Dassault Systèmes, Lund, Sweden, dag.brueck@3ds.com

Abstract

System Structure and Parameterization (SSP) is a tool independent standard to define complete systems. Dymola now supports import and export of SSP files; this paper describes how the SSP support was implemented and discusses some of the constraints and unavoidable compromises.

Keywords: Modelica, SSP, implementation

1 SSP and supporting processes

System Structure and Parameterization (Mai 2019), known as SSP, is a tool independent standard to define complete systems. In a typical use of SSP, the system description consists of several interconnected Functional Mockup Units (FMUs), nested system descriptions, and their parameterizations and other related data, as shown in Figure 1.

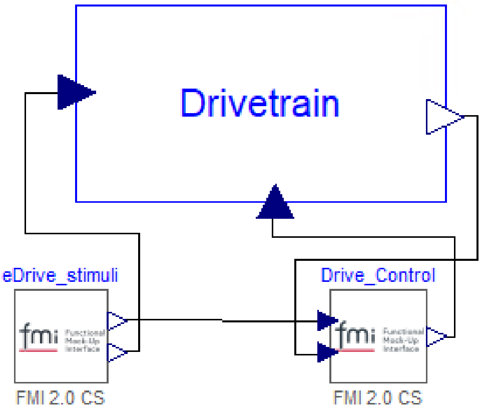

Drivetrain
eDrive_stimuli
Drive_Control
fmi Functional Mock-Up Interface
fmi Functional Mock-Up Interface
FMI 2.0 CS
FMI 2.0 CS

Figure 1. Top-level system comprising two FMUs and a drivetrain subsystem described hierarchically.

SSP is suitable for representing simple model structure with additional support for parameter sets and (some) simulation setup. However, SSP was designed to meet several needs:

- SSP for designing a simulation structure. Components are described with its inputs and outputs and its required parameters, but no behavior.
- SSP as a template for implementation, based on the interfaces and parameters defined in the previous step.
- SSP as central parameterization description and database (Hällqvist et al. 2021). Several data sets can be defined and documented in a portable manner.
- SSP as a ready-to simulate system description. This is the main use we envision in our environment.
- SSP for reuse of system structure during different phases of development, for example, an SSP defined originally for software-in-the-loop can also be reused for hardware-in-the-loop testing.

A key objective is that the SSP files can be transferred between multiple environments for architecture specification, detailed design and model implementation, or post-processing analysis. A demonstrator using such a multi-tool workflow was developed by ProSTEP ivip Smart System Engineering (Rude et al. 2021), as shown in Figure 2. For this application, several tools were used in a collaborative manner: Dymola (Dassault Systèmes 2019; Elmqvist 2014), easySSP (eXXcellent 2022), PMSF FMI Bench (Mai 2023) and Tracy (Vettermann 2021).

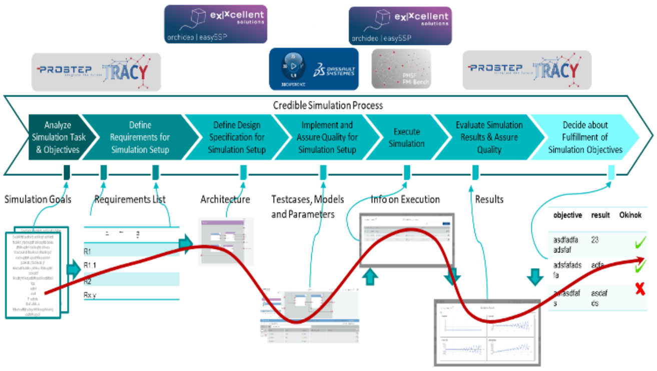

exXcellent solutions
orchideo | easySSP
exXcellent solutions
orchideo | easySSP
PROSTEP
TRACY
DASSAULT SYSTEMES
PMSF FMI Bench
PROSTEP
TRACY
Credible Simulation Process
Analyze Simulation Task & Objectives
Define Requirements for Simulation Setup
Define Design Specification for Simulation Setup
Implement and Assure Quality for Simulation Setup
Execute Simulation
Evaluate Simulation Results & Assure Quality
Decide about Fulfillment of Simulation Objectives
Simulation Goals
Requirements List
Architecture
Testcases, Models and Parameters
Info on Execution
Results
R1
R1.1
R2
Rx.y
objective
result
Ok/nok
asdfadfa adsfaf
23
adsfafads fa
adfa
asdfaf s
asdaf ds

Figure 2. Traceability workflow using SSP from ProSTEP ivip Smart System Engineering.

SSP is commonly stored as a zip-file containing several files and a hierarchy of directories. A simplified view of the SSP structure is given in Figure 3.

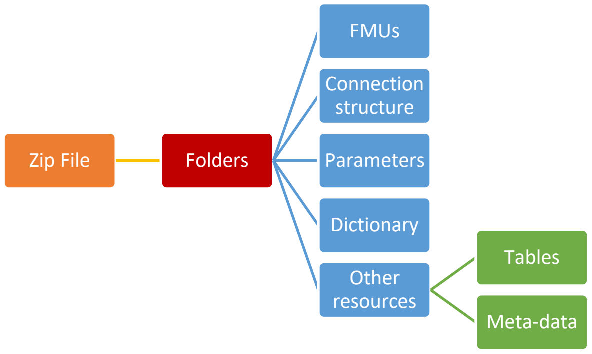


Figure 3. Simplified structure of the SSP file.

SSP is open for any extensions using standardized or vendor-specific annotations and directories with additional meta-data. Under the catch-phrase "Credible Decision Process" (CDP), the SSP community has expressed great interest in adding standardized meta-data. The SET Level project has developed a process framework for integrating simulation into the development and validation of system models (Heinkel and Steinkirchner 2022a; Heinkel and Steinkirchner 2022b). The process supporting CDP is shown in Figure 4.

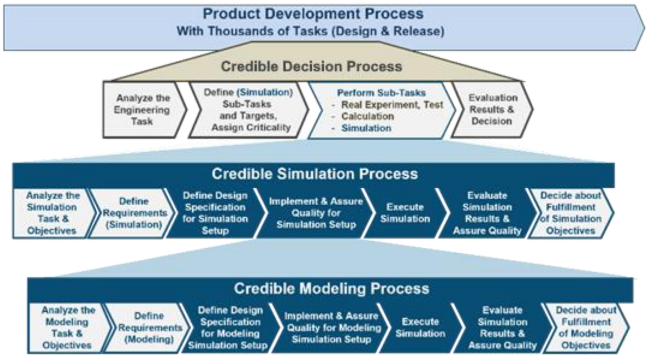

Product Development Process
With Thousands of Tasks (Design & Release)
Credible Decision Process
Analyze the Engineering Task
Define (Simulation) Sub-Tasks and Targets, Assign Criticality
Perform Sub-Tasks
- Real Experiment, Test
- Calculation
- Simulation
Evaluation Results & Decision
Credible Simulation Process
Analyze the Simulation Task & Objectives
Define Requirements (Simulation)
Define Design Specification for Simulation Setup
Implement & Assure Quality for Simulation Setup
Execute Simulation
Evaluate Simulation Results & Assure Quality
Decide about Fulfillment of Simulation Objectives
Credible Modeling Process
Analyze the Modeling Task & Objectives
Define Requirements (Modeling)
Define Design Specification for Modeling Simulation Setup
Implement & Assure Quality for Modeling Simulation Setup
Execute Simulation
Evaluate Simulation Results & Assure Quality
Decide about Fulfillment of Modeling Objectives

Figure 4. SET Level credible decision process.

An extensive Simulation Resource Meta Data (SRMD) file format was defined to store meta-data for simulation resources as well as meta-data for models, tools, maps and scenarios, and simulation tasks (Heinkel et al. 2022c). Similar approaches are LOTAR (LOTAR International 2023), MIC (IRT SystemX 2020) and MoSSEC (MoSSEC Project 2021), each shaped by the needs of their respective communities.

It is worth noting that SSP does not specify any simulation semantics, although you can argue that one is implied because much of the definition is based on FMI and the interconnection structure. However, including components of other types (e.g. Modelica) is within the scope of SSP.

SSP is developed as a project in the Modelica Association (MAP-SSP). The first version of the SSP standard was published in March 2019 (Mai 2019), and progress is now made to align SSP 2.0 with the FMI 3.0 specification.

Several tools supporting SSP are now emerging; the remainder of this paper describes how SSP support was implemented in Dymola and discusses some of the constraints and unavoidable compromises. A detailed comparison with other Modelica tools could not be made due to Dassault Systèmes policies.

2 Mapping SSP to Modelica

Dymola is a development and simulation environment for Modelica models, with support for importing Functional Mockup Units (FMUs) and running simulations. This means that much of the support needed for SSP "comes for free", the remaining task being to map SSP structures to Modelica models in a sensible way. The mapping is summarized in Figure 5.

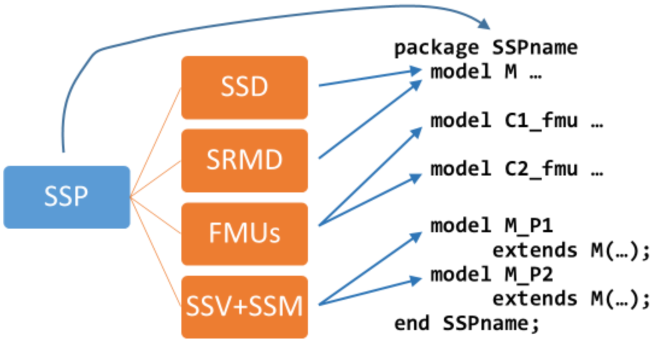


Figure 5. Mapping SSP key elements to Modelica

2.1 Package as container for all artifacts

The first design choice was that everything imported from an SSP would be collected in a single Modelica package in order to prevent contamination of the namespace.

The package name is derived from the SSP name, and certain SSP documentation is stored as package documentation.

2.2 System Structure Description (SSD)

The top-level system, i.e. the `SystemStructure.ssd` file, is converted to a Modelica model, including the internal component, connector and connection structure. A restriction at present is that Dymola only supports one top-level SSD.

Embedded systems in the form of nested SSDs are also imported as locally defined models, and the corresponding component is created in the top-level model.

2.3 Functional Mockup Unit (FMU)

All FMUs embedded in the SSP file are imported into the enclosing package. This import relies entirely on the available FMU import capabilities of Dymola to create a Modelica wrapper model, which means that any combination of ME and CS, or FMI version is supported.

The FMU import processes the port definitions of the FMU and creates the corresponding Modelica connectors.

For that reason, any connector definitions defined in the corresponding SSP component are resolutely discarded; the FMU ports are used instead.

2.4 Parameters and connectors

SSDs can have both parameters and connector definitions. They are mapped to the closest matching Modelica type. For example, a connector of type `ssc:Real` and kind "input" is represented by the MSL type `RealInput`. Units in the SSD are applied to the Modelica model without any checking until the model is translated in Dymola.

The Binary type is used as a special case to represent more complex Modelica types (see below), but in the general case represent a problem for a Modelica environment. There is no natural mapping to an anonymous array of bytes; this is an unsolved problem that SSP shares with FMI.

2.5 Parameter sets

One of the strong points of SSP is that it combines simulation models with parameter sets, stored in one or more SSV-files. There are also optional mapping files (SSM) that allows parameters to mapped to model variables with different names, and in that process perform linear transformations for e.g. unit conversion.

In Dymola, we apply a common Modelica idiom to parameter sets. For each parameter set, we create a new model that extends from the main model defined in the SSP, and then we provide the parameters as a modifiers in the extends-clause. This process allows us to keep the natural structure of the SSP file in a manner that maps naturally to Modelica. It also allows further parameter changes by inheritance.

2.6 Documentation and meta-data

The question of simulation quality, from measurement data via modeling assumption to simulation setup, has received considerable attention. As a result, the Credible Simulation Process (Heinkel and Steinkirchner 2022a) is applied to SSP. A key aspect of CSP is the ability to store meta-data in the form of key-value pairs, following standardized templates defined by organizations or corporations.

Dymola extracts meta-data stored in the proposed Simulation Resource Meta Data (SRMD) format and converts it to model annotations that are displayed and edited as part of the documentation.

2.7 Simulation setup

SSP does not have any simulation semantics in itself, although possibly something can be derived from the underlying reliance on FMU components. The simulation setup in SSP is conversely restricted to simulation start and stop times.

Dymola has the capability to store a more extensive "experiment" annotation in Modelica models, and that information is shared in SSPs by the use of a proprietary SSP annotation (lacking further standardization).

3 Example of an imported SSP

Using a simple SSP example, the XML code with many details omitted is shown in Listing 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<ssd:SystemStructureDescription >
  <ssd:System name="Example">
    <ssd:Connectors>
      <ssd:Connector name="pos"
         kind="output" />
    </ssd:Connectors>
    <ssd:Elements>
      <ssd:Component name="stimulus"
        type=
          "application/x-fmu-sharedlibrary"
        source=
          "resources/StimulusFMU.fmu">
        <ssd:Connectors>
          <ssd:Connector name="y"
             kind="output">
             <ssc:Real/>
             <ssd:ConnectorGeometry ... />
          </ssd:Connector>
        </ssd:Connectors>
        <ssd:ElementGeometry ... />
      </ssd:Component>
      <ssd:Component name="controller" .. />
      <ssd:Component name="drivetrain" .. />
    </ssd:Elements>
    <ssd:Connections>
      <ssd:Connection
         endElement="controller"
         endConnector="ref"
         startConnector="y"
         startElement="stimulus">
         <ssd:ConnectionGeometry ... />
      </ssd:Connection>
      <ssd:Connection ... />
      <ssd:Connection ... />
      <ssd:Connection ... />
    </ssd:Connections>
    <ssd:SystemGeometry ... />
  </ssd:System>
  <ssd:DefaultExperiment stopTime="4" />
</ssd:SystemStructureDescription>
```

Listing 1. Simple SSP example.

The representation of parameters in SSP is currently subject to discussion; see also (Brück 2023).

Importing the SSP file with three FMUs, the generated Modelica model is displayed as Figure 6.

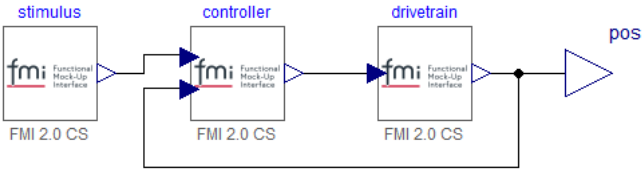


Figure 6. Diagram of SSP file imported into Dymola.

The generated Modelica code, with graphical annotations removed for clarity is shown in Listing 2.

```
model Example "Model for position servo"
  // Connectors
  Modelica.Blocks.Interfaces.RealOutput
      pos annotation (...);
  // Components
  parameter Real k(value=200)
      "Gain of controller";
  parameter Modelica.Units.SI.Time
      T(value=10)
      "Time constant of controller (T>0)";
  parameter Modelica.Units.SI.Radius
      r(value=0.5) "Radius of load";
  parameter Modelica.Units.SI.Mass
      m(value=80)  "Mass of load";
  StimulusFMU_fmu stimulus
      annotation (...);
  ControllerFMU_fmu controller
      annotation (...);
  DrivetrainFMU_fmu drivetrain
      annotation (...);
equation
  // Connections
  connect(stimulus.y, controller.ref)
      annotation (...);
  connect(controller.y, drivetrain.u)
      annotation (...);
  connect(drivetrain.pos, controller.pos)
      annotation (...);
  connect(drivetrain.pos, pos)
      annotation (...);
  annotation (experiment(StopTime=4));
end Example;
```

Listing 2. Generated Modelica code after import.

If the SSP file contains parameter sets (SSV and optionally SSM files), then additional Modelica models are created that provide parameter settings, see Listing 3.

```
model HighGain "Servo with high gain"
  extends Example(k=400, T=12);
end HighGain;
```

Listing 3. Applied parameter set.

It is worth noting that Dymola creates a Modelica wrapper model for each imported FMU. These models are however no different for FMUs in SSP-files from other imported FMUs.

4 Mapping Modelica to SSP

Being a modeling and simulation environment for Modelica, Dymola has to support model export to SSP.

4.1 Models with FMUs

The most straightforward use case is a top-level Modelica model populated with interconnected FMUs as components. This kind of structure can be exported to a standard SSP file under the assumption that you only use types that can be expressed in SSP. For example, Modelica.Blocks.Interfaces.RealInput and RealOutput can be mapped to `ssc:Real` with kind=”input” and “output” respectively. FMUs are copied from wherever they are located into the SSP file’s resource directory.

Similarly, local variables and parameters of built-in types, connections and description string have corresponding attributes in SSP. More advanced concepts such as inheritance (*extends*) and model templates (*replaceable/redeclare*) cannot be represented.

4.2 Need for annotations

SSP has a general escape mechanism called “annotations” that allows a tool to store arbitrary information with a proprietary encoding. Each annotation is tagged by the tools, e.g. com.3ds.dymola.

Dymola uses such annotations to store model documentation, the full experiment (simulation) setup, commands and standardized Modelica figure annotations. It is worth noting that the only simulation setup attributes defined in SSP are start and stop times.

Dymola can store model meta-data (key-value pairs, in user-defined groups). Most meta-data are stored in SSP annotations, the exception being meta-data groups being identified as being in SRMD format. Such SRMD meta-data is stored in separate SRMD files in the SSP extra directory.

Graphical annotations in Modelica models are not stored in SSP. The possibility to store the entire model text as an annotation is not used by Dymola.

4.3 Native Modelica models

A natural extension (in a Modelica tool) is the possibility to use Modelica components in addition to FMUs and export it as an SSP file. Such an extension would increase the expressive power of SSP, for example by connecting physical connectors such as mechanical flanges or fluid pipes.

SSP was designed with such openness in mind, although the specification only mentions FMI by name and provides a restricted set of types appropriated from FMI. A noteworthy point is that SSP does not define any simulation semantics, this follows implicitly from the semantic meaning of connecting its components.

Dymola supports an extended SSP format that has been proposed as part of SSP 2.0 (Brück 2023). The proposal aims to be a practical compromise with the following key properties:

- Components and connectors can have native Modelica types, such as a rotational inertia. We have chosen to represent them with SSP’s Binary type as a generalization of the concept.
- Acausal connectors have been added (in addition to in, out and inout).
- Parameter values can be arbitrary expressions.

- Modelica components use references to Modelica types; the Modelica models themselves are not stored in the SSP file.

It is hoped that the community can gain experience of using Modelica components in SSP and that it eventually lead to adaption by SSP.

4.4 Round tripping

In this context, round tripping means the ability to read and write between internal and external data formats without any loss of information. A stricter sense of the term would require an identical external representation.

Dymola is only partially successful in this respect. Overall structure and simulation behavior is very well preserved, which can be expected as FMUs are copied in and out.

The read-edit-store cycle for SSPs is not so well developed and will need improvement in the future. In particular, annotations from other tools may be lost during editing.

5 Conclusions

Reasonable, although not complete, support for SSP has been implemented in Dymola. It takes advantage of earlier functionality for e.g. importing FMUs, hence is able to map most SSP concepts to Modelica models.

The development effort is entirely guided by practicality, with the aim of providing high-quality simulation capabilities to SSP. The degree of success has to be judged by its users.

Acknowledgements

The author wants to thank the members of the SSP design group, in particular Pierre Mai and Peter Lobner, for constructive collaboration.

References

Brück, Dag (2023). "Modelica Models in SSP" in Proc. 15th International Modelica Conference, Aachen, Germany.

Dassault Systèmes (2019). "What is Dymola?", https://www.3ds.com/fileadmin/PRODUCTS/CATIA/DYMOLA/PDF/What-is-Dymola-2020x.pdf.

Elmqvist, Hilding (2014). "Modelica Evolution - From My Perspective" in Proc. 10th Modelica Conference, Lund, Sweden.

eXXcellent (2022). "orchideo | easySSP", https://www.easy-ssp.com/.

Heinkel, Hans-Martin, P. R. Mai, R. Aue, J. Bou, C. Bühler, C. Franke and A. Puetz (2022). "SRMD format and classifications for metadata". https://gitlab.setlevel.de/open/processes_and_traceability/traceability_data/-/blob/main/SETLevel_SRMD_and_classifications_for_metadata.pdf.

Heinkel, Hans-Martin and K. Steinkirchner (2022a). "Credible Simulation Process". https://gitlab.setlevel.de/open/processes_and_traceability/credible_simulation_process_framework/-/blob/main/Credible-Simulation-Process-v1-2.pdf.

Heinkel, Hans-Martin and K. Steinkirchner (2022b). "Credible Modeling Process", https://gitlab.setlevel.de/-open/processes_and_traceability/credible_simulation_process_framework/-/blob/main/Credible-Modeling-Process-v1-0.pdf.

Heinkel, Hans-Martin, P. R. Mai, R. Aue, J. Bou, C. Bühler, C. Franke and A. Puetz (2022c). "SRMD format and classifications for metadata", https://gitlab.setlevel.de/open/processes_and_traceability/traceability_data/-/blob/main/SETLevel_SRMD_and_classifications_for_metadata.pdf.

Hällqvist, Robert, R. C. Munjulury, R. Braun, M. Eek and P. Krus (2021). "Engineering Domain Interoperability Using the System Structure and Parameterization (SSP) Standard" in Proc. of the 14th International Modelica Conference, Linköping, Sweden.

IRT SystemX (2020). "Model Identity Card (MIC)", https://mic.irt-systemx.fr/mic.

LOTAR International (2023). "LOTAR - Long Term Archiving and Retrieval", https://lotar-international.org/.

Mai, Pierre R. et al. (2019). "System Structure and Parameterization". https://ssp-standard.org/publications/SSP10/SystemStructureAndParameterization10.pdf.

Mai, Pierre R. (2023). "PMSF FMI Bench", https://pmsf.eu/products/fmibench/.

MoSSEC Project (2021). "Modelling and Simulation information in a collaborative Systems Engineering Context," ISO 10303-243:2021, http://www.mossec.org/.

Rude, Stefan, F. Fischer, H. Esen, P. R. Mai, K. Steinkirchner, P. Lobner, D. Brück and H.-M. Heinkel (2021). "prostep ivip SmartSE: Traceable simulation results in a heterogeneous environment", https://youtu.be/qVXD0sZG5f8.

Vettermann, Steven, C. Bühler and K. Steinkirchner (2021). "Traceability Software Demonstrator TRACY," 29 April 2021, https://setlevel.de/assets/mid-term-presentation/Traceability.pdf.

An FMI- and SSP-based Model Integration Methodology for a Digital Twin Platform of a Holistic Railway Infrastructure System

Ozan Kugu[1] Shiyang Zhou[1] Rebecca Nowak[2] Gabor Müller[3] Stefan H. Reiterer[3] Alexander Meierhofer[3] Stefan Lachinger[4] Lukas Wurth[1] Manfred Grafinger[1]

[1]Institute of Engineering Design and Product Development, TU Wien, 1060 Vienna, Austria,
ozan.kugu@tuwien.ac.at
[2]VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH, 1220 Vienna, Austria,
rnowak@vrvis.at
[3]Virtual Vehicle Research GmbH, 8010 Graz, Austria, alexander.meierhofer@v2c2.at
[4]AIT - Austrian Institute of Technology GmbH, 1210 Vienna, Austria, stefan.lachinger@ait.ac.at

Abstract

Nowadays, the digitalization of large-scale railway infrastructure systems is a major trend, which helps to reduce the life-cycle costs of the railway transportation. For this purpose, the Digital Twin (DT) technology can be used to interoperate different digital data and models, belonging to the railway infrastructure system, in a virtual platform for predictive maintenance, diagnostics and condition monitoring in the railway sector. However, the simulation models of the infrastructure system are tool-dependent, lack ease-of-use and platform compatibility. Therefore, we have to customise them in order to make them more representative and then integrate easily and tool-independently into the DT platform. For this purpose, we propose to use the Functional Mock-up Interface (FMI) and System Structure Parameterization (SSP) technologies as open interface standards between the models and software tools. In this work, we demonstrate the application of the FMI and SSP standards separately for two use cases, which include a multibody simulation (MBS) model of a railway vehicle and residual life time (RLT) calculation of a steel bridge.

Keywords: FMI, FMU, SSP, Model Integration, Digital Twin, Railway Digitalization

1 Introduction

In recent decades, the railway sector has made a significant contribution to both local and long-distance public and freight transport. The holistic, large-scale railway infrastructure system has played a major role in this. The system is very complex and consists of different subsystems such as railway vehicle, track, turnout, tunnel and bridge. These are to be controlled, maintained, monitored, diagnosed and visualized, which is complex and time-consuming. Therefore, the whole system has to be easy to control and simply presented to the train operators and infrastructure managers for their operational, supervision and maintenance tasks while expanding life cycle time and reducing costs at the same time. This is certainly possible if the system can be used in a user-friendly and interactive virtual environment. Nowadays, the DT technology is foreseen as a suitable method to digitalize the railway system, as it is already applied in different kinds of systems in many sectors such as automotive, aircraft, etc., incl. the railway infrastructure. For example, (Kaewunruen, AbdelHadi, et al. 2022) proposed to use it for efficient maintenance, resilience and sustainability of railway bridge infrastructures. (Hamarat, Papaelias, and Kaewunruen 2022) also used it to analyze and simulate fatigue damage in railway turnouts. Unfortunately, there are difficulties in applying the railway infrastructure system to a DT platform by using different subsystem assets, adapters and interfaces, which is even more sophisticated when considering the proper communication and interaction of the assets with each other. As (Ahmadi et al. 2021) mentioned, uncertainties of measured system parameters and variables due to external factors (e.g. noise, weather), abnormalities caused by time-frame mismatches between system and model, and incomplete system interpretation, understanding and not 100% reliability of the DT are the overall challenges to overcome while operationally implementing the DT platform.

In order to overcome the difficulty of asset integration mentioned above, first, we propose to use the FMI standard, which helps us to adopt simulation models of different railway infrastructure subsystems, to make them easy to use, tool independent, platform compatible and suitable for intellectual property (IP) protection. FMI is an open standard, providing an interface between dynamic simulation models and various software tools, and is already supported by more than 170 tools. The models are supposed to be packed into the Functional Mock-up Unit (FMU), which is the simulation unit of the FMI consisting of a model description file (xml), implementation in source and/or binaries, additional data and functionality. (https://fmi-standard.org/)

For this work, the second standard interface we propose to use is SSP. It consists of one or more FMUs, transferable parameter description formats such as system structure definition (.SSD), signal directories (.SSB), parameter values (.SSV), parameter mapping (.SSM), all as packed in System Structure Package (.SSP) (Pierre R. Mai 2018). Many industrial and academic members also contribute to the SSP standard. The technology helps to keep and interoperate different FMUs, belonging to a physical system, together with parameter mapping, storage and exchange in a whole SSP-file. (`https://ssp-standard.org/`)

Models and data of different railway infrastructure subsystems need to interoperate with each other properly, which is a very complex and elaborate task to handle. Finally, we need to ensure user control and visual representation of the railway infrastructure under consideration of user-friendliness and user-interactivity. Therefore we will integrate these models and data into a DT platform, called R4F Platform and proposed by (Zhou, Dumss, et al. 2022).

In addition to building the FMI- and SSP-based interfaces between the models and the platform, we need to design and optimize the entire integration process of the platform, which helps us to achieve the right visual output and enhance the user control. For this work, we use Jenkins pipelines for the continuous integration (CI) of these models. Jenkins is used, because it is open-source, time-saving, user-friendly, tool-independent and provides many plugins (cf. (Mysari and Bejgam 2020)). Besides, it shows durable, pausable, versatile characteristics and its workflow is code-based (`https://www.jenkins.io/doc/book/pipeline/`). In addition, we use a pipeline auto-generation technology, which is able to automatically and dynamically generate the Jenkins pipeline from a graph data-model containing different model characteristics by using a graph database management system (DBMS), which helps us to store, version and manipulate pipeline graphs used to auto-generate CI pipelines, in the platform (see (Reiterer, Schiffer, and Benedikt 2022) for more details on the implementation). This also helps to reduce the configuration effort of the pipeline workflow by eliminating the need to write pipeline code for each change that may be applied. Most importantly, the auto-generation technology helps to maintain and manage the reusability, traceability, parameter changes and data structure applicability of different use cases, making them even more understandable to the end user.

This paper aims to show how to seamlessly integrate models into the R4F Platform by using both the FMI and SSP standards as a model integration methodology. In this work, first section 2 shows examples of related research on DT for railways and the application of the standards. In section 3, we explain the proposed model standardisation and simulation approaches to integrate the models into the platform in detail. In section 4, we describe the FMU and SSP export, simulation process and result comparison of the aforementioned two use cases as a demonstration of the two approaches. In section 5, we show the proposed integration process, which helps to realize the complete model integration in the platform. Finally, in section 6, we present the conclusion and outlook of this work.

2 Related Work

2.1 Digital Twin for Railways

Using a DT has the potential to supervise and regulate a physical system in real time, leading to enhanced system performance, decreased maintenance expenditures, and enhanced safety. In the railway industry, DT can be applied to multiple areas such as predictive maintenance, fault diagnosis, dynamic analysis, and condition monitoring. This technology incorporates data from multiple sources, including sensors, cameras, and historical records, to create a unified platform capable of modeling and simulating the physical system.

In 2018, (Kaewunruen and Xu 2018) investigated a DT-aided Building Information Modelling (BIM) application that was used to adopt a 3D model of a station building and transform it into a 6D building information model for planning, design and operational purposes. During their research, they identified some limitations and risks of the BIM adoption such as the lack of a model standard, IP issues (copyright, data privacy), relatively high project costs and lack of model accuracy due to inaccurate modeling and data entry control, high model complexity and inaccurate design data. These drawbacks need to be considered and overcome when implementing a DT application. After that (Kaewunruen and Lian 2019) established and developed a 6D BIM of a railway turnout system by using the DT technology to enhance information flow, visualized maintenance, cost estimation and collaboration among different stakeholders in terms of life cycle management, for railway turnout systems. These insights can benefit engineers, project managers, technicians, and senior management.

(S. Zhang et al. 2021) created a framework for DT-assisted fault diagnostics and real-time health monitoring of railway point machines. However they focus on a single subsystem of the railway infrastructure, from which models and data of different subsystems are to be integrated into the DT appropriately to make the DT system more comprehensive. (Zhou, Dumss, et al. 2022) also proposed to continuously integrate the models and data from different railway subsystems into the conceptual model-based R4F Platform, which represents the fully connected and digital version of the holistic railway infrastructure system with a 6-layer system architecture. This work aims to enhance the model integration process occurring in the platform as mentioned before.

2.2 Functional Mock-up Interface (FMI)

The FMI standard came up first in 2010 as a new open-source interface standard for Model Exchange (ME) 1.0 (`https://fmi-standard.org/assets/releases/FMI_for_ModelExchange_v1.0.pdf`) to export different dymamic system models by generating C-code either in source or binary form (Balakirsky et al. 2010). After that, (Bastian et al. 2011) proposed to use the

FMI for Co-Simulation (CS), consisting of co-simulation interface and description schema, to simulate coupled subsystems time-dependently. After that (Blockwitz et al. 2012) introduced the FMI 2.0 providing further features such as combination and unification of both ME and CS interfaces in one document (modelDescription.xml), enhancement of the interface variables and their classification (causality & variability, parameter tuning during simulation), saving and restoring the FMU state, improved dependency description by using an element called ModelStructure, Jacobian matrices (e.g. for implicit integration methods or FMU linearization) and improved unit definitions. In 2014, (Bertsch, Ahle, and Schulmeister 2014) evaluated the FMI technology by using the FMU Compliance Checker and FMI Cross Checking methods to further improve the maturity of FMI-based simulations. They also found the technology very promising for sharing different simulation models between different stakeholders in collaboration with OEMs (Original Equipment Manufacturers) from an industrial perspective.

(Thule et al. 2018) used the FMI standard for their work, where they implied the high potential of the technology to co-simulate different simulation models with an FMI-suitable tool. Besides, (Pieper and Obermaisser 2018) suggested applying the FMI method to the Software-in-the-Loop simulation via the Internet or LANs, which is used to test networked railway systems, to provide an interface for the simulation supported by many different tools. In this way they were able to increase the tool-independency of the simulation, which is also one of the main reasons for using the standard for this work. Furthermore, (Hartmann 2020) managed to co-simulate a couple of two subsystems (a physical barge and crane), implemented in a DT system, by using the FMI methodology. He also found the results very promising due to their high accuracy although further validation of the results in a real-life environment and the development of a real-time DT co-simulation are needed in the future. In addition, (Golightly et al. 2022) used FMI to design a rail multi-model for rail decarbonisation. They also suggested using the technology to overcome IP issues due to different software tools, models and skills, although there are some limitations such as limited modelling quality, validation of results and lack of comparison between multi-modelling and single model.

Based on the aforementioned research, the FMI methodology shows a great potential for adopting different simulation models for co-simulation in a DT platform. Therefore, this work aims to apply the technology to different models of the railway subsystems such as the railway vehicle or railway steel bridge belonging to the two use cases for their integration into the R4F Platform.

2.3 System Structure and Parameterization (SSP)

The SSP standard was first presented at the 1st Japanese Modelica Conference 2016 (`https://ssp-standard.org/literature/`) as a standardized format to connect a network of components such as FMUs, to store and apply their parameters to these components, and also to ensure protection of the IP of these parameters. (Ochel et al. 2019) developed an application called OMSimulator to parameterize, compose and exchange FMI-based models for both co-simulation and model exchange by using the SSP technology in the application. As another example, (Hällqvist et al. 2021) used the SSP standard for parameter specifications and exchange between different simulation and geometric models interoperating in an aircraft vehicle system and therefore found it very promising to develop their automated simulation method.

For this work, we also consider the SSP technology promising for the further development of the co-simulation of one or more railway subsystem FMUs with additional supported parameter description sets. Therefore, this research paper aims to demonstrate the methodology by packaging all the FMU and parameter files into one file, belonging to each of the two use cases. The one file will then be used for its FMU-based simulation as integrated into the R4F Platform.

3 Methodology

In this section, first, we describe the approach followed to design and optimize the interface between the model and R4F Platform by using both the FMI and SSP standards (see Figure 1). Secondly, we explain the path we followed to realize the simulation of the FMI- and SSP-standardized model in the platform in Figure 2, which helps to demonstrate both previously mentioned use cases for the railway digitalization in this work.

3.1 Model Standardisation Approach

On the left side of Figure 1, we specify the available models belonging to the various railway infrastructure subsystems as the first step. These models consist of default input parameters, simulation algorithm and output channels, which are connected to each other, ready to be simulated and visually demonstrated by the default software tool of the model. In the next step, they are formatted into the FMU either with solver (FMI for Co-Simulation) or without (FMI for Model Exchange), which is possible by using suitable FMI tools (`https://fmi-standard.org/tools/`). This makes FMI-standardised models more tool-independent, easier to use and more platform-compatible, because the end user doesn't need to open the model's default software tool for simulation, but can easily redirect the FMU file anywhere else and then bring it into simulation inside the R4F Platform. The FMU file consists of a required model description XML file, which describes the model (incl. the registered inputs and outputs), an optional binaries folder, including file(s) for operating system (OS) compatibility of the FMU, an optional sources folder with all C-sources used for FMU compilation and linking, and an optional resources folder with resources needed by the FMU to read data from model specific files during initialization ((https://fmi-standard.org/assets/releases/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf), p. 66). As the last step, the FMU is packed into the SSP format, which can be exported by a limited number of tools, most of which are already commercial (https://ssp-standard.org/tools/). The SSP file consists of one or more FMUs representing all the simulation models of the railway infrastructure system, which shows a way to interoperate the models with each other by using one file in the platform. The SSP also includes the SSD and SSV files, which are important to describe, easily exchange, keep input and output parameters and apply them into the FMU(s). In addition, we create an extra folder in the SSP file, where we separately upload additional necessary input data sets and simulated results. The reason for keeping these input data sets separate is to keep all the input data modular and reusable, thus reducing data complexity. This also helps the end user to easily find what they want to see and set up as input. The simulated results, which are the model simulation results from the default software tool, are also needed to confirm the reliability of the SSP simulation by comparing them with the SSP simulation results. All these aspects of the SSP mentioned above are essential for integrating the models into the R4F Platform in terms of easy and platform-independent configuration, mapped description, backup storage and IP protection of the parameters for this work.

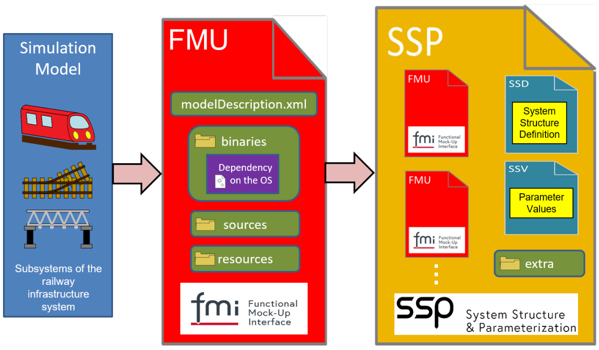

Simulation Model
Subsystems of the railway infrastructure system
FMU
modelDescription.xml
binaries
Dependency on the OS
sources
resources
fmi Functional Mock-Up Interface
SSP
FMU
FMU
SSD
System Structure Definition
SSV
Parameter Values
extra
ssp System Structure & Parameterization

Figure 1. Model standardisation approach.

3.2 Model Simulation Approach

Figure 2 shows how we implement the simulation process of the SSP-standardized model in the R4F Platform. The SSP file of the model consists of the FMU file(s), the SSV file for input parameterization, simulated results, additional input data sets and the SSD file with the pre-registered output channels needed for result analysis, validation and visualization to the end user.

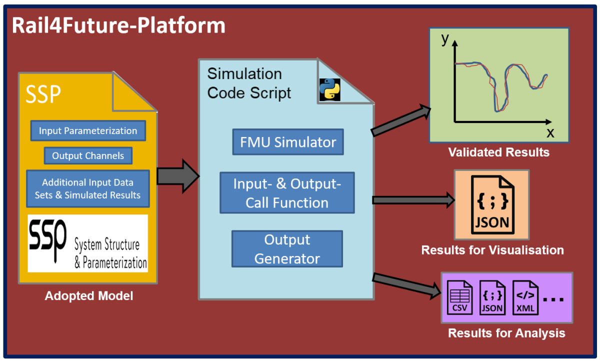


Figure 2. Model simulation approach.

After putting the SSP file into the platform, we provide a simulation code script, which can extract the FMU file from the SSP one, and all the data belonging to the FMU from the additional input data sets and simulated results with an input call function after executing it. After that, the code execution can put the FMU into simulation with an FMU simulator, then extract outputs from the simulation with an output call function, and finally generate all the outputs with an output generator in the platform. Surely, the functional components of the script should have the necessary software tool, packages and libraries, which can be provided from their official web pages or publicly available repositories freely as found out for this research work before. In the R4F Platform, the simulation code script can be executed by the Jenkins pipeline technology as we tested with the two previously mentioned use cases before, which is the key success of this work to integrate the models of different railway infrastructure subsystems into the platform.

The final step is to obtain the results of the entire simulation in order to analyse, validate and finally visualise them in the platform for the end user. First, the results of the SSP simulation are compared with the simulated results, generated from the simulation in the default software tool, in a plot (Validated Results). Second, the results to be visualized are converted into JavaScript Object Notation (JSON) format for visualization to the end user, as this open standard file format, with its human-readable key-value paired data structure, is well suited to the visualization part of the R4F platform. Finally, the results can be generated in different file formats (e.g. CSV, JSON, XML...), which the end user can analyze in different well-structured data types such as tables, arrays, lists, etc.

However, there are some challenges to be faced, namely dependencies and limitations, that we need to consider for the SSP simulation in the platform. For example, valid commercial license servers, Virtual Private Network (VPN) in some cases, OS-dependency of the FMU file and some open-source tools with necessary libraries and packages need to be connected and provided in the platform to make this kind of simulation successful. Therefore, it is essential have knowledge and experience of the model simulation process, data management, software installation, configuration and license management. Furthermore, the importance of dealing with IT-specific problems (e.g. system crashes, unsuccessful command executions) cannot be denied as we have realized specifically for this research.

4 Use Case Examples

In this section, first, we define the two use cases: 1) RLT calculation of a steel bridge and 2) MBS model of a railway vehicle. Second, we explain the conversion of their models into the FMU- and SSP-formats in detail. Then we demonstrate the simulation process of their SSP-standardized models in the R4F Platform. Lastly, we compare their SSP-simulation results and results provided by their default simulation tool with some plotted figures for validation purposes.

4.1 Use Case 1: Residual Life Time Calculation of a Steel Bridge

The use case demonstrates the life time and damage sum calculation of a steel bridge by using a Python simulation model with its input files containing demo data such as train data, influence lines, bridge positions and detail categories. For this work, the whole asset of the use case was provided by AIT - Austrian Institute of Technology GmbH. Validation of the structural life time with respect to fatigue failure is done for five different details of the designed bridge, visually shown by addressing their positions with points in Figure 3, which is provided by VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH. Each of the detail positions is assigned a detail category representing the cyclic stress $\Delta\sigma_c$ in N/mm² corresponding to $2 * 10^6$ load cycles of the fatigue strength curve as given in EN1993-1-9. Detail categories and the X, Y and Z coordinates of the detail positions are given in Table 1.

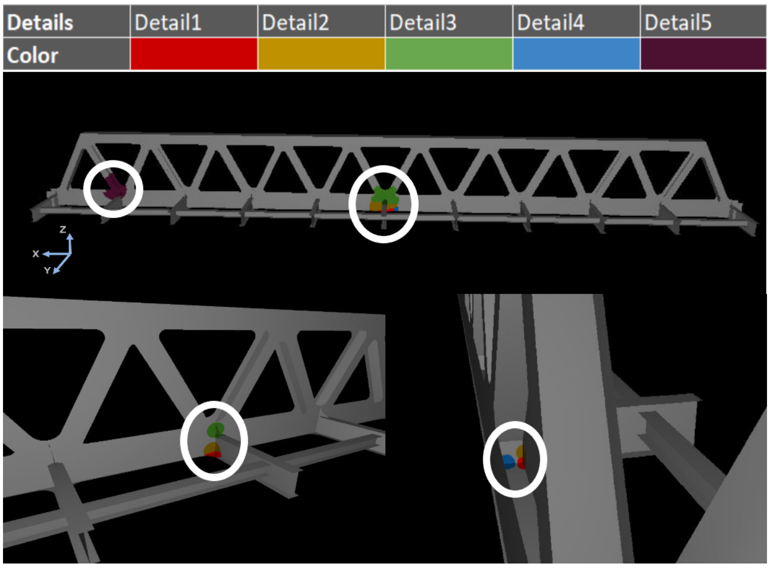
Table 1. Detail categories and positions of the steel bridge.

Detail	*Category [N/mm2]*	*X*	*Y*	*Z*
Detail1	80	0,057	0,306	-0,332
Detail2	80	0,009	0,265	-0,174
Detail3	36	0	0,32	0,56
Detail4	90	0	0	-0,332
Detail5	90	17,107	-0,088	0,866

In this section, first, we explain how the FMU and SSP formats are exported from the whole Python simulation model of the RLT calculation algorithm. Second, we demonstrate the entire simulation process of the use case using the model simulation approach mentioned before. Lastly, we compare the Python- and SSP-simulation results of the use case to each other by showing two diagrams for the result validation.

4.1.1 FMU Export

As the first step, we need to understand the Python simulation model consisting of different mathematical formulas with all its simulator, input and output components and bindings. After successfully testing the model simulation, we need to pack the model into the FMU format for co-simulation, which is possible by using the pythonfmu package (Hatledal, H. Zhang, and Collonval 2020). The next step is to prepare the model in a different Python code including all the formulas, input, output call functions and output generators in a pre-defined and imported pythonfmu class called FMI2Slave (`https://pypi.org/project/pythonfmu/`). Lastly, we execute a pythonfmu build command calling the prepared code to get the FMU format of the RLT calculation algorithm. In the FMU file, two different dll files in the bina-



Details	Detail1	Detail2	Detail3	Detail4	Detail5
Color	🟥	🟨	🟩	🟦	🟪

Figure 3. The steel bridge with highlighted detail points.

ries folder are found, which shows the OS-compatibility of the file in two different OSs (Linux & Windows).

4.1.2 SSP Export

After achieving the FMU file, we need to pack the FMU file into the SSP format to bring it into the R4F Platform as a complete model package with parameter description. For this purpose, we use the Model.CONNECT^{TM}, a commercial system integration tool provided by AVL List GmbH, because the tool can import FMU(s) for co-simulation and export SSP including the FMU(s). The SSP file also contains one SSD file describing and reserving all the registered parameters incl. inputs and outputs from the FMU-packaging process of the model.

4.1.3 Simulation Process

Figure 4 demonstrates the entire simulation process of the use case in a virtual machine, a prototype of the R4F Platform, due to the model simulation approach. First, we convert the prepared Python code with pythonfmu into the FMU format and lastly we achieve the SSP format of the model as mentioned in the previous subsection. From the SSP file, the FMU file is extracted and then simulated by running a simulation code script, which we wrote in the open-source Python programming language. Most importantly, the language supports the FMU-simulation for the use case by using the Python library called fmpy (`https://pypi.org/project/FMPy/`), which includes input and output call functions. In the use case of this work, the simulation code is actually the integrated version of the Python simulation of the prepared code included at the beginning. Moreover, we use a 2D-graphics package called matplotlib (Hunter 2007) in the simulation code to create static, interactive and publication quality plots. Additionally, we apply json and csv packages to the code to generate output in CSV and JSON formats after the simulation code script is executed by running the Jenkins pipeline. As outputs, the simulation gives out two plots from the matplotlib for result validation, CSV and JSON files for result analysis, and eventual visualization to the end user after reading all the provided CSV input files and Python-calculated life times and damage sums, which are the simulation results of the Python simulation model provided at the beginning.

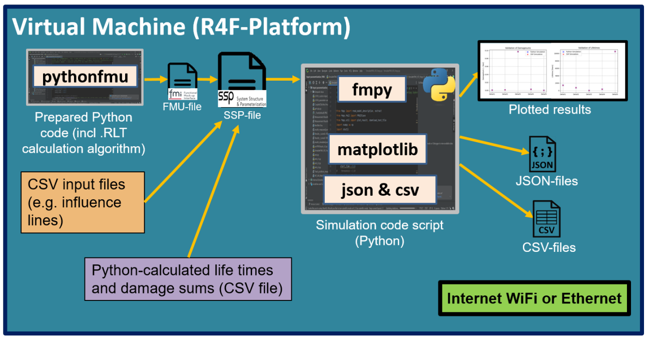

Virtual Machine (R4F-Platform)
pythonfmu
Prepared Python code (incl .RLT calculation algorithm)
FMU-file
SSP-file
CSV input files (e.g. influence lines)
Python-calculated life times and damage sums (CSV file)
fmpy
matplotlib
json & csv
Simulation code script (Python)
Plotted results
JSON-files
CSV-files
Internet WiFi or Ethernet

Figure 4. Simulation process of the RLT bridge.

4.1.4 Comparison Analysis for Result Validation

Figure 5 shows two plots, where the results of both Python- and SSP-simulations (life times and damage sums) are compared to each other due to the five defined detail categories.

As realized from the result consistency between the Python- and SSP-simulation in Figure 5, the SSP-simulation works properly and is suitable to be integrated into the R4F Platform.

4.2 Use Case 2: Multibody Simulation of a Railway Vehicle

The use case demonstrates the drive of a railway vehicle on a track. For this work, we use a generalized MBS model of the vehicle, which is parameterized, based on the Manchester Benchmark (Iwnicki 1999) and provided by Virtual Vehicle Research GmbH (ViF). We analyze the model in Simpack version 2022x, a commercial software tool from Dassault Systèmes, that provides a reliable way of understanding the dynamic behaviour of the railway vehicle due to the interaction between the vehicle and track. In the interaction, the irregularities of the track geometry, causing dynamics forces on the vehicle and track, are certainly to be taken into account in the MBS, because the track irregularities can lead to potential wheel damage and, in severe cases, derailments.

In this section, first, we explain the FMU- and SSP-export of the model due to the model standardization approach mentioned before with necessary methods, tools, inputs and outputs. After that, we describe the simulation process of the SSP-standardized model in a figure and finally compare its results with the Simpack simulation outputs resulted from the same input parameters.

4.2.1 FMU Export

After understanding and testing the whole simulation process of the MBS vehicle in Simpack, we need to define the necessary input parameters and output channels used for the FMU export. In the case of this work, first, we set up the track irregularities previously. Second, we define four different scenario parameters to realize a curved / straight vehicle drive with / without passengers for the input parameterization: 1) Track curve radius; 2) Track superelevation; 3) Additional mass (passenger+luggage); 4) Vehicle speed (constant). Besides, eight output channels of the vertical deflection of the primary spring (4 for each one of the two bogies) and one of track position are registered to the MBS model, because they are used for fatigue life calculation of the primary spring and therefore shown in a graph. Then we can directly obtain the FMU format of the model, including the Simpack solver, by using the Simpack tool. For this work, we export the MBS model into FMI 2.0 for Co-Simulation, which is the latest FMI version supported by the Simpack 2022x. Furthermore, we discovered that the FMU file is dependent on the OS, where the Simpack 2022x was installed and the

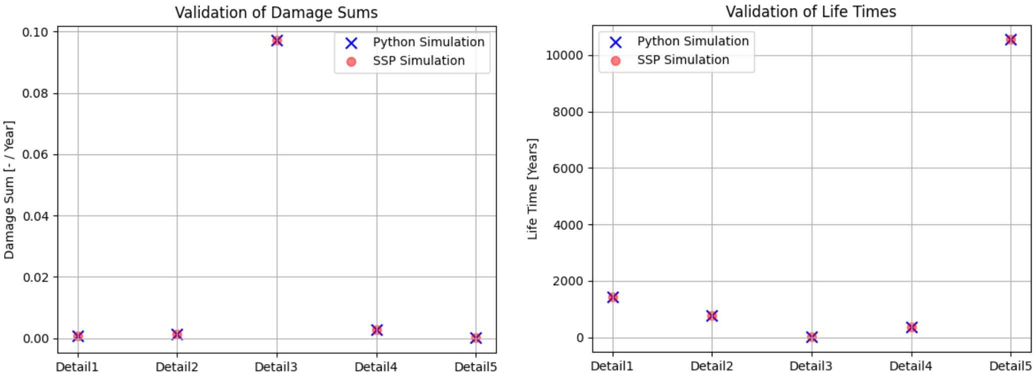


Figure 5. Simulation result comparison based on demo data of the RLT bridge.

file was exported from the tool before. It means that the file has only one dll file, working for the one OS, in the binaries folder. Therefore, the OS of the R4F Platform must definitively be taken into account to realize the simulation of the model in the platform.

4.2.2 SSP Export

After creating the FMU format of the MBS model, we need to convert it into the SSP format and then simulate it in the R4F Platform. For the task of this work, we use the same tool Model.CONNECT$^{\mathrm{TM}}$.

4.2.3 Simulation Process

After achieving the SSP-standardized model, it is simulated by running the Jenkins pipeline in the platform in a proper way. Therefore, we need to build, design and optimize the simulation process of the model based on the proposed model simulation approach mentioned above. After that, we need to adapt the whole process to the platform.

In Figure 6, we describe the considered simulation process of the MBS model with useful software, interfaces, file formats, software packages, adapters, inputs and outputs in the virtual machine. First, the FMU file in the SSP is read by a simulation code script after the FMU- and SSP-export of the MBS model. We wrote the script in the open-source Python language as the one in the previously mentioned use case. We apply the fmpy package for the FMU-simulation of the use case in the script as well. In addition, we use a Python library called bs4 in the code script, because it helps to call inputs directly from the SSV of the SSP file, where the end user can see and configure the clearly shown input parameters before starting the model simulation. Besides, like in the previously mentioned use case, we propose to use the matplotlib, json and csv packages in the script as output generators, which enable to create JSON file(s) for visualization, CSV for analysis and 2D-plot(s) to validate the SSP simulation with the Simpack simulated data, extracted from the previous Simpack simulation of the MBS model, in CSV and/or TXT format. Additionally, we use the Command Line Interface (CLI) technology to allow the end user to enter their inputs directly into the CLI window for demonstration purposes as an alternative to the input parameterization with the SSV file mentioned before.

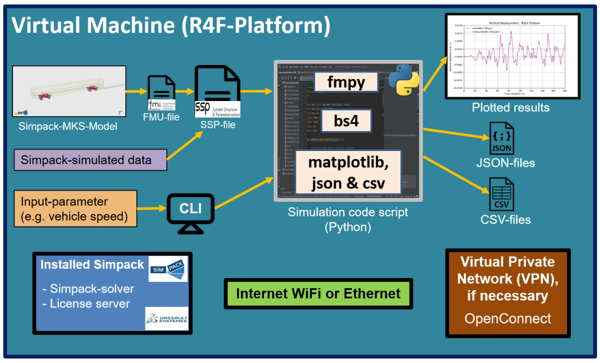


Figure 6. Simulation process of the MBS vehicle.

Furthermore, to overcome difficulties and limitations such as the connection to an available Simpack license server (either with VPN or without), OS-dependency of the FMU and internet availability in the simulation process, we installed and implemented the Simpack tool, a VPN service provider (e.g. OpenConnect tool) and an additional license server file, checked by the Simpack server to verify the licensing, in the platform.

4.2.4 Comparison Analysis for Result Validation

After running the simulation of the SSP-standardized model in the platform, we need to compare the Simpack- and SSP-simulation results to each other to validate the functionality of the SSP file in the proposed simulation process. In Figure 7, we show the vertical deflection of one primary spring, extracted from one of the previously mentioned output channels, due to the track position of the railway vehicle, which is loaded (passengers+luggage) and driven with a constant speed of 120 km/h on a tangent track without superelevation (straight drive) as an example.

As realized from the result consistency between the Simpack- and SSP-simulation in Figure 7, the SSP-simulation works properly and is suitable to be integrated into the R4F Platform.

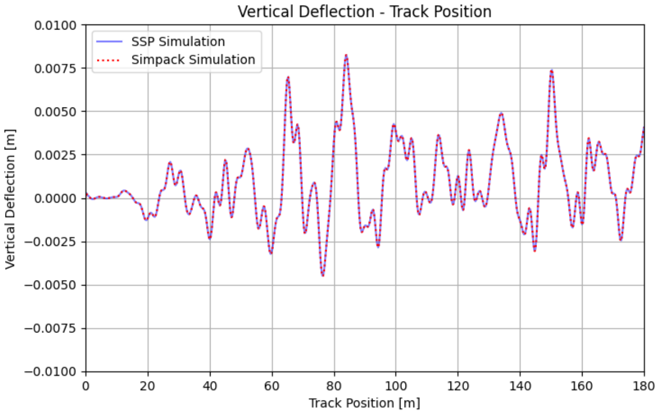


Figure 7. Simulation result comparison of the MBS vehicle.

5 Proposed Integration Process

Figure 8 shows the proposed integration process to be applied to the R4F Platform, which helps to manage and maintain the communication, interaction of the FMI- and SSP-standardized railway infrastructure models, provided from the Asset Provider, their final visual representation and enhanced user control for the end user properly. First, to continuously integrate the models in the platform, we propose to use the Jenkins pipeline technology in this research paper as mentioned before. Second, we will apply a version control system to the platform to manage the configuration, storage, exchange and archiving of necessary files and source codes related to the two use cases (SSP file, simulation code script, input & output files, pipeline code) in a software repository collaboratively. Besides, we discovered the Source Code Management (SCM) system in the Jenkins pipeline, which helps the pipeline to extract the updated sources from the repository. The reversed process is also possible by pushing the new output files back to the repository for result analysis as done by executing particular useful commands in the pipeline code before. Furthermore, we will use the pipeline auto-generation technology with a graph DBMS in the platform as mentioned previously (see (Reiterer, Schiffer, and Benedikt 2022) for example). In this research work, we created and configured a Jenkins pipeline, then described and modeled its workflow by developing a simple control script and a pipeline graph for testing purposes. Lastly, a visualization prototype is under development which demonstrates visualization and interaction techniques for the R4F Platform according to the use cases. It will allow users to set simulation input parameters and explore simulation results. This is done using a combination of linked views like 2D charts, 3D views, maps, and specially designed visualizations for exploring simulation results in a spatio-temporal context.

As we experienced in the model integration process of the two previously mentioned use cases before, there are some dependencies and limitations such as license server, VPN, internet and OS-dependency to overcome for the accomplishment of the integration process in the R4F Platform. For this research work, the Asset Provider exports the FMI format from various tools (e.g. pythonfmu, Simpack), SSP format directly from the Model.CONNECT^{TM} tool and essential Python packages from the Python tool other than providing the railway submodels to the platform. Besides, we surely need to do some internal configurations in the platform such as connecting the platform to a license server (e.g. Simpack license server), providing the OS, internet WiFi and/or Ethernet for wireless/wired network connection, 5G for mobile broadband network connection with relatively high internet speed and VPN service client for the license server connection in some cases.

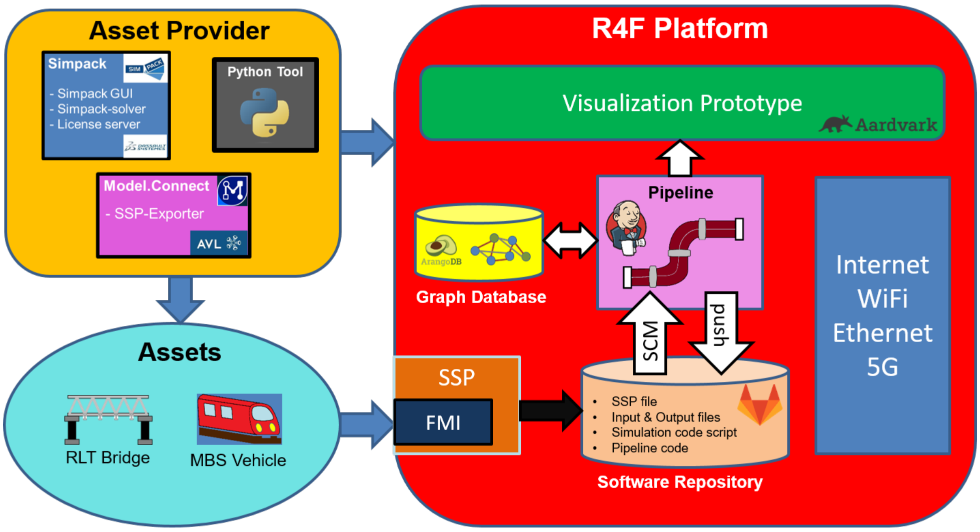


Figure 8. Proposed integration process.

6 Conclusion and Outlook

On the basis of the relatively precise result consistencies shown in the two use cases, the FMI- and SSP- standards are promising to be applied to different railway submodels for their integration into the platform. As experienced in this work, the FMI technology is a potentially useful interface standard supported by many different tools to adapt the models to the platform in terms of tool-independency, platform-compatibility and ease-of-use (portable as file). Moreover, the SSP-package of the models shows great benefits to enhance the IP protection, clear description, storage and exchange of different parameters belonging to the models. In addition, the SSP simulation of both use cases is carried out by executing the Jenkins pipeline interacted with a software repository belonging to a version control system, which also shows high potential in adaptiveness of the simulation to the R4F Platform.

Of course, there are more challenges to overcome to fully realize the model integration and interoperation in the platform. In the future, different use cases should interoperate with each other and be further described by using the FMI and SSP technologies. For example, a surrogate model (machine learning with the MBS model), proposed by (Zhou, Meierhofer, et al. 2023) to reduce the computational effort of the traditional MBS, is going to be applied to the platform by using both suggested interface standards. Additionally, an anti-slip control system is planned to be integrated into the platform as co-simulated with the MBS model to be able to co-accelerate and co-brake the railway vehicle. Finally, the proposed integration process is going to be further developed and optimized in a virtual machine by using containerization, container orchestration and deployment technologies, which assist to integrate and deploy the FMI-standardized models and their belonging data continuously, time- and energy-sustainably as a major benefit in future.

Acknowledgements

The authors would like to acknowledge the financial support of the COMET project Rail4Future (882504) within the COMET Competence Centers for Excellent Technologies from the Austrian Federal Ministry for Climate Action (BMK), the Austrian Federal Ministry for Digital and Economic Affairs (BMDW), the Vienna Business Agency and the Styrian Business Promotion Agency (SFG). The Austrian Research Promotion Agency (FFG) has been authorised for the COMET program management. We further thank for the support from ÖBB-Infrastructure AG and Siemens Mobility Austria GmbH.

References

Ahmadi, Miad et al. (2021). “Adapting digital twin technology in electric railway power systems”. In: *2021 12th Power Electronics, Drive Systems, and Technologies Conference (PEDSTC)*. IEEE, pp. 1–6.

Balakirsky, Stephen et al. (2010). *Simulation, Modeling, and Programming for Autonomous Robots*. Springer.

Bastian, Jens et al. (2011). “Master for co-simulation using FMI”. In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 63. Linköping University Electronic Press, pp. 115–120.

Bertsch, Christian, Elmar Ahle, and Ulrich Schulmeister (2014). “The Functional Mockup Interface-seen from an industrial perspective”. In: *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 096. Linköping University Electronic Press, pp. 27–33.

Blockwitz, Torsten et al. (2012). “Functional mockup interface 2.0: The standard for tool independent exchange of simulation models”. In: *Proceedings*.

Golightly, David et al. (2022). “A feasibility assessment of multi-modelling approaches for rail decarbonisation systems simulation”. In: *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit* 236.6, pp. 715–732.

Hällqvist, Robert et al. (2021). “Engineering domain interoperability using the system structure and parameterization (ssp) standard”. In: *Modelica Conferences*, pp. 37–48.

Hamarat, Mehmet, Mayorkinos Papaelias, and Sakdirat Kaewunruen (2022). “Fatigue damage assessment of complex railway turnout crossings via Peridynamics-based digital twin”. In: *Scientific reports* 12.1, p. 14377.

Hartmann, Bjørn (2020). “Digital Twin Monitoring of Offshore Knuckle Boom Crane”. MA thesis. NTNU.

Hatledal, Lars Ivar, Houxiang Zhang, and Frederic Collonval (2020). “Enabling Python Driven Co-Simulation Models With PythonFMU.” In: *ECMS*, pp. 235–239.

Hunter, J. D. (2007). “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

Iwnicki, SD (1999). *The Manchester Benchmarks for Rail Vehicle Simulation. Department of Mechanical Engineering, Manchester Metropolitan University, England, Supplement to Vehicle System Dynamics, Vol. 31, ISSN 0042–3114*.

Kaewunruen, Sakdirat, Mohannad AbdelHadi, et al. (2022). “Digital Twins for Managing Railway Bridge Maintenance, Resilience, and Climate Change Adaptation”. In: *Sensors* 23.1, p. 252.

Kaewunruen, Sakdirat and Qiang Lian (2019). “Digital twin aided sustainability-based lifecycle management for railway turnout systems”. In: *Journal of Cleaner Production* 228, pp. 1537–1551.

Kaewunruen, Sakdirat and Ningfang Xu (2018). “Digital twin for sustainability evaluation of railway station buildings”. In: *Frontiers in Built Environment* 4, p. 77.

Mysari, Sriniketan and Vaibhav Bejgam (2020). “Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible”. In: *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. IEEE, pp. 1–4.

Ochel, Lennart et al. (2019). “Omsimulator–integrated fmi and tlm-based co-simulation with composite model editing and ssp”. In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. 157. Linköping University Electronic Press.

Pieper, Tobias and Roman Obermaisser (2018). “Distributed co-simulation for software-in-the-loop testing of networked railway systems". In: *2018 7th Mediterranean conference on embedded computing (MECO)*. IEEE, pp. 1–5.

Pierre R. Mai (2018-10-11). *MAP "System Structure and Parameterization" – Current Status and Plans*. Modelica Association. URL: https://ssp-standard.org/publications/2018_American_Modelica_Conference/2018_Usermeeting_SSP-StatusandPlans.pdf (visited on 2023-05-12).

Reiterer, Stefan H, Clemens Schiffer, and Martin Benedikt (2022). "A Graph-Based Metadata Model for DevOps in Simulation-Driven Development and Generation of DCP Configurations". In: *Electronics* 11.20, p. 3325.

Thule, Casper et al. (2018). "Towards the verification of hybrid co-simulation algorithms". In: *Software Technologies: Applications and Foundations: STAF 2018 Collocated Workshops, Toulouse, France, June 25-29, 2018, Revised Selected Papers*. Springer, pp. 5–20.

Zhang, Shiyao et al. (2021). "A digital-twin-assisted fault diagnosis of railway point machine". In: *2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI)*. IEEE, pp. 430–433.

Zhou, Shiyang, Stefan Dumss, et al. (2022). "A Conceptual Model-based Digital Twin Platform for Holistic Large-scale Railway Infrastructure Systems". In: *Procedia CIRP* 109, pp. 362–367.

Zhou, Shiyang, Alexander Meierhofer, et al. (2023). "A Machine-Learning-based Surrogate Modeling Methodology for Submodel Integration in the Holistic Railway Digital Twin Platform". In: *Procedia CIRP* 119, pp. 345–350.

PNRG – A Library for Modeling Variable Structure Energy Grids in Modelica using Energetic Petri Nets

Christian Gutsche[1, 2] Zizhe Wang[1, 2] Sebastian Götz[2] Volodymyr Prokopets[2] Uwe Aßmann[2]

[1]Boysen–TU Dresden–Research Training Group, Dresden, Germany
[2]Chair of Software Technology, Technische Universität Dresden, Dresden, Germany, {christian.gutsche, zizhe.wang, sebastian.goetz1, volodymyr.prokopets, uwe.assmann}@tu-dresden.de

Abstract

Operating energy grids with a high share of renewable energy sources (RES) requires system reconfiguration as a response to environmental condition changes. To understand them better, simulations are needed and Modelica is an excellent choice for that. Energy grids with event-based reconfigurations are an instance of variable structure systems (VSS). However, the full support of VSS in Modelica is challenging and topic of ongoing research. Petri nets (PNs) offer a formalism for modeling VSS. The capability to simulate PNs in Modelica gives an opportunity to model VSS in Modelica. This paper presents an approach to utilize PNs in Modelica for modeling variable structure energy grids. Therefore, we introduce energetic Petri nets, a special type of PNs and an experimental library called PNRG for PN-based energy system modeling is presented. Furthermore, possibilities and limits of modeling VSS energy grids are discussed and an outlook how to develop this technique is provided.

Keywords: Energy Grids, Sector Coupling, Variable Structure Systems, Context-oriented Programming, Petri Nets

1 Introduction

The climate change forces societies to radically change the way of providing energy to their residents. Energy grids with a high share of renewable energy sources (RES) have to handle high fluctuations between the production and consumption of different energy media. To mitigate the negative effects and optimize the energy efficiency, so called sector coupling is investigated. This principle describes the combination of different energy systems. For example, electrical, gas and heat systems are coupled so that an oversupply of one energy medium can compensate the lack of another one. Sector coupling is nowadays broadly discussed in science, as (Fridgen et al. 2020) describes. However, the complexity of systems grows with an increased sector coupling and therefore modeling of energy grids gets even more important. With Modelica, there is a widely used modeling language which is well suited to simulate such complex systems.

The Modelica library TransiEnt (Senkel et al. 2021) is an excellent choice to simulate exactly this type of energy systems (Heckel et al. 2022). It provides a magnificent level of detail and includes many components which are needed for realistic energy grid simulations. However, TransiEnt only works with the Dymola environment for now.

Another problem with simulating energy grids with a high share of RES and sector coupling is the missing possibility to fully describe and simulate so called variable structure systems (VSS) in Modelica. While Specific types of VSS can be simulated, mostly basic definitions as in (Utkin 1977), some more advanced VSS types start to cause problems (Tinnerholm 2022). For example, when the number of equations or variables is varying.

While Modelica simulations based on PNs are also limited in the way of simulating VSS with dynamically changing equations or varying numbers of variables and components, they allow to describe the variability of models and their state transitions. To make component based modeling using PNs accessible in Modelica, a special type of PNs, energetic PNs will be defined. They are also necessary to realize instantaneous effects on PN components within the energetic Petri net after local changes. These energetic Petri nets will be used to implement a Modelica package which allows to model VSS energy grids. In the following, two example models using this package are presented and it will be shown, that VSS can be modeled with that.

1.1 Variable Structure Energy Grids

The power supply of RES power plants is highly depended on the availability of natural energy sources like sun and wind energy. Therefore ensuring a reliable power supply from renewable energy sources requires a focus not only on sunny and windy days but also on cloudy days with low wind conditions, particularly when operating power grids with a high share of RES. Various storage systems are discussed to be used for mitigating these effects. As (Alhamali et al. 2016) show, there are a lot of possible solutions with different advantages and disadvantages. All storage systems need additional infrastructure which might be expensive. In some cases the location does not allow to install specific plants.

Another way to react to supply or demand changes is to activate and deactivate producers or consumers during

DOI
10.3384/ecp204727

operation time. Additionally, special grid designs can improve the resilience of power grids. (Liu et al. 2022) provides a general overview of so called micro grids (MGs) to stabilize larger power grids. The investigation of such designs is an interesting objective for research. When simulating the variability of such power grids, structural changes must be performed during the simulation's run time.

1.2 Variable Structure Systems in Modelica

Systems with a variable structure are quite natural. For example, opening windows when the air quality is bad or covering windows when it is too sunny outside are such structural changes of a room system. Therefore, it is not surprising that VSS are studied for a long time as (Utkin 1977) already recognises in 1977. However, mathematically these VSS are not easy to describe. Especially for modeling there is a lot of ongoing research in this field. Some theoretical background is provided by (Benveniste et al. 2019).

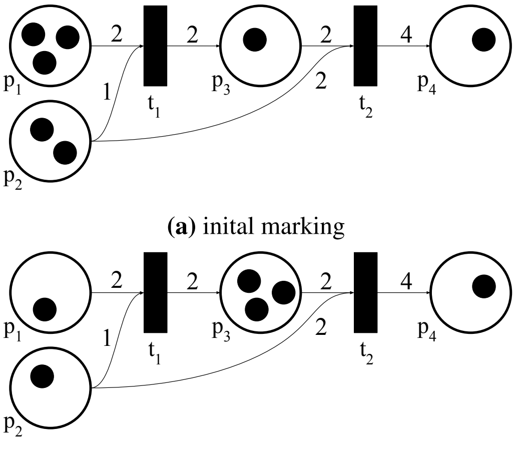
For Modelica there is a significant interest in increasing the possibilities for VSS modeling. Different approaches to simulate VSS in Modelica have been investigated and proposed, e.g. MOSILAB (Nytsch-Geusen et al. 2005), MoVasE (Esperon, Mehlhase, and Karbe 2015) and DySMo (Mehlhase 2015). Also, other solutions are proposed like the modeling language Sol introduced by (Zimmer 2010). However, all these approaches realized VSS in Modelica only partly or implemented VSS only in a subset of Modelica. Recently, (Tinnerholm, Pop, Sjölund, et al. 2020) presented an approach to make the modeling of VSS available in Modelica using a new compiler based on the Julia programming language which is discussed in detail in (Tinnerholm, Pop, and Sjölund 2022).

While investigating VSS, two aspects need to be considered. The studied system must be described clearly. This means not only the components itself have to be defined well but also how these components behave in different states of the system. The other aspect is the controlling of the systems states. In complex systems a lot of conditions can occur under which state transitions shall be performed. But not only the definition of these rules can become very complicated. Also the approval of a correctly working state control is an important but complex topic itself. A secure formalization of these state transitions must therefore be defined. So called PNs are an excellent choice for systems with a large and complex state space.

1.3 Petri Nets

PNs were firstly described by Carl Adam Petri in his PhD thesis (Petri 1962) but got extensively expanded afterwards. The following definition of a basic Petri net is given in (Murata 1989).

Definition 1.1 (Petri Net)**.** A Petri net is a graph defined by the 5-tuple $PN = (P,T,F,W,M_0)$ with the following definitions:



(a) inital marking

(b) after firing

Figure 1. An example of a simple discrete Petri net. (a) shows the inital state of the PN and (b) the state after the first firing which is also the final state.

1. $P = \{p_1, p_2, ..., p_m\}, m \in \mathbb{N}$ is a finite set of places
2. $T = \{t_1, t_2, ..., t_n\}, n \in \mathbb{N}$ is a finite set of transitions
3. $F \subseteq (P \times T) \cup (T \times U)$ is a set of arcs
4. $W : F \rightarrow \mathbb{N}_{>0}$ is a weight function
5. $M_0 : P \rightarrow \mathbb{N}$ is the initial marking
6. $P \cap T = \emptyset$
7. $P \cup T \neq \emptyset$

This definition results in a directed graph composed of places and transitions which are connected with arcs. Every place p has a number of tokens $M(p) \in \mathbb{N}$ and every arc has a weight w, here denoted as w_{in} when it points to a transition and w_{out} when it points to a place. How the PN works is explained with the help of Figure 1.

A transition t fires if $W(p_i,t) \leq M(p_i)$ for all places p_i which are connected to the transition via the arcs with the weights $w_{in,i}$. In Figure 1 transition t_1 has two input places p_1 and p_2. The weight of the arc between p_1 and t_1 is $W(p_1,t_1) = 2$ while the weight for p_2 and t_1 is $W(p_2,t_1) = 1$. Since $M(p_1) = 3 \geq 2 = W(p_1,t_1)$ and $N(p_2) = 3 \geq 1 = (p_2,t_1)$ the transition fires. Transition t_2 can not fire since $M(p_3) = 1 < 2 = W(p_3,t_2)$ which is the weight of the arc between p_3 and t_2. The arc weights define the number of tokens which are taken from the input places and given to the output places. Therefore, both p_1 and p_2 now have 1 token and since the output weight of t_1 is $W(t_2,p_3) = 2$, the place p_3 has now 3 tokens. Since $M(p_1) = 1 < 2 = W(p_1,t_1)$ and $M(p_2) = 1 < 2 = W(p_2,t_2)$, neither the firing condition of t_1 nor t_2 is fulfilled after the first firing so that there will be no further firing. The PN is in its final state.

There are many extensions to these basic discrete Petri nets. Here, only some notable extensions are mentioned.

The introduction of inhibitor arcs, firstly described in (Agerwala 1974) is simple yet powerful. They allow to check if a token number is smaller than a specific threshold, and enable firing in this case without consuming tokens. Originally this threshold was 1 and therefore inhibitor arcs were used to check if places are empty. However inhibitor arcs can also be used with thresholds > 1. Test arcs (David and Alla 2008) allow to check if a token number is larger than a specific threshold and they enable the firing of transitions without consuming tokens. They can be used to prioritize transitions and therefore avoid conflicts when two transitions have the same input place.

Another important extension are continuous Petri nets (David and Alla 1987). For those the tokens are real numbers. Their transition firing is not described by discrete token transport but continuous flows. Therefore, those are a limit case of discrete PNs with token numbers $\gg$ arc weights (David and Alla 2008). How this type of PNs works is explained by the following example. A continuous Petri net consisting of a continuous place p_1 with no input arc, an initial token number of $M(p_1) = 2$ and an output arc which connects p_1 to a transition t_1 with $W(p_1,t_1) = 4$ would behave the following way. The token number of p_1 would decrease with an constant rate of $4\,\frac{1}{\mathrm{s}}$ until the token number is 0. So the number of tokens at the time t is given as the equation $M(p_1,t) = -4t + 2$ for $t < 0.5\,\mathrm{s}$ and afterwards $M(p_1) = 0$. These Petri nets are powerful since they can represent systems of ordinary differential equations.

Petri nets containing discrete and continuous places and transitions are called hybrid petri nets. The modeling possibilities with Petri nets are significantly enhanced with these hybrid PNs since continuous systems like fluids flowing in a pipe system can be modeled with such PNs.

In Modelica, the open-source library PNLib (Proß and Bachmann 2012) can be used for integrating PNs for modeling systems. PNLib implements the basic discrete PNs as well as the test and inhibtor arcs extension, continuous Petri nets and other special transitions types.

1.4 Petri Nets and Variable Structure Systems

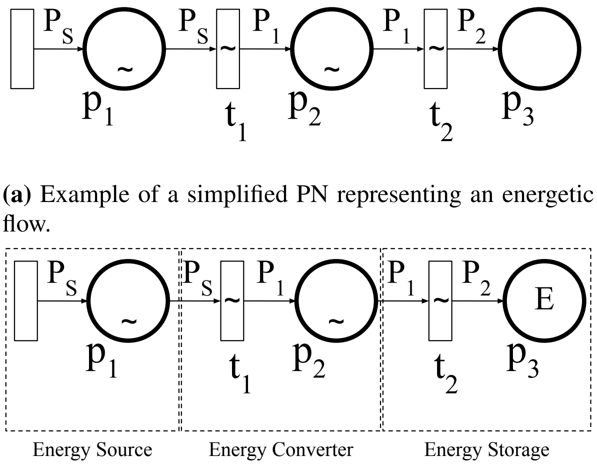
Why are those automata interesting for studying VSS? Petri nets can be used as state machines what is well described in (David and Alla 2008) but PNs are much more powerful. They can define many sub-states in parallel and how these affect each other. Additionally there are numerous verification methods to investigate the Petri net properties like reachability or liveness. An overview for some of these model checking approaches is given in (Wolf 2019). Therefore, PNs are interesting for controlling the state transitions of VSS.

However, this is not the only connection between VSSs and Petri nets. In (Mai et al. 2018), so called adaptive Petri nets were introduced. In these adaptive PNs, subnets can be activated and deactivated based on the marking of the PN. Therefore, the Petri net itself is a variable structure system where the state transition controlling is integrated in the system itself. Furthermore, (Mai et al. 2018) presents an algorithm to flatten adaptive Petri nets to basic PNs with inhibitor arcs.

In (David and Alla 2008) several application examples for modeling based on Petri nets are described, such as gas storage systems or a four-stroke engine. Examples for PN-based modeling in Modelica are presented in (Proß and Bachmann 2012).

For the goal of enabling the expression of VSS in Modelica, the application of Petri nets appears to be a promising approach. In the following, a way to use PNs for modeling power grids will be presented.

2 Energy Grid modeling and Energetic Petri Nets



(a) Example of a simplified PN representing an energetic flow.

(b) Possible interpretation of the different parts of the energetic Petri net.

Figure 2. An example of a simple energetic Petri net. Energetic places and transitions are denoted with a "~". Transitions and places without a "~" are continuous transitions and places.

The modeling of VSS energy grids is not possible in Modelica when using large physical models. Therefore, a method reducing the modeling to a problem accessible in Modelica is needed. An alternative approach based on Petri nets is capable of doing this and is presented in the following.

Modeling electrical grids with Petri nets was investigated for example in (Lu et al. 2016). However, these modeling techniques lack user friendliness and do not contain sector coupling. The component based modeling in Modelica would help to increase the usability of this modeling method. To study the usage of Petri nets for the component based modeling of VSS power grids, a simplified base model is used which just takes the power generation and consumption into account. This means other properties like phase angles or the frequency that are also crucial for the grid stability are neglected in this approach. However, the main goal of the model discussed here is to validate the assumption, that it is possible to model VSS using Petri nets.

Modeling the power generation and consumption leads to the idea to model the flow of energy through the power grid. Since in continuous Petri nets arc weights represent the derivative of the place tokens, the natural choice for PN representations for energy and power are the following: The tokens in places represent stored energy and therefore, the transition arcs will represent the electrical power of components.

However, energy can not be stored in every case. So during modeling, it is important to make sure that energy is not stored in places where it should not be stored. To ensure this, two possible solutions can be implemented. The first method is to realize all grid components in a single transition whenever energy is not allowed to be stored inbetween, instead of creating one transition for every grid component with places between the transitions. This is very disadvantageous, e.g. for component-based modeling or when a sub-net needs to be activated adaptively. Therefore, it is preferable to define so called energetic Petri nets that cover the needs of energetic flow modeling by definition.

Definition 2.1 (Energetic Petri Net)**.** An energetic Petri net is a hybrid Petri net with inhibitor arcs and:

1. Energetic places P_E with $N \in \mathbb{N}_{>0}$ input and $M \in \mathbb{N}_{>0}$ output arcs, that behave like continuous places with:

$$\sum_{i=1}^{N} w_{\text{in},i} = \sum_{j=1}^{M} w_{\text{out},j} \tag{1}$$

2. Energetic transitions T_E with $K \in \mathbb{N}_{>0}$ input and $L \in \mathbb{N}_{>0}$ output arcs that behave like continuous transitions with:

$$\forall j : w_{\text{out},j} = f\left(w_{\text{in},1}, \dots, w_{\text{in},K}\right) \tag{2}$$

and

$$\sum_{i=1}^{K} w_{\text{in},i} \leq \sum_{j=1}^{L} w_{\text{out},j} \tag{3}$$

for normal arcs and no restriction for inhibitor or test arcs.

This definition ensures three properties. At first, the mentioned flow in places where no energy should be stored is given. Also, energy can not be produced but only converted with loss due to the efficiencies of the components. For dynamic simulations it is also important to not only have constant weights but weights that can change over time. A definition for such PNs was already introduced in 1978 by (Valk 1978). These self-modifying PNs allow to represent more complex ordinary differential equation systems with PNs using the token numbers of places as arc weights. In combination with equation 2 this leads to a Petri nets where the change of an arc weight leads to instantaneous changes of weights in following arcs within the energetic PN.

How can these Petri nets be used to model energy flows? As shown in Figure 2, power sources like wind or solar energy can be modeled as continuous transitions without input arcs and varying weights of the output arcs. The output arc weight should be the source power P_S. Since these natural energy sources cannot be stored directly, the source transition is connected to an energetic place which instantaneously transfers the energy to an energetic transition which can represent simplified power plants converting the energy to electrical energy using a physical equation $P_1 = f(P_S)$. The last place is an ordinary continuous one and can be interpreted as an ideal energy storage.

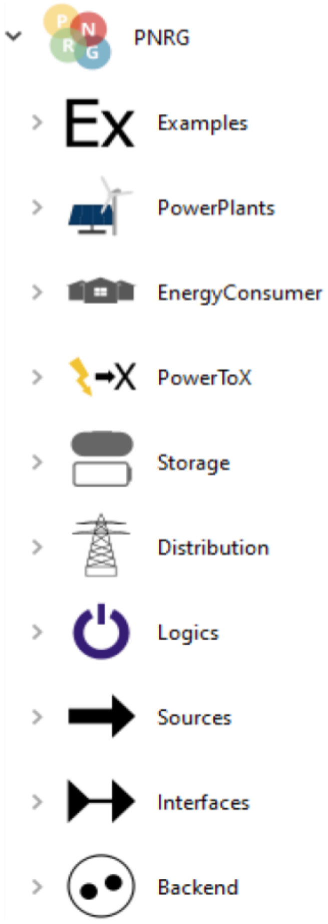


Figure 3. Structure of the PNRG package.

3 PNRG Library – Petri Net based Renewable Energy Grids

For a user-friendly and reusable modeling, the different sub-nets representing components of the energy grid should be programmed as Modelica models that can be connected accordingly. Therefore, a package based on the PNlib Petri nets was developed for component-wise mod-

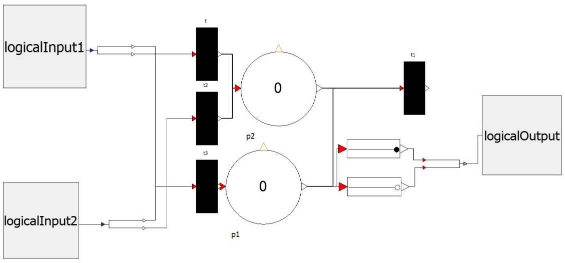


Figure 4. Example of an PNlib implementation of an AND gate. A token number of 1 represents true and a token number of 0 represents false.

eling of different energy grids. However, the package is still in an experimental state. In this section we will briefly introduce the main aspects of this package.

Figure 3 shows the structure of the PNRG package. Examples for specific Models are photovoltaic power plants, wind power plants, electricity consumers, electrolysers, batteries, transformers and logic components like AND gates. To implement the energetic places and transitions, the continuous place and transition models from PNlib were modified.

3.1 Control Layer

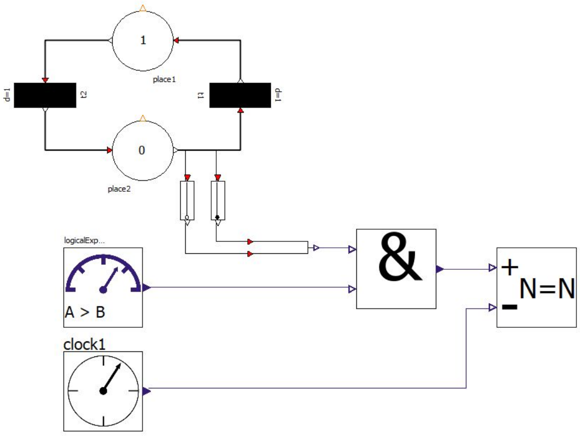


Figure 5. Example for different logical elements based on Petri nets. A simple PNlib Petri net, a module to convert boolean expressions to logical inputs and a clock are connected to an integer controller using an AND gate.

As described in 1.4, the first application of PNs for VSS is the control of state transitions. For switching PNRG components on and off, logical interfaces are defined. These logical interfaces have a combination of test and inhibitor arcs as input and output. Both of these arcs are needed to express logical expressions satisfactorily. This allows to build logical gates. Exemplary, an implementation of an AND gate built using PNlib Petri nets is shown in Figure 4.

In principle, every PNlib place connected to test and inhibitor arcs can be used for these logical interfaces. Additionally, a periodical clock and a module that converts a boolean expression to a logical input are implemented. Another handy component is a module called integer controller. This module can be used for incrementally increasing or reducing an integer number by activation the respective logical input. This can be used, for example, if the number of active wind turbines should be variable during simulation time. The application layer components that can be activated or deactivated have logical inputs as interfaces to realize this functionality. An example for such an activation or deactivation is the charging or discharging of a battery which can be both stopped and reactivated depending of the current power supply and demand.

Figure 5 shows an example for the usage of these control layer components. Here, an integer controller is controlled by a clock for decreasing its number and it is increased when place 2 has a token number of 1 and for two variables A and B, the requirement $A > B$ is fulfilled.

In principle, controlling via the definition of rules can also be done with standard Modelica Boolean Blocks. However, in some cases it might be desirable to only use Petri nets in the model, e.g. for verification. Then it is helpful to use the components provides by PNRG.

3.2 Application Layer

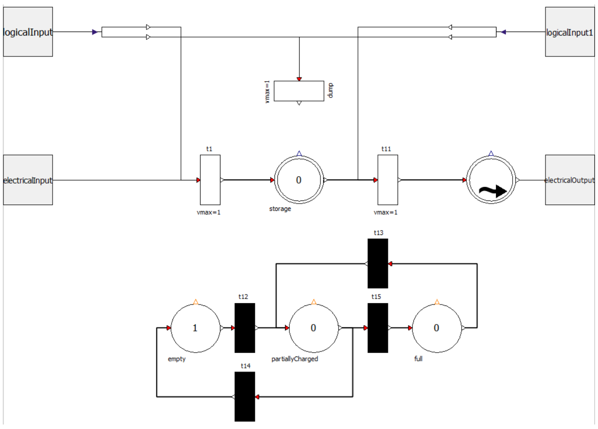


Figure 6. The implementation of a battery. the upper interfaces are the logical inputs that control the charging and discharging. The energy storage is modelled as a place. The Petri net at the bottom is a inner control net that indicates the state of charging.

To model energy grids, typical components like power plants, storage systems or consumers are implemented with Petri nets. In the following, some basic implementations are presented.

In the most simplified way, components like photovoltaic (PV) power plants can be treated as an energy converter abstracted as a transition. The solar power P_{in} can be read in from a file and fed into a Petri net transition output. Using energetic PNs, a model that fulfills constraints of an energetic flow can be ensured and separated into components. The PV power plant can then be modeled as a transition with the input weight:

$$w_{in} = P_{in} \tag{4}$$

and the output weight:

$$w_{out} = P_{in} \cdot A \cdot \eta \tag{5}$$

where A is the area of the PV panels and η is its efficiency. The angle between panel and sun or the temperature of the plant is neglected here but could be added in principle. The transitions output will then be connected to an energetic place. Afterwards consumers or storage systems could be connected to the energetic PN, where energy can be stored in normal continuous places.

With this implementation, the activation and deactivation of components can be achieved by blocking transitions with inhibitor and test arcs so that they are not able to forward energy. An example for a battery with charging and discharging that can be activated is shown in figure 6.

Simplifications that are applied are a constant charging and discharging power of batteries and the ignoring of energy losses during charging, discharging and storing of the energy. Also, power grids are assumed to have no energy losses. Wind turbines are idealized and do not include inertia effects.

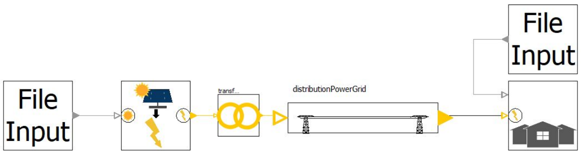


Figure 7. Simple example of an energy grid modeled with PNRG.

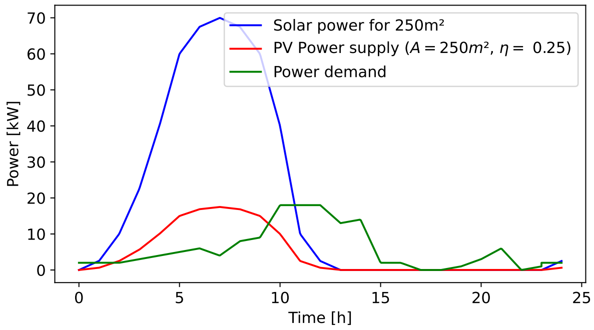


Figure 8. The power of the sun for 100 m^2, a PV power plant with 250 m^2 are and an effiency of 0.25 as well as a consumer as shown in Figure 7.

A simple example without VSS is shown in Figure 7. The grid consists of a PV power plant, a transformer, a distribution power grid and a consumer. The solar radiation power per area and the consumed power are read in from files. These values are artificially chosen but shall be used for demonstration purposes. The PV power plant has an area of $A = 250\,\mathrm{m}^2$ and an efficiency of $\eta = 0.25$. The results are shown in Figure 8. It can be seen that the power of the PV plant follows the course of the solar power. Also it is clearly visible that the power supply exceeds the demand from $t = 2$h to $t = 9$h and can not fulfill it after $t = 10$h.

4 Modeling VSS Energy Grids with Energetic Petri Nets

To study how VSS can be modeled with PNRG, two examples were investigated. Both examples include conditional charging and discharging of a battery and a variable number of active wind turbines. The second example also contains instances of conditional connections.

4.1 Example 1 – Adaptive Activation of Wind Turbines and Battery Charging and Discharging

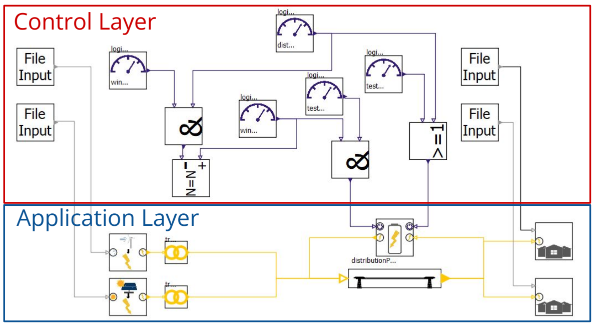


Figure 9. Example of an VSS energy grid modeled with PNRG in Modelica. The battery is charged only if enough power can be supplied from the wind and solar power plants. To control the difference between power supply and demand, additionally the number of active wind turbines can be controlled and the discharging of the battery can be activated.

The first example is shown in Figure 9. The grid contains wind power plants and PV power plants, a battery as a storage system and two consumers. The control layer is shown in Figure 9 marked with a red rectangle. It contains the file inputs and the rules that control the variability of the system. These rules are defined as following:

If the power supply is larger than the demand of the consumers, then:

1. Activate the battery charging if charging power would not overload the power supply.
2. If it is still too high, deactivate wind turbines.

If the power supply gets smaller than the demand of the consumers, then:

1. Reactivate wind turbines.
2. If all turbines are active, discharge the battery.

The number of active wind power plants is controlled with the integer controller. These rules should lead to a

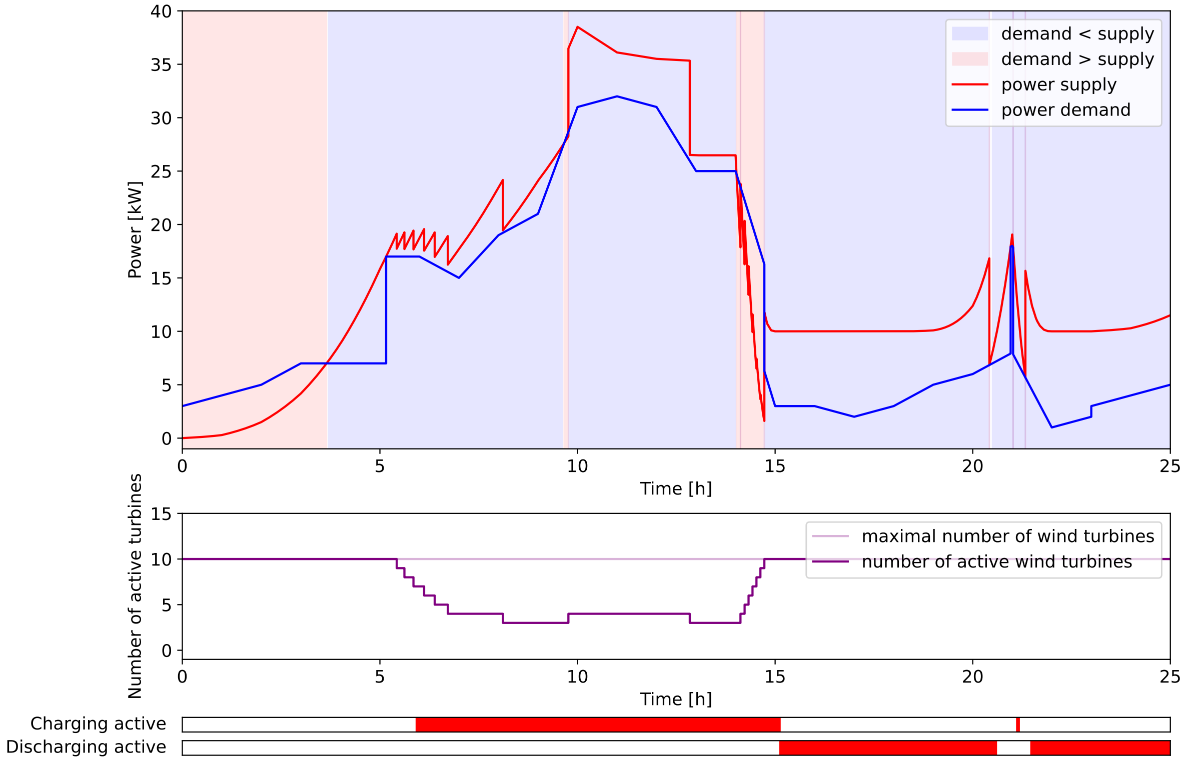


Figure 10. The upper plot shows the power supply and demand for the simple VSS grid from Figure 9. The background is colored according to the difference between power supply and demand. The plot shows that the rules lead to the desired grid behavior and prove that VSS can be modeled in this case. The lower plot shows the number of active wind turbines. In the two bars at the bottom, the red color shows if the battery charging and discharging is active.

system, where it is tried to have an power supply larger than the consumption but the supply should not exceed the demand unnecessarily.

The resulting curves for power supply and demand are shown in Figure 10. The background is colored blue if the demand is smaller than the supply and it is red otherwise. The lower part in Figure 10 shows if the battery is charging or discharging indicated by a red bar. It also shows the number of active wind turbines.

Figure 10 shows that at the moment when the power supply covers the demand and the charging power of the battery at $t = 5$h, the battery start to charge. Since the power supply is still rising, the wind turbines are deactivated successively. At around $t = 10$h the power demand starts to rise above the supply so a wind turbine is reactivated but again deactivated at $t = 13$h when the power demand decreases. After about 14h the available wind and solar energy decreases significantly so despite reactivating all wind turbines step by step, the power demand can not be covered. So the battery charging is deactivated at $t = 15$h and since this is not enough the discharging of the battery starts. A short natural energy peak from $t = 20$h and $t = 22$h stops the discharging for a short period. At the beginning of the simulation, the discharging is inactive despite a shortage of the power supply because the battery is initialized empty and therefore cannot discharge.

In conclusion, the result confirms that the system behaves like desired and that the rules change the system accordingly. This also proves that VSS can be modeled using Petri nets for this case.

4.2 Example 2 – Conditional Connections

After example 1 showed that PNs can model VSS in Modelica, a more complex model shall be investigated now. As shown in Figure 11 the system consists of two grids. Both of them are similar to the model presented in example 1 but only one consumer is connected to each of the grids. Also, the number of wind turbines and the PV area is different. However, the main difference is that one of the PV power plants is not statically connected to the upper distribution grid but dynamically connectable to one of both grids during run time.

The implemented rules for the batteries and the number of active wind power plants is basically the same as in example 1. Some modifications are done to avoid conflicts for the different state of the conditional connection. The upper PV power plant will always be connected to the distribution grid, with the smaller difference between supply and demand. In this difference, the power supply of the PV power plant itself is not included.

In Figure 12 the results of the simulation are shown. As expected the PV power plant will be connected to the grid with a smaller difference between supply and demand. While the supply can not cover the demand for both grids, the PV power plant is reconnected from the lower to the

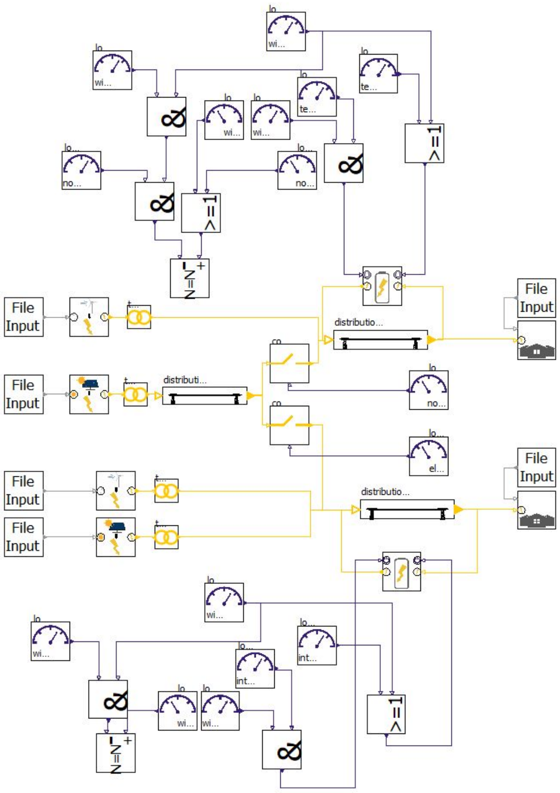


Figure 11. Example for a more complex VSS energy grid. A conditional connection can connect a solar energy park to either the upper or the lower distribution grid. The charging and discharging of the battery as well as the controlling of the number of active wind turbines is controlled like in example 1 but adjusted to work with the conditional connection.

upper grid at $t = 1\,\mathrm{h}$ because at this moment the lack of power is higher. The actual positive affects can be seen from $t = 4\,\mathrm{h}$ where for the upper grid the supply can cover the demand by itself and the lower grid which is connected to the PV power plant can only cover its demand because of the connection. Here the connection even leads to an activation of the battery charging.

It should be noted, that a clear definition of rules is very important for the success of such simulations. Inconsistencies will lead to crashes of the simulation. Especially for complex systems where multiple components affect each others activation and deactivation rules, it can get very difficult to define these rules sufficiently well.

5 Conclusion

This paper presented an approach to model VSS energy grids in Modelica based on Petri nets. While the implementation is still experimental and the technical components are simplified, it was shown that Petri nets are suitable for modeling systems with a variable structure that changes during simulation time in Modelica. To demonstrate this, two examples for energy grids with different VSS types were presented. These two examples showed that different VSS can be modeled with PNs. Both models behaved like expected due to the defined rules for state transitions.

To ensure that the used Petri nets keep properties of the flow of energy while enabling component-based models, so called energetic places and transitions were defined. Based on the PNlib Modelica library, these special Petri nets were implemented to create a library called PNRG that can be used to model VSS energy grids.

However, the capabilities of modeling VSS are limited. While activating or deactivating components is possible, adding variables, objects or components dynamically in run time is currently not possible.

6 Outlook

Although the presented examples showed that Petri nets can be used to model VSS, a lot of work has to be done for excellent energy grid modelling using Petri nets. In the first place, the realism of the implementations for different components should be increased. For example, additional effects influencing the charging and discharging powers of the batteries could be implemented. One approach could be to use existing models and libraries for these detailed calculations. Also, more parameters should be implemented to increase the realism of the modeling. Implementing parameters like voltages, phase angles and frequencies are needed to verify how stable the simulated energy grids under state transitions are.

As described in the introduction, one of the motivations for studying VSS is to simulate sector coupling systems. Therefore a logical next step would be to implement and study components like electrolysers, gas power plants or gas and heat consumers.

Also an advanced formalism to define the state transitions to avoid conflicts is an interesting subject for further investigations. Especially for the control sub-net, verification methods that are defined for Petri Nets could be studied to avoid conflicts. (Svadova and Hanzálek 2004) presents a method for generating reachability graphs for hybrid Petri nets that is also applied in (Lu et al. 2016) for the analysis of the micro grid models. Therefore, this might be one approach for an analysis tool for the here described grids.

In the end, the results from this approach need to be verified with other energy grid simulation models, which are not based on Petri nets. This comparison is important, not only to evaluate how realistic the single components work but also to validate the realism of the whole grid.

Acknowledgements

The authors would like to thank the Boysen–TU Dresden–Research Training Group for the financial and general support that has made this contribution possible. The Research Training Group is co-financed by the Friedrich and Elisabeth Boysen Foundation and the TU Dresden. The work is also partially supported by the German Research Foundation (SFB/TRR 339, Project ID453596084).

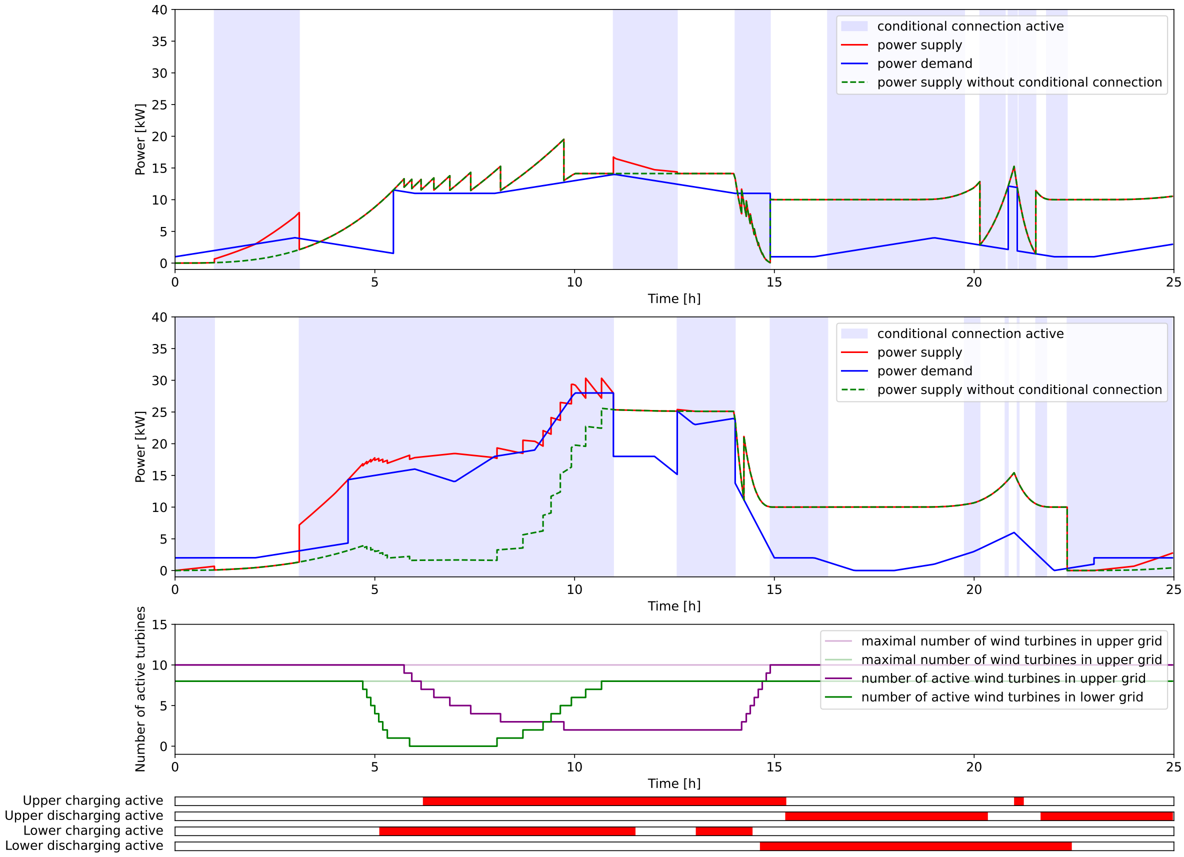


Figure 12. Simulation output for the second example. The upper plot shows the power supply and demand for the upper distribution grid from Figure 11, the second plot for the lower grid. The background is colored according to if the PV power plant is connected to the respective grid. The lower plot shows the number of active wind turbines. In the four bars at the bottom, the red color shows if the battery charging and discharging is active.

References

Agerwala, Tilak (1974). *Complete Model for Representing the Coordination of Asynchronous Processes*. The JohnsHopkins University, Baltimore, Maryland.

Alhamali, Abdulwahab et al. (2016-12). "Review of Energy Storage Systems in electric grid and their potential in distribution networks". In: pp. 546–551. DOI: 10.1109/MEPCON.2016.7836945.

Benveniste, Albert et al. (2019-10). "Multi-Mode DAE Models - Challenges, Theory and Implementation". In: *Computing and Software Science: State of the Art and Perspectives*. Vol. LNCS-10000. Lecture Notes in Computer Science. Springer, pp. 283–310. DOI: 10.1007/978-3-319-91908-9_16. URL: https://inria.hal.science/hal-02333603.

David, René and Hassane Alla (1987). "Continuous Petri nets". In: *8th European Workshop on Application and Theory of Petri Nets, Zaragoza*, pp. 275–294.

David, René and Hassane Alla (2008-07). "Discrete, Continuous, and Hybrid Petri Nets". In: *Control Systems Magazine, IEEE* 28, pp. 81–84. DOI: 10.1109/MCS.2008.920445.

Esperon, Daniel, Alexandra Mehlhase, and Thomas Karbe (2015-07). "Appending Variable-Structure to Modelica Models (WIP)". In: Proceedings of the Conference on Summer Computer Simulation 2015.

Fridgen, Gilbert et al. (2020). "A holistic view on sector coupling". In: *Energy Policy* 147, p. 111913. DOI: https://doi.org/10.1016/j.enpol.2020.111913. URL: https://www.sciencedirect.com/science/article/pii/S0301421520306248.

Heckel, Jan-Peter et al. (2022). *Investigation of dynamic interactions in integrated energy systems*. DOI: 10.15480/882.4678. URL: http://hdl.handle.net/11420/13866.

Liu, Tao et al. (2022-05). "Stability and Control of Power Grids". In: *Annual Review of Control, Robotics, and Autonomous Systems* 5. DOI: 10.1146/annurev-control-042820-011148.

Lu, Xiaoyu et al. (2016). "Hybrid Petri nets for modeling and analysis of microgrid systems". In: *IEEE/CAA Journal of Automatica Sinica* 3.4, pp. 349–356. DOI: 10.1109/JAS.2016.7510070.

Mai, Carl et al. (2018-02). "Adaptive Petri Nets -A Petri Net Extension for Reconfigurable Structures". In: Proceedings of the Tenth International Conference on Adaptive, Self-Adaptive Systems, and Applications.

Mehlhase, Alexandra (2015-06). "Konzepte für die Modellierung und Simulation strukturvariabler Modelle". PhD thesis. DOI: 10.13140/RG.2.1.3041.2007.

Murata, Tadao (1989). "Petri nets: Properties, analysis and applications". In: *Proceedings of the IEEE* 77.4, pp. 541–580. DOI: 10.1109/5.24143.

Nytsch-Geusen, Christoph et al. (2005-04). "MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics". In: Proceedings of the 4th International Modelica Conference, Hamburg, Germany.

Petri, Carl Adam (1962). *Kommunikation mit Automaten*. Technische Hochschule Darmstadt. URL: https://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/.

Proß, Sabrina and Bernhard Bachmann (2012-11). "PNlib - An Advanced Petri Net Library for Hybrid Process Modeling". In: Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany, pp. 47–56. DOI: 10.3384/ecp1207647.

Senkel, Anne et al. (2021). "Status of the TransiEnt Library : transient simulation of complex integrated energy systems". In: *14th International Modelica Conference 2021*. Linköping Electronic Conference Proceedings, pp. 187–196. URL: http://hdl.handle.net/11420/11525.

Svadova, Martina and Zdeněk Hanzálek (2004-11). "An algorithm for the evolution graph of extended hybrid Petri nets". In: vol. 5, 4905–4910 vol.5. ISBN: 0-7803-8566-7. DOI: 10.1109/ICSMC.2004.1401308.

Tinnerholm, John (2022). *A Composable and Extensible Environment for Equation-based Modeling and Simulation of Variable Structured Systems in Modelica*. Linköping UniversityLinköping University, Software and Systems, Faculty of Science Engineering. Licentiate Thesis. DOI: 10.3384/9789179293680.

Tinnerholm, John, Adrian Pop, and Martin Sjölund (2022). "A Modular, Extensible, and Modelica-Standard-Compliant OpenModelica Compiler Framework in Julia Supporting Structural Variability". In: *Electronics* 11.11. ISSN: 2079-9292. DOI: 10.3390/electronics11111772. URL: https://www.mdpi.com/2079-9292/11/11/1772.

Tinnerholm, John, Adrian Pop, Martin Sjölund, et al. (2020-11). "Towards an Open-Source Modelica Compiler in Julia". In: pp. 143–151. DOI: 10.3384/ecp2020174143.

Utkin, Vadim I. (1977). "Variable structure systems with sliding modes". In: *IEEE Transactions on Automatic Control* 22.2, pp. 212–222. DOI: 10.1109/TAC.1977.1101446.

Valk, Rüdiger (1978). "Self-modifying nets, a natural extension of Petri nets". In: *Automata, Languages and Programming*. Ed. by Giorgio Ausiello and Corrado Böhm. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 464–476. ISBN: 978-3-540-35807-7.

Wolf, Karsten (2019). "How Petri Net Theory Serves Petri Net Model Checking: A Survey". In: *Transactions on Petri Nets and Other Models of Concurrency XIV*. Ed. by Maciej Koutny, Lucia Pomello, and Lars Michael Kristensen. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 36–63. ISBN: 978-3-662-60651-3. DOI: 10.1007/978-3-662-60651-3_2. URL: https://doi.org/10.1007/978-3-662-60651-3_2.

Zimmer, Dirk (2010). *Equation-based modeling of variable-structure systems*. Swiss Federal Institute of Technology, Zürich. URL: https://doi.org/10.3929/ethz-a-006053740.

Energy Efficiency Measures for Existing Factory Buildings

Xenia Kirschstein[1] Anja Schaffarczyk[1] Miriam Schuster[1] Nadja Bishara[1]

[1]Energy Efficient Construction Unit, Institute of Structural Mechanics and Design, Technical University of Darmstadt, Germany {kirschstein,schaffarczyk,schuster,bishara}@ismd.tu-darmstadt.de

Abstract

To contribute to carbon neutrality, energy efficiency measures in existing factories must be evaluated holistically, considering not only production and technical building equipment but also the building itself. In this study, a package is introduced as part of a simulation library which aims to identify integrated energy efficiency measures. The package enables the user to simulate building related efficiency measures independently or combined with machines and technical building equipment. Special focus is placed on the efficiency measure hereafter referred to as enclosure, which designates a thermally activated construction around a number of machines to facilitate the capturing of waste heat emitted to the ambient air. A comparison with measured data shows a good agreement of the return temperature for stationary conditions. Furthermore an application example for the package is given.

Keywords: building stock, refurbishment, TABS, industrial buildings, energy system optimization

1 Introduction

The goal of the *European Green Deal* adopted in 2019 is to achieve carbon neutrality in Europe by 2050 (Bundesministerium für wirtschaftliche Zusammenarbeit und Entwicklung 2023). Buildings alone account for 40% of the global energy consumption (Nejat et al. 2015) and thereby significantly contribute to the global greenhouse gas emissions. Reducing energy consumption and using greener energy sources for the building operation is therefore crucial to achieve lower overall carbon emissions. Industrial buildings are no exception, but pose a special challenge because of the different industrial processes that take place in them. The need for the integration of building simulation into the planning procedure to assess energy saving potentials as well as non-energy benefits like thermal comfort is frequently addressed in the literature (e.g. by Gourlis and Kovacic (2017) and Bleicher et al. (2014)). Weeber, Ghisi, and Sauer (2018) propose a procedure model how to integrate energy building simulation in the assessment of energy efficiency measures in factories, using *IDA-ICE* as building simulation software. They also emphasize that although computer simulations are considered a valuable tool among architects and civil engineers, they are still underrepresented in planning processes. Smolek et al. (2018) explain this by too much complexity and too high costs of the building design tools for the purpose of industrial energy management. They address this problem by developing a building model consisting of manageable *cubes*.

Maier et al. (2021) describe the holistic approach followed by the *ETA factory* in Darmstadt, where building, production equipment and building services are interlinked mainly by thermally activated building structures (TABS). Low temperature waste heat captured by TABS can then for example be lifted to a usable temperature level using a heat pump (as e.g. described by Ramschak et al. (2018)).

As part of the research project *ETA im Bestand*, the simulation library *ThermalIntegrationLibrary* (TIL) has been developed for the identification and evaluation of energy efficiency measures in existing factories and was described by Theisinger et al. (2023). A special focus of this library is the interconnection of building, building equipment and machines. As *Modelica* is a popular tool for hydraulic grids and sophisticated control strategies (described e.g. by O'Donovan, Falay, and Leusbrock (2018)), it was chosen as a modelling language for all subsystems. Thereby time-consuming co-simulation and the usage of commercial building simulation software could be avoided.

Within the TIL the *FactoryBuildings* package aims to provide the user with easy-to-use basic building models as well as example factory buildings assembled from data collected within the research project. These example models can be dragged and dropped to be simulated along with the machine models as described by Theisinger et al. (2023). The *FactoryBuildings* package is intended to create a link between the available information of an existing building, the experience of the person responsible for the building, and the often time intensive model set up and simulation in *Modelica* by facilitating the modeling to such an extent that the barrier for use is minimized and a plausibility check with experience is possible.

The structure of the package is presented in Section 2.1. The correct implementation of the enclosure model introduced in Section 2.1 is validated using measurement data obtained from an experiment with a small demonstrator in Section 2.2. Furthermore an application example of the package is given, applying a few simple efficiency measures to an exemplary production hall (Section 2.3). The respective results are presented in Section 3.

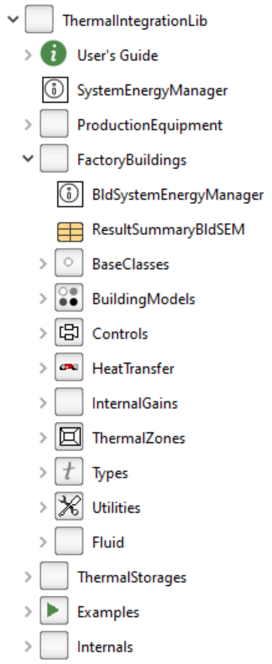
DOI
10.3384/ecp204737

2 Methods

2.1 The FactoryBuildings package

The *FactoryBuildings* package relies to a large extent on the well known *Buildings* library described by Wetter et al. (2014) in addition to the *ModelicaStandardLibrary* (Modelica Association and contributors 2020), both in terms of models and structure of the package. The package browser is shown in Figure 1 and contains the following sub-packages:

- *BaseClasses*: Contains the framework for the use of the building model within the factory system model described by Theisinger et al. (2023).
- *BuildingModels*: Contains examplary models consisting of one or more thermal zones ready to use in a factory model along with the machines. Furthermore examples are provided to evaluate building related energy efficiency measures including the machine waste heat as load profiles only.
- *Controls*: Contains a semi-ideal heater to facilitate simulating heating and cooling loads using a range of set temperatures over a whole year, described in Detail below.
- *HeatTransfer*: Contains data as known from the *Buildings* library.
- *InternalGains*: Contains models to facilitate the input of boundary conditions regarding heat sources and sinks used in *PartialBuildingModel* (described below).
- *ThermalZones*: Contains the base class *PartialBuildingModel* which makes use of the *Buildings.ThermalZones.Detailed.MixedAir* model, extended by *ProductionHall*, which facilitates modelling thermal zones like production halls and office spaces, and *Enclosure*, which can be used to model room-in-room concepts. The *ProductionHall* and *Enclosure* Model are described in more detail below. Furthermore modelling examples and examplary industrial building models are provided.
- *Types*: Contains types like weekdays for convenience.
- *Utilities*: Contains utilities used throughout the library.



- ThermalIntegrationLib
  - User's Guide
  - SystemEnergyManager
  - ProductionEquipment
  - FactoryBuildings
    - BldSystemEnergyManager
    - ResultSummaryBldSEM
    - BaseClasses
    - BuildingModels
    - Controls
    - HeatTransfer
    - InternalGains
    - ThermalZones
    - Types
    - Utilities
    - Fluid
  - ThermalStorages
  - Examples
  - Internals

Figure 1. TIL package browser

The *ProductionHall* model consists of the *MixedAir* model as well as of a collection of different convenient submodels for ventilation, infiltration, schedule based internal gains, thermal bridges and various for clarity conditionally generated connectors, which facilitate a coupling with technical building equipment. A simplified room temperature control allows for a quick generation of useful heating and cooling loads. Furthermore the inclusion of efficiency measures like an enclosure or TABS is facilitated by conditionally generated submodels and parameters reduced to the essentials. For the mapping of TABS the *Buildings* model *ParallelCircuitsSlab* is integrated. The *Enclosure* model is based on the same principle as *ProductionHall* but does not allow external walls and uses temperature inputs rather than a weather bus. It should be set up by connecting its room-facing constructions to corresponding surfaces of the surrounding room in order to allow the radiation calculation.

As the time step of the simulation is variable and at the same time the thermal zone should not be thermally conditioned within a certain temperature range, the implementation of an ideal heater/ cooler is not trivial. Our solution is described in the following: The semi-ideal heater model checks whether the current room temperature falls below or exceeds a certain minimum and maximum threshold. The error between the current value and the threshold is fed into a first order function if the current temperature is not between the minimum and maximum limit. A gain constant with a standard value of $V \cdot 1.3\ \frac{\mathrm{kg}}{\mathrm{m}^3} \cdot 1005\ \frac{\mathrm{J}}{\mathrm{kg{\cdot}K}}$, where V denotes the zone volume, may be specified by the user to be multiplied with the error (in K). A corresponding heat flow is then prescribed to the heat port respecting a specified minimum and maximum power for heating and cooling. The graphics view of the model is shown in Figure 2. It may be advisable to adjust the gain constant depending on how the zone is equipped, which is why we recommend to check the results of the air temperature before proceeding.

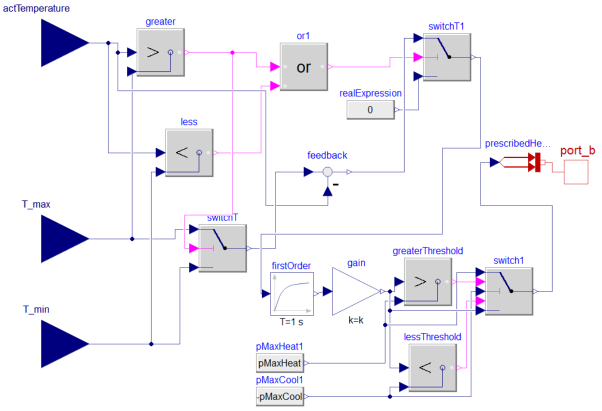


Figure 2. Graphics view of the semi-ideal heater model

2.2 Validation of the enclosure model

Measured data obtained from an experiment with a small demonstrator (1.82 m x 0.4 m x 1 m) is used to validate the correct implementation of the enclosure model. The experimental setup is shown in Figure 3.

The envelope of the demonstrator consists of vacuum insulated micro-reinforced high-performance concrete (mrUHPC) embedded in a steel frame. The respective properties are summarized in table 1.

Table 1. Properties of the construction layers

Property	Concrete	Vacuum insulation
Layer thickness [m]	0.05	0.02
Thermal conductivity $[\frac{\mathrm{W}}{\mathrm{m{\cdot}K}}]$	5 *	0.007
Specific heat capacity $[\frac{\mathrm{J}}{\mathrm{kg{\cdot}K}}]$	1100	900
Density $[\frac{\mathrm{kg}}{\mathrm{m}^3}]$	2500	195

* Hauser (2016)

The four walls are thermally activated each using a capillary tube mat located in the centre of the concrete layer. The connections are arranged according to the Tichelmann system so that supply and return lines of each mat sum up to about the same length, the two shorter mats are connected in series. The dimensions of the cross-linked polyethylene (PEX) capillary tubes are 3.5 x 0.5 mm arranged in parallel at a distance of 10 mm between one collector pipe each for supply and return. The supply and return pipes outside the demonstrator (about 4 m each) are non-insulated.

When the measurement is started the air temperature in the demonstrator is approx. 13.8 °C. At this time, the water is not yet circulating through the system. 28 minutes after the start of the measurement the pump supplying the TABS is turned on (total volume flow rate of 0.19 $\frac{\mathrm{m}^3}{\mathrm{h}}$, supply temperature of approx. 9.7 to 10.1 °C). The water temperatures are measured with PT1000 sensors with a tolerance of ± 0.3 K at 21 °C. The surface temperatures are measured with NTC sensors with a tolerance of ± 0.2 K at 0 to 70 °C. The ambient temperature during the experiment is approx. 21.4 °C.

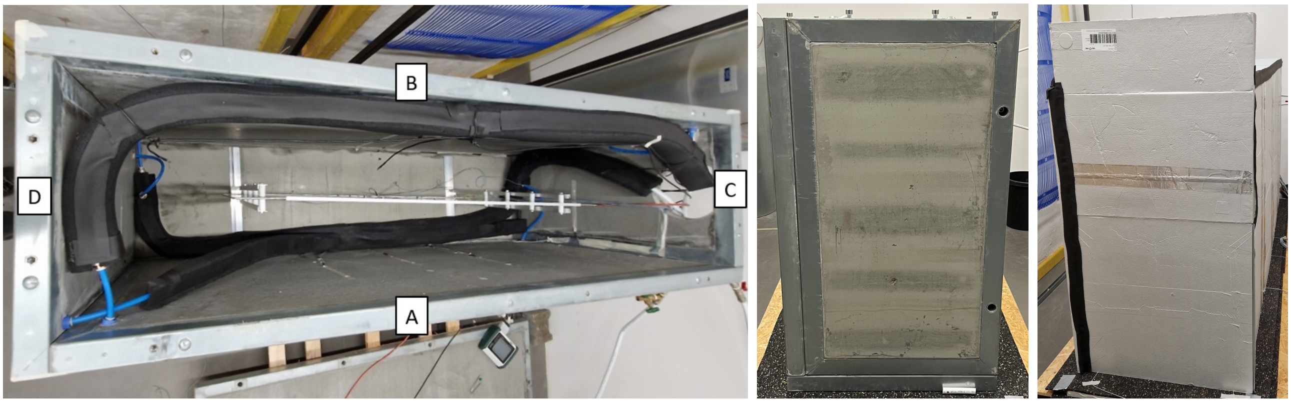
The model is set up in *Dymola*. The inlet temperature, mass flow rate and surrounding temperature are read in from a file containing measured data over 2.5 hours. The demonstrator is modelled using the *Enclosure* model with TABS. Inside the demonstrator, the connecting insulated pipe between the short wall TABS is considered as well as the longer supply or return pipe of each wall respectively. To account for the thermal losses of the non-insulated pipes outside the demonstrator, a series of experiments is conducted and the heat transfer coefficient is fitted according to the measured temperature difference to 15 $\frac{\mathrm{W}}{\mathrm{m}^2{\cdot}\mathrm{K}}$. Furthermore an air exchange by infiltration of 0.15 h^{-1} and a heat flow rate of 0.05 $\frac{\mathrm{W}}{\mathrm{m}^2{\cdot}\mathrm{K}}$ by thermal bridges to account for the steel frame are assumed. The model is shown in Figure 4. The results are described in Section 3.1.

2.3 Evaluation of energy efficiency measures

Besides the application of the building model as a production environment the *FactoryBuildings* package also enables the user to evaluate building related energy efficiency measures without the machine models. Including the machine waste heat as load profiles offers the advantage of a reduced computational effort, which facilitates the simulation over a whole year. The TIL thereby offers a solution for the relevant time frames for both production and building efficiency measures.

For the simulation of the building related efficiency measures the same models as for the factory simulation can be used. The *BldSystemEnergyManager* model extending the *SystemEnergyManager*, which centrally evaluates the energy demands of the factory components (as described by Theisinger et al. (2023)), can be used to compare a reference building with a refurbished building version.

To illustrate the application described above, an exemplary production hall is equipped with efficiency measures. The production hall was constructed 1994 and has a rectangular base area of about 7800 m^2. The external walls consist of sandwich- and cassette constructions with 8 cm PUR foam resp. 12 cm mineral wool. The flat roof is insulated with 12 cm PIR and the floor is a 20 cm thick concrete slab. The windows are double glazed. Furthermore air change rates of 0.15 h^{-1} by infiltration and of 1.5 h^{-1} by ventilation are assumed and weather data of Frankfurt, Germany is used, while the ground temperature oscillates between 0 and 20 °C. As internal gains 130 people with a heat output of 75 W each are considered along with lighting of 15 $\frac{\mathrm{W}}{\mathrm{m}^2}$ at an efficiency of 12%. 50 machines with 10 kW waste heat output to the room air (taken from an exemplary machine within the research project) are operated from 8:00 till 17:00. The minimum and maximum set temperatures are specified as 19 and 25 °C when the space is occupied resp. 16 and 30 °C when the space is not occupied.



(a) Demonstrator without ceiling element with designation of the walls **(b)** Uninsulated **(c)** Insulated

Figure 3. Experimental setup

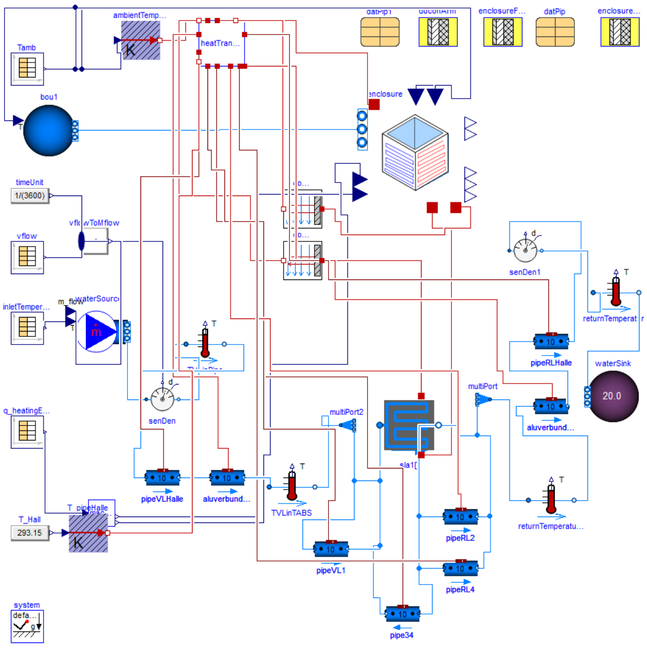


Figure 4. Graphics view of the *Dymola* model of the experimental setup

To improve the energy efficiency of the industry hall, firstly a retrofit insulation of the floor is considered, as the energy demand of the hall is heating dominated. This could be an option for the owners in the course of a major restructuring. Secondly modern lighting is integrated with 15 $\frac{\mathrm{W}}{\mathrm{m}^2}$ at an efficiency of 40%. Furthermore a 500 m^2 PV plant with 100 kWp is integrated. The integration of renewables is not an efficiency measure in the classical sense, but should still be considered when changes of the building envelope and technical building equipment are planned. The two versions of the model are dragged and dropped in the model containing the *BldSystemEnergyManager*. For the evaluation, the user may specify energy prices and and $\mathrm{CO_2}$ factors. The results are described in section 3.2.

3 Results

3.1 Validation of the enclosure model

In Figures 5 and 6 the simulated and measured return temperatures and inner wall surface temperatures of the enclosure described in Section 2.2 are compared. While in the transient cool down of the construction at the beginning of the experiment the simulated and measured return temperatures differ up to 1.4 K, the difference decreases to 0.2 K while steady state is approached. This behaviour can be observed using the epsilon-NTU as well as the finite difference method implemented in the *Buildings* radiant slab model for the heat transfer between fluid and slab, and corresponds to its limitation to steady state applications as described in the user's guide (Wetter et al. 2014). The rise in the measured return temperature before the pump starts is due to the fact that the fluid temperature in the pipes where the sensor is located is initially not in equilibrium with the ambient air temperature. The jump in the simulated return temperature at the beginning of the experiment is due to the initial temperatures of the water volumes in the various pipes which differs from the respective room temperatures and could only be modelled approximately.

The surface temperature sensors are located approx. in the centre of the inner wall surfaces respectively. As shown in Figure 6, the measured and simulated surface temperatures differ up to 0.6 K two hours after the pump is switched on and even more at the beginning of the cool down. While in the model the heat transfer seems to be overestimated at the beginning of the cool down for the two larger walls and underestimated when steady state is approached, the opposite seems to be true for the two smaller walls. Apart from the above mentioned restrictions of the radiant slab model, the two main reasons for the deviation are firstly that the whole air volume is assumed to be completely mixed. This is not the case in reality, where an uneven temperature distribution in the air and at the construction surfaces can be observed. Secondly, the simulated values are to be understood as an av-

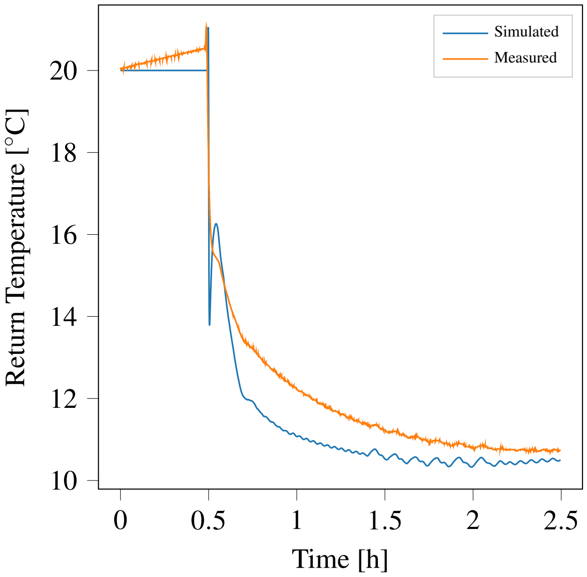


Figure 5. Simulated and measured return temperatures

erage over the whole surface area while the sensor measures the temperature at one point.

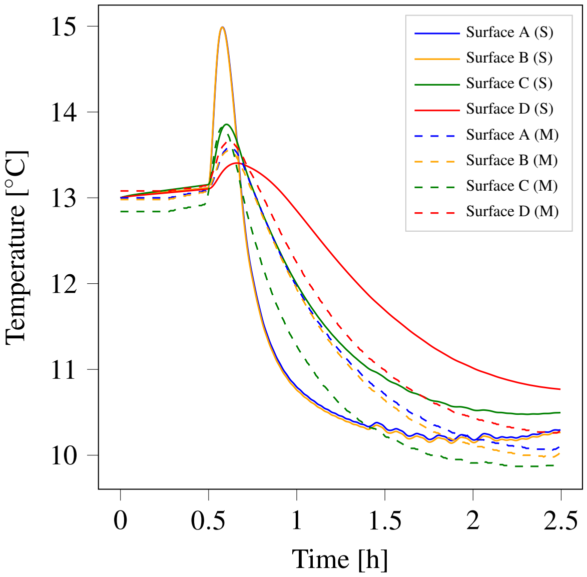


Figure 6. Simulated (S) and measured (M) inside surface temperatures

The results show that the *Buildings* model *ParallelCircuitsSlab* is reasonably well suited for an estimation of the behaviour of the considered type of capillary tube mats within the *Enclosure* model, e.g. for the capturing of temporally constant machine waste heat, keeping in mind the depicted limitations.

3.2 Evaluation of energy efficiency measures

Using a script with convenient plot settings provided in the TIL, an overview of the annual heating, cooling, electrical and total energy demands, energy costs and greenhouse gas emissions is given based on the user inputs. In this case, the insulation of the floor leads to a reduction of the heating demand by 80%, but at the same time to an increase of the cooling demand by 350%. Therefore an increased ventilation of additional 1.5 h^{-1} is introduced when the room temperature exceeds a maximum value of 24 °C as well as the outside temperature. However, the effect is not high, as the heat capacity of the constructions is very low. The electricity costs and emissions decrease noticeably, but the relative effect on the overall energy costs and emissions is not high. It can be safely assumed that this would not be the case taking into account the electricity consumption of the machines. The results for the comparison of the total energy costs and emissions is shown in Figure 7. The overall costs and emissions are more than halved and more evenly distributed throughout the year in the refurbished building, due to the above described reasons.

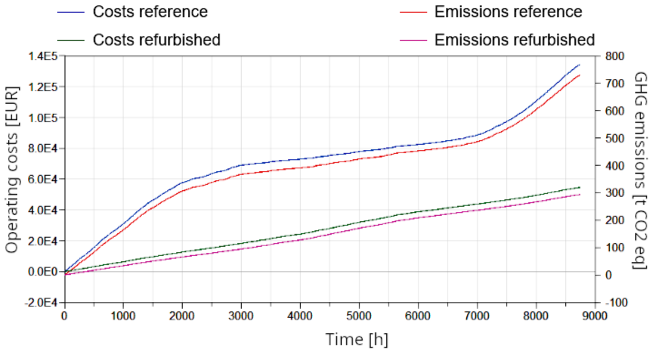


Figure 7. Results for the energy efficiency measures in *Dymola*

The assumptions made in this example are not representative for industrial buildings in general and are merely meant to depict the application of the *FactoryBuildings* package. As the utilization and boundary conditions for different industrial buildings are manifold, it is important for the user to make suitable assumptions for their specific case.

4 Conclusions and Outlook

The presented package enables the user to identify and evaluate energy efficiency measures especially in factory buildings. It relies largely on the *Buildings* library, but provides a simplified user experience by adding some functionalities and restricting the user's choice to a necessary minimum. The building model can either be simulated with simplified assumptions for machine waste heat to lay the focus on the building related efficiency measures, or act as an environment for the production equipment, typically simulated in a shorter time frame.

To propose and analyze more sophisticated energy efficiency measures than the ones presented in Section 2.3, it is important to have some knowledge about boundary conditions like the type of production equipment and details about the technical building equipment. Therefore, in future work a case study with a real factory will be performed where all important parameters are known a priori and customized efficiency measures can be evaluated using the *FactoryBuildings* package. The application of the package as part of the *ThermalIntegrationLibrary* is intended to serve as a starting point for more in-depth energy system analyses.

Acknowledgements

The authors gratefully acknowledge the financial support of the project ETA im Bestand (grant agreement No. 03EN2048A) which is funded by the Federal Ministry for Economic Affairs and Climate Action (BMWK) and managed by the management agency Project Management Jülich (PTJ).

The structure of the base model as interface to the rest of the *ThermalIntegrationLibrary* has been developed by LTX Simulation GmbH.

Furthermore the authors would like to thank Michael Frank, Yang Xue, Daniel Jiang and Aditya Hiremath for their support.

References

Bleicher, Friedrich et al. (2014-01). “Co-simulation environment for optimizing energy efficiency in production systems”. In: *CIRP Annals* 63.1, pp. 441–444. ISSN: 0007-8506. DOI: 10.1016/j.cirp.2014.03.122. URL: https://www.sciencedirect.com/science/article/pii/S0007850614001255 (visited on 2023-08-15).

Bundesministerium für wirtschaftliche Zusammenarbeit und Entwicklung (2023). *European Green Deal*. URL: https://www.bmz.de/de/service/lexikon/european-green-deal-118284.

Gourlis, Georgios and Iva Kovacic (2017-02). “Building Information Modelling for analysis of energy efficient industrial buildings – A case study”. In: *Renewable and Sustainable Energy Reviews* 68, pp. 953–963. ISSN: 1364-0321. DOI: 10.1016/j.rser.2016.02.009. URL: https://www.sciencedirect.com/science/article/pii/S1364032116002173 (visited on 2023-08-15).

Hauser, Stephan (2016). “DUCON® – Mikrobewehrter Hochleistungsbeton”. In: *Beton Kalender 2017*. John Wiley Sons, Ltd. Chap. IX, pp. 491–517. ISBN: 9783433606803. DOI: https://doi.org/10.1002/9783433606803.ch9.

Maier, Andreas et al. (2021). “Energieeffizienz weitergedacht: die thermisch aktivierte Gebäudehülle als Teil einer Prozesskette”. In: *Mauerwerk* 25.4, pp. 156–168. ISSN: 1437-1022. DOI: 10.1002/dama.202110016.

Modelica Association and contributors (2020). “Modelica Standard Library 4.0.0”. In: URL: https://github.com/modelica/ModelicaStandardLibrary.

Nejat, Payam et al. (2015). “A global review of energy consumption, CO 2 emissions and policy in the residential sector (with an overview of the top ten CO 2 emitting countries)”. In: *Renewable and Sustainable Energy Reviews* 43, pp. 843–862. ISSN: 1364-0321. DOI: 10.1016/j.rser.2014.11.066. URL: https://www.sciencedirect.com/science/article/pii/S1364032114010053.

O’Donovan, Keith, Basak Falay, and Ingo Leusbrock (2018). “Renewables, storage, intelligent control: how to address complexity and dynamics in smart district heating systems?” In: *16th International Symposium on District Heating and Cooling, Dhc2018*. Ed. by I. Weidlich. Vol. 149. Amsterdam: Elsevier Science Bv, pp. 529–538. DOI: 10.1016/j.egypro.2018.08.217.

Ramschak, Thomas et al. (2018). “High solar fraction by thermally activated components”. In: *Proceedings of the Ises Eurosun 2018 Conference - 12th International Conference on Solar Energy for Buildings and Industry*. Ed. by A. Haberle. Freiburg: Intl Solar Energy Soc, pp. 133–139. ISBN: 9783982040806. DOI: 10.18086/eurosun2018.06.11.

Smolek, Peter et al. (2018-06). “Hybrid Building Performance Simulation Models for Industrial Energy Efficiency Applications”. In: *[Journal of Sustainable Development of Energy, Water and Environment Systems]* [6].[2], [381]–[393].

Theisinger, Lukas et al. (2023-03). “Energieeffizienzmaßnahmen im Industriebestand: ETA im Bestand: Modellbasierte Identifikation und Bewertung von Energieeffizienzmaßnahmen in der metallverarbeitenden Industrie”. In: *Zeitschrift für wirtschaftlichen Fabrikbetrieb* 118.3, pp. 127–132. ISSN: 2511-0896. DOI: 10.1515/zwf-2023-1033.

Weeber, Max, Enedir Ghisi, and Alexander Sauer (2018-01). “Applying Energy Building Simulation in the Assessment of Energy Efficiency Measures in Factories”. In: *Procedia CIRP*. 25th CIRP Life Cycle Engineering (LCE) Conference, 30 April – 2 May 2018, Copenhagen, Denmark 69, pp. 336–341. ISSN: 2212-8271. DOI: 10.1016/j.procir.2017.11.148.

Wetter, Michael et al. (2014). “Modelica Buildings library”. In: *Journal of Building Performance Simulation* 7.4, pp. 253–270. DOI: 10.1080/19401493.2013.765506.

Hybrid Power Systems Simulation and Optimization Utilizing SSP and FMI

Dai Araki[1] Magnus Sandell[2]

[1]Toshiba Digital Solutions Corporation, Japan, dai.araki@toshiba.co.jp

[2]Toshiba Europe Ltd., UK, Magnus.Sandell@toshiba-bril.com

Abstract

Collaborative model-based development of the hybrid power system often requires large-scale co-simulation and system parameter optimization. In this study, we investigate an architecture for parallel processing simulation of SSP (System Structure and Parametrization) and FMI (Functional Mock-up Interface), which enables high-speed computation by multi-core distribution. We combine Bayesian optimization and co-simulation, then we build a collaborative development platform for hybrid power systems design. We report performance experiments using hybrid electric vehicle simulation model published by JAMBE (Japan Automotive Model-Based Engineering center).

Keywords: Model exchange, FMI, SSP, Distributed co-simulation

1 Introduction

Hybrid power systems are a combination of power generation and energy storage systems (batteries and fuel cells). As shown in Figure 1, a hybrid power system consists of systems for both power supply and power demand stakeholders, and each stakeholder has its own methods and tools for system design and analysis. Hybrid electric vehicles (HEVs) also consist of various components such as engines, electric motors, DC-DC converters, DC-AC inverters, and batteries, and OEMs and suppliers work together for system design and analysis.

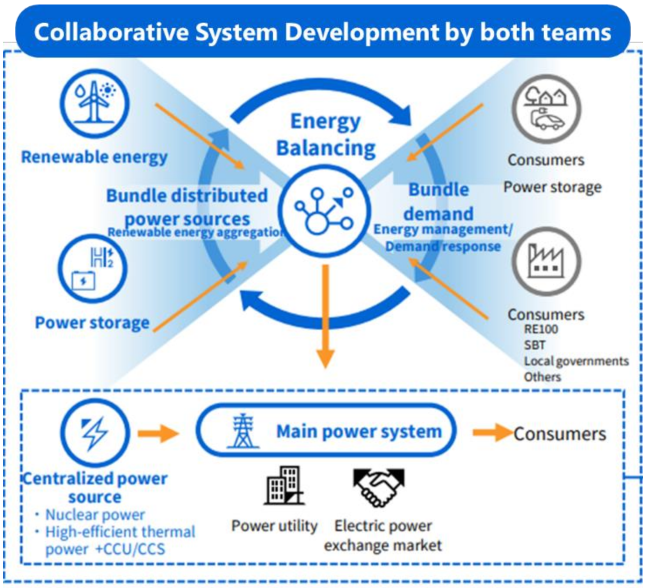

Collaborative System Development by both teams
Renewable energy
Energy Balancing
Consumers
Power storage
Bundle distributed power sources
Renewable energy aggregation
Bundle demand
Energy management/ Demand response
Power storage
Consumers
RE100
SBT
Local governments
Others
Main power system
Consumers
Centralized power source
・Nuclear power
・High-efficient thermal power +CCU/CCS
Power utility
Electric power exchange market

Figure 1. DX in Hybrid Power Supply Systems Design.

This means that system designing should consider different perspectives, which requires the use of several different tools and methodologies. However, connecting tools and exchanging information between different teams often leads to inefficiencies in system development. Since connecting tools from different vendors is not guaranteed to work and not supported by each vendor, it is necessary for the tool users to build and maintain their own environment for connecting tools.

For these reasons, interoperability standards for model exchanging between tools are important. FMI (Functional Mock-up Interface) and SSP (System Structure and Parametrization) are standardized by Modelica Association to establish interoperability between tools for model exchanging at various levels of abstraction.

SmartSE project of prostep ivip Association has published “SmartSE Recommendation” (SmartSE 2023) for simulation-based system design and decision making in collaborative development between multiple design teams across companies based on the utilization of FMI and SSP standard.

1.1 FMI (Functional Mock-up Interface)

FMI (Functional Mock-up Interface) is common interface and file format that allows simulation models to be passed between different tools. FMI standard is available on the official website (ref. FMI website). There are many tools over hundreds that support the FMI standard, and models can be exchanged between these tools.

FMU (Functional Mock-up Unit), which is zipped compressed file format, contains modelDescription.xml file in XML document format and a library file in binary format (DLL on Windows systems, SO on Linux systems) that implements the model's definition expressions or solver functions.

The format of modelDescription.xml is defined by the standard, and it contains information such as the names, types, and other attributes of the input and output signals of the model stored in the FMU, and a list of parameters that can be set and changed from outside the FMU.

There are two types of FMI standards “Model Exchange” interface and “Co-simulation” interface. Model Exchange FMU contains only the model equations.

Co-simulation FMU includes both model equations and solver.

Model Exchange interface assumes that the model equations in multiple FMUs are aggregated and computed by a single solver. This makes it difficult to speed up the calculation by distributed parallelization, and it is difficult to guarantee consistency between the results of a single solver and those of distributed parallel computation. The reason is that each individual Co-simulation FMU contains a model and a solver, so each FMU can be computed independently.

1.2 SSP (System Structure and Parametrization)

SSP (System Structure and Parametrization) is standard format for describing model and signal connection structures and other parameters necessary to conduct multi-domain co-simulation combining multiple FMUs. Specification of SSP can be obtained from the official SSP website (ref. SSP website).

Like FMU, SSP is a compressed file in zip format, and its interior consists of several sub-formats and its interior consists of several sub-formats. SSD (System Structure Description) is an XML file that describes the hierarchical structure, connection relationships, and functional structure of the entire FMU network.

1.3 Distributed co-simulation standards

In collaborative development through model exchange, there are many opportunities for large-scale simulation that combine model components created by multiple teams. Most commercial simulation tools calculate models sequentially, so when model parts are exchanged between companies or design teams using FMI and SSP, the simulation speed decreases as the number of FMUs for model parts increases.

Distributed co-simulation is expected to speed up large-scale simulations by running models in parallel. Distributed co-simulation may be run in a multi-core distribution within a single machine, on a set of machines interconnected via a local area network, or on globally distributed computers communicating via the Internet.

Typical distributed co-simulation standards include IEEE 1516 HLA and DSP.

IEEE 1516 HLA (High Level Architecture) (IEEE 1516) standardizes distributed co-simulation and standard interface for connecting multiple heterogeneous simulators. HLA defines controller called RTI (Run-time Infrastructure) which provides services such as data distribution and time synchronization among multiple connected simulators (called federates in HLA). Through this RTI, simulators are combined in a star-like network configuration for distributed simulation.

DCP (Distributed Co-simulation Protocol) (ref. DCP website) is a new protocol developed for the purpose of connecting real-time systems (HILS or prototype machines) and simulators. DCP protocol has two modes: real-time mode that connects a real-time system (actual device) and a simulator, and non-real-time mode that is used to connect virtual simulators. In both modes, the role of controller (equivalent to RTI in HLA standard) is limited compared to HLA. For data communication, each simulator node communicates directly with the other simulator node on a point-to-point basis. DCP protocol configures mesh-type network for distributed simulation. In the real-time mode of DCP protocol, each node runs using its own timer, so there is no explicit synchronization of time between nodes. In the non-real-time mode, the controller sends clock signal to each node, and the simulator on each node runs explicit synchronized to the clock signal.

1.4 Contributions of this paper

In this paper, we design a multi-core distributed simulator which performs high speed co-simulations that connect many model parts (FMUs). Next, we combine Bayesian optimization and distributed co-simulation to create a toolset that can automatically perform parameter optimization of hybrid power systems design.

2 Computation of distributed co-simulation

This section considers distributed co-simulation for SSP and FMI that can perform parallel computation with multi-core distribution.

The upper part of Figure 2 shows the sequence of execution of the calculations of each simulator connected to the distributed simulation, the time synchronization for distributed simulation, and the data distribution. The lower part of Figure 2 shows the case of simulation by sequential computation without distributed computation as a comparison.

In the upper part of Figure 2, each of the three simulators computes independently and in parallel until the logical time set as the synchronization timing (coupling point), at which point the simulators exchange output values with the other simulators. After that, the three simulators again perform calculations in parallel and exchange output values with the other simulators at the next synchronization timing. This process is repeated. This figure shows a simple case in which the communication step size of the three simulators is the same and does not vary, but there may be other cases in which the synchronization period varies, or in which each simulator has a different synchronization period or synchronization timing.

In contrast, in the case of sequential calculations as in the lower part of Figure 2, which do not involve distributed calculations, the three calculations are repeated in sequence using a set of simulators. Most commercial simulation tools are considered to be sequential.

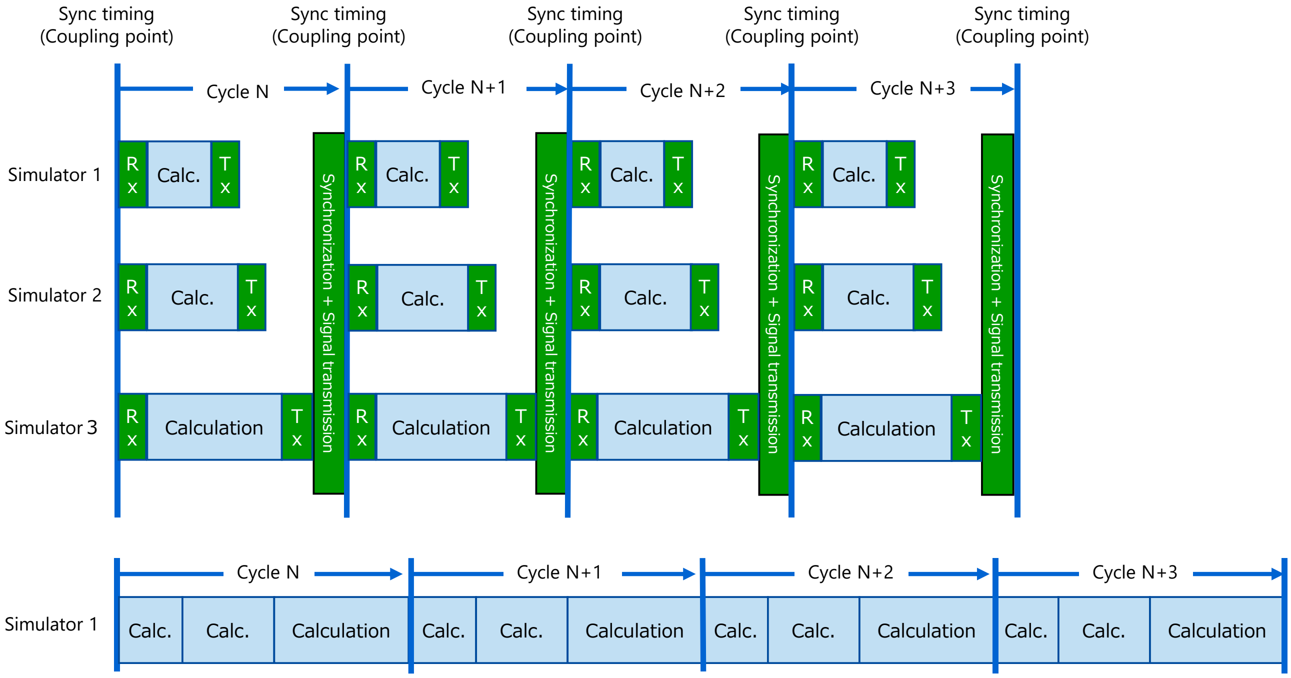


Figure 2. Comparison of computation between distributed co-simulation and sequential co-simulation.

Comparing distributed and sequential computation cases, the distributed simulator is expected to increase simulation speed compared to the sequential case. However, the simulation speed does not increase linearly with the degree of parallelism. In the case of distributed computation, there is the overhead of synchronization and signal transmission, so it is important how lightly this process can be made and how much the computation time can be shortened. It should also be assumed that the computational complexity of each model component is not uniform, so even if distributed computation is used, there is a tendency for the speed to decrease when there is a model with a large computational complexity.

In Figure 2, all simulators synchronize at the same period, but in general, each simulator may synchronize at a different period. DSP described in section 1.3 is suitable for distributed co-simulations that synchronize at a single cycle, but not for distributed simulations that synchronize at different cycles, because it synchronizes by distributing clock signals from the controller node to the slaves. On the other hand, IEEE 1516 HLA is suitable for both single-period and multi-period synchronization because it schedules slave nodes by supervising the global time in the controller node. Therefore, this paper adopts the IEEE 1516 HLA mechanism.

3 Distributed SSP-FMI co-simulation

3.1 Architecture of distributed simulator

This section shows the architecture of SSP-FMI simulator with distributed computation shown in Figure 3. SSP-FMI simulator was developed using distributed co-simulation platform VenetDCP from Toshiba Digital Solutions (ref. VeneDCP website).

"FMI Executable" loads and executes Co-simulation 2.0 interface FMU file. FMI standard defines the function APIs used to initialize the FMU and execute model computation. These functions are stored in DLL binary library file in the FMU. "FMI Executable" unzips FMU file, obtains information of input/output signals and parameters from modelDescription.xml file, and calls the function APIs in DLL binary library file to drive and simulate the imported FMU. In co-simulation where multiple FMUs are running, multiple FMI executables are launched for individual FMUs to achieve parallel distributed computation.

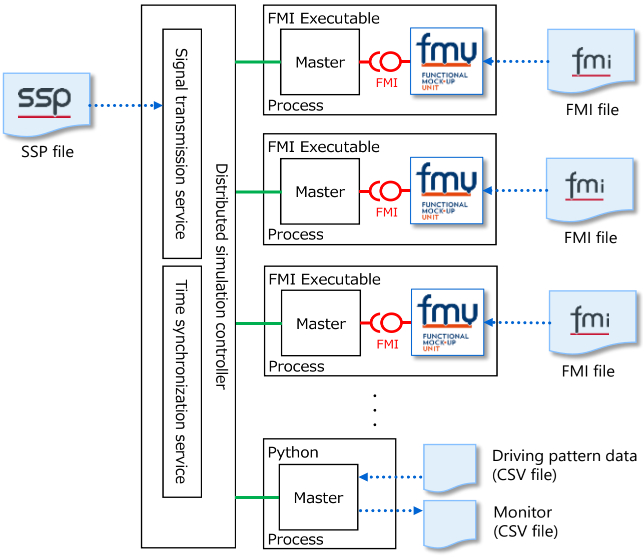
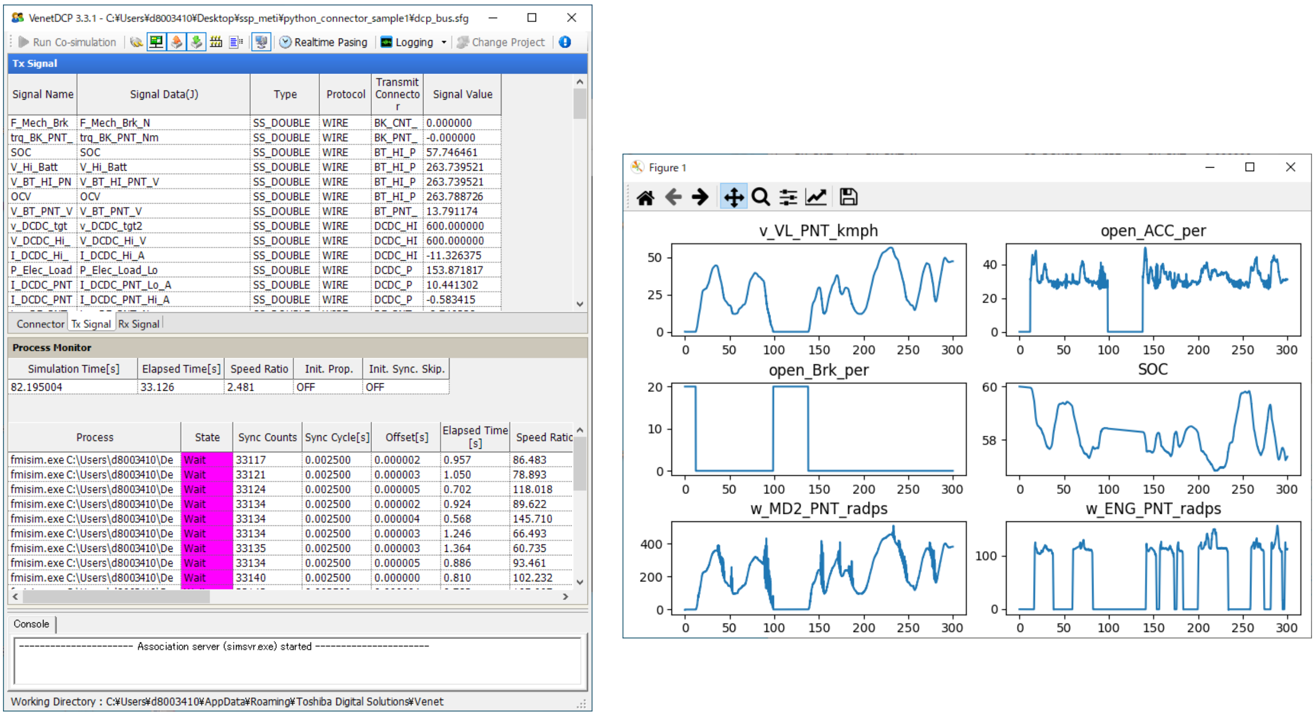


Figure 3. Architecture of distributed SSP & FMI simulator.

Recording and monitoring of the test data time series signal input and output signal time series are performed using Python.



VenetDCP 3.3.1 - C:¥Users¥d8003410¥Desktop¥ssp_meti¥python_connector_sample1¥dcp_bus.sfg

Tx Signal

Signal Name	Signal Data(J)	Type	Protocol	Transmit Connector	Signal Value
F_Mech_Brk	F_Mech_Brk_N	SS_DOUBLE	WIRE	BK_CNT_	0.000000
trq_BK_PNT_	trq_BK_PNT_Nm	SS_DOUBLE	WIRE	BK_PNT_	-0.000000
SOC	SOC	SS_DOUBLE	WIRE	BT_HI_P	57.746461
V_Hi_Batt	V_Hi_Batt	SS_DOUBLE	WIRE	BT_HI_P	263.739521
V_BT_HI_PN	V_BT_HI_PNT_V	SS_DOUBLE	WIRE	BT_HI_P	263.739521
OCV	OCV	SS_DOUBLE	WIRE	BT_HI_P	263.788726
V_BT_PNT_V	V_BT_PNT_V	SS_DOUBLE	WIRE	BT_PNT_	13.791174
v_DCDC_tgt	v_DCDC_tgt2	SS_DOUBLE	WIRE	DCDC_HI	600.000000
V_DCDC_Hi_	V_DCDC_Hi_V	SS_DOUBLE	WIRE	DCDC_HI	600.000000
I_DCDC_Hi_	I_DCDC_Hi_A	SS_DOUBLE	WIRE	DCDC_HI	-11.326375
P_Elec_Load	P_Elec_Load_Lo	SS_DOUBLE	WIRE	DCDC_P	153.871817
I_DCDC_PNT	I_DCDC_PNT_Lo_A	SS_DOUBLE	WIRE	DCDC_P	10.441302
I_DCDC_PNT	I_DCDC_PNT_Hi_A	SS_DOUBLE	WIRE	DCDC_P	-0.583415

Process Monitor

Simulation Time[s]	Elapsed Time[s]	Speed Ratio	Init. Prop.	Init. Sync. Skip.
82.195004	33.126	2.481	OFF	OFF

Process	State	Sync Counts	Sync Cycle[s]	Offset[s]	Elapsed Time [s]	Speed Ratio
fmisim.exe C:\Users\d8003410\De	Wait	33117	0.002500	0.000002	0.957	86.483
fmisim.exe C:\Users\d8003410\De	Wait	33121	0.002500	0.000003	1.050	78.893
fmisim.exe C:\Users\d8003410\De	Wait	33124	0.002500	0.000005	0.702	118.018
fmisim.exe C:\Users\d8003410\De	Wait	33134	0.002500	0.000002	0.924	89.622
fmisim.exe C:\Users\d8003410\De	Wait	33134	0.002500	0.000004	0.568	145.710
fmisim.exe C:\Users\d8003410\De	Wait	33134	0.002500	0.000003	1.246	66.493
fmisim.exe C:\Users\d8003410\De	Wait	33135	0.002500	0.000003	1.364	60.735
fmisim.exe C:\Users\d8003410\De	Wait	33134	0.002500	0.000005	0.886	93.461
fmisim.exe C:\Users\d8003410\De	Wait	33140	0.002500	0.000000	0.810	102.232

Console

```
--------------------- Association server (simsvr.exe) started ---------------------
```

Working Directory : C:¥Users¥d8003410¥AppData¥Roaming¥Toshiba Digital Solutions¥Venet

Figure 4. Example of SSP-FMI Simulation (series parallel hybrid vehicle model).

"Distributed simulation controller" provides the signal transmission service and the time synchronization service between FMI Executables and Python.

This mechanism corresponds to the RTI (Run-time Infrastructure) of the IEEE1516 HLA. The signal exchange service also reads SSP file and sets up the signal connection relationship between FMUs and parameters.

Since "FMI Executable," "Python," and "Distributed simulation controller" are independent processes, each process will be distributed across multiple CPU cores when run on a multi-core CPU machine. The number of processors and CPU core allocation can be changed using the processor affinity option in Microsoft Windows. Processor Affinity, also called CPU pinning, allows the user to assign a process to use only a few cores.

Inter−process communication between "FMI Executable," "Python," and "Distributed simulation controller" uses shared memory between processes on the same machine and TCP communication between processes on different machines.

Figure 4 shows the execution screen of the SSP-FMI simulator, with the "Distributed simulation controller" screen on the left and the simulation output signal time series on the right, plotted as a graph using Python's Matplotlib library (ref. Matplotlib website).

3.2 Performance evaluation using hybrid vehicle simulation

This section reports the performance evaluation of distributed SSP-FMI simulation by using series parallel hybrid electric vehicle (HEV) simulation model shown in

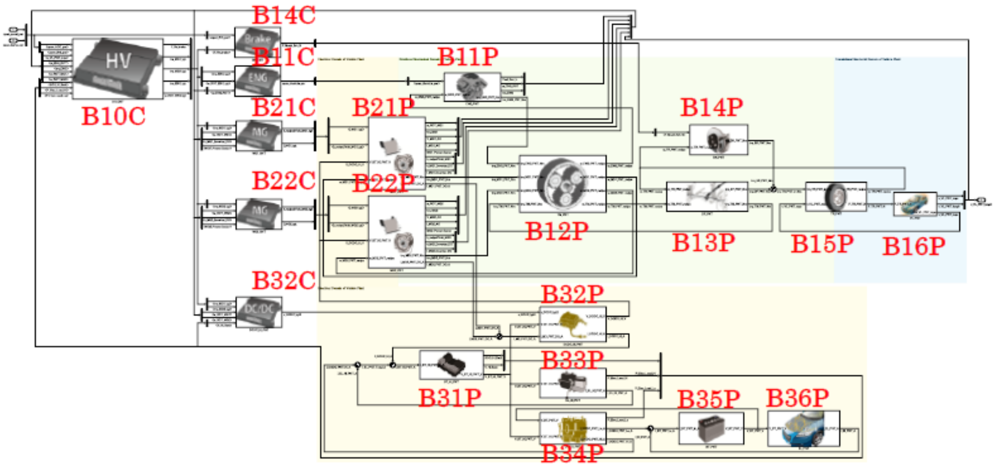


Figure 5. Series parallel hybrid electric vehicle model (ref. JAMBE HEV model)

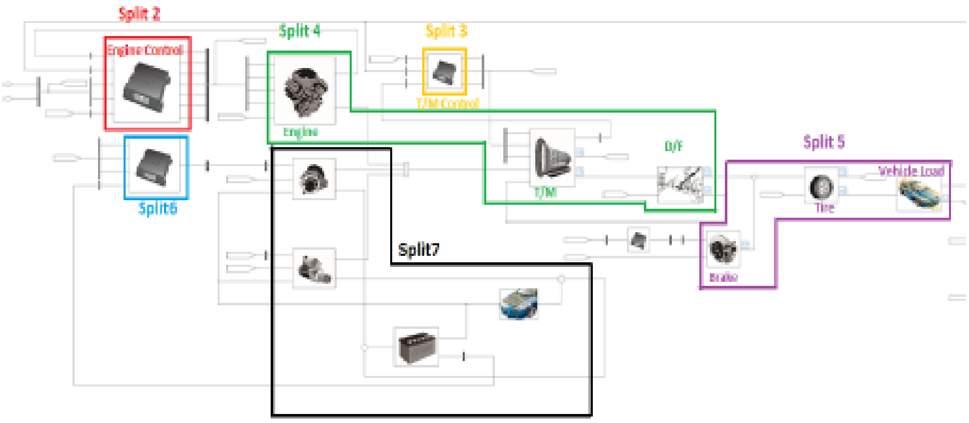


Figure 6. CVT vehicle simulation model (ref. JAMBE CVT model)

Figure 5 and continuously variable transmission (CVT) engine vehicle simulation model shown in Figure 6. Both simulation model is published by JAMBE (Japan Automotive Model-Based Engineering center).

The original model was built in MathWorks Simulink. We divided Simulink model and exported in FMI format. The number of FMI files for the hybrid vehicle model is 21 files divided by the block units of BXXX in Figure 5, and the CVT vehicle model is 7 files divided by the block units of color marks in Figure 6.

Each simulation was performed with the input of 1800 second standard driving pattern of WLTC (Worldwide harmonized Light vehicles Test Cycles) class 3b developed by UNECE (United Nations Economic Commission for Europe). Both simulations were run with a sampling time of 2.5 millisecond which is the same as the original JAMBE model.

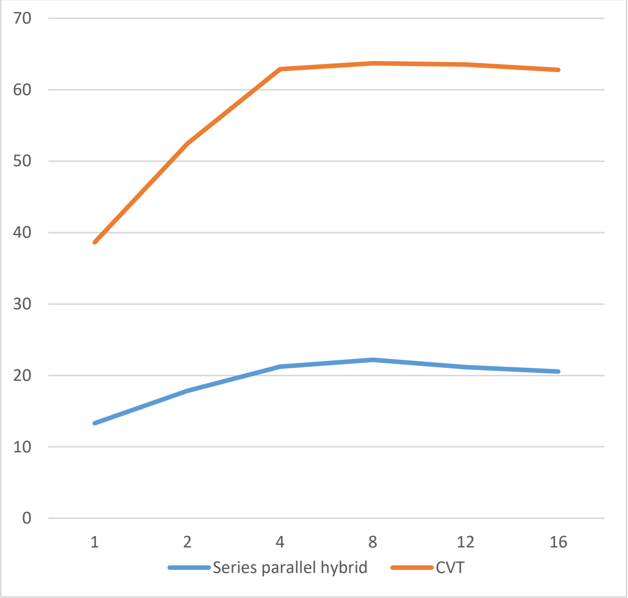


Figure 7. Number of CPU cores (threads) and RTF (Core i7-10870H)

We measured the RTF (real-time factor) of simulation speed using two different machines. The first machine equips Intel Core i7-10870H processor with 8 cores and 16 threads. The second machine equips Intel Core i7-8700 processor with 6 cores and 12 threads.

We used Windows 10 as the OS. The SSP-FMI simulator we developed allows the user to select the number of CPU cores (threads) used in the calculation using the processor affinity option of Windows, and we compared the simulation speed when using a single CPU and when using multiple CPU cores (threads).

Figure 7 shows the measurement results on the machine with Intel Core i7-10870H processor. The vertical axis represents RTF. The horizontal axis is the number of CPU cores (threads) used. The upper measurement is for a CVT vehicle simulation with 7 FMUs, and the lower measurement is for a hybrid vehicle simulation with 21 FMUs. Figure 8 shows the measurement results using the machine with Intel Core i7-8700 processor, and the notation is the same as in Figure 7.

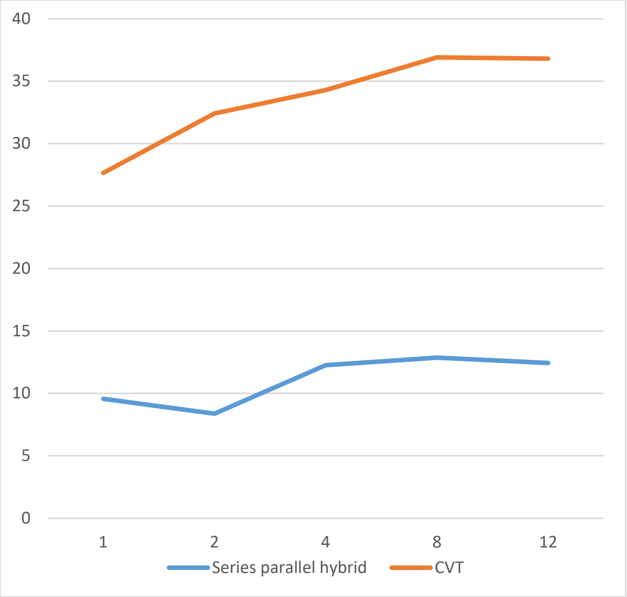
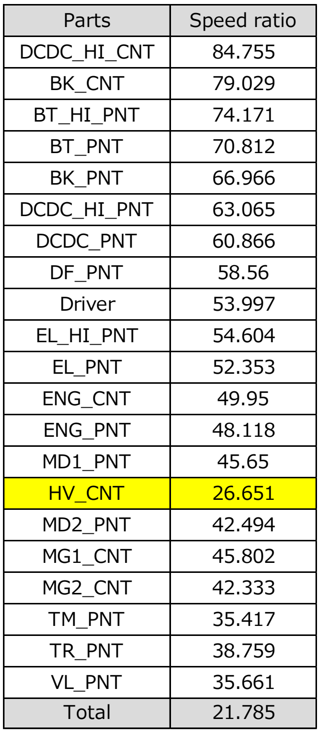


Figure 8. Number of CPU cores (threads) and RTF (Core i7-8700)

The measurement results show that increasing the number of CPUs used in a distributed calculation can increase the speed by up to a factor of two compared to a calculation using a single CPU. However, it was also found that speed increases only up to 8 CPU cores (threads) and that speed tends to decrease slightly when more than 8 CPU cores (threads) are used. This is thought to be because excessive use of processor CPU cores (threads) affects the execution of non-simulation processes and has the opposite effect on the performance of distributed computation.

Table 1. Comparison of RTF for each FMI model. (Series parallel hybrid vehicle)



Parts	Speed ratio
DCDC_HI_CNT	84.755
BK_CNT	79.029
BT_HI_PNT	74.171
BT_PNT	70.812
BK_PNT	66.966
DCDC_HI_PNT	63.065
DCDC_PNT	60.866
DF_PNT	58.56
Driver	53.997
EL_HI_PNT	54.604
EL_PNT	52.353
ENG_CNT	49.95
ENG_PNT	48.118
MD1_PNT	45.65
HV_CNT	26.651
MD2_PNT	42.494
MG1_CNT	45.802
MG2_CNT	42.333
TM_PNT	35.417
TR_PNT	38.759
VL_PNT	35.661
Total	21.785

Table 1 compares RTF of each individual component FMU in a hybrid vehicle simulation using 21 FMUs with 8 threads on a Core i7-10870H, measuring the execution time that "FMI Executable" was running. It can be seen that the simulation speed varies by a factor of several depending on the complexity of the model included in the FMU and the amount of calculation. The overall simulation RTF is 21.785 while the RTF of the single FMU of HV_CNT (hybrid control controller) is close to this at 26.651, indicating that the calculation of HV_CNT is the overall speed-determining factor. This shows that the overall simulation speed tends to be dragged down by the computationally intensive FMU, and that no further speed-up can be expected in the hybrid vehicle simulation even if the number of threads is increased to 8 or more.

By measuring the overall RTF and RTFs of each individual FMU in this manner, it is believed that it is possible to determine the optimal number of CPU cores (threads) that will provide the maximum simulation speed in the parallel distributed SSP-FMI simulator.

4 System parameter optimization using SSP-FMI co-simulation

4.1 Framework of system parameter optimizetion

This chapter describes an application of SSP-FMI co-simulation to system parameter optimization.

Collaborative and rapid development of hybrid power supply systems often requires various configuration and many control parameters to be optimized. It also requires to facilitate model exchange between partners while keeping confidentiality of model. We think distributed co-simulation utilizing model interoperability standard FMI and SSP and Bayesian optimization will be solve the problems.

Figure 9 illustrates the framework of collaborative development platform for hybrid power suppy systems. In this framework, model parts are collected from partners in FMI format and optimum parameter set can be searched by Bayesian optimizer and distributed co-simulation.

4.2 Optimization set-up

A flowchart for optimization functionality is shown in Figure 10, where an initial value is first generated (either randomly or by user input). The SSP file for the co-simulation is then modified, where the values of the parameters to be optimized over are changed. This allows the co-simulation to be run for the chosen parameter values.

Once the co-simulation has finished, the cost function to be optimized can be extracted from the output. Based on the parameter and output values, an optimization module can determine the next point to evaluate.

4.3 Optimization algorithms

Compared to most optimization problems, the cost function in co-simulation is often time- and resource-consuming. The system may comprise a large number of subsystems and even with parallel and distributed simulation, it can take a long time to evaluate its performance. Furthermore, the inner workings of the subsystems are often not known to the co-simulation master as they may originate from different vendors or developers. Hence, we can treat the cost function as a black box which is expensive to evaluate.

For this kind of optimization problem, a suitable algorithm is Bayesian optimization (Brochu09). This works by placing a Gaussian prior on the function and updating the posterior distribution based on the observed input and output values. This can be used to compute the best next value to evaluate, where criteria such as “probability of improvement” and “expected improvement” can be used. The advantage of Bayesian optimization is that it requires few evaluations of the cost function, as opposed to, e.g., evolutionary methods

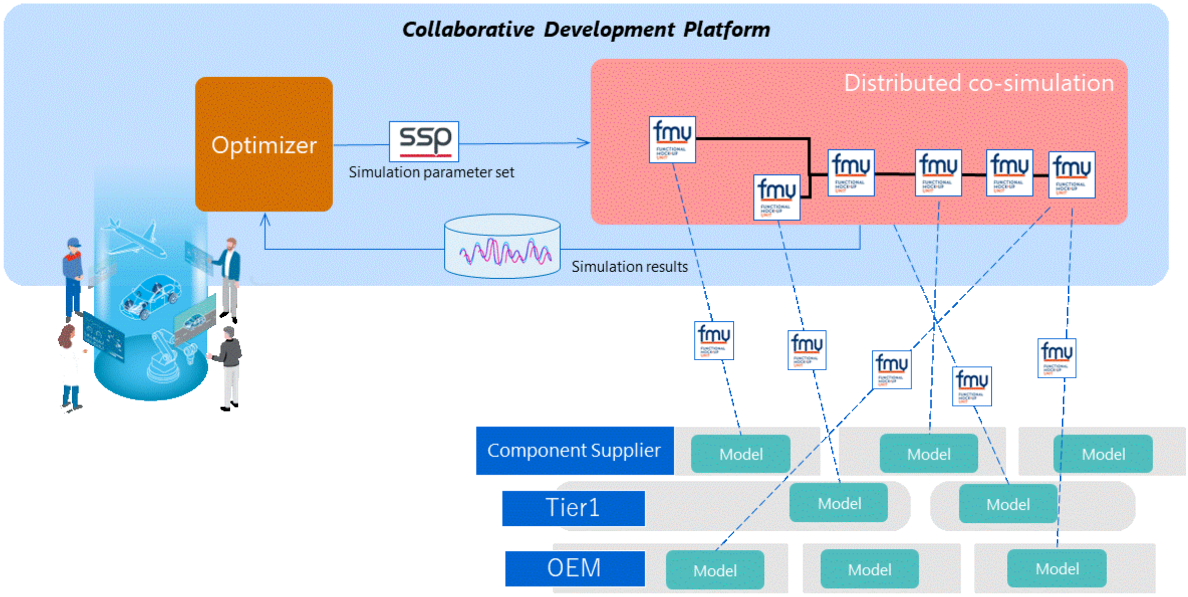


Figure 9. Framework of collaborative development platform.

(NSGA-II, genetic algorithms, particle swarms, etc.) which need many evaluations (Emmerich18).

It should be pointed out that it is possible to consider more than one cost function. Multi objective optimization is common in large and complex systems, where there are many metrics by which a system performance can be measured in. These are often conflicting, and an optimal trade-off is sought. Bayesian optimization can be extended to multi objective optimization, e.g., with efficient algorithms such as TSEMO (Bradford18).

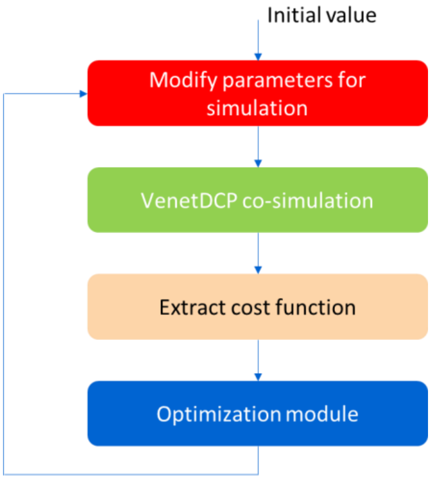


Figure 10. Optimization principle with co-simulation.

4.4 Optimization example

As an example of system parameter optimization, we considered the JAMBE HEV model. In particular we considered the role the clutch thresholds play in the propulsion. As shown below, the clutch helps activate electric only (Figure 11a) or hybrid-electric assist (Figure 11b). This is determined by, among other things, a threshold to open and a threshold to close it (measured in vehicle speed, km/h). These two thresholds were chosen as the system parameters to optimize over. The cost function was set as the fuel consumption during the WLTC class3b test drive cycle shown in Figure 12.

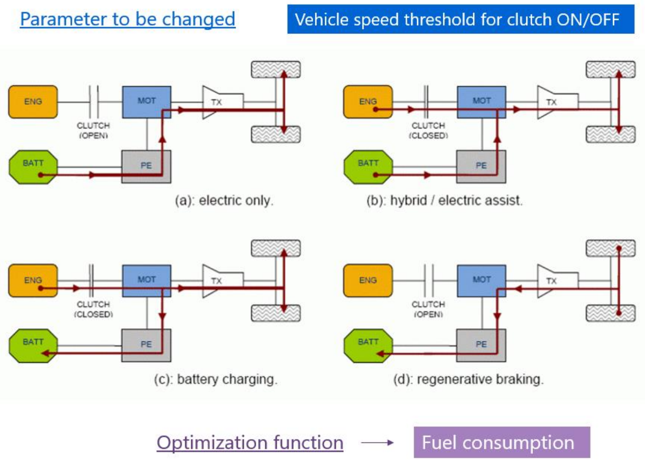


Figure 11. Scenario for the considered JAMBE model.

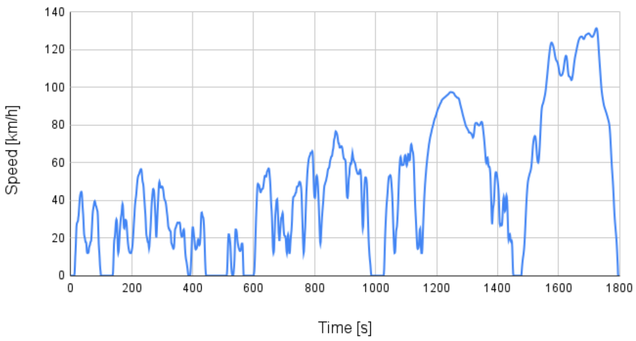


Figure 12. WLTC Class 3b test driving cycle consisting of Low, Medium, High and Extra High phases.

4.5 Optimization results

Using the MATLAB toolbox, Bayesian optimization of the clutch thresholds was implemented. Note that this allows the *Optimization module* in Figure 10 to output a suggested next value to evaluate in the next co-simulation, which was based on the “expected improvement” criterion. The cost function was extracted from the co-simulations as the fuel consumption after the 1800 seconds drive cycle. This is measured in km/l, so we are looking for its maximum.

The results are shown in Figure 13, where the fuel consumption (in km/l) is plotted against the two clutch thresholds parameter which defined in the JAMBE HEV model. HV_CNT_Clutch_ON_threshold_vel_kmph means that closing clutch is possible above this speed. HV_CNT_Clutch_OFF_threshold_vel_kmph means that opening clutch is prohibited below this speed.

To appreciate the optimization result, we also performed an exhaustive evaluation for all feasible parameter values. This involved around 400 co-simulations, whereas the optimization approach only used 10. Compared with the default parameter values which defined in the JAMBE HEV model (red circle), the optimized values (blue circle) showed a 2% improvement in fuel consumption. Although this is quite small, more gains can be had by considering other blocks, possibly in combination with each other.

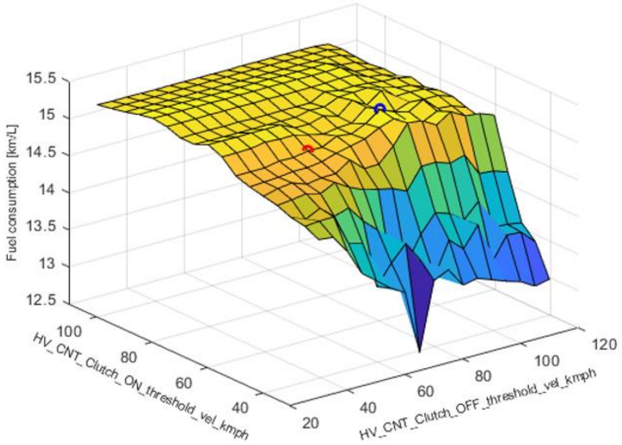


Figure 13. Optimization results for the co-simulated JAMBE model. The red and blue circles represent the default and optimized operational values, respectively.

5 Conclusion

FMI and SSP standards establish model exchange at various levels of abstraction and interoperability between tools. In this paper, we investigated a configuration of SSP-FMI simulator that enables parallel computation by multi-core distribution. We also examined application of SSP-FMI simulator to the system parameter optimization.

We are planning to apply collaborative development platform to the development of electric vehicles (integration of batteries, BMS, power trains, vehicle dynamics, etc), hybrid electric aircrafts (hydrogen fuel cells with batteries and high-performance electric motors) and offshore wind turbines (optimize efficiency and cost by wind & wave prediction).

We also plan to support co-simulation interface of FMI3.0 standard, whose official specification will be issued in 2022, and the newly introduced Scheduled Execution (SE) interface with the distributed parallel simulation in this paper.

References

SmartSE (2023). “SmartSE Recommendation V3 (Smart Systems Engineering Collaborative Simulation-Based Engineering Version 3.0), prostep ivip Association, January 2023. https://www.prostep.org/fileadmin/downloads/PSI_11_V3_SmartSE_Rec_and_Part_A-I.zip

FMI - Functional Mock-up Interface) https://fmi-standard.org

SSP - System Structure and Parameterization https://ssp-standard.org

IEEE1516. “IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)”, DOI： 10.1109/IEEESTD.2010.5553440 https://standards.ieee.org/ieee/1516/3744/

DCP - Distributed Co-Simulation Protocol https://dcp-standard.org/

VenetDCP - Distributed Co-Simulation Platform https://www.global.toshiba/ww/products-solutions/manufacturing-ict/venetdcp.html

Matplotlib - Visualization with Python https://matplotlib.org/

JAMBE HEV model. “Fuel efficiency models and manuals for series parallel hybrid 2 vehicles” https://www.jambe.jp/system/link.aspx?cid=200091

JAMBE CVT model. “Fuel efficiency model and manual for CVT” https://www.jambe.jp/system/link.aspx?cid=200101

Brochu, E., Cora, M., and de Freitas, N. (2009). “A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning”. Technical Report TR-2009-023, Department of Computer Science, University of British Columbia. arXiv:1012.2599.

Emmerich, M.T.M., Deutz, A.H. A tutorial on multi objective optimization: fundamentals and evolutionary methods. Nat Comput 17, 585–609 (2018). https://doi.org/10.1007/s11047-018-9685-y

Bradford, E., Schweidtmann, A.M. & Lapkin, A. “Efficient multi objective optimization employing Gaussian processes, spectral sampling and a genetic algorithm”. J Glob Optim 71, 407–438 (2018). https://doi.org/10.1007/s10898-018-0609-2

Simulation of Vehicle Headlamp Levelling systems

Filip Cieslar[1] Martin Düsing[1]

[1]HELLA GmbH & Co. KGaA, Czech Republic, Germany,
{filip.cieslar,martin.duesing}@hella.com

Abstract

Adjustment systems are used in vehicle headlamps to regulate the flare on the street. The kinematic system within the headlamp is driven automatically based on level sensor signals and can additionally be manually set to a start position. In modern cars the automatic vehicle headlamp levelling is legal duty due to the strong *cut-off line (COL)* between dark and light. This COL can be measured in a workshop but not during operation. Due to the complex kinematics including nonlinear contacts, friction and damping a Modelica model is used to calculate the position of the *COL*. The results show a characteristic hysteresis of the horizonal position during automatic movement. The simulation results are compared to measurements and show good agreement.

Keywords: Cut-off line, hysteresis, headlamp

1 Introduction

Modern vehicle headlamps have a strong *COL* between dark and light. Different loads can cause oncoming traffic to be blinded unintentionally (Hignett 1970). To prevent that a vehicle headlamp levelling system adjusts the *COL* based on information of level sensors at the car axles. The mechanical system to adjust the horizontal or vertical position of the *COL* is strongly nonlinear because of contacts, friction, elasticity and damping such that a hysteresis behavior of the *COL* position vs. the adjustment size can be observed (Opgen-Rhein et al. 2004).

Overall hysteresis can be defined as a difference between the direction of a process which changes the system. In our case this process is a levelling of the COL, and the direction of movement can lead to different ending position of COL. Thus, the hysteresis behavior can be considered as parasitic.

The aim of this paper is to model the dynamics of the levelling system to understand the origin of hysteresis and to predict the position in detail anytime. The model is used in early predevelopment phase of few headlamp projects to verify design ideas and optimize the adjusting system. In this work the model is described and validated with measured data of a real headlamp.

2 Hysteresis behavior of Vehicle Headlamp Levelling

As hysteresis in a headlamp levelling system, we define the difference between start and end position of COL after a defined levelling process. Each automotive company have different measurement processes, but principally it is always forward and backward (or exactly in opposite order) movement. Thus, we compare starting position of COL assuming as 0 and the end position as *h*. The distance between headlamp and wall on which is the COL projected is 10m. The position of the COL is measured in a special testing lab with adequate equipment. The COL level is digitally evaluated while the leveling is done by stepper motor connected to the adjusting system.

Each automotive company has special requirements about the COL level which should be kept. This prevents the unwanted position of the light on the street based on dynamic leveling during travel. The range of vertical (horizontal) COL travel varies but, in all cases, is smaller than 5°, in the 10 m distance equals to 0.87 m. Hysteresis in such a system can be significant and without proper focus on result will not fulfill the requirements. To prevent that the simulating model is created. The Modelica environment is chosen because of its multi domain capabilities because in these systems there are several physical phenomena.

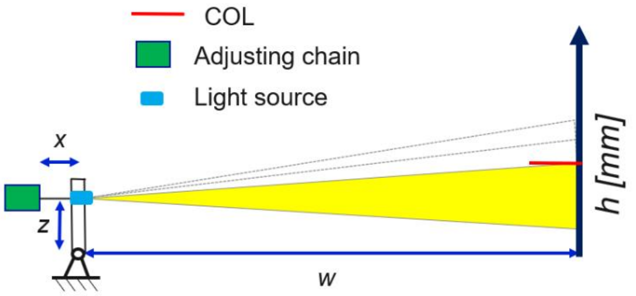


Figure 1 Vertical hysteresis test scheme

3 Modelica Model

The geometry of the hysteresis test shown in **Figure 1** leads to

$$h = x \cdot \frac{w}{z} \tag{1}$$

$$h = \mathrm{w} \cdot \tan\varphi \tag{2}$$

where φ is the angle of rotation of the light source.

Geometry of most of headlamps is within following limits: z = 90…130 mm, x= 2…10 mm. Based on that we realize that any movement in x will lead to around 100 times bigger movement of COL in h. To create a model which is able to simulate such a system we need to pay attention to the adjusting system and include the following:

- Geometry of each part
- Backlashes between parts
- Elasticity of parts
- Contact deformations between parts
- Forces and friction in the system

The adjustment system is designed with several mechanical parts which are connected to each other. The adjustment can be done with an electrical stepper motor and by a manual rotational movement of the so-called customer interface which lead to mechanical movement of the whole adjustment system with stepper motor including. The Hysteresis test is done separately for both versions, but the system is connected as one. As it was written before this system contains several parts and each connection can generate unknown hysteresis behavior. From physical point of view, we can observe the following behavior on an example connection of 2 parts:

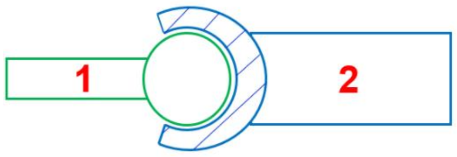


Figure 2 Connection of part1 and 2 in adjustment system

Then the hysteresis contribution of such a contact can be defined as:

$$h = h_b + h_d + h_e \tag{3}$$

Where:

h is total hysteresis

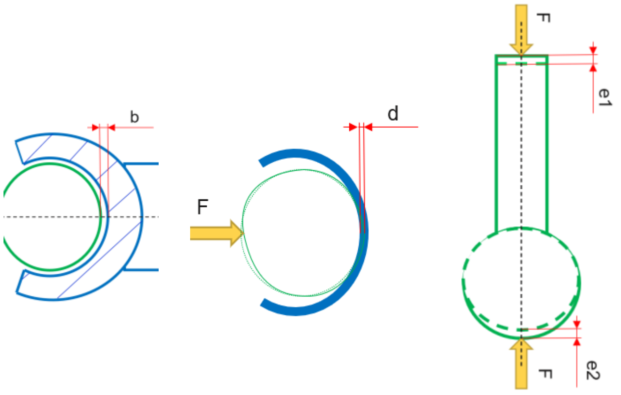
h_b is backlash part of hysteresis

h_d is part of hysteresis caused by deformation

h_e is elasticity part of hysteresis

Backlash	Contact def.	Elasticity
$\Delta h_b = b$	$\Delta h_d = d = (\frac{9F^2}{16E^2R})^{\frac{1}{3}}$	$\Delta h_e = e = \frac{F}{E \cdot S} \cdot l_0$

(Popov et al., 2019)



Δh_b= 0.001 mm	Δh_d= 0.008 mm	$\Delta\ h_e$= 0.0025 mm

$$h = h_b + h_d + h_e = 0.0115 \approx \mathbf{1.15\ mm}$$

Figure 3 Example hysteresis contribution with backlash, contact and elasticity.

In Figure 3 the contributions of each physical part shown in Figure 2 of the total hysteresis are shown. The radius of the ball is R and S is the area. Young's modulus is referred to as E. This particular example shows that the backlash does not have to be the main contributor to the hysteresis even thought that it usually is. The part of Figure 3 describing contact deformation includes manufacturing tolerances and imperfections of the production. Such a contact can be described by the Hertzian contact deformation law or a variation of it. The elasticity of parts contributes as well, due to the different orientation of internal force during leveling. This could be described by Hooke’s law. Hertzian deformation and elastic deformations depend on internal forces in the system. The tangential internal forces are functions of the frictions between elements in the system. Therefore, the problem of hysteresis contribution is pretty complex and needs to be simulated. Some factors can be simplified by linearization of the behavior.

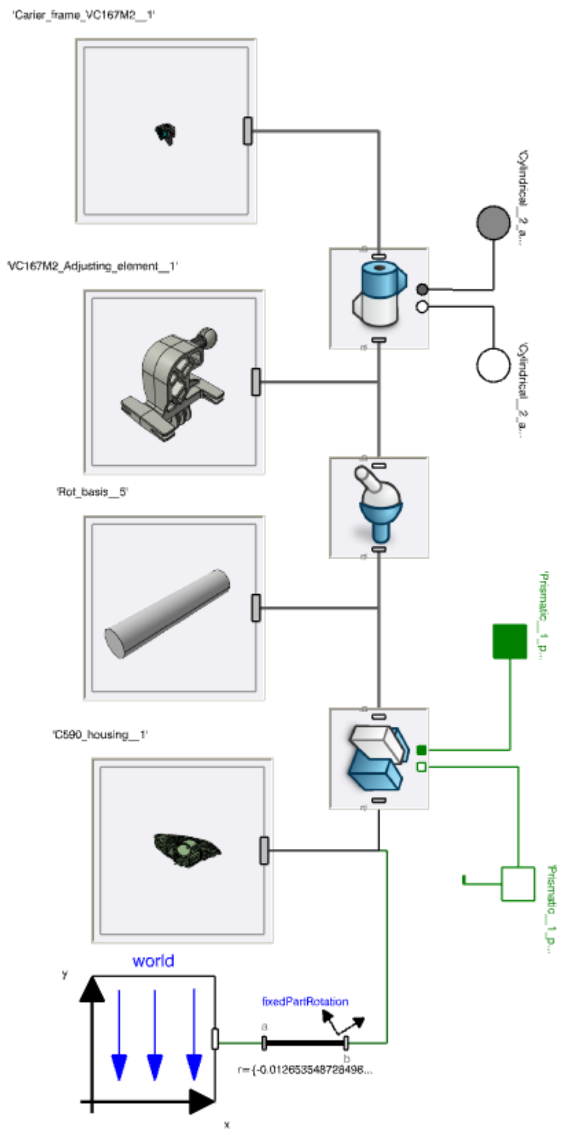


Figure 4 Basic kinematics scheme created in 3DExperience

To create a Modelica model capable of simulating the hysteresis of a COL during the development of a headlamp the software Dymola 2022x and Behavioral modeling (Dymola) in 3DExperience 2022x platform from Dassault systems are used. Very useful is a CATIA interface for kinematics. The kinematics can be easily defined in CATIA. In the next step a Modelica model can be automatically generated with 3DExperience. This Modelica model is based on the precise geometric parameters of the system, such as geometry, center of gravity, inertia tensors and mass of the body. It can be used within 3DExperience or exported to Dymola without loss of functions.

Figure 4 shows a part of subsystem of the main body of a headlamp and its adjusting system. Parts are connected through joints. Controlled joints need to be connected to other subsystems which define their behaviors in terms of torques, forces, elasticities, or backlashes.

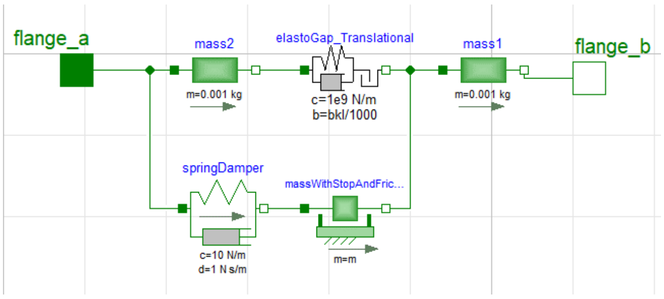


Figure 5 Subsystem of backlash with friction and hard stop

To model friction the subsystem in Figure 5 is used. It is a model with Stribeck characteristic, which is the most accurate and proven technique to simulate friction in mechanical systems. It combines static friction which is higher than the dynamic friction, when parts are moving. In addition, velocity dependency is included into this behavior. Thus, in comparison to a very simple Coulomb model of friction, this model can simulate more precisely parts which are sliding in guiderail. The behavior of that part is very dependent on friction. The Stribeck characteristic is described in Figure 6.

4 Simulation Results

The simulation model shown in the previous chapter was used in development of several headlamps. In the variety of projects, the design team is forced to use different adjustment systems. Some parts are changed due to space management but as the simulation model is developed as a parametric model, it is possible to adjust it to the current design. The simulation procedure is set up with the same parameters as the one in the measuring laboratory for every project. A big advantage of the model is that additional parameters like forces inside the system or input torques are tracked automatically.

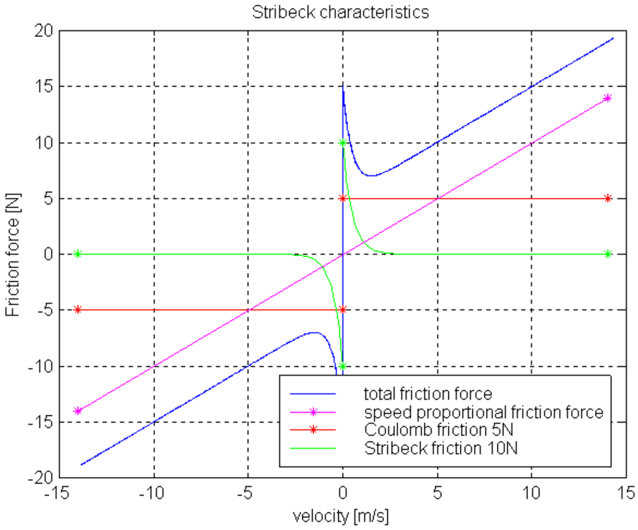


Figure 6 Friction definition using Stribeck characteristic

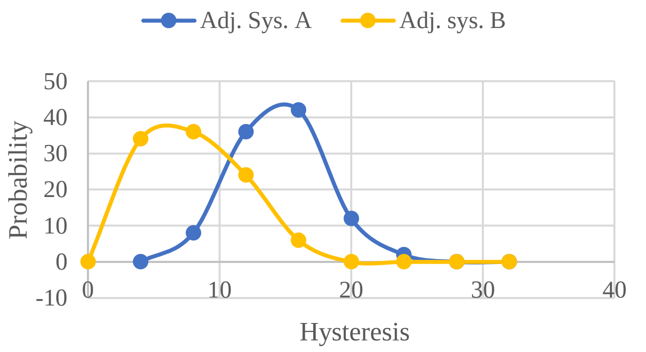


Figure 7 Hysteresis results histogram of 2 adjusting systems

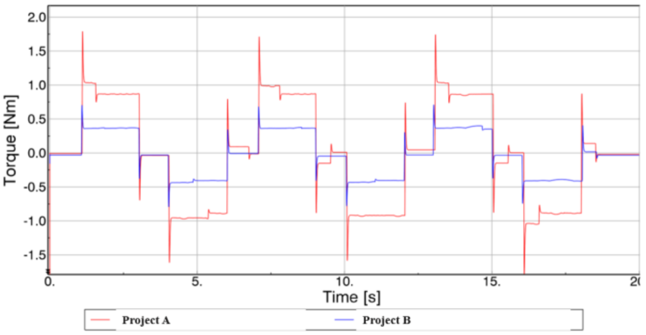


Figure 8 Driving torque needed to level during hysteresis test – project compared

Because of the high number of parameters and their manufacturing tolerances there is a need to perform hundreds or thousands of simulations during the design phase (Brück et al., 2002). A Monte Carlo approach was applied to include statistical characteristic of that problem. The simulation is run 500 times with randomly selected parameters. The distributions are chosen based on knowledge of tolerances and measured real production samples. The results can predict the possibility of a hysteresis test fulfilling requirements with current design. Example of results of two different adjusting systems used in headlamps are shown in Figure *7*. The difference between A and B Adjusting systems are in the design and the parts are used. Due to restricted space inside of a headlamp the A system is smaller and less robust. This is why his behavior leads to worse hysteresis results.

In Figure 8 two torque characteristics are shown. It is a comparison of two different projects. It can be observed that Project A is dealing with way more higher torque values. This can lead into problems within the adjusting system and the design needs to be changed. This is a good example how a simulation result can help a design team in early development phase without any real testing and measurement.

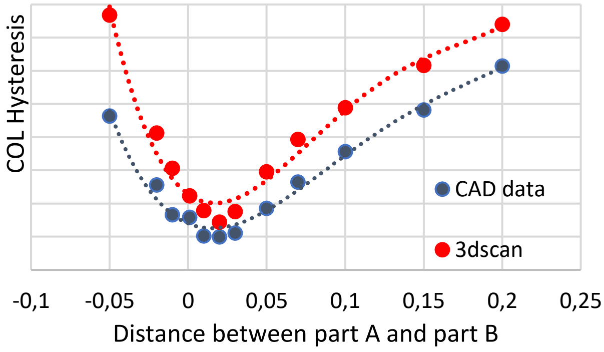


Figure 9 Comparison of simulation result based on geometry from CAD and real 3Dscan (the difference is influence of manufacturing process)

The graph in Figure 9 shows a comparison of simulation results with different 3D data. The blue line illustrates a result based on data from an early phase of development in CAD environment. To reduce the influence of the manufacturing process on the simulation result, manufactured parts were scanned and analyzed. The simulation result is shown in red. This was done later in the project phase and proved that it can significantly change the hysteresis result.

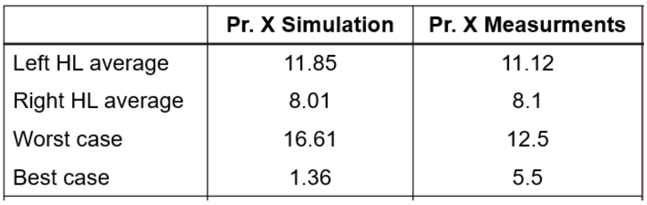
Therefore, in next projects it is planned to make collaboration with Moldflow simulations department. Moldflow can predict the manufacturing imperfections or dislocations and this data can be used as input for system model for hysteresis instead of using CAD data from design. This makes the simulation output more reliable concerning precision of both models, but also makes the result more realistic. Realizing this feature in early phase of the project is a key point to benefit from digitalization of testing process and will save a lot of time and money in next phases of each project.

5 Comparison with Measurements

All simulation models contributing into development process must be validated and verified. Otherwise, the project team cannot rely on simulation results or make simulation-based decision and must prove the simulation ideas with proper measuring test. In that case there is no significant cost reduction which is the biggest reason to do simulations at all. A once validated and verified model can be used in a lot of following projects it can save a lot of money with reducing numbers of physical testing, development of wrong design ideas, speeding up the troubleshooting process etc.

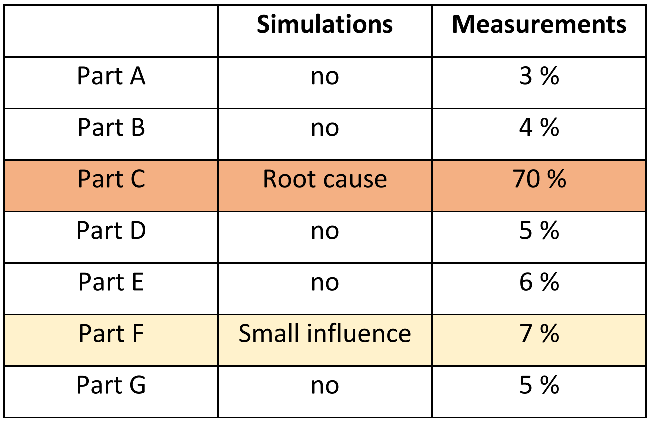
However, in our case verification of such a model is especially complex. System model of headlamp for hysteresis of adjusting processes deals with a lot of complex and hard to measure parameters such as friction, current position of components, very precise dimensions of the parts and forces. In the hysteresis test only, the final value is measured and even thought it fits to the simulation value, it is not certain that the model can identify the root cause properly.

Tables 1 and 2 show examples from projects. In Table 1 the simulated hysteresis in mm in x-direction is compared to a measurement and shows quite good agreement. Table 2 shows an analysis of the simulation results to identify the biggest contributor to the total hysteresis. In this case obviously Part C is the root cause of 70% of hysteresis in the whole system.



	Pr. X Simulation	Pr. X Measurments
Left HL average	11.85	11.12
Right HL average	8.01	8.1
Worst case	16.61	12.5
Best case	1.36	5.5

Table 1 Prediction of hysteresis value: Simulation vs measurements



	Simulations	Measurements
Part A	no	3 %
Part B	no	4 %
Part C	Root cause	70 %
Part D	no	5 %
Part E	no	6 %
Part F	Small influence	7 %
Part G	no	5 %

Table 2 Identifying the root cause in the system

6 Conclusion

In this paper a headlamp levelling system and its issues with hysteresis of the cut-off line between dark and light at the 10m wall is shown. A Modelica model based mostly on the Modelica standard library, using the CAD interface within the 3DExperience platform was described and proven as capable to be a great help in the development of a headlamp. The Simulation calculates and prints several internal parameters inside the adjusting system as torques and forces. Furthermore, the simulation model was improved with measured data of real parts through 3D scan. Verification methods were used to evaluate the precision of the simulation results. However, there are still some uncertain factors that were not considered in the model such as temperature of headlamp. In the future there is an ambition to upgrade the model to be even more valuable in the development process.

References

Opgen-Rhein, Peter; Bertram, Torsten; Seuss, Jurgen; Karas, Peter; Stryschik, Dieter (2004). “A Hardware in-the-Loop Based Process Improves Quality and Decreases the Development Period of a Dynamic Headlamp Levelling System”. *IFAC Proceedings Volumes.* 37.14, pp. 37-42. DOI: 10.1016/S1474-6670(17)31077-7.

Hignett, H. J. (1970). “Vehicle Loading and Headlamp Aim”. *Road Research Laboratory /UK*. Report No LR 329. pp. 20

D. Brück, H. Elmqvist, S. E. Mattsson und H. Olsson (2002). “Dymola for Multi-Engineering Modeling and Simulation”. *Proceedings of the 2nd International Modelica Conference*. Oberpfaffenhofen, 55-1 - 55-8.

Popov VL, Heß M, Willert E (2019) Viscoelastic materials. In: *Handbook of Contact Mechanics.* Springer Verlag, Berlin

Piecewise-Steady-State Modelica Simulations for the Conceptual Design Phase of Industrial Processes

Raphael Agner[1] Jonas Grand[1] Andrin Duss[1] Beat Wellig[1]

[1]CC Thermal Energy Systems and Process Engineering, Lucerne University of Applied Sciences and Arts, Switzerland, {raphael.agner, jonas.grand, andrin.duss, beat.wellig}@hslu.ch

Abstract

The conceptual design of industrial processes is challenging as relatively little information about the eventually selected equipment and their operation is known in this early design stage. Furthermore, the systems are increasingly integrated with themselves, and their design must be addressed systematically. Simulation can assist in better understanding the effects of design decisions on the resulting system performance. To facilitate the simulation of industrial processes in this early design phase, this paper proposes an approach to modeling system components specifically aimed at employing known key design parameters and assuming steady-state behavior of the process for a certain period of time (e.g. one hour). A solution over a longer period of time (e.g. for a year) can then be obtained by simulating a multitude of such shorter periods, leading to the piecewise-steady-state solution. The proposed approach is developed with an exemplary case study, based on a real industrial site. The resulting model computes the annual load profile within the range of seconds for the given case study.

Keywords: Piecewise-Steady-State, Conceptual Design, Process Simulation, Energy Systems

1 Introduction

The design and the optimization of industrial processes must be addressed systematically. In the conceptual design phase, engineers evaluate how different process-requirements can be fulfilled with various technologies and with different combinations of possible unit operations. This generally includes the design of heat exchanger networks including energy conversion units such as (absorption) heat pumps, gas turbines, or (organic) Rankine cycles. However, the plant design is restricted to consider a limited number of operating points due to its requirement of manual execution of specific steps. Nevertheless, many processes are undergoing changes over time, such as annual load variation or intra-day operation changes. This also applies with increasing intensity to energy supply systems, especially when considering (local) renewable energy sources. In the conceptual design phase, it may neither be of high relevance nor appropriate to attempt to address these fluctuations with detailed dynamic simulations of the processes as this level of detail may be too intensive on engineering cost, while the necessary boundary conditions and especially process parameters are neither known nor relevant in this phase. Therefore, the processes can often be described sufficiently accurately when assuming piecewise-steady-state behavior of the process over a certain period of time, in which the operating conditions do vary only negligibly and which are significantly longer than the dynamic response of the individual components. The evaluation of a multitude of such periods allows for the quantitative analysis of the system through a larger period of time. The, in this phase, unknown system dynamics can therefore be neglected and the models can be reduced to their steady-state characteristics.

Modelica is often used for the simulation of such thermal systems. As it can be seen on the Modelica Library overview (Modelica Association 2023a), various libraries – open-source and commercial – address these systems specifically. Furthermore, Modelica software developers are developing an increasing amount of functionalities specifically targeting the steady state evaluation of models. These tools mainly target the calculation of one steady-state operation point. In this way, they may also be used to simulate design points of industrial processes. In the conceptual design phase, the customarily required set of parameters describing the physical entity of each system component is, however, not known.

The research gap addressed in this paper is the change in the modeling requirements compared to existing libraries when simulating industrial processes in the conceptual design phase. Using an example from a waste incineration plant, this paper describes how Modelica may be used as a simulation framework to model the steam turbine process with corresponding heat utilization based on its steady-state characteristics and available design parameters. The simulations are performed to analyze the performance of the system on an annual basis. The novelty introduced is the use of piece-wise steady-state modelica models that employ design reference values as system parameters instead of estimated physical properties that are not known in the given design-phase.

2 Case Description

The analysis described in this paper is part of an ongoing project and lays the foundation for comparing system performance considering different yet to-be-defined system

DOI
10.3384/ecp204757

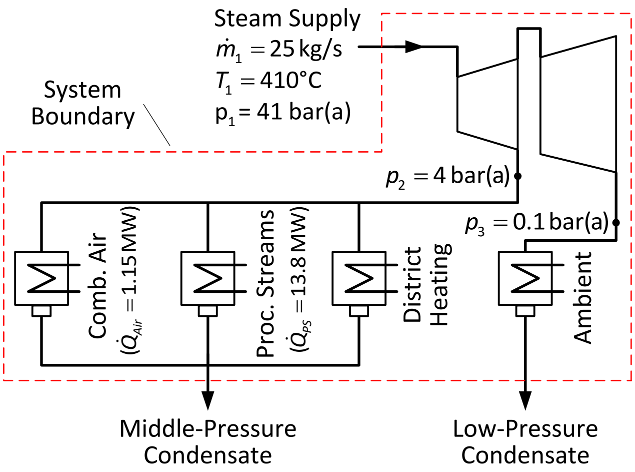


Figure 1. Schematic of the system under consideration.

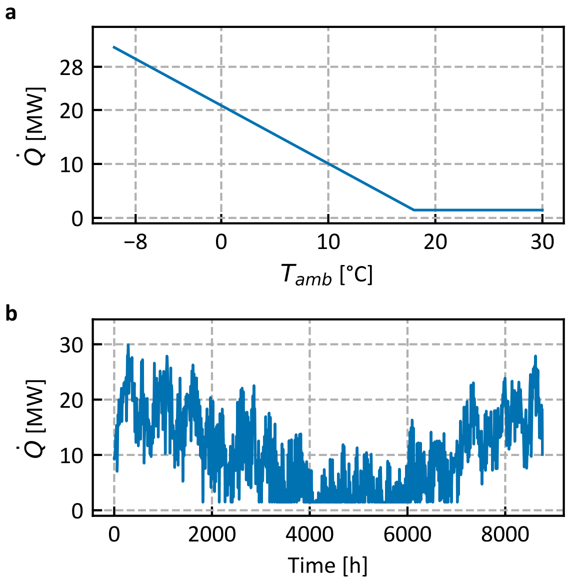


Figure 2. Heat load curve (a) and annual load profile of the assumed district heating network (b) when applying the heat load curve of (a)

configurations. Various energy conversion units may be included in the analysis in the future step. The core part of the analysis, the steam turbine and heat utilization process of a waste incineration plant, is described subsequently. The data is anonymized and simplified from the real case under investigation by the authors.

Overall, the core part of the system (depicted in Figure 1) is governed by the supply of steam from a waste incineration furnace, which can be assumed to work continuously over the year, delivering steam at $p_1 = 41\,\text{bar}$ and $T_1 = 410\ ^\circ\text{C}$ with a mass flow rate of $\dot{m}_1 = 25\,\text{kg/s}$. Subsequently, the steam is expanded in a two-stage steam turbine. The turbine considered is equipped with a steam bleed on the medium pressure level at $p_2 = 4\,\text{bar}$ (between stages one and two). Different heating demands are to be covered by this bleed steam, including combustion air preheating, a district heating network and various process streams (summarized in one aggregated heating demand called process streams). While most processes are due to their link to the main waste furnace constant, the district heating demand varies by nature. To account for this variation, in this early design phase, a simple linear heating curve, specified by a heating limit of $T_{HL} = 18\,^\circ\text{C}$ and a nominal heating load of $\dot{Q}_{nom} = 28\,\text{MW}$ at the nominal ambient temperature for the heating system design of $T_{amb,nom} = -8\,^\circ\text{C}$ is assumed. This nominal ambient temperature corresponds to the climatic conditions in Zurich (SIA 2010), the heating limit is chosen based on the experience of the authors with similar systems. To account for the domestic hot water consumption, a constant load of 1.5 MW is added to the heating curve. This value corresponds to a split of 84 % space heating demand and 16 % domestic hot water demand based on the Swiss end-energy consumption statistics (Kemmler and Trachsel 2022). The annual load profile is then calculated with the design reference year data of the Meteonorm database (Meteotest AG 2023) for Zurich which provides the hourly averages of the ambient temperature. Since district heating systems possess rather large inertia (storage, volume of the pipes and inertia of the heat consumers), a rolling-mean filter was applied with an assumed four hours averaging window. Figure 2 shows the assumed heating demand in relation to the ambient temperature in the form of the heat load curve and as the resulting hourly profile, which is later imported into the Modelica simulation.

The remaining streams are defined by their constant heat load as follows:

- Combustion air preheating: $\dot{Q}_{Air} = 1.15\,\text{MW}$
- Process Streams: $\dot{Q}_{PS} = 13.8\,\text{MW}$

3 Model Description

Since the parameters of the physical entity of the planned components in the system under consideration are only specified in later design phases, the models used in the proposed simulation approach must be formulated differently than typically seen in thermal energy system simulation. In the case of a condenser, for example, it shall be prescribed that the steam be condensed fully to its bubble point or subcooled to a given temperature. Thus the model shall accept corresponding parameters instead of the parameters of an eventually selected component, such as its heat transfer area or its heat transfer coefficient. Naturally, this specification affects the resulting models' structure and reduces their universal applicability. Additionally, boundary conditions must be chosen such that the overall model is well-defined. Considering these restrictions, an overview description of the derived models is given in the following sections.

DOI
10.3384/ecp204757

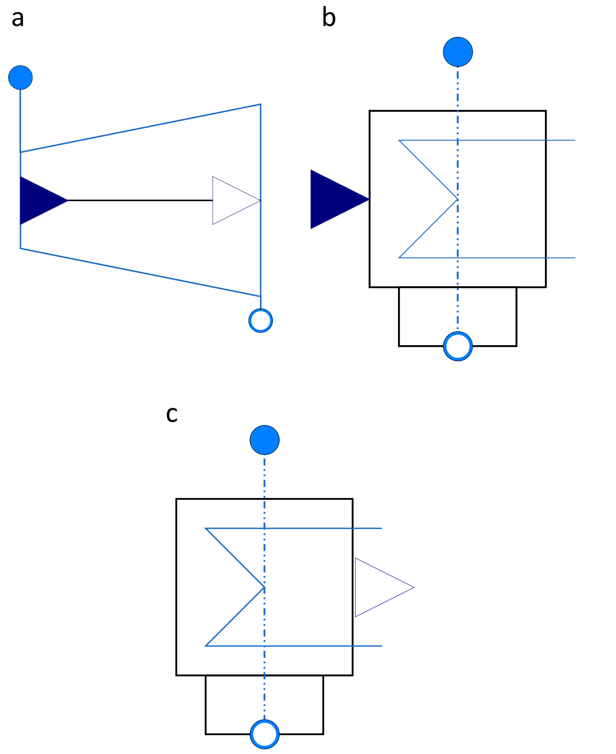


Figure 3. Icon of (a) one turbine stage, (b) of the condenser with prescribed heat flow rate and (c) of the low-pressure condenser after the second turbine stage.

3.1 Steam Turbine

In the considered model, the individual turbine stage is modeled based on typically available design parameters:

- Outlet pressure p_ω
- Isentropic efficiency η_s
- Combined mechanical and electrical efficiency $\eta_{el.+mech.}$

Based on these three parameters and the given inlet conditions governed in the overall model (see section 3.4), the model can be implemented to describe the steady-state characteristics of the turbine stages as follows (See e.g. Baehr and Kabelac (2012) for further references):

$$P_{el} = \dot{m}\ (h_\alpha - h_\omega)\ \eta_s\ \eta_{el.+mech.} \tag{1}$$

Where P_{el} is the generated electric power of the turbine stage, $\dot{m}$ the steam mass flow rate, h_α and h_ω the specific enthalpies at the in- and outlet of the turbine, respectively. Using the isentropic efficiency, h_ω is expressed as follows:

$$h_\omega = h_\alpha - (h_\alpha - h_{\omega,s}(p_\omega))\eta_s \tag{2}$$

Where $h_{\omega,s}(p_\omega)$ is the enthalpy at the outlet of the turbine if the expansion to p_ω would occur isentropically.

The turbine stage is therefore parameterized with its efficiencies and the outlet pressure of this stage. These values can be obtained at the very beginning of any design process of a steam turbine process.

3.2 Steam Condensers

To model the utilization of the middle-pressure steam (after the first turbine stage) for different heating requirements, a set of condenser models was developed (see Figure 3b and Figure 3c).

For the simplest case of a prescribed heat flow (Figure 3b), the model requires only the outlet condition of the condensate (temperature or steam quality) as a parameter. The prescribed heat flow is implemented as an input signal to the block.

In the given case study, the pressure losses can be neglected as there are no subsequent components after the condensers, but could also be later integrated as an additional parameter of the models.

Therefore, the outlet enthalpy on the steam side of the condenser can be calculated with the specified outlet temperature or steam quality and the pressure at the inlet (prescribed by the upstream component). With this known outlet enthalpy h_ω the energy balance is as follows:

$$\dot{Q} = \dot{m}\ (h_\alpha - h_\omega) \tag{3}$$

With the described case of a prescribed heat flow rate $\dot{Q}$ for the middle-pressure condensers, the mass flow rate $\dot{m}$ is calculatable, and Modelica's equation-based modeling concept can be applied directly.

The resulting variation of $\dot{m}$ directly employs the typically used control concept of a mass flow variation with a throttle valve on the steam side. However, no control parameters have to be specified in this, proposed way.

The condenser of the low-pressure steam after the second turbine stage must be modeled slightly differently, as the heat flow rate may not be specified by an input, but it is the result of the condensation of the entire steam flow after the second turbine stage. With a slight alteration of the condenser model described above, this objective can be achieved. The energy balance (Eq. 3) remains the same. The only alteration in the Modelica code is removing the heat flow rate input. Additionally, the mass flow rate of this component must be specified in the overall model, utilizing the proper boundary conditions as described in the following section. Again, the only parameter needed for this component is the outlet temperature or the steam quality of the condensate.

3.3 Boundaries and Data Sources

The boundaries are adapted from the Modelica Standard Library (Modelica Association 2023b) but altered to prescribe pressure, enthalpy and mass flow rate at the steam source and none of them at the condensate sink. Therefore, they must be used together to result in a well-defined system of equations.

The heat load of the district heating system is imported from the *Combi Time Table* block of MSL 4 (Modelica Association 2023b) utilizing the Extern Data Library (Beutlich and Winkler 2021) to facilitate data exchange with the external data source.

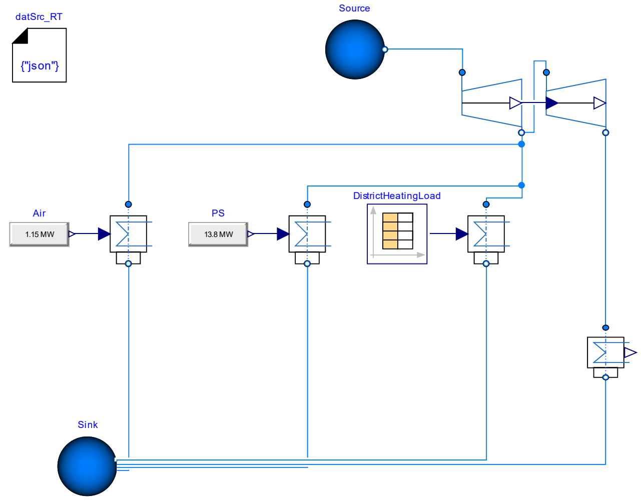


Figure 4. Complete model of the considered part of the steam turbine process with two turbine stages and four condensers.

3.4 Resulting Overall Model

When modeling the overall process as shown in Figure 1 with the components as derived in the previous sections, the Modelica model of Figure 4 is resulting. For the following simulation study, the parameters according to Table 1 have been set, where the outlet conditions of the condensers are prescribed with the respective temperature T or steam quality x:

Table 1. Used parameter values in simulation study

Variable	Value
η_s	0.8
$\eta_{el.+mech.}$	0.9
$p_{\omega,2}$	p_2
$p_{\omega,1}$	p_3
$T_{\omega,cond,Air}$	50 °C
$T_{\omega,cond,PS}$	70 °C
$T_{\omega,cond,DH}$	100 °C
$x_{\omega,cond,RC}$	0 (bubble point)

Annual system simulations were performed to evaluate the functionality and performance of the described modeling approach. The steady-state solution of each timestep (in this case, one hour) is computed. The calculated steady-state solutions for each hour of operation are then analyzed to create an overview of the resulting operating points of the eventual system. The specification of the boundary conditions is given in section 2 and the parameterization is given in section 3.4.

4 Simulation Results

This initial simulation aims to validate the model's functionality and to perform first investigations of the effects of varying middle-pressure steam utilization on electricity production. Using Dymola® 2023 with the Dassl (Petzold 1993) algorithm and the Visual Studio 2019 C++ Compiler, a total CPU-time for the integration of approximately 1.5 seconds results on a Notebook with an i7-1265U CPU and 16 GB of RAM.

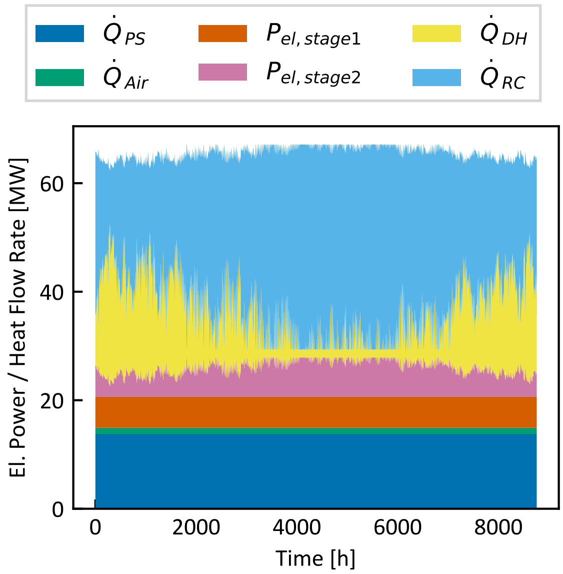


Figure 5. Stack plot of extracted heat flow rates and electric powers from the system.

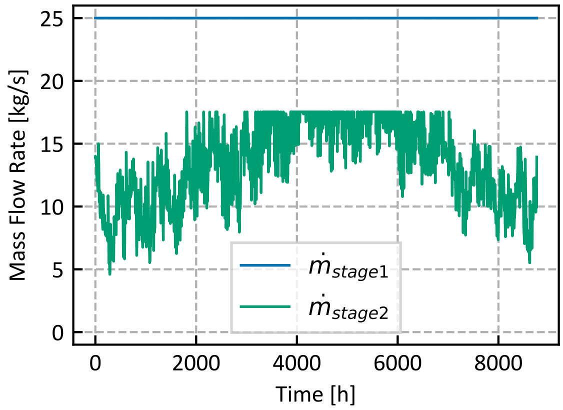


Figure 6. Mass flow rates through the two turbine stages throughout the year

Figure 5 shows the duties of the different components under consideration. Electricity and heat extracted from the process are added up in the vertical of the stack plot to give an overview of the shares of the different quantities over time. It can be seen that roughly 65 MW of heat or electricity is extracted from the plant throughout the year. The total extracted load varies slightly as the condensate leaves the system at different enthalpy levels, and the turbines' mechanical and electrical efficiency $\eta_{el.+mech.}$ leads to further losses. The condensate of the middle-pressure steam is leaving the system in the range of 50-70 °C, while the condensate of the low-pressure steam at the recooler is leaving the system at 46°C (i.e., the bubble temperature at 0.1 bar) (See section 3.4). The plot's three lowest entries depict the constant heat extraction for the process streams, the combustion air preheating and the electricity extracted through the first turbine stage. Thereafter, there is mainly a trade-off between heat utilization for the district heating network and electricity production with the second turbine stage. The recooling heat flow corresponds with the electricity production of the second stage, as there is a direct link between the two components.

When analyzing the turbine more thoroughly by studying the steam mass flow rate of each turbine stage as shown in Figure 6, it can be seen that i) the mass flow rate of the first stage is expectedly not varying as no steam consumers are in parallel to this stage, an ii) the steam mass flow rate through the second stage varies greatly in the chosen concept. A part load ratio in terms of the mass-flow rate of 26% would result (minimum vs. maximum of mass flow rate in stage 2). Since this might pose challenges for the detailed design of the turbine and the corresponding controls, changes in the conceptual design may be investigated to improve system operability.

5 Discussion

The chosen approach of piecewise-steady-state system simulation for the considered steam process leads to a relatively lightweight model that computes annual load simulations in the scale of seconds. To be able to obtain these results, the overall model must be formulated in a way, that the given physical relationships are sufficient. The Modelica modeling concept assists in formulating such models as the balance equations can almost directly be entered in the respective Modelica source code. The graphical representation and interactive connection of the different components allow additionally the adaption of changes of the components and model re-utilization for similar applications.

The chosen approach for system parameterization using the design values such as isentropic efficiency and condenser outlet temperatures instead of the physical quantities like heat transfer area or turbine geometry enables the modeling of the system as it should be designed in the later design phase without having to fine-tune component specifications with the latter sets of parameters. The presented case-study demonstrates the applicability of the proposed modelling approach, while more sophisticated design tasks must be addressed in the future to showcase the ability of such models to support design decisions. Comparing to the traditional design workflow where only few data points are calculated manually the herein proposed approach offers insight into a large range of possible operating conditions and enables the study of the individual process parameters in detail. This approach thus allows to make design decision based on the performance in various operating conditions and enables additionally the quantification of the performance of the system for a representative period of time (e.g. one year).

On a numerical point of view, the removal of the dynamics of the models should allow for the use of lower-order integration methods to further reduce the computation time. A comparison of different integration methods is thus also needed in the future.

6 Conclusion

In this paper, a novel approach for the simulation of thermal energy systems in the conceptual design phase has been presented. It was shown, that the models should be formulated differently when simulating a process in the conceptual design phase than in later phases. The developed simulation models consisting of steam turbines and condensers enable piecewise-steady-state Modelica simulations for the analysis of the annual performance of the example case-study in the scale of seconds in integration time. The main contributions can be summarized as (i) the formulation of the components to be compliant with the available parameters and (ii) the implementation as simplified models treating the systems as piecewise-steady-state in their behavior. Further development of the chosen approach is identified in the development of a general-purpose model library and in the extension of the available unit operations with a focus on energy conversion units. Furthermore, more exhaustive case studies need to be analyzed to demonstrate the viability of the chosen approach. This would also allow to identify possible challenges in the solvability of models with the proposed approach and to derive more general modelling guidelines.

Acknowledgements

This work has been supported by the Swiss Federal Office of Energy SFOE (project DeCarbCH within the "SWEET" program, contract number Si/502260-01 and project PICC, contract number Si502519-01). Further financial support was provided by the Lucerne University of Applied Sciences and Arts, Switzerland.

References

Baehr, Hans Dieter and Stephan Kabelac (2012). “Thermodynamik der Wärmekraftanlagen”. In: *Thermodynamik: Grundlagen und technische Anwendungen*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 523–568. ISBN: 978-3-642-24161-1.

Beutlich, Thomas and Dietmar Winkler (2021-09). “Efficient Parameterization of Modelica Models”. In: *Proceedings of the 14th International Modelica Conference*. Ed. by Martin Sjölund et al. Linköping Electronic Conference Proceedings 181. Linköping, Sweden: Modelica Association and Linköping University Electronic Press, pp. 141–146. DOI: 10.3384/ecp21181141.

Kemmler, Andreas and Tim Trachsel (2022). *Analyse des schweizerischen Energieverbrauchs 2000 - 2021 nach Verwendungszwecken*. Tech. rep. Bundesamt für Energie BFE.

Meteotest AG (2023). *Meteonorm*. Computer Program, last accessed: 2023-06-05. URL: http://www.meteonorm.com/de/downloads.

Modelica Association (2023a). *Modelica Libraries*. Web Page, last accessed: 2023-06-05. URL: https://modelica.org/libraries.html.

Modelica Association (2023b). *Modelica Standard Library Version 4.0.0*. Computer Program, last accessed: 2023-06-05. URL: https://github.com/modelica/ModelicaStandardLibrary.

Petzold, Linda R. (1993). *Description of DASSL: a differential/algebraic system solver*.

SIA (2010). *Merkblatt 2028, Klimadaten für Bauphysik, Energie- und Gebäudetechnik*. Code of Practice.

Modeling and Control Design of an Educational Magnetic Levitation System

Anton Haumer[1]

[1] Faculty of Electrical Engineering and Information Technology, Ostbayerische Technische Hochschule Regensburg, Germany, anton.haumer@oth-regensburg.de

Abstract

A magnetic levitation system is a perfect educational example of a nonlinear unstable system. Only with suitable control, a small permanent magnet can be held floating stable below a coil. After modeling and simulation of the system, control of the system can be developed. At the end, the control algorithm can be coded on a microcontroller, connected to a pilot plant.

Keywords: mechatronics, magnetic levitation, time-discrete control, functional mockup interface

1 Introduction

Using a ready-to-use system like Zeltom's depicted in **Figure 1** allows to set the focus on the development of the control algorithm and gives quick results.

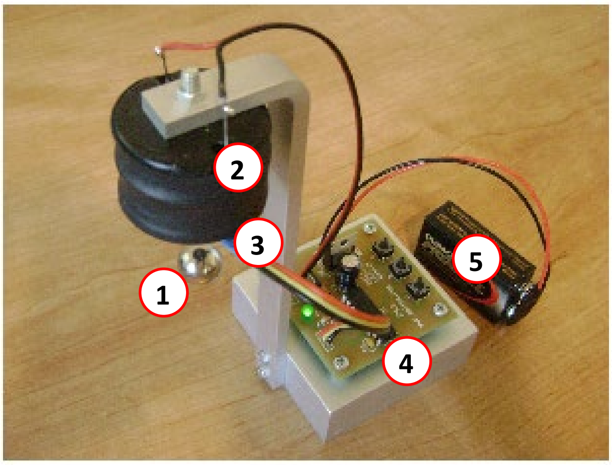


Figure 1 *Magnetic Levitation System*
https://www.zeltom.com/emls.html

1.1 Description of the System

Zeltom's system (**Figure 1**) consists of:

1. Permanent magnet (disc or sphere)
2. Coil (with iron core)
3. Hall effect sensor (below coil)
4. PCB with controller and power electronics
5. Voltage source (9 V battery)

The permanent magnet is attracted by the iron core of the magnet and the magnetic field excited by the current flowing through the coil. Gravitational force acts in opposite direction. Inverting the coil current wouldn't result in repelling the magnet, but causes the magnet to flip by 180°.

The magnet's position is detected by a Hall effect sensor, placed at the bottom of the coil. Not only does the magnetic field of the permanent magnet affect the Hall effect sensor, but the magnetic field caused by the coil current, too.

1.2 Equations of the System

The transient behavior of the coil current is described by Equation 1:

$$v = R \cdot i + L \cdot \frac{di}{dt} \tag{1}$$

Induced voltage due to the moving permanent magnet can be neglected.

The equation of motion of the permanent magnet can be written as Equation 2:

$$m \cdot \ddot{d} = f(d,i) - m \cdot g \tag{2}$$

The position of the magnet is measured along the d-axis in upward direction as shown in **Figure 2**. Position $d = 0$ is located at the bottom of the coil. Therefore, only positions of the permanent magnet along the negative half of the d-axis are meaningful.

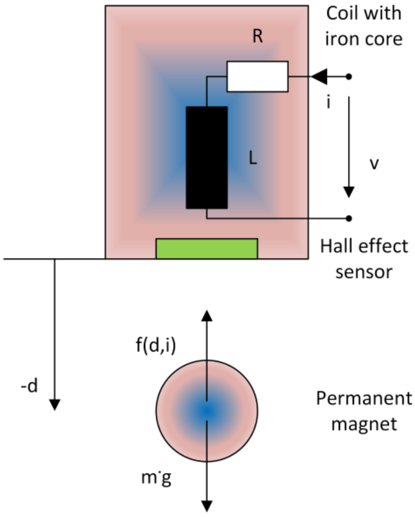


Figure 2 *Illustration of the system*

The force between coil and magnet is strongly nonlinear and either the function has to be determined by finite element simulations of the magnetic field or by measurements at the real system. Fortunately Zeltom has performed such measurements and provides a set of equations by (Zeltom 2009) - Equation 3 for the force:

$$f(d,i) = k \cdot \frac{i}{d^4} \tag{3}$$

Unfortunately Equation 3 doesn't take into account the force between the permanent magnet and the iron core even in absence of coil current $i = 0$. Although the influence on control around the equilibrium is small, the equation can be enhanced as shown in Equation 4:

$$f(d,i) = k' \cdot \frac{i_C + i}{d^4} \tag{4}$$

Equation 4 introduces a new parameter i_C which describes the force between the permanent magnet and the iron core and adapts parameter k such way that the original equilibrium at position d_0 with coil current i_0 described by Equation 5 remains unchanged:

$$k \cdot \frac{i_0}{d_0^4} = k' \cdot \frac{i_C + i_0}{d_0^4} = m \cdot g \tag{5}$$

Specifying the position d_C where the force between the permanent magnet and the iron core just meets the gravitational force (Equation 6) allows calculation of the new parameters (Equations 7 and 8):

$$k' \cdot \frac{i_C}{d_C^4} = m \cdot g \tag{6}$$

$$k' = k \cdot \left(1 - \frac{d_C^4}{d_0^4}\right) \tag{7}$$

$$i_C = \frac{m \cdot g}{k'} \cdot d_C^4 \tag{8}$$

The output of the Hall effect sensor has been approximated by Zeltom in Equation 9:

$$e = \alpha + \frac{\beta}{d^2} + \gamma \cdot i \tag{9}$$

This voltage e will be corrupted by some noise. The influence of this effect can easily be investigated by adding a noise signal from `Modelica.Blocks.Noise` as described by (Klöckner 2014).
The parameters of the system are summarized in **Table 1** and taken from (Zeltom 2009) as well as (Thiele 2019).

Equilibrium is investigated with a stationary model, setting all derivatives in Equations 1 and 2 to zero, additionally exploiting Equation 4 (and 9, if the output of the Hall effect sensor is of interest).

Figure 3 compares the steady-state characteristic $i_0 = f(d_0)$ according to Equation 3 (blue) and 4 (red, dashed). Bear in mind that equilibrium for these points of operation is unstable without appropriate control.

Table 1 *Parameters of the system*

Parameter	*Value*	*Unit*
R	2.41	Ω
L	15.03	mH
m	3.02	g
k	$1.731 \cdot 10^{-8}$	$\mathrm{N} \cdot \frac{m^4}{A}$
d_0	−2	cm
→ i_0	273.75	mA
d_C	−1	cm
→ i_C	18.25	mA
→ k'	$1.623 \cdot 10^{-8}$	$\mathrm{N} \cdot \frac{m^4}{A}$
α	2.48	V
β	0.292	$mV \cdot m^2$
γ	0.48	S

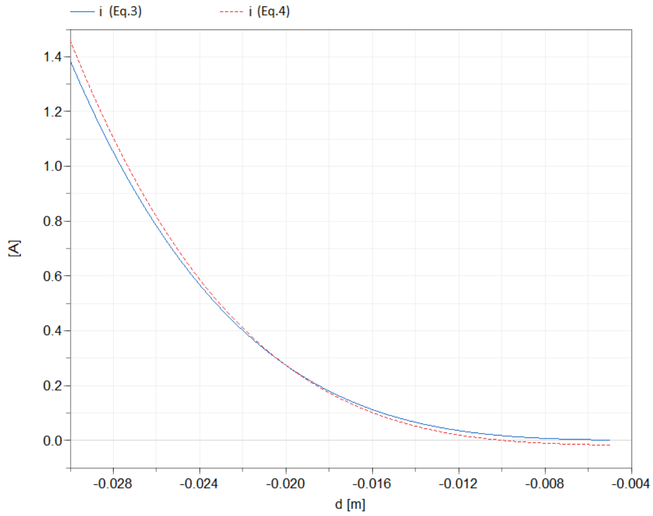


Figure 3 *Steady-state characteristic of the system*

2 Controller Design

Splitting the system into three control loops allows simple and stable control, similar to an electric drive:

- A series connection of resistor and inductor, with or without induced voltage (Equation 1).
- The current causes force (Equation 4). The force accelerates the mass (Equation 2). Integral of acceleration gives velocity.
- Integral of velocity gives position.

Table 2 *Comparison between DCPM and MagLev*

DCPM	*MagLev*
$v_A = R_A \cdot i_A + L_A \cdot \frac{di_A}{dt} + v_i$	$v = R \cdot i + L \cdot \frac{di}{dt}$
$v_i = k\phi \cdot \omega$	$v_i \approx 0$
$\tau = k\phi \cdot i_A$	$f = f(i,d)$
$J \cdot \frac{d\omega}{dt} = \tau - \tau_L$	$m \cdot \ddot{d} = f - m \cdot g$
$\omega = \frac{d\varphi}{dt}$	$d = \int \dot{d} \cdot dt$

DOI
10.3384/ecp204763

Table 2 shows a comparison between a DC machine excited by permanent magnets and the magnetic levitation system under investigation. Despite the fact that the force formula is nonlinear and the velocity is not directly accessible, the equations are equivalent.

According to (Schröder 2020) such a system can be controlled using cascaded control. **Figure 14** shows a block diagram of the whole system including control.

2.1 Current Controller

Input: Current control error
Output: Reference voltage
Taking into account the dead time caused by the time discrete communication between controller and hardware based on switching frequency $T_d = 1/f_{switch}$, the controllers can be designed as for continuous control. The dead time is replaced by the first element of its series expansion, i.e. a first order delay.
Optimal control for the innermost control loop results in a PI controller, parameterized according to the magnitude optimum (Dierk Schröder 2020, Equation 10 and 11):

$$k_P = \frac{L}{2 \cdot T_d} \tag{10}$$

$$T_I = \frac{L}{R} \tag{11}$$

The output of the controller is limited to the source voltage (battery), therefore an anti-windup measure has to be implemented.
The transfer function of the closed loop can be approximated by a first order delay with substitute time constant $T_{sub} = 2 \cdot T_d$.

2.2 Speed Controller

Input: Speed control error
Output: Reference force → Reference current
Assume the actual position d is known, we can establish a transformation from force to current inverting Equation 4. The output of the speed controller is limited by a force dependent on the maximum admissible current. Therefore an anti-windup measure has to be implemented. Using the actual position d, the maximum admissible force can be calculated from the maximum admissible current using Equation 4.
Optimal control of the speed control loop results in a PI controller, parameterized according to the symmetrical optimum (Dierk Schröder 2020, Equation 12 and 13):

$$k_P = \frac{m}{2 \cdot T_{sub}} \tag{12}$$

$$T_I = 4 \cdot T_{sub} \tag{13}$$

Furthermore, a low-pass filter for the reference speed has to be implemented. This fulfills the physical law that a step in speed would require infinite force.

2.3 Position Controller

Input: Position control error
Output: Reference speed
A simple P-controller should be sufficient, since the speed controlled system subsequently integrates speed to position, which guarantees accuracy without permanent deviation between reference and actual position. It is possible to define an upper limit for the proportional gain (Equation 14) to avoid overshooting position. In reality we will have to reduce proportional gain to find the optimal setting.

$$k_P \leq \frac{1}{16 \cdot T_{sub}} \tag{14}$$

2.4 Observer

Since neither position nor velocity is measured directly, both of them have to be calculated from the output of the Hall effect sensor, using some sort of observer. Using the measured current, we can evaluate Equation 15 after inverting Equation 9, neglecting noise:

$$d = -\sqrt{\frac{\beta}{e - \alpha - \gamma \cdot i}} \tag{15}$$

Velocity is the derivative of position.

2.5 Time Discrete Control

According to (Latzel 1995), the controller blocks may be transformed from a continuous version to a time discrete version without changing the parameterization, as long as the sample period – which is chosen as the inverse of the switching frequency – is short compared to the system's time constants. The shortest time constant is the coil's time constant. Using a switching frequency of $1\,kHz$ (or higher) this constraint (Equation 16) is fulfilled:

$$\frac{1}{f_{switch}} = 1\text{ ms} \ll \frac{L}{R} = 6.23\,ms \tag{16}$$

All controller tasks are triggered once per sample period, but additionally the desired sequence has to be kept:

- A/D-conversion (sample)
- Position controller
- Speed controller
- Current controller
- D/A-conversion (hold)

Two versions of the control blocks have been implemented, a triggered and a clocked one.

The triggered blocks (**Figure 15**) use slightly time shifted triggers to guarantee the order of execution of the blocks.

The clocked blocks (**Figure 16**) based on Modelica.Clocked (Otter 2012) are executed within the same clock partition. The order of execution relies on the tool, sorting the blocks according to the signal flow.

3 The MagLev Library

The library developed by the author contains examples, components, DC/DC-converter and control blocks. It is available on github under the BSD 3-Clause Revised License:
https://github.com/AHaumer/MagLev
The complete system model is depicted in **Figure 15** and **Figure 16**. The discrete block `e2d` calculates position from the output of the Hall effect sensor and the actual sampled current (Equation 15) and performs a time discrete differentiation of the position to obtain velocity.
The discrete block `adda` samples sensed and holds actuating variables with $sampleTime = \frac{1}{f_{switch}}$:

- ← Source voltage
- ← Coil current
- → Reference voltage

When the control part (grey background in **Figure 15**) communicates with a co-simulation FMU of the system (light blue background in **Figure 15**) this block can be omitted.
All parameters are summarized in a parameter record which also calculates steady state position and controller parameter. This ensures consistent parameterization of all components and makes it easy to switch to a different system, e.g. Zeltom offers additionally a MagLevPlus system with an enhanced coil providing higher force acting on the magnet.

3.1 Components

This sub-library contains the coil and the magnet, modeled in an object-oriented style.
The coil has an additional translational mechanical connector to determine the magnet's position and provide the force caused by the magnetic field of the coil current, as shown in **Figure 4**. This force is calculated according to Equation 4. Additionally, Equation 9 is evaluated to determine the output signal of the Hall effect sensor, including noise which is modeled using `Modelica.Blocks.Noise` (Klöckner2014).
The magnet model is a simple point mass whose vertical acceleration is determined by the magnetic force provided by the translational mechanical connector and the gravitational force.

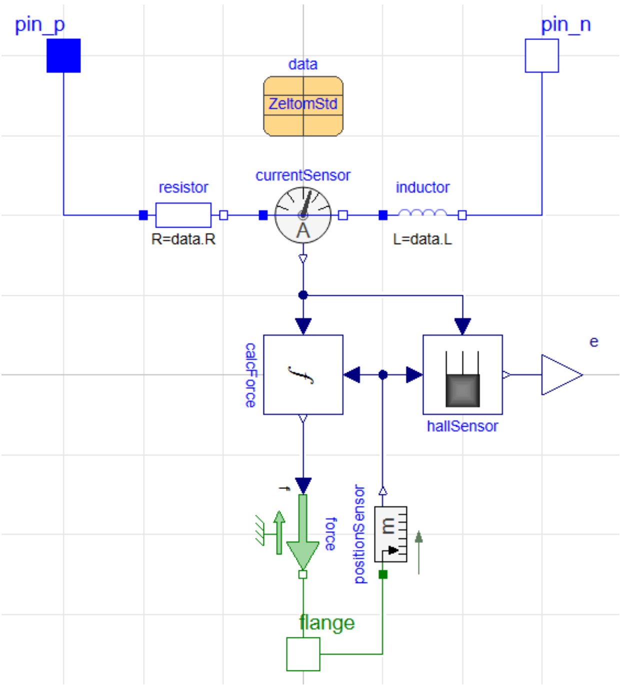


Figure 4 Model of the coil

3.2 DC/DC-Converter

The DC/DC-converter is provided as an averaging implementation and a switching version. The averaging version avoids switching effects and provides higher performance. For a detailed proof of concept the switching version is used.
The averaging version shown in **Figure 5** just prescribes the reference voltage to the output and pulls current from the input such way that power on both sides is equal. This is achieved with a fast integral controller. Of course reference voltage is limited between actual source voltage and zero in case of a buck converter.

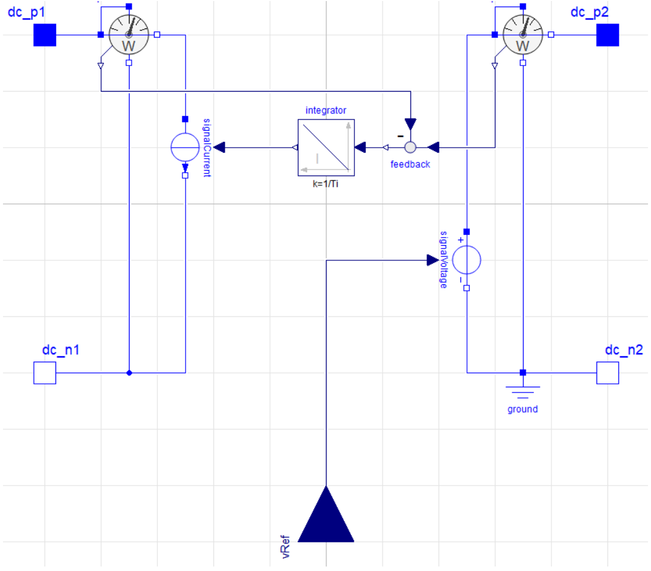


Figure 5 Averaging DC/DC-converter

The switching H-bridge shown in **Figure 6** has the same functionality as the model implemented in `Modelica.Electrical.PowerConverters.DCDC` but modeled with a different layout. It is used as a buck converter. The `SignalPWM` is taken from `Modelica.Electrical.PowerConverters.DCDC` but is adapted by replacing the sawtooth PWM reference signal by a triangular one. This allows to sample the current at the beginning (or in the middle) of the switching period, finding the average of the current with good accuracy as shown in **Figure 7**. The block C in the controller time line designates the time span to execute the control algorithm.

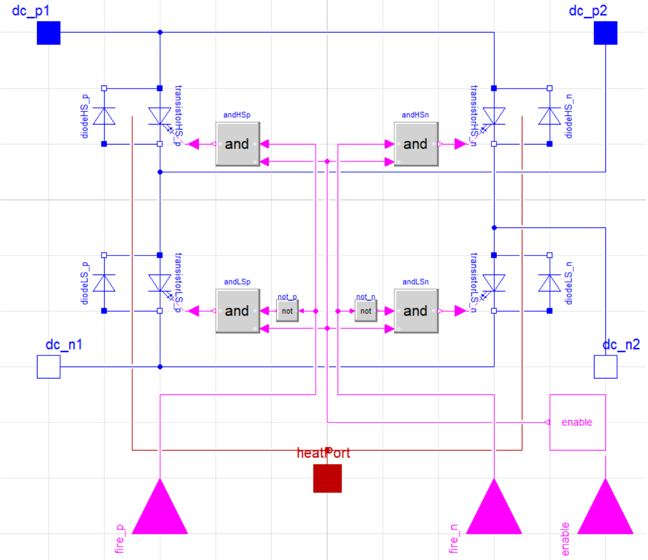


Figure 6 Switching DC/DC-converter (H-bridge)

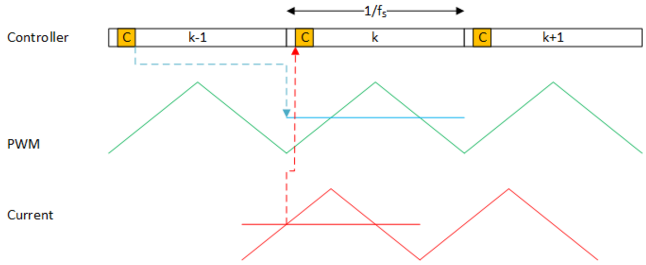


Figure 7 Timing: Switching period and PWM

3.3 Control Blocks

For the anti-windup measure necessary in both the continuous and time discrete PI controller, two solutions have been implemented:

- Back-Calculation: The difference between limited and unlimited output multiplied by the inverse of the proportional gain is subtracted from the integrator's input.
- Clamped: The integral part is stopped when the output exceeds the limit.

The observer block `e2d` implements Equation 15 and takes an approximate derivative.

The continuous version of the limited PI controller is taken from `Modelica.Electrical.Machines.Examples.ControlledDCDrives.Utilities` with some enhancements. The equations of the continuous PI controller do not avoid iteration.

The time discrete version implements an equivalent algorithm as shown in **Listing 1**, using an explicit (forward) Euler. Iteration is avoided by calculating a prediction of the unlimited output. This function is called both from the triggered block and the clocked block within when-clauses.

Listing 1 *Time discrete PI controller*

```
function piStep
  input Real u "Reference signal";
  input Real u_m "Measured signal";
  input Real kp "Proportional gain";
  input SI.Time Ti "Integral time constant";
  input SI.Time Ts "Sample period";
  input Real kFF "Gain of feed-forward";
  input Real ff "Feed-forward signal";
  input AntiWindup antiWindup;
  input Real yMin "Lower limit of output";
  input Real yMax "Upper limit of output";
  input Real pre_x "Previous state";
  output Real x "State";
  output Real y "Result";
protected
  Real e "Control error";
  Real predict "Prediction of output ";
  Real cropped "Cropped part of output";
algorithm
  e := u - u_m;
  predict := kp*e + kp*(pre_x + Ts/Ti*e) +
    kFF*ff;
  cropped := predict -
    min(max(predict, yMin), yMax);
  x := pre_x + Ts/Ti*
    (if antiWindup ==  BackCalc then
      (e - cropped/kp)
    else (if abs(cropped) > small then 0
      else e));
  y := min(max(kp*e+kp*x+kFF*ff,yMin),yMax);
end piStep;
```

4 Simulation Results

Simulation of the time discrete position controlled system shows satisfactory results:

In **Figure 8** the initial position of the magnet is $d_0 = -2\ cm$, the initial velocity is zero. The reference position is given by a trapezoid which is nearly a pulse series. The real position of the magnet follows the reference position very well. The observer's result `e2d.d` shows only negligible deviation from the real position taken from the magnet model.

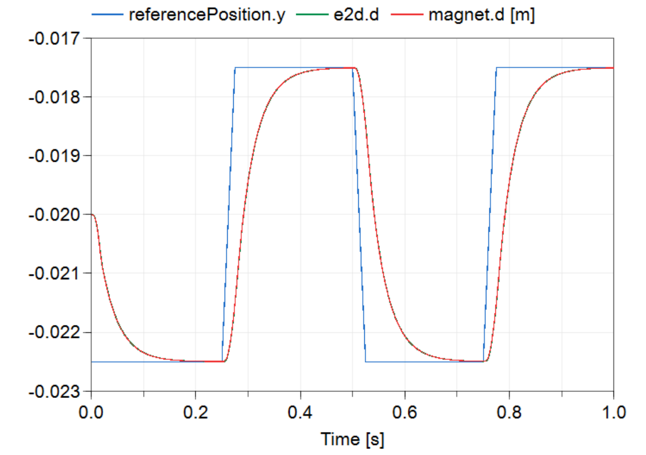


Figure 8 Time discrete position control: Positon trajectory

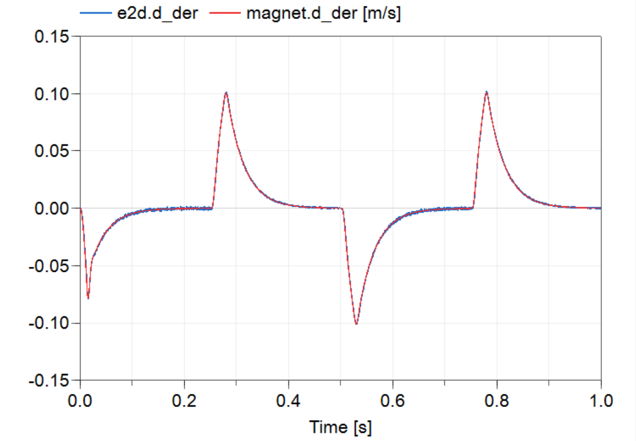


Figure 9 Time discrete position control: Speed trajectory

Figure 9 compares velocity obtained in the observer block with velocity in the magnet model. The influence of the noise added to the output signal of the Hall effect sensor can be seen at the trajectory of the velocity signal `e2d.d_der` reconstructed by the observer. Thus it is possible to estimate the impact of noise on the control performance.

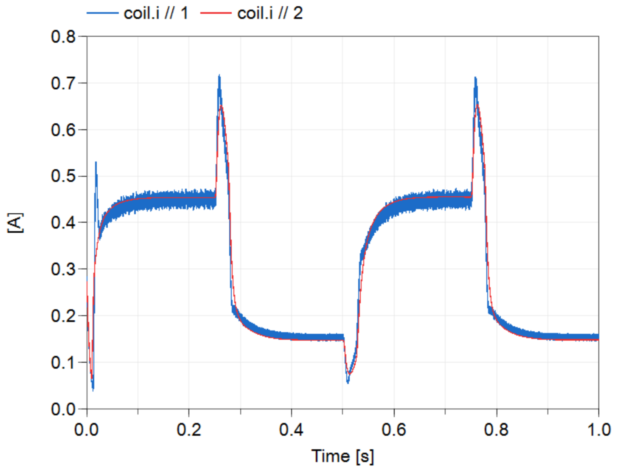


Figure 10 Coil current of discrete and continuous version

The coil current simulated with discrete control and switching DC/DC converter is shown in blue in **Figure 10**. The same figure shows the coil current obtained with continuous control and averaging DC/DC converter in red. The discrete results have been obtained using a switching frequency $f_{swtch} = 2.5\,kHz$. For the continuous results additionally noise of the Hall sensor signal has been set to zero.

Differences between the two trajectories:

- Switching versus averaging DC/DC-converter
- Time discrete versus continuous control
- Sample/hold versus first order delays
- Absence of noise in the continuous model

The continuous model neglects some effects but is capable of being simulated in real-time. It shows a good estimation of the behavior under control and the energy consumption from the source. The energy consumption over the shown cycle differs only by 0.6% from the discrete version.

Both the triggered and the clocked version of the discrete models provide a more detailed insight in the systems behavior, but the triggered version takes 7.5-times and the clocked version 6-times the as long as the continuous version to simulate.

5 Controller and Power Electronics

Zeltom's printed circuit board (No. 4 in **Figure 1**) is replaced by a rapid prototyping system available in the lab:

The TI F28069M LaunchPad shown in **Figure 11** contains a TI C2000 µC and can be programmed either using a software tool provided by TI or utilizing Simulink.

The latter one has the advantage that the system model can be exported from Modelica as functional mockup unit. This FMU is subsequently imported to Simulink, the control algorithm can get thoroughly tested under realistic conditions before downloading it to the µC.

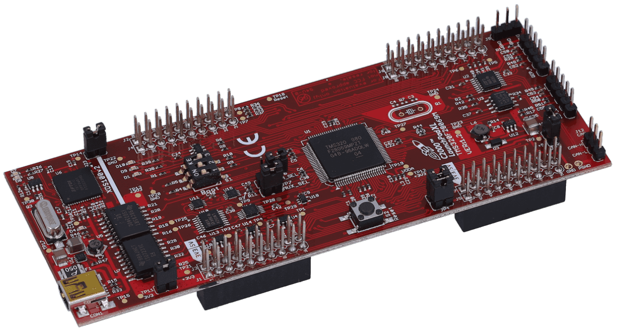


Figure 11 TI 28069M LaunchPad https://www.ti.com/tool/LAUNCHXL-F28069M

The TI BoosterPack DRV8848 shown in **Figure 12** provides two H-bridges suitable for $4..18\,V/1\,A\,RMS$. One half of one of the two H-bridges is used as a step-down converter to supply the coil with variable voltage.

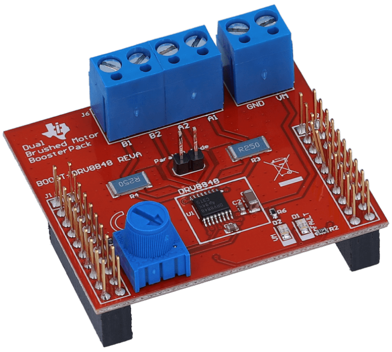


Figure 12 *BoosterPack DRV8848*
https://www.ti.com/tool/BOOST-DRV8848

For sure this choice of controller and DC/DC converter is overkill, but it is available in the lab and the workflow is well documented.

6 Conclusions and Outlook

This project proves the advantages of using Modelica, especially for educational purposes:

1. Develop a clearly arranged open source model of the physical system using Modelica.
2. Design a control concept, testing it with Modelica.
3. Export a functional mockup unit of the physical system from Modelica.
4. Import the FMU to Simulink. Implement the controller in Simulink and test it acting on the FMU.
5. Download the control algorithm to the embedded controller, proving the control concept in reality.

Steps 1 – 3 can be performed by a teacher, preparing a students' project.

A physical model of the magnetic levitation system has been implemented, as well as both averaging and switching versions of a DC/DC-converter.

For the concept of control a conventional cascaded control with current controller, speed controller and position controller has been chosen. Final simulations show satisfactory results.

Furthermore, the system model can be exported as functional mockup unit shown in **Figure 13**. This FMU can be imported in Simulink.

It is a students' project to implement the control structure in Simulink, subsequently downloading the control algorithm to the embedded controller. Thus the concept can be proven: The magnet should follow the reference position signal.

Since the students' project is ongoing work, it is not sure that the magnetic levitation system with the new control board can be presented at the conference, but following the results of the Modelica simulations it can be expected to be a success.

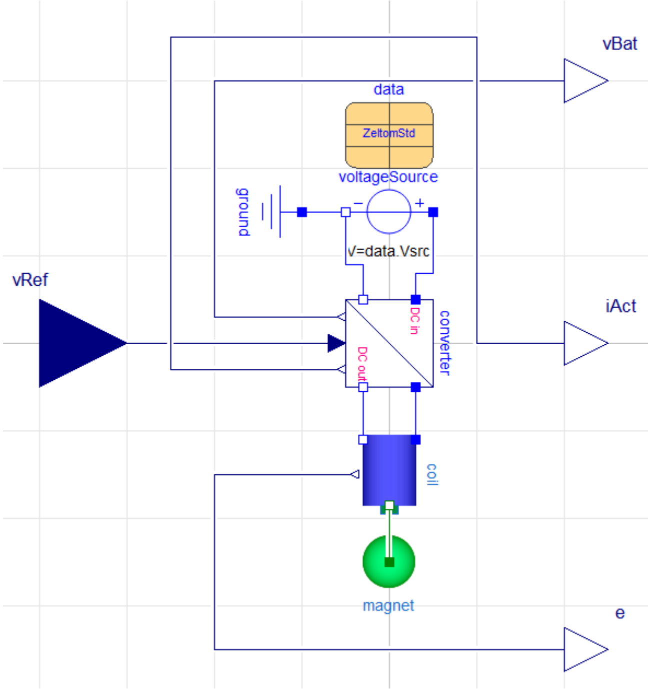


Figure 13 *Functional mockup unit of the system*

Acknowledgement

Without the discussions with Martin Otter it wouldn't have been possible to develop the discrete control blocks based on `Modelica.Clocked`.

References

Hilding Elmqvist, Martin Otter, Sven-Erik Mattsson (2012). "Fundamentals of Synchronous Control in Modelica". Proceedings of the 9th International Modelica Conference 2012. DOI:10.3384/ecp1207615.

Andreas Klöckner, Franziskus L. J. van der Linden, Dirk Zimmer (2014). "Noise Generation for Continuous System Simulation". Proceedings of the 10th International Modelica Conference 2014, pp. 837-846. DOI 10.3384/ecp14096837.

Wolfgang Latzel (1995). "Einführung in die digitalen Regelungen" (in German). VDI. ISBN 978-3-642-95779-6.

Martin Otter, Bernhard Thiele, Sven-Erik Mattsson (2012). "A Library for Synchronous Control Systems in Modelica". Proceedings of the 9th International Modelica Conference 2012. DOI:10.3384/ecp1207627.

Dierk Schröder (2020). "Elektrische Antriebe – Regelung von Antriebssystemen" (in German). 5th Edition. Springer Vieweg. ISBN 978-3-662-62700-6.

Bernhard Thiele, Bernt Lie, Martin Sjölund, Adrian Pop, Peter Fritzson (2019). “Controller Design for a Magnetic Levitation Kit using OpenModelica’s Integration with the Julia Language”. Proceedings of the 13th International Modelica Conference 2019, pp. 303-311. DOI 10.3384/ecp19157303.

Zeltom LLC (2009). “Electromagnetic Levitation System - Mathematical model”. URL: https://www.zeltom.com/documents/emls_md.pdf

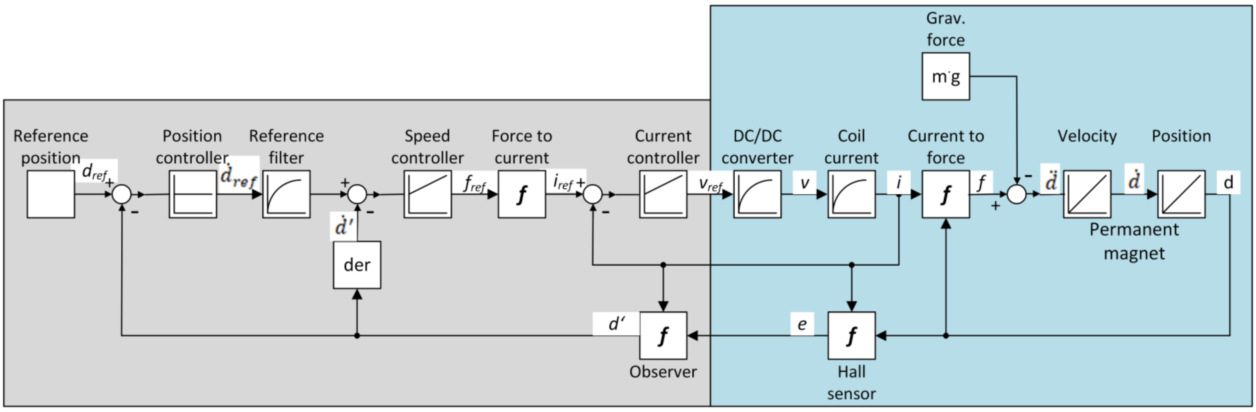


Figure 14 Block diagram of the system

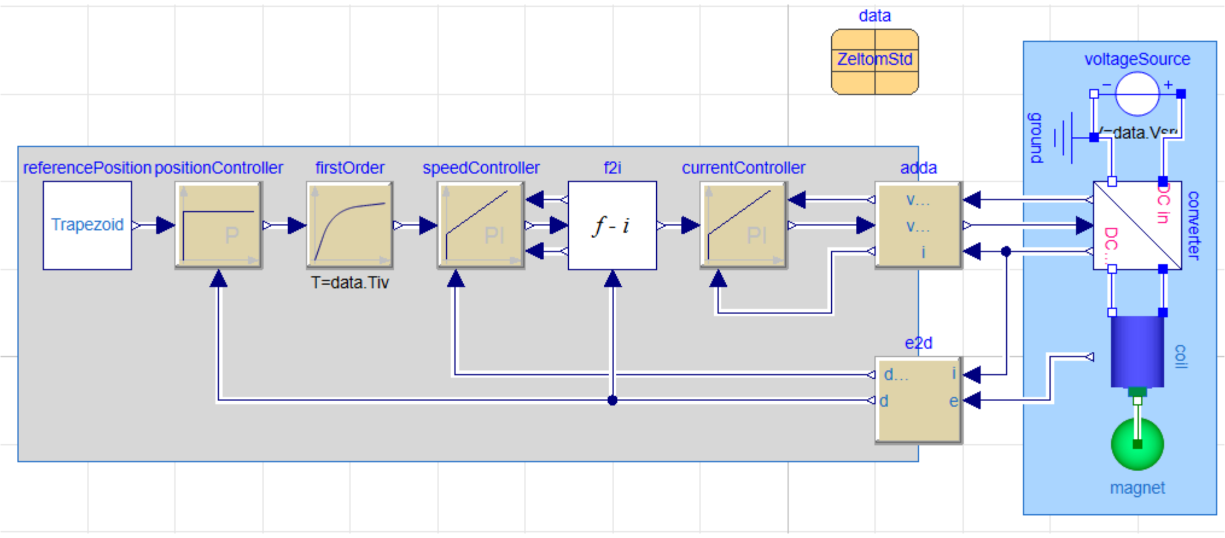


Figure 15 Complete System Model using triggered blocks

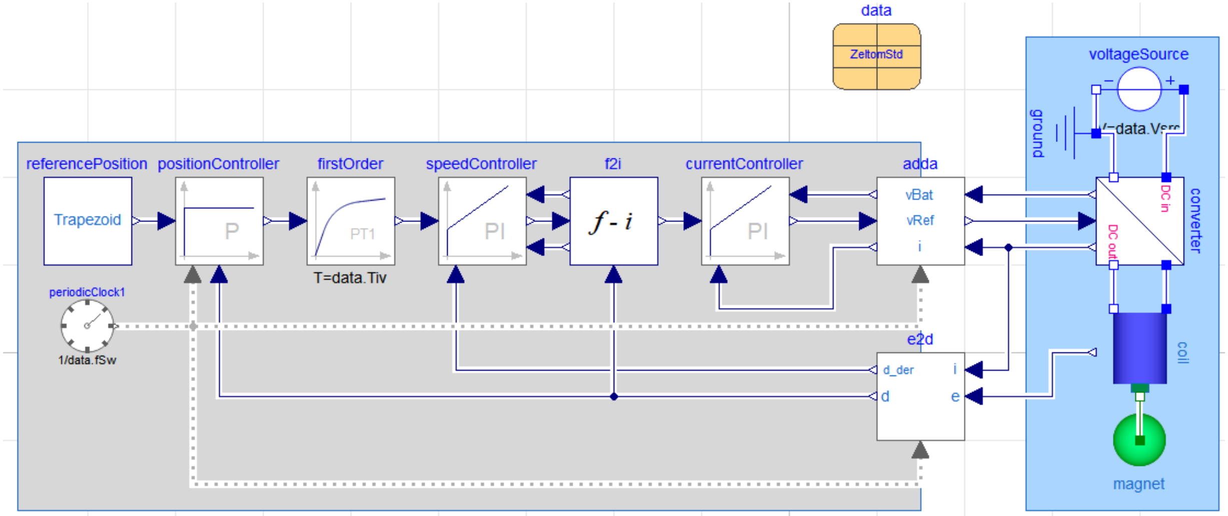


Figure 16 Complete System Model using clocked blocks

Coupling of Thermal and Electrical Systems for the Simulation of ECS Architectures

Nicolás Ablanque Mejía[1] Sriram Karthik Gurumurthy[2] Santiago Torras Ortiz[1] Antonello Monti[2]
Joaquim Rigola[1] Carles Oliet[1]

[1]Universitat Politècnica de Catalunya - Barcelona Tech (UPC), Heat and Mass Transfer Technological Center (CTTC), Spain, {nicolas.ablanque, santiago.torras, joaquim.rigola, carles.oliet}@upc.edu
[2]ACS, EONERC, RWTH Aachen University, Germany, {sgurumurthy, amonti}@eonerc.rwth-aachen.de

Abstract

This work is focused on the coupling of two complex models based on different underlying physics: a vapor compression refrigerating system and its electrical drive system. The main challenge was to correctly handle the large simulation time constant difference which is three orders of magnitude smaller for the electrical system. The two models have been originally developed following very specific requirements (i.e. high numerical robustness and low time consumption) for their suitable use in simulations of large and complex aircraft Environmental Control Systems (ECS). The direct coupling of both systems has been observed to cause numerical instabilities, therefore, a coupling approach based on non-invasive dynamic relaxations has been implemented. The resulting combined simulations have shown to be numerically stable for the complete range of operating conditions and for a wide range of time steps.

Keywords: Multi-physics, Vapor compression system, Electrical drive system, Systems coupling

1 Introduction

The aviation industry has witnessed significant advancements in recent decades, supported by remarkable innovations. While aviation has made substantial contributions to society, certain critical issues resulting from the growth of commercial air travel need attention. The foremost concern is its impact on climate change, primarily due to the release of CO2 emissions. To tackle these challenges, the industry has embraced a strategy that involves replacing mechanical, hydraulic, and pneumatic systems with electrically driven systems. This shift has given rise to the term "More Electric Aircraft" (MEA) within the industry (Sarlioglu and Morris 2015). The ECS is responsible for maintaining cabin temperature and pressure. It traditionally uses the bleed air from the main engines for its operation. However, under the MEA paradigm, the ECS has undergone a transition to be electrically fed (Sarlioglu and Morris 2015).

The novel ECS architectures within the MEA framework may include a Vapor Compression System (VCS) powered by an electrical motor drive with the aim to provide higher efficiency additional cooling power to the cabin. In this paper, the combined simulation of the two aforementioned systems is addressed. On the one hand, the VCS system consists of a single-stage compression configuration including a centrifugal compressor, an air-to-refrigerant condenser (to eject heat into the aircraft ram air ducts), and a refrigerant-to-liquid evaporator (to indirectly extract heat from the cabin). This VCS provides additional cooling power to the cabin. It can be turned on and off during the flight according to the cabin cooling needs. The model developed to simulate the VCS system has been introduced in a previous work (Ablanque et al. 2023). On the other hand, the electrical system consists of a three phase inverter which drives a Permanent Magnet Synchronous Motor (PMSM). The Field Oriented Control strategy (FOC) has been adopted for the control of the PMSM drive (Irwin 1997). The PMSM drive is equipped with speed and torque control. The speed requested by the VCS is delivered by the electrical system.

In this paper, the goal is to successfully couple the thermal and electrical systems and conduct numerous simulation tests to validate the correct operation of the system. Discussions about the coupling strategy of two systems, the time steps adapted for the simulations and the results of the simulations are provided. Since the angular speed of the PMSM and VCS are different, the shafts of the two system are coupled through a gear box. The two systems are based on different domains of physical laws and ideally require different time steps for optimal simulation. The electrical system typically requires very small time steps for Electro-Magnetic Transient (EMT) simulations (in the range of 1 to 100 μs) while the VCS time step is significantly larger (about 1 second). The electrical system is modelled in the dynamic phasor domain instead of EMT domain for increasing the time step to the range of 1 second such that it is compatible with the VCS system (Gurumurthy et al. 2022; Loka et al. 2022; Demiray 2008). To avoid non-smooth changes of torque from the VCS side and non-smooth changes of speed from the electrical side, a non-invasive dynamic relaxation strategy has been introduced. The dynamics of the individual thermal or electrical models are not internally modified with this approach. It consists in numerically relaxing the torque

DOI
10.3384/ecp204771

and speed values exchanged across the mechanical flanges of the two systems. The coupled VCS-PMSM drive system is tested for various scenarios such as initial switch-on of the system, various speed reference changes, and more importantly, the starting-up and switching-off of the two systems while the simulation is being conducted. This paper also analyzes the computation CPU time of the model for various time steps to show the computational effectiveness of the proposed thermal-electrical system.

The paper is organized as follows: Section 2 describes the VCS and its testing procedures, followed by numerical simulation results of the VCS. The electrical system description, testing procedure and numerical simulation results are carried out in Section 3. The coupling strategy of the thermal-electrical system, the multi-physics simulation results, and the computational burden evaluation of the coupled simulations are carried out in Section 4. Finally, the conclusions and future work are presented in Section 5.

2 Vapor Compression System Model

2.1 Description

The vapour compression system model represents a typical single-stage configuration with four fundamental components: a centrifugal compressor, a refrigerant-to-air condenser, an expansion device, and a refrigerant-to-liquid evaporator. Additional minor components are also considered in the model such as connecting pipes, a reservoir, a by-pass valve and a super-heating sensor. Figure 1 shows the model internal scheme.

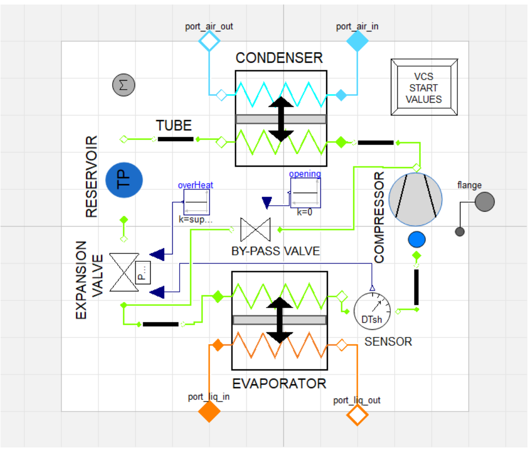


Figure 1. Vapor compression system scheme.

The vapor compression system model has been developed for its use within large thermal-electrical architectures of aircraft environmental systems. It must be thermally and electrically coupled to other subsystems. More specifically, the condenser exchanges heat with a ram air circuit (through inlet and outlet air fluid connections), the evaporator exchanges heat with a liquid circuit (through inlet and outlet liquid fluid connections), and the compressor exchanges mechanical variables with the electric drive system (through a single mechanical connection).

2.2 Numerical Assessment and Tests

To successfully conduct simulations at the architecture level is particularly challenging due to many reasons such as the high number of components being solved, the high number of interactions between systems, the different physics being considered, and the necessity of low computing time consumption. Therefore, to prevent resolution issues, the vapor compression system model has been developed to fulfill demanding numerical robustness requirements and has been subjected to comprehensive series of tests focused on achieving the following aspects:

- The model must initialize correctly independently of the boundary condition values (the values at the boundary conditions are shared with other systems and they can vary dramatically during the initialization calculations).
- The model must be correctly simulated independently of the main simulation set-up parameters such as the stop time and the interval length (the set-up is common to all architecture systems).
- The model must be robust for all possible combinations of fluid boundary types. The fluid connections can be defined from the pressure values at both ends (P-P), or alternatively, from the pressure and the mass flow rate values at opposite ends (M-P). The model must also handle different input signal types for the boundary condition variables such as constant, step, ramp or sine.
- The vapor compression system model must be able to be turned on and/or off as many times as necessary and at any moments of the architecture simulation. In such cases, the simulation of the other systems involved should continue without being affected.
- The CPU time needed for the model to simulate dynamic and steady-state cases must be relatively low to prevent numerical bottlenecks at the architecture level (the maximum time required for steady-state simulations has been set at 10 seconds).

2.2.1 Initialization and steady-state tests

A complete set of runs has been generated to test the model robustness during initialization and the corresponding resolution time for steady-state conditions. The dataset has been built-up taking into account different values for all the boundary conditions (i.e. compressor speed, air and liquid temperatures, air and liquid mass flow rates, and air pressure) covering the whole physical range of possibilities and all its possible combinations. In addition, different fluid boundary condition types (see Figure 2) and different values for the interval length were also taken into consideration.

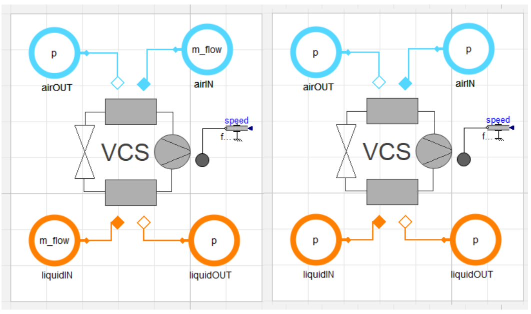


Figure 2. Boundary condition types: M-P (left) and P-P (right)

The tested runs consisted of 5400 cases (900 cases for each combination of boundary condition type and interval length). The simulation stop time was set at 2000 seconds so that the steady-state condition can be reached before. The main statistics obtained from the model simulations are summarized in Table 1.

Table 1. Initialization and steady-state convergence of VCS model: main statistics

Boundary type	Interval length [s]	Mean CPU time [s]	Failure rate [%]
M-P	4	1.40	0
M-P	2	1.58	0
M-P	1	1.86	0
P-P	4	1.42	0
P-P	2	1.59	0
P-P	1	1.88	0

The results have shown that the model initialization is successful for all the cases without being affected by the interval lengths used or the combination of boundary values. The mean simulation time calculated for all cases is 1.6 seconds which is well below the threshold of 10 seconds required (in fact no cases were found to have a simulation time above 10 seconds).

2.2.2 Start-up and shut-down tests

A complete set of cases has also been generated to test the model robustness during shut-down and start-up demanding transients. These cases are mostly based on the same combinations and characteristics defined in the previous section. However, instead of reaching a steady-state condition, the system is now subjected to consecutive shut-downs and start-ups every 1000 seconds (the stop time for simulations is now 5000 seconds).

In this case, the whole set of cases has also been successfully simulated without experiencing any numerical issue. Illustrative results for a particular case are shown in Figure 3 in terms of the VCS refrigerant mass flow rate and the CPU time consumption (the mass flow rate is presented in dimensionless form due to confidentiality reasons). It can be observed from the CPU time evolution that the solver requires additional efforts during the main dynamic events.

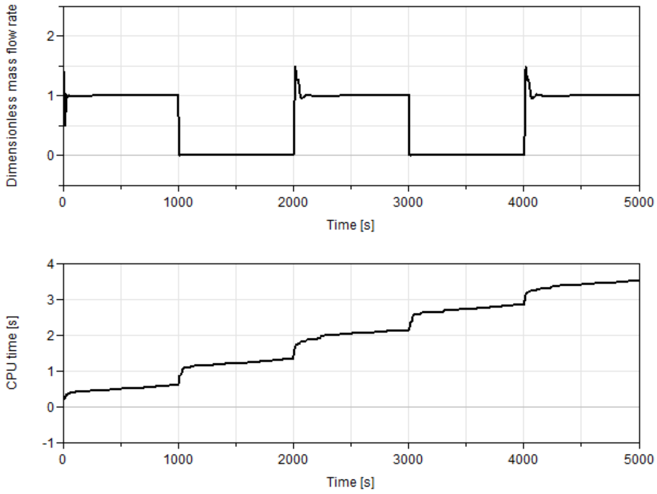


Figure 3. VCS model refrigerant mass flow rate and CPU time consumption during start-up shut-down test

2.2.3 Additional transient tests

Finally, it is worth mentioning that the model has been subjected to additional tests to ensure its robustness for different dynamic changes of the boundary condition variables based on the input signal type as shown in Figure 4 for the step and sine cases.

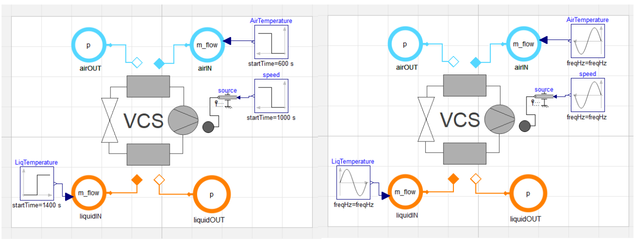


Figure 4. Boundary variable signal types: step (left) and sine (right)

The whole data-set prepared to test signals has also been successfully simulated by the model. Illustrative results for a particular case with sine signals are presented in Figure 5 in terms of the VCS compressor suction and discharge dimensionless pressures.

3 Electrical Drive System

3.1 Description

The model of the electrical system considered is shown in Figure 6. The system consists of a three-phase 2-level converter that interfaces a DC power supply and a permanent magnet synchronous motor (PMSM). The motor controller is equipped with field oriented control to track the reference speed supplied to motor controller. The motor controller achieves speed reference tracking by actively controlling the output voltage vectors of the three-phase inverter. The electrical drive system would be coupled

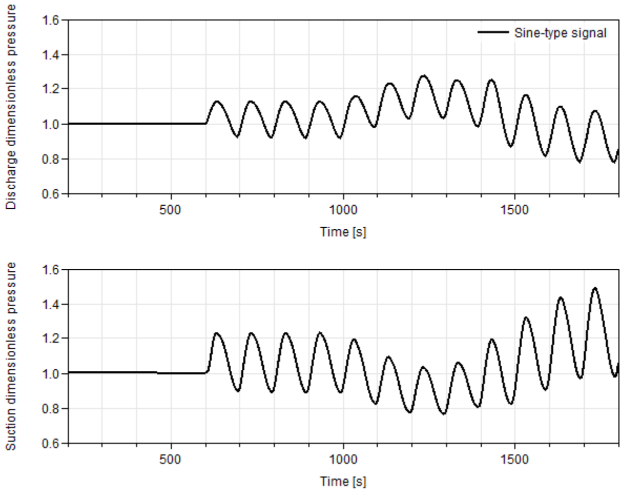


Figure 5. VCS suction and discharge pressures under transient conditions (sine signal)

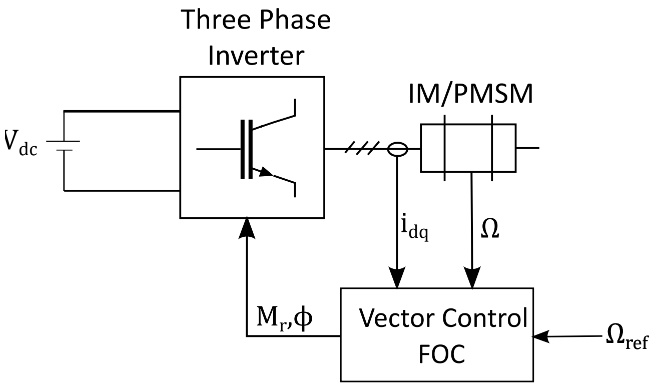


Figure 6. Electrical drive system

with the VCS system and therefore the electrical model needs to be robust and stable when load torque changes and speed changes occur. A short description of the electrical drive system is provided followed by detail simulation results to test the electrical motor.

3.1.1 PMSM

The PMSM is modelled in the dynamic phasor domain similar to (Irwin 1997; Loka et al. 2022). The flux in the stator λ_{ps} depends only on the stator currents i_{ps} since there is no electrical circuit in the rotor (Irwin 1997; Krause et al. 2013). The stator flux real part depends on the permanent magnet flux λ_f from the rotor as shown in Eq. (1).

$$\begin{bmatrix}\lambda_{ps,re}\\ \lambda_{ps,im}\end{bmatrix} = \begin{bmatrix}L_{sd} & 0\\ 0 & L_{sq}\end{bmatrix}\begin{bmatrix}i_{ps,re}\\ i_{ps,im}\end{bmatrix} + \begin{bmatrix}\lambda_f\\ 0\end{bmatrix} \tag{1}$$

L_{sd} and L_{sq} represent the direct and quadrature axis self-inductance. The stator currents are controlled by the applied stator voltage v_{ps} and electrical supply angular frequency ω_s of the three phase inverter as shown in (2).

$$\frac{d\lambda_{ps}}{dt} = v_{ps} - r_s i_{ps} - j\omega_s\lambda_{ps} \tag{2}$$

The electrical torque T_e developed by the PMSM is given by (3). The differential equation corresponding to the mechanical angular speed of the motor Ω_m is given by (4) where T_L is the load torque applied on the motor from the coupled external system , B is the coefficient of rotational friction, J is the inertia of the PMSM.

$$T_e = \frac{3}{2}p(\lambda_f i_{ps,im} + \Delta L.i_{ps,re}i_{ps,im}) \tag{3}$$

$$J\frac{d\Omega_m}{dt} = T_e - B\Omega_m - T_L \tag{4}$$

The PMSM model is included to have the electrical loss, mechanical loss and efficiency calculations.

The PMSM is controlled through field oriented control (FOC). The controller consists of three levels, the outer level is a speed controller modelled by a PI controller. The speed controller synthesizes the torque reference. By Maximum Torque per Ampere (MTPA) strategy (Irwin 1997), the reference currents are calculated. An inner current controller tracks the stator current of the PMSM by actively controlling the output voltage vectors of the inverter.

3.1.2 Three phase inverter

We consider a 2-level inverter with B6C topology for the three-phase inverter. Harmonics introduced by the three phase inverter are neglected and the three-phase inverter is modelled as a fundamental phasor. A model that considers switching harmonics can be found in (Gurumurthy et al. 2022; Holmes and Lipo 2003; Ruan et al. 2018). The dynamic phasor of the phase to neutral voltage of phase A v_{an} is given by (5).

$$\langle v_{an}\rangle_0 = \frac{M_r V_{dc}}{4}e^{j\varphi} \tag{5}$$

The voltage on the DC link is V_{dc} and the control variables are Modulation ratio M_r and phase-shift angle φ. The conduction and switching loss evaluation of the power electronic switches are performed and these calculations are included within the model (Loka et al. 2022; Bierhoff and Fuchs 2004; Acquaviva et al. 2020).

3.2 Numerical testing and simulation results

To test the numerical robustness of the PMSM drive model, several tests are conducted. The drive model needs to control the speed of the motor adhering to the speed reference command provided by the VCS system. The speed reference needs to be maintained while supplying the required load torque and power that the VCS system requires. Furthermore, when the speed reference is increased, the torque required by the VCS system also increases. This adds further dynamic constraints that needs to be taken care by the motor controller.

The following tests are proposed to validate the drive operation. Notice that the speed and torque values are referred to the electrical motor side and not to the VCS side.

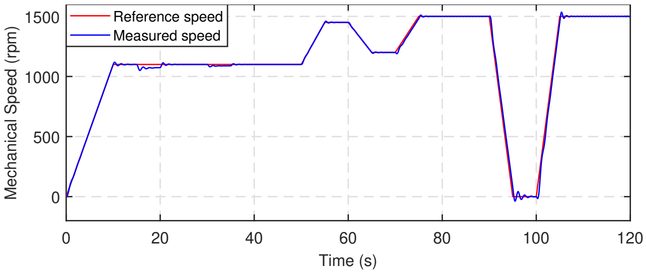


Figure 7. Comparison of reference speed and actual motor speed

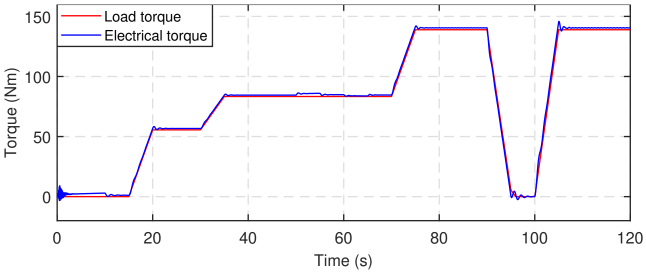


Figure 8. Comparison of applied load torque and electrical torque produced by PMSM

A specific gear ratio value had to be considered for the conversion of speed and torque values to the VCS side.

- At t = 0s to t = 10s, the motor is switched-on and accelerated to a speed reference of 1100 rpm with torque 0 Nm.
- At t = 15s to t = 20s, the torque is increased linearly to 55.5 Nm while the motor tries to maintain the speed.
- At t = 30s to t = 35s, the torque is increased linearly to 83.3 Nm while the motor tries to maintain the speed.
- At t = 50s to t = 55s, the motor is accelerated to a speed reference of 1450 rpm.
- At t = 60s to t = 65s, the motor is decelerated to a speed reference of 1200 rpm.
- At t = 70s to t = 75s, the motor is accelerated to a speed reference of 1500 rpm and simultaneously, the torque is also linearly increased to 139 Nm.
- At t = 90s to t = 95s, the motor is controlled to speed reference 0 rpm and 0 Nm torque to validate a stand-still behavior.
- At t = 100s to t = 105s, the motor is controlled to speed reference 1500 rpm and 139 Nm torque to validate the recovery from stand-still position.

Fig. 7 shows the comparison of reference speed and the actual motor speed. The start-up of the motor is smooth without oscillations and the speed reference changes are also accurately tracked without oscillations and steady state error.

During sudden load torque changes at t = 15s and t = 30s, the speed does not change drastically due to fast torque control action as shown in Fig. 8. It can be observed that the electrical torque is greater than the mechanical load torque due to frictional losses.

The power losses occurring in the PMSM are shown in Fig. 9, which are consisting of electrical losses and mechanical losses. The electrical losses are dependent on the operating currents which in-turn depend on the required torque. The mechanical frictional losses mainly depends on the operational speed of the motor. The overall peak efficiency of the PMSM is approximately 98 %.

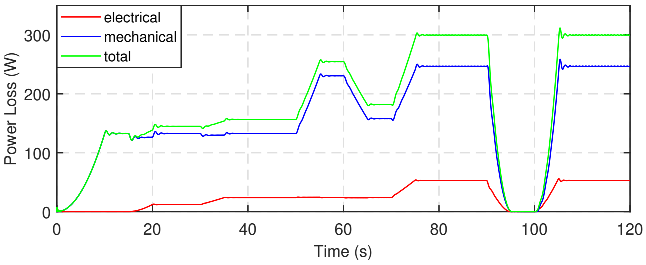


Figure 9. Power losses in the PMSM

The inverter losses consists of switching losses and conduction losses occurring in the IGBTs and the diodes. The inverter losses are shown in Fig. 10. The conduction and switching losses are functions of operational currents and voltage which are mainly dependent on the operational torque of the motor. The three phase inverter peak efficiency is approximately 97.5 %.

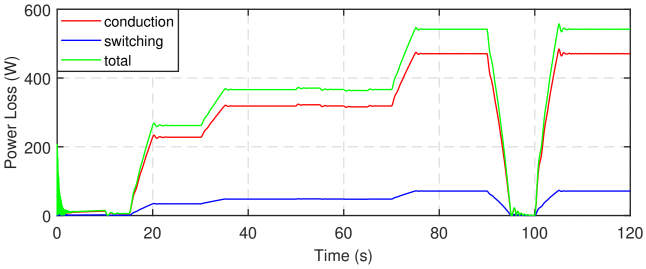


Figure 10. Power losses in the three phase inverter

4 Coupling of Thermal and Electrical Systems

4.1 Description

The VCS model has been coupled with the electrical drive system as shown in Figure 11. The VCS is based on fluid-dynamic and thermal physics while the drive system is based on electric physics. The latter system is aimed to power the compressor and to regulate its speed based on

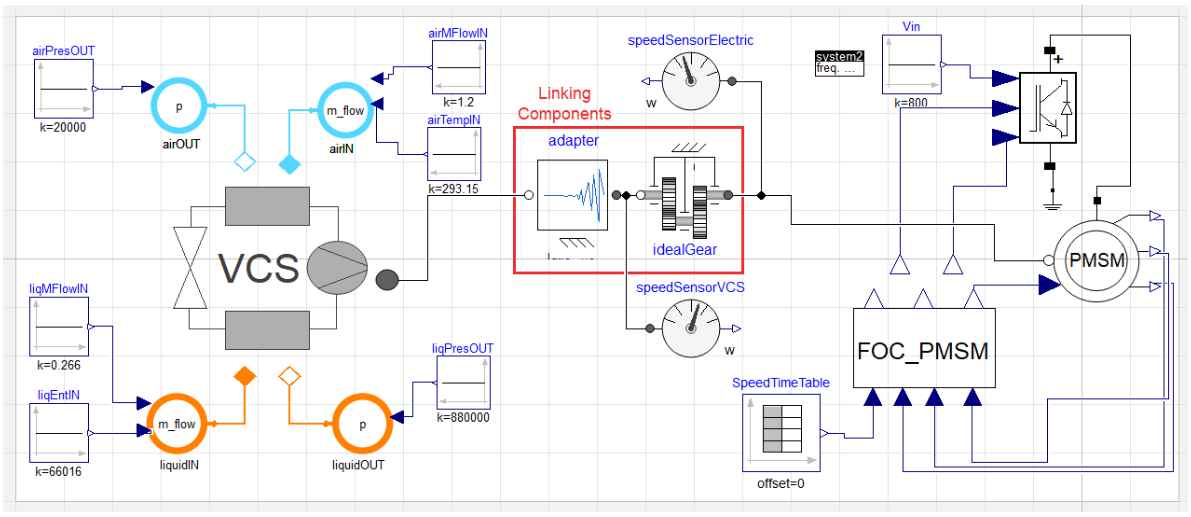


Figure 11. VCS and electrical drive system coupling scheme

an input signal provided by an external control mechanism (for the present tests the signal has been provided by appropriate source blocks). The two systems are mechanically linked so that the compressor rotating shaft parameters, namely, torque and speed, are shared by the two systems.

Two additional elements have been added to the mechanical linking. Firstly, an ideal gear component from the Modelica Standard Library has been used to match the operating characteristics of both systems. And secondly, due to uncontrolled oscillations observed when direct mechanical coupling was conducted, an additional non-invasive component integrating dynamic relaxation equations has been added. This component acts as a numerical hinge for the solver to avoid unstable numerical iterations when steep changes are experienced by both shared values: the torque and the speed. The aforementioned connecting element reduces the torque that the electrical system receives, and similarly, it moderates the speed that the compressor of the thermal system receives. In this way, rapid changes, such as start-ups or shut-downs are dampened, and the resolution does not collapse. The equations used are as follows:

$$\frac{d\phi}{dt} = \frac{(\phi_{aux} - \phi)}{\alpha_\phi} \tag{6}$$

$$\frac{d\tau}{dt} = \frac{(\tau_{aux} - \tau)}{\alpha_\tau} \tag{7}$$

Where ϕ and τ are the rotation angle and the torque in the coupling shaft, respectively. The time constant for relaxation, α, is used to avoid numerical instabilities and can be modified by the user.

4.2 Simulation

4.2.1 Expected performance

The simulations have been conducted in order to test the system capability and to achieve the following characteristics:

- Robustness during the initialization (i.e. resolution at time step zero) which is usually a critical numerical aspect for the solver due to unknown values of the variables at the previous time step.
- Ability to conduct simulations at different fixed time steps (i.e. interval length defined in the simulation set-up). This characteristic is also crucial as the two systems studied herein are intended to be simulated together with all the other thermal and electrical systems of the complete ECS architecture.
- Robustness to handle start-up and complete switch-down conditions during the simulations. This numerically challenging feature is necessary due to the intermittent use of the VCS within the ECS operation.
- Low CPU resolution time to reduce the impact on the time consumption for the whole ECS simulation. This will allow real time based simulations and also to carry out simulations of large data-sets for design, prediction and/or control studies.

4.2.2 Results

The simulation baseline case to test all the aforementioned requirements is described as follows: an initial start-up procedure (time = 0 s), followed by a rapid complete shut-down of the system (at time = 500 s), then a rapid start-up again (at 1500 s), and finishing at stop time of 3000 s. The solver used for all cases was the default integration method DASSL.

Figure 12 shows the compressor rotating speed evolution for the baseline case dynamic simulation. The figure shows both the compressor speed value for the VCS and the rotational speed value provided by the electrical drive in order to see the impact of the mechanical relaxation applied to the systems coupling. Similarly, Figure

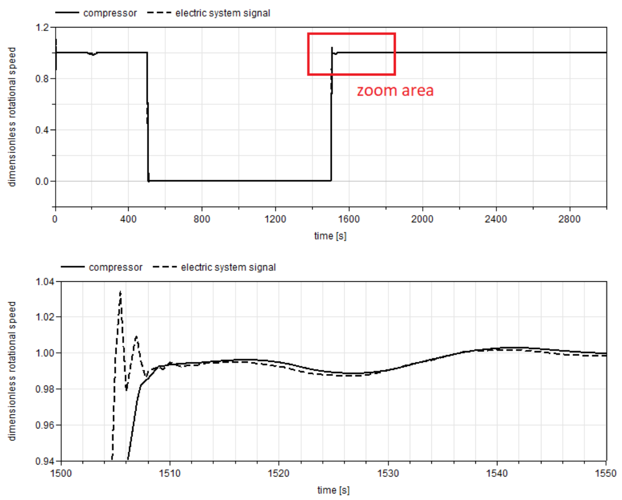


Figure 12. Rotational speed evolution. Top: whole simulation period. Bottom: zoom area in a start-up event

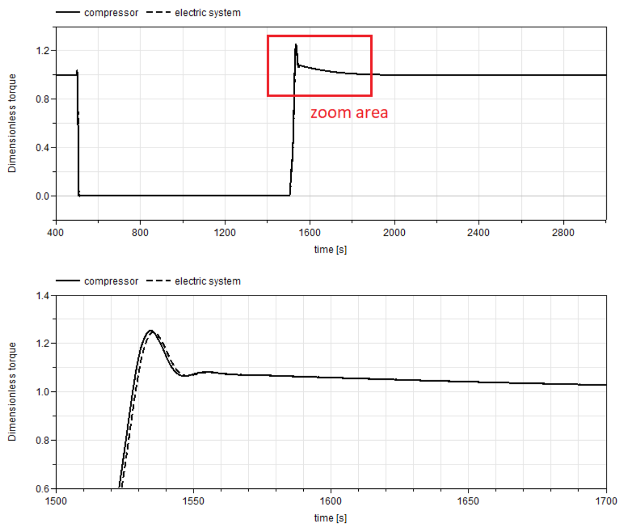


Figure 13. Torque evolution. Top: whole simulation period. Bottom: zoom area in a start-up event

13 shows the compressor torque evolution. The values of both shared variables, namely, the rotational speed and the torque, are practically the same for both systems during the whole simulation except for the moment of fast dynamic changes. The dynamic relaxations applied at the mechanical coupling have proven to provide numerical robustness for such transitions with a minimum cost (the discrepancies observed at such critical moments are not significant for the global result).

The baseline case has been simulated considering different values for the time step (i.e. interval length of the simulation set-up) to evaluate its numerical robustness but also its computational time consumption. The results are summarized in Table 2. The results show very good performance in terms of real time factor for all cases ($RTF = CPUtime/realtime$) and also, as expected, important resolution time differences between the time step values considered.

Table 2. Time consumption and time step assessment

Time step [s]	CPU time [s]	real time factor
5	26.4	0.0088
1[a]	24.9	0.0083
0.5	30.7	0.0102
0.1	51.2	0.0170
0.05	69.3	0.0231
0.01	99.3	0.0331
0.005	171.9	0.0573
0.002[b]	349.7	0.1165

[a]Typical value for VCS, [b] typical value for electrical drive

[a]

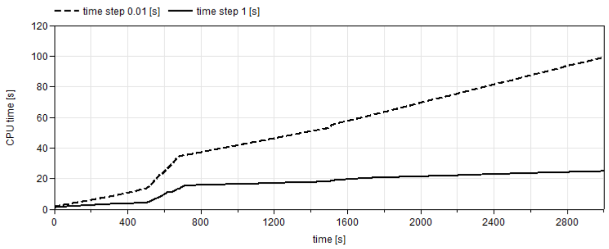


Figure 14. CPU time evolution for two different time steps

Finally, Figure14 shows the CPU time consumption evolution for two representative cases with different time steps. This Figure allows to see the solver stress level at different moments. In this sense, based on the CPU time slope, we can observe that the shut-down procedure and the transition to the off condition are more stressful for the solver than the start-up procedure.

5 Conclusions

The present work dealt with the linking of two models based different underlying physics: a vapor compression system and its electrical drive. The linking has been conducted using mechanical components to represent the compressor shaft.

- Both models have been independently tested to ensure appropriate numerical performance in terms of robustness and CPU time consumption.
- The combined simulation of the resulting thermal-mechanical-electrical model has been assisted with dynamic relaxations for the applied to the mechanical linking parameters, namely, rotational speed and torque.
- The combined model has proven to be numerically robust at critical conditions (initialization, start-up and shut-down procedures), independently of the simulation set-up configuration, and with low time consumption (adequate to simulate large data-sets of cases and real time calculations).

- The combined model is suitable for integrated simulations in large ECS architectures where many other systems are involved.

Acknowledgements

This project has received funding from the Clean Sky 2 Joint Undertaking (JU) under grant agreement No 886533. The JU receives support from the European Union's Horizon 2020 research and innovation programme and the Clean Sky 2 JU members other than the Union.

Horizon 2020
European Union Funding
for Research & Innovation

Clean Sky 2
JOINT UNDERTAKING

Carles Oliet, as a Serra Húnter Associate Professor, acknowledges the Catalan Government for the support through this Programme.

Disclaimer: The contents presented in this article reflect only the author's point of view: the authors and Clean Sky JU are not responsible for any use that may be made of the information it contains.

References

Ablanque, Nicolás et al. (2023-08). "Vapour Compression Cycle Modelling for Use within Large Thermal Systems". In: *26th International Congress of Refrigeration (ICF2023)*. DOI: 10.18462/iir.icr.2023.0790.

Acquaviva, Alessandro et al. (2020). "Analytical conduction loss calculation of a mosfet three-phase inverter accounting for the reverse conduction and the blanking time". In: *IEEE Transactions on Industrial Electronics* 68.8, pp. 6682–6691. DOI: 10.1109/TIE.2020.3003586.

Bierhoff, Michael H and Friedrich W Fuchs (2004). "Semiconductor losses in voltage source and current source IGBT converters based on analytical derivation". In: *2004 IEEE 35th Annual Power Electronics Specialists Conference (IEEE Cat. No. 04CH37551)*. Vol. 4. IEEE, pp. 2836–2842. DOI: 10.1109/PESC.2004.1355283.

Demiray, Turhan (2008). "Simulation of power system dynamics using dynamic phasor models". PhD thesis. ETH Zurich. URL: https://doi.org/10.3929/ethz-a-005566449.

Gurumurthy, Sriram Karthik et al. (2022). "Hybrid Dynamic Phasor Modeling Approaches for Accurate Closed-Loop Simulation of Power Converters". In: *IEEE Access* 10, pp. 101643–101655. DOI: 10.1109/ACCESS.2022.3208963.

Holmes, D Grahame and Thomas A Lipo (2003). *Pulse width modulation for power converters: principles and practice*. Vol. 18. John Wiley & Sons. URL: https://ieeexplore.ieee.org/servlet/opac?bknumber=5264450.

Irwin, J David (1997). *The industrial electronics handbook*. CRC press.

Krause, Paul C et al. (2013). *Analysis of electric machinery and drive systems*. Vol. 75. John Wiley & Sons. URL: https://ieeexplore.ieee.org/servlet/opac?bknumber=5265638.

Loka, Nasrulloh Ratu Bagus Satrio et al. (2022). "Surrogate Modelling of Dynamic Phasor Simulations of Electrical Drives". In: *IECON 2022–48th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, pp. 1–6. DOI: 10.1109/IECON49645.2022.9968552.

Ruan, Xinbo et al. (2018). *Control techniques for LCL-type grid-connected inverters*. Springer. DOI: 10.1007/978-981-10-4277-5.

Sarlioglu, Bulent and Casey T Morris (2015). "More electric aircraft: Review, challenges, and opportunities for commercial transport aircraft". In: *IEEE transactions on Transportation Electrification* 1.1, pp. 54–64. DOI: 10.1109/TTE.2015.2426499.

A Penalty Function-based Modelica Library for Multi-body Contact Collision

Ziheng Zhu[1] Xueqi Ma[1] Dedong Liang[1] XingPei Jiang[1] Yuhui Liu[1] Chen Lu[1] Fanli Zhou[1]

[1]SuZhou Tongyuan Soft. & Ctrl. Tech. Co. Ltd, Suzhou, China,
{zhuzh, maxueqi, liangdd, jiangxp, liuyh, chenl, zhoufl}@tongyuan.cc

Abstract

Contact collisions are prevalent in mechanical multi-body systems and have always been a significant limiting factor for engineering technology development. This paper examines the fundamental types of contact in multi-body dynamics systems and explores their inherent topological relationships. Based on the multi-body dynamics theory and penalty function contact algorithm, this paper constructed the multi-body dynamics contact model using Modelica, which is a multi-domain unified modeling language. To enhance the applicability of the contact model library in the modeling of multi-body system, the contact model provides a connection interface compatible with the multi-body library in the Modelica standard library.

Keywords: Contact, multi-body dynamics, penalty function, Modelica, MWORKS

1 Introduction

Multibody dynamics primarily investigates the relationships between forces and motion among multiple bodies. In actual multibody dynamics systems, the majority of the relationships between bodies are contact-based. In multibody dynamics simulations, it is challenging to simulate the contact relationships between objects with precision. Consequently, conventional contact relationships are transformed into modeling components, such as motion pairs to enhance modeling efficiency and simulation accuracy. Nevertheless, there exist numerous simulation scenarios where modeling using contact relationships between geometric entities is necessary and cannot be simplified, such as simulation of cam mechanism motion, quadruped robot gait simulation, and Contact process between gears (Dahl M et al. 2017) (Bortoff S A 2020). Currently, basic contact analysis functions are necessary for multibody dynamics software.

Various contact modeling methods have been proposed to address diverse engineering application problems. For instance, Magalhaes H et al. proposed a modeling idea applied for the orbital dynamics contact model based on the Hertzian contact theory (Magalhães H et al. 2020). Additionally, Dahl M et al. introduced a modeling method that applies the Hertzian contact algorithm to the gear contact process (Dahl M et al. 2017). Safaeifar H et al. addressed the issue of hysteresis damping coefficient and proposed a novel modeling approach (Safaeifar H et al. 2020). Bortoff S A et al propose an implicit, event-driven, penalty-based method for modeling rigid body contact and collision in the design and analysis of control algorithms for precision robotics (Bortoff S A 2020).

Referring to the penalty function contact algorithm, this paper constructs a rigid body dynamic contact model library based on the Modelica language and the reusable Modelica mechanical library. Devoted to enhancing the contact analysis capabilities of the Modelica language within the domain of rigid body dynamics. At present, numerous scholars have developed a variety of rigid body dynamic contact model libraries based on the Modelica language. Oestersotebier F et al. employed the geometric analysis method to establish a contact model library specifically for simple contact surfaces (Magalhães H et al. 2020). Furthermore, Hofmann A and Otter M et al. implemented a collision library, utilizing a penalty-based collision approach and incorporates the Bullet Physics Library for collision detection(Hofmann A et al. 2014) (Otter M et al. 2005).

Currently, most mainstream multibody dynamics simulation software features contact analysis function, which is considered as a crucial technical indicator. Multi-body dynamics simulations generally involve the analysis of geometric, mechanical, and mathematical models. The geometric model provides an intuitive representation of the model structure. The mechanical model expands on the geometric model by adding four mechanical elements: kinematic constraints, driving constraints, force elements, and external forces or torques. These elements are assembled according to the kinematic constraints and initial position conditions. This assembly process utilizes a solver to compute the expression and establish a mechanical model. The solver generates the coefficient matrices for the system motion equation of the mechanical model. These matrices can be utilized to derive the system's mathematical model and analyze the model's kinematic and dynamic characteristics.

MWORKS.Sysplorer is a system-level integrated platform for the design and simulation verification of multi-domain industrial products that fully supports the Modelica unified modeling standard across multiple domains. This paper aims to develop a contact model library utilizing the capabilities of this platform.

2 Basic Concept

2.1 Contact Mechanism

Contact friction arises when two distinct surfaces make contact and interact through rubbing, creating a contact state characterized by friction. In the field of physics, surfaces in contact demonstrate the following properties: (1) they prevent penetration; (2) they can transmit both normal pressure and tangential friction forces; (3) they do not transmit normal tensile forces. These characteristics enable free separation among surfaces.

2.2 Penalty Function Method for Contact Mechanism

In the case of contact between rigid bodies, the contact force can be calculated using a spring-damper model that takes into account the penetration depth, resulting in a force, and the penetration velocity, leading to a damping force, between the bodies. To calculate the contact force between bodies in such scenarios, the spring-damper model is frequently employed(Machado M et al. 2012).

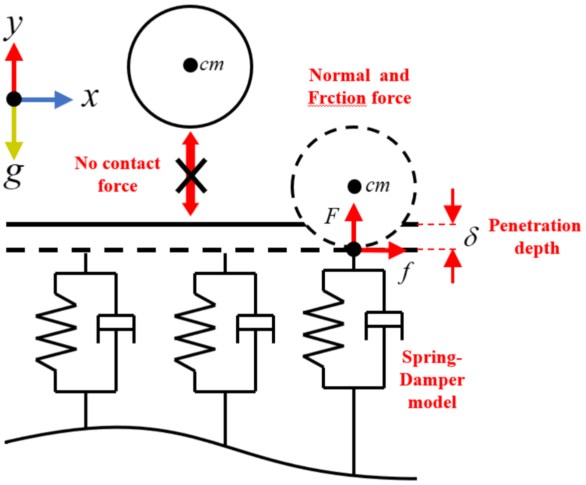


Figure 1 Two-dimensional point-face contact force equivalent schematic

Currently, there are two methods available for computing contact forces in multibody systems: the regression-based and penalty-based contact algorithms. The penalty-based contact algorithm is generally smoother and faster in numerical simulation. The contact model in this study is established solely using the penalty-based contact algorithm. When calculating the contact force between two components using the impact function, it is mainly composed of two parts: the elastic force generated by mutual penetration and the damping force generated by relative velocity. The generalized form can be expressed as:

$$F = K\delta^n + C(\delta)V(t) \tag{1}$$

In which,

F— Normal contact force, measured in N.

K— Contact stiffness, measured in N/m. Generally, a higher stiffness value makes numerical integration more difficult, but if it is too small, the contact situation of the component cannot be accurately simulated. Usually, the contact stiffness is set to 10^8 N/m, The reasonable range of stiffness values can be estimated by Hertz contact theory.

n — Force exponent, used to calculate the contribution value of the material stiffness term in the instantaneous normal force. For contact with high stiffness, n >1 otherwise n <1. For metals, it is usually taken as 1.3~1.5, and for rubber, it is usually taken with a typical value of 1.5.

δ — Penetration depth between objects, a variable that depends on time, measured in mm.

$C(\delta)$ — Contact damping as a cubic function of penetration depth, measured in Ns/m, Generally, 0.1~1% of the stiffness value is taken.

$V(t)$ — Normal relative velocity between contact objects as a function of time, measured in m/s.

In this paper, the contact penetration depth is used to characterize the size of the contact force, mainly considering that the contact model is constructed based on three geometric elements: point, line and surface. It is more intuitive to calculate the contact force between points, lines and surfaces by using penetration depth, and it is easy to expand in building complex geometric contact models.

If the normal penetration depth of the contact point is less than the critical penetration value, the damping force varies cubically with the penetration depth. Conversely, if the penetration depth is greater than or equal to the critical value, the damping force will attain its maximum value, as illustrated in the following figure:

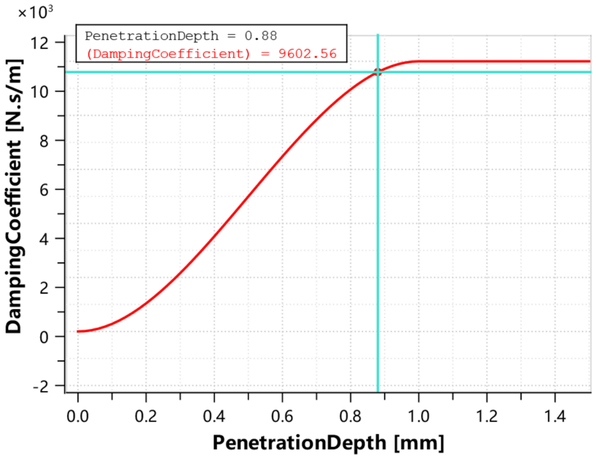


Figure 2 Relationship curve between damping and penetration depth

Firstly, the essence of the penalty function contact algorithm can be analyzed through the point-surface contact model. According to the diagram, contact is deemed to transpire when the distance between the point and the surface is below zero, signifying the penetration of the two rigid bodies.

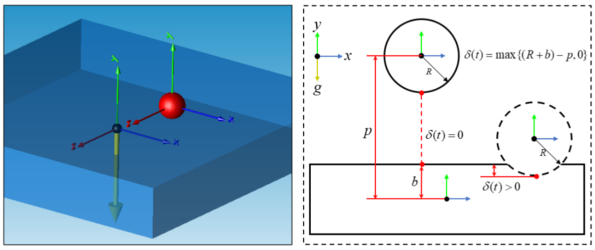


Figure 3 Two-dimensional schematic diagram of point-surface contact

Here, the penetration depth $\delta(t) = \max\{(R+b) - p(t), 0\} > 0$ is used to characterize the degree of penetration between the two rigid bodies. A contact is considered to exist when δ(t)>0, with larger values indicating a more pronounced penetration phenomenon and resulting in greater interaction forces. In this paper, when the rigid bodies are in contact, the normal pressure generated by the contact between the rigid bodies is equivalently modeled using a spring-damper model, and its value is related to the penetration depth $\delta(t)$ and the relative velocity $V_v(t)$ of the surface normal. The specific expression is as follows:

$$\textit{Contact elasticity}:$$

$$F = \begin{cases} \max\{K\delta^n(t) + C(\delta(t))V_v(t), 0\} & \delta(t) > 0 \\ 0 & \delta(t) \le 0 \end{cases} \tag{2}$$

$$\textit{Contact damper}:$$

$$C(\delta) = Step(\delta, 0, 0, \delta_m, C_m) \tag{3}$$

The friction force F_f generated by the contact between rigid bodies is equivalent to a velocity based Coulomb friction model, and its value is calculated according to the contact force F and friction coefficient u. The friction coefficient is related to the relative slip velocity $V_t(t)$ of the two objects in contact(Sextro W et al. 2003).

$$F_f = -F \cdot u(V_t(t)) \tag{4}$$

According to the difference of the relative sliding speed of the two contact objects, the friction process transitions between dynamic friction and static friction. The equivalent formula of friction coefficient and slip velocity is calculated by a cubic step function. The expression of the cubic step function is as follows:

$$Step(x, x_0, h_0, x_l, h_l) =$$
$$\begin{cases} h_0 & x \le x_0 \\ h_0 + \left(\frac{(h_l - h_0)(x - x_0)^2}{(x_l - x_0)^2}\right)\left(\frac{3(x_l - x_0) - 2(x - x_0)}{x_l - x_0}\right) & (x > x_0 \\ & \& x < h_l) \\ h_l & else \end{cases} \tag{5}$$

The specific expression of friction force is as follows:

$$u = Step(V_t(t), V_s, -1, V_s, 1) \cdot Step(abs(V_t(t)), V_s, C_{st}, V_{tr}, C_{dy}) \tag{6}$$

V_s is the relative slip velocity corresponding to the maximum static friction;

C_{st} is static friction coefficient;

V_{tr} is the relative slip velocity corresponding to the dynamic friction;

C_{dy} is the coefficient of dynamic friction.

The following diagram shows the relationship between the friction coefficient and the relative slip velocity.

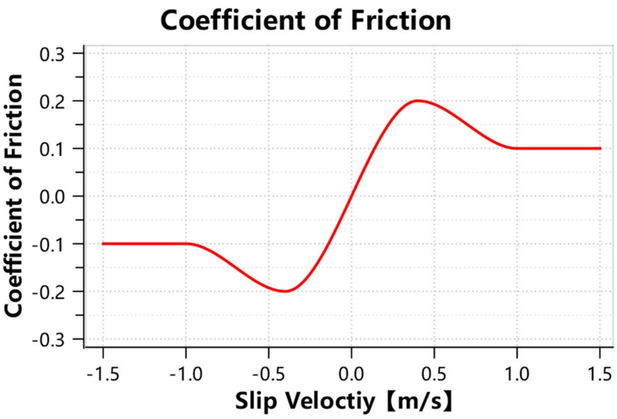


Figure 4 Relation curve of friction coefficient and relative slip velocity

When the slip velocity gradually increases, the friction coefficient gradually increases from 0 to 0.2 of static friction, and then gradually transitions to 0.1 of dynamic friction.

According to the above formula, it can be seen that the value of the interaction force is only related to the velocity vector $V_t(t)$ and the penetration depth $\delta(t)$ between the rigid bodies. The velocity vector $V_t(t)$ can be conveniently obtained using the Modelica multi-body library, and only the penetration depth value $\delta(t)$ needs to be calculated. Therefore, the essence of contact algorithm is actually the process of calculating the penetration depth $\delta(t)$.

3 Overview of Contact Model Library

The contact model library mainly includes examples library, basic component library, friction component library, function library and icon library. The basic component library provides 9 typical contact models, mainly including Sphere-line contact, Sphere -surface contact, and line-line contact. On this basis, six contact models are derived, including line-surface contact, Sphere

-curve contact, Sphere -cylinder, line-cylinder, and Sphere -to- Sphere contact. Each model takes into account the contact force and friction characteristics between rigid bodies, and has a mechanical connection interface, which is compatible with the Modelica multi-body library. The friction component library serves the purpose of calculating the friction force during object contact and forming the contact model components. The function library mainly includes the penalty function contact algorithm function, the damping coefficient calculation function and the geometric analysis algorithm function, etc., which are called repeatedly during the contact simulation. The resulting library structure is illustrated in the accompanying figure.

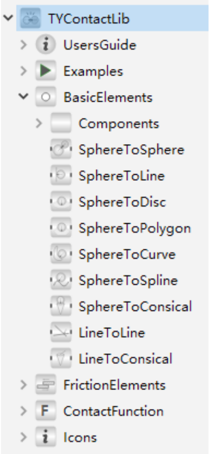

TYContactLib
- UsersGuide
- Examples
- BasicElements
  - Components
  - SphereToSphere
  - SphereToLine
  - SphereToDisc
  - SphereToPolygon
  - SphereToCurve
  - SphereToSpline
  - SphereToConsical
  - LineToLine
  - LineToConsical
- FrictionElements
- ContactFunction
- Icons

Figure 5 The structure of contact model library

4 Implementation of Contact Models

Based on MWORKS.Sysplorer platform and Modelica mechanical standard library, this paper uses Modelica language to build the contact model mentioned above. The platform has an open Modelica model library customization function to meet different modeling needs. Provides model text, model ICONS, components and other views of the model browse and edit. Support curve display of result data, 3D animation display and rich curve calculation and operation functions.

When analyzing the dynamic process and kinematic process of multi-body contact, the model must describe not only the contact force and contact friction force resulting from the interaction between rigid bodies but also consider the geometric shape information of the objects to determine the contact state and contact area. The following modeling ideas can be employed:

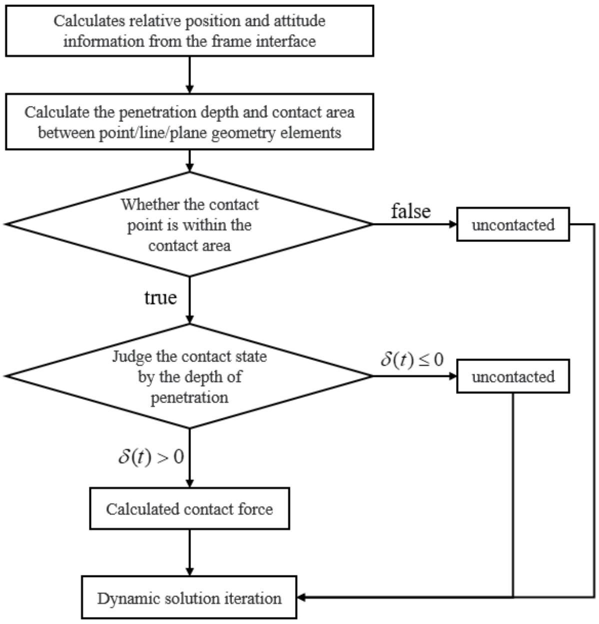

Calculates relative position and attitude information from the frame interface
Calculate the penetration depth and contact area between point/line/plane geometry elements
Whether the contact point is within the contact area
false
uncontacted
true
Judge the contact state by the depth of penetration
$\delta(t) \le 0$
uncontacted
$\delta(t) > 0$
Calculated contact force
Dynamic solution iteration

Figure 6 Modeling approach for contact models

The contact model serves as a bridge that connects the multi-body models in the Modelica standard library. It acquires the pose information between rigid bodies through the frame interface, and utilizes the geometric information of the relative local coordinate system between two rigid bodies as parameters to analyze the contact state between their geometric elements.

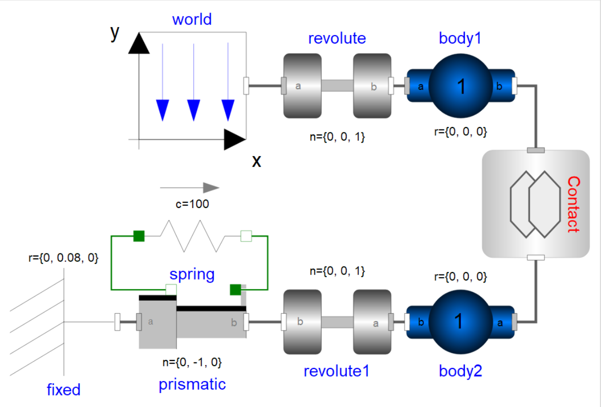


Figure 7 Example of a multibody model including contact

The example model, depicted in the above figure, references the body module from the multi-body library. Users can define the shape of the object according to the actual situation or import geometric models in specific formats. Additionally, users are required to set the geometric data of the object as parameters in the contact model in accordance with specific formats.

5 Contact Detection

The contact algorithm solves only the problem of determining the acting force when contact occurs between rigid bodies. The contact process between rigid bodies involves considering the contact area between the geometric elements of the rigid bodies. For instance, when dealing with the contact problem between a point and a plane, this article addresses the contact process between a point and a finite surface region. Thus, the key to developing a geometric contact model is determining whether the geometric elements lie within the boundary region.

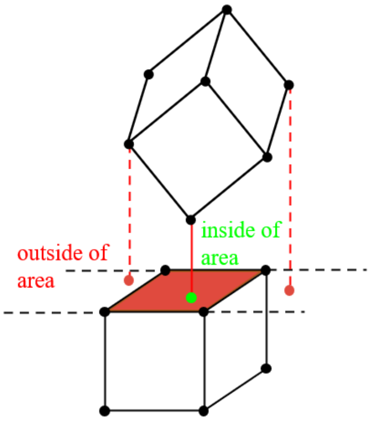


Figure 8 Schematic diagram of point-surface contact area between cubes

This paper mainly refers to the discrete element method and the geometric analysis method to detect the contact area(Oestersötebier F et al. 2014) (Elmqvist H et al. 2015) (Van Den Bergen 2003). Here, this paper based on the theory of geometric graphics, all rigid bodies are described by basic geometric figures such as points, lines, and surfaces, so that the contact problem between rigid bodies can be transformed into the contact problem between three geometric elements. In addition, in order to improve the efficiency of the software to solve the model, all the contact models constructed in this paper use the geometric analysis method to analyze the contact state between objects.

5.1 Point-Line Contact Boundary Detection

Point-line contact involves two main issues: calculating the distance between a point and a line, and determining whether a point is within the line segment. The diagram below shows how to determine if a point lies within the line segment.

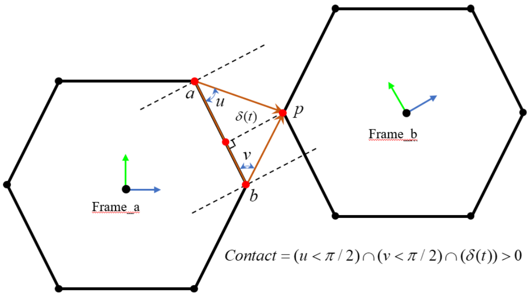

a
u
$\delta(t)$
p
v
b
Frame_a
Frame_b
$Contact = (u < \pi/2) \cap (v < \pi/2) \cap (\delta(t)) > 0$

Figure 9 Point-segment region judgment schematic diagram

With frame_a as the reference frame, calculate $\overrightarrow{ab}$, $\overrightarrow{ap}$, $\overrightarrow{bp}$ respectively, and then calculate the angle u between the vectors $\overrightarrow{ab}$ and $\overrightarrow{ap}$, as well as the angle v between the vectors $\overrightarrow{ba}$ and $\overrightarrow{bp}$.When $u < \pi/2$ and $u > \pi/2$, it is judged that the point p is in the accessible area of the line segment ab. When the penetration depth $\delta(t) > 0$ occurs, there will be contact force between the two rigid bodies.

5.2 Point-Surface Contact Boundary Detection

In fact, any polygon or curved surface can be made up of several triangular surfaces. Therefore, solving the problem of determining the region between points and triangular surfaces enables the easy solution of the problem of determining the region of complex geometric bodies. This article uses the centroid method and the schematic diagram below to determine whether the point lies within the region of the triangular surface.

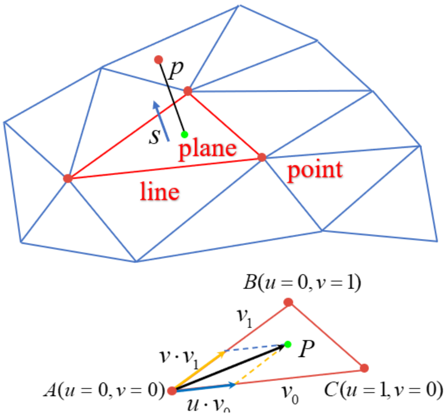


Figure 10 Point and Triangle Face Region Judgment Diagram

Algorithm for determining if a point is within a triangle face region:

$$\begin{cases} u = \dfrac{(v_1 \cdot v_1)\cdot(v_2 \cdot v_0) - (v_1 \cdot v_0)\cdot(v_2 \cdot v_1)}{(v_0 \cdot v_0)\cdot(v_1 \cdot v_1) - (v_0 \cdot v_1)\cdot(v_1 \cdot v_0)} \geq 0 \\ v = \dfrac{(v_0 \cdot v_0)\cdot(v_2 \cdot v_1) - (v_0 \cdot v_1)\cdot(v_2 \cdot v_0)}{(v_0 \cdot v_0)\cdot(v_1 \cdot v_1) - (v_0 \cdot v_1)\cdot(v_1 \cdot v_0)} \geq 0 \end{cases} \tag{7}$$

When $u, v \geq 0$ the contact point is inside the triangle area, contact force calculation will be performed. Otherwise, contact force calculation will not be performed. To determine whether a point is inside a rectangular area, it can be divided into two triangular areas, which makes it possible to perform the contact force calculation.

5.3 Line-Line Contact Boundary Detection

The point-surface contact algorithm in the example of contact between cubes only considers the process of vertex-surface contact. To prevent interference issues, as illustrated in the figure, multiple geometric points on the geometric body must be selected for contact calculations with the surface. However, the number of geometric points significantly affects the model's solving efficiency and accuracy. Therefore, for computational efficiency, an analytical method should be used to establish a line-line contact model instead.

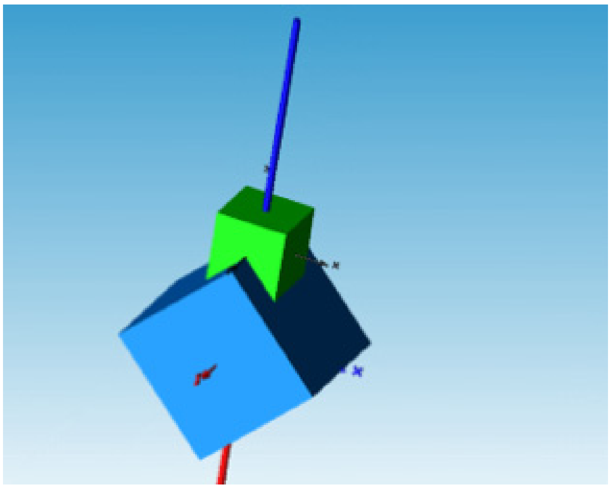
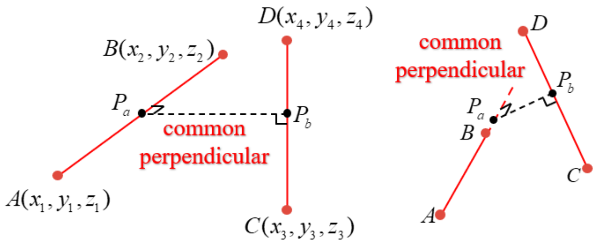


Figure 11 Cubic Contact Interference Phenomenon Considering Only Point-surface Contact

The line-to-line contact model requires consideration of two situations: contact between coplanar line segments and contact between non-coplanar line segments. The former can be achieved by point-to-line contact, while the latter involves determining the positional information of the contact points at each moment, as the contact point between non-coplanar line segments changes over time. To address this, this paper proposes the use of the theory of perpendicular lines, which establishes that there is only one straight line perpendicular to two non-coplanar lines. By calculating the vector information of the perpendicular line, the distance between contact points at each moment and the penetration depth $\delta(t)$ can be determined.



$0 < m,n < 1$ $m > 1, 0 < n < 1$

Figure 12 Schematic diagram of line-line contact judgment method

The specific algorithm is shown in the figure above. Assuming that there are two line segments AB and CD in space, P_a and P_b are the intersection points of line segments AB and CD with the common perpendicular, and the coordinate values of P_a and P_b can be described by the following equations.

$$P_a \Rightarrow \begin{cases} X = x_1 + m(x_2 - x_1) \\ Y = y_1 + m(y_2 - y_1) \\ Z = z_1 + m(z_2 - z_1) \end{cases}$$
$$P_b \Rightarrow \begin{cases} U = x_3 + n(x_4 - x_3) \\ V = y_3 + n(y_4 - y_3) \\ W = z_3 + n(z_4 - z_3) \end{cases} \tag{8}$$

When $0 \le m,n \le 1$, it indicates that P_a and P_b are on line segments AB and CD. Otherwise, P_a and P_b are on the extended lines of the respective line segments AB and CD respectively. With the expressions of P_a and P_b, the distance between the two points can be calculated as follows:

$$P_aP_b = \sqrt{(X-U)^2 + (Y-V)^2 + (Z-W)^2} \tag{9}$$

To find the common perpendicular of two line segments is equivalent to finding the minimum value of P_a, P_b:

$$\min f(m,n) = \sqrt{(X-U)^2 + (Y-V)^2 + (Z-W)^2} \tag{10}$$

Then taking the partial derivative of the function $f(m,n)$ with respect to m,n, a system of two linear equations in two variables can be obtained:

$$\begin{cases} \dfrac{\partial f(m,n)}{\partial m} = 0 \\ \dfrac{\partial f(m,n)}{\partial n} = 0 \end{cases} \tag{11}$$

With the above equation system, we can obtain the value of m,n and determine the distance and position vector between the contact points. Utilizing this information, we can apply the contact algorithm formula to develop the line-line contact model. The application of the line-line contact model ensures that the aforementioned interference problem is eliminated in the cubic contact simulation results.

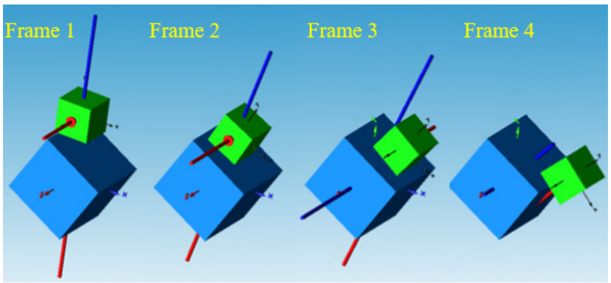


Figure 13 Schematic diagram of point and cylinder contact judgment method

6 Model Extension

6.1 Contact between Point and Cylinder Surface

Drawing on the modeling ideas for point-line contact, it can be extended to the modeling of point-cylinder surface contact, as depicted in the following schematic diagram. By obtaining the position vector of the current point element (P1\P2) relative to the coordinate system LCI of the cylinder geometry, combined with the position vector of the line element endpoints relative to the cylindrical geometry coordinate system, the spatial geometric information of which line segment the point element is about to contact can be obtained in real-time.

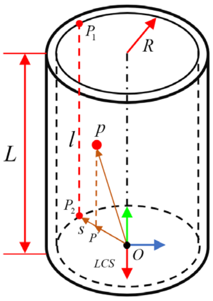


Figure 14 Geometric information diagram of contact line segment

6.2 Point-Curve Contact

The point-curve contact model is equivalent to the point-line contact model with multiple line segment elements, and is based on the point-line contact modeling algorithm. To create the point-curve contact model, geometric information from several points on the curve is read, and cubic spline interpolation is performed to fit the curve. The interpolated curve is then discretized into line segments. By selecting more discrete points, the contact curve can be made to resemble the actual curve more closely.

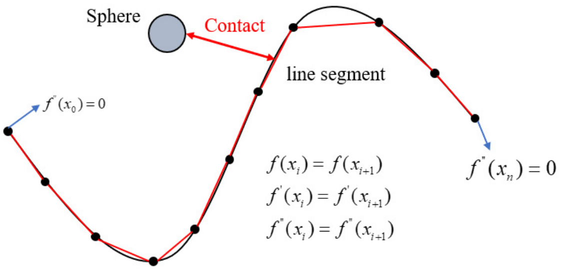


Figure 15 Realization method of contact between point and spline curve

The cubic spline interpolation algorithm actually divides the curve interval $[a,b]$ into n intervals $[(x_0,x_1),(x_1,x_2),\ldots,(x_{n-1},x_n)]$, with a total of $n+1$ points. Each interval is described by a cubic spline function:

$$f_i(x)=a_i x^3+b_i x^2+c_i x+d_i \tag{12}$$

The cubic spline interpolation should satisfy the following conditions:

(1). Each segment should be a cubic spline function;

(2). It should satisfy the interpolation condition, i.e., $f_i(x_i)=y_i(i=0,1,\ldots,n)$;

(3). The curve should be smooth, i.e., $f(x)$, $f'(x)$, $f''(x)$,should be continuous.

7 Simulation Results

The main objective of this chapter is to focus on the aforementioned contact model. It presents three representative examples implemented in MWORKS and utilizes the Dassl algorithm for solving and analysis.

Example 1: Simulating the contact process between a ball with a radius of 30mm and a disk with a radius of 250mm. The z axis of the disk is connected with the world coordinate system by a rotating pair, and a position sine excitation is applied to the rotating pair to analyze the motion of the ball under the action of contact force. The following example demonstrates the construction process.

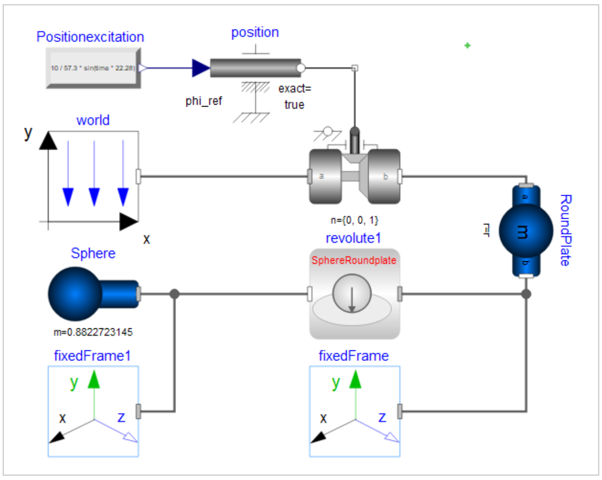


Figure 16 Sphere falling on a swinging round plate

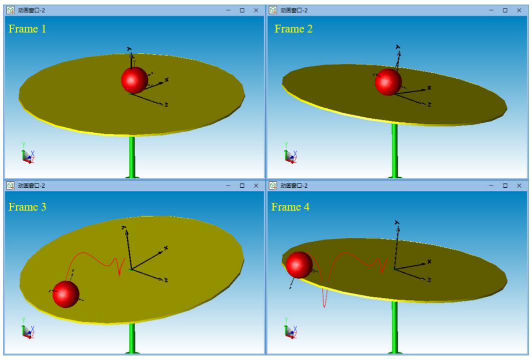


Figure 17 Animation of a sphere falling on a swinging round plate

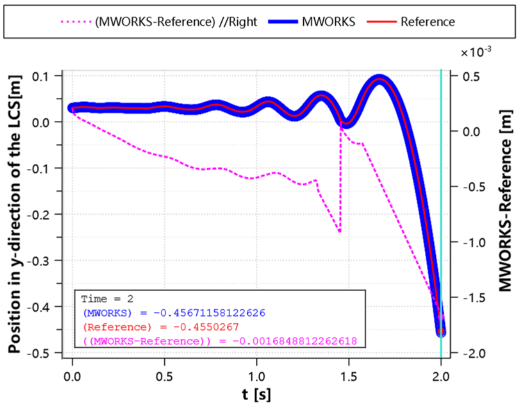


Figure 18 The displacement curve in the Y direction of the center of the sphere

The simulation results are compared with those obtained from Adams, a commercial dynamic simulation software. Figure 17 depicts the simulated curves with 2s. The curves of the two simulation results are in good agreement, and the cumulative error of the position at 2s is 1.6mm.

Example 2: The linear contact model is utilized to simulate the operational principle of the differential. Initially, an equal resistance moment is applied to both wheels, causing them to rotate at the same speed. Subsequently, the resistance moment on the left wheel is increased, leading to the occurrence of differential behavior between the two wheels.

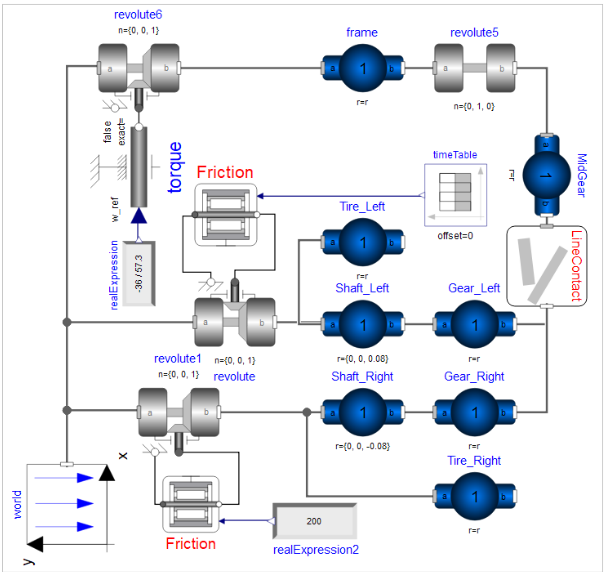


Figure 19 An example of the application of the line contact model

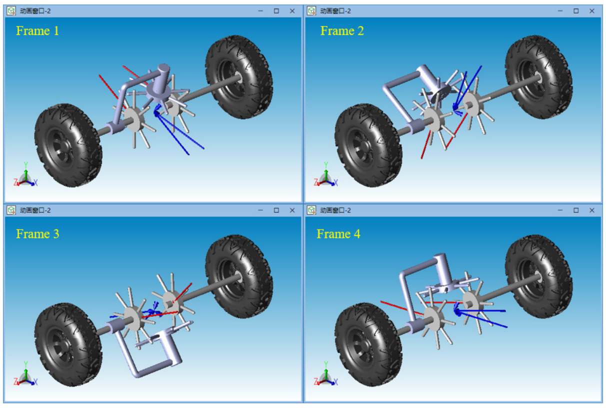


Figure 20 An example animation of a line-line contact model (In the figure, the single arrow is the interacting force vector and the double arrow is the interacting torque vector)

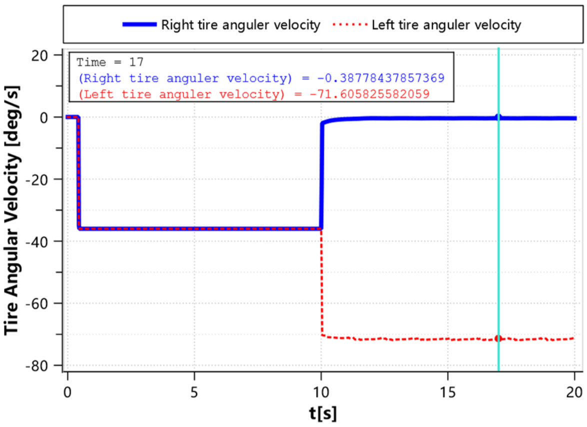


Figure 21 Curve of speed change of two tires

From the graph, it is evident that initially both wheels have the same resistance and speed. However, at 10s, when a resistance moment is applied to the right wheel, a differential phenomenon arises between the two wheels. The speed of the right wheel approaches zero, while the speed of the left wheel doubles its original value.

Example 3: Apply point and curve contact to construct the following two high pair transmission processes.

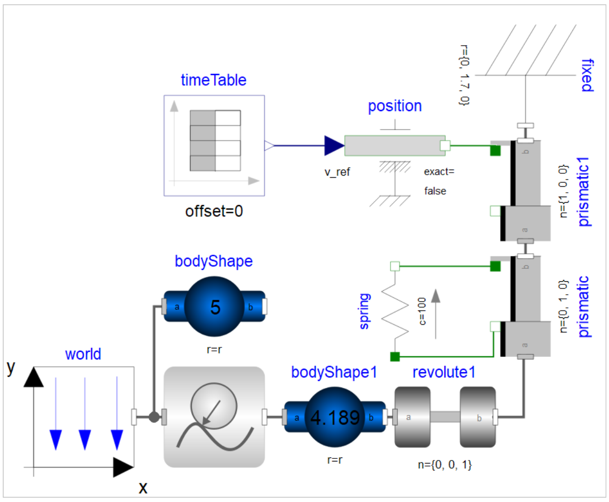


Figure 22 Example of application of point and spline contact model

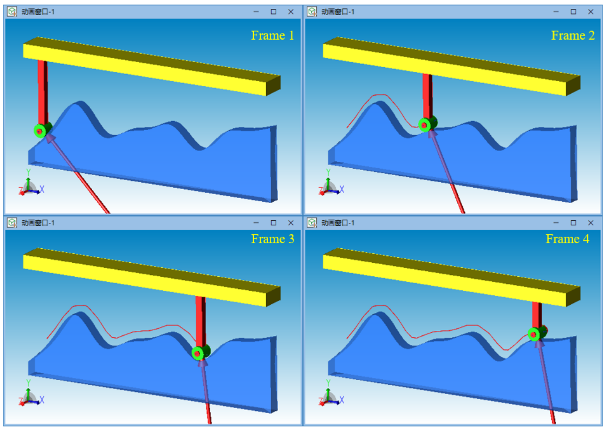


Figure 23 Point and spline curve contact model animation

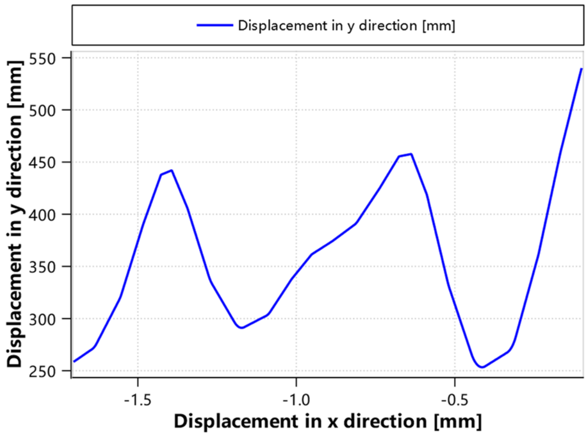


Figure 24 Cylinder displacement curve

Example 4: The point and cylinder contact model is applied to simulate the dynamic contact process of several small balls rolling successively on a funnel until they fall into the cylinder. The following is an example of the build.

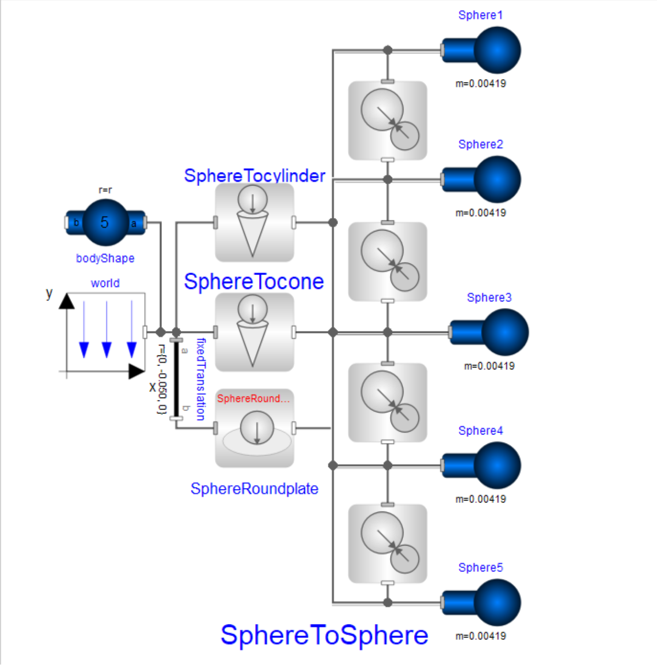


Figure 25 Example of the point-cylinder contact model

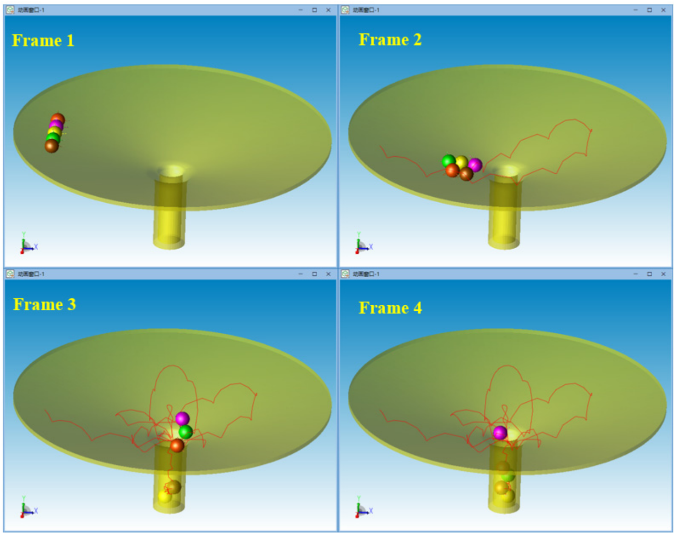


Figure 26 Animation of multiple spheres tumbling down funnels

The example model is constructed by point-cylinder contact model, point-disc contact model and ball-ball contact model. Taking this as an example, multiple typical contact models can be combined to construct more complex contact scenarios.

8 Summary and Outlook

In this paper, the contact library primarily focuses on developing fundamental models for point-surface, point-line, and line-line contacts. These three contact types form the foundation for complex geometric contacts. Furthermore, this paper extends the scope of rigid body dynamic contact models to include more significant variations, such as point contact with spline curves, point contact with cylindrical surfaces, and point contact with polygonal surfaces.

The above discussion represents only a fraction of the broader field of rigid body contact dynamics, leaving significant room for further advancements in contact model development. For instance, when dealing with complex geometric contact problems, it is crucial to optimize model solving efficiency through contact search algorithms, analyze the impact of different contact force calculation methods on the accuracy and efficiency of model solutions, and address contact force errors stemming from the transition between geometric elements.

References

Dahl M , Wettergren H , Tidefelt H(2017). “Modelica Spur Gears with Hertzian Contact Forces”. International Modelica Conference. 2017. DOI: 10.3384/ecp17132755.

Bortoff S A(2020). “Modeling contact and collisions for robotic assembly control”.Proceedings of the American Modelica Conference. 2020.

Magalhães,H. Marques, F., Liu, B. et al (2020). “Implementation of a non-Hertzian contact model for railway dynamic application". Multibody Syst Dyn 48, 41–78.

Safaeifar H, Farshidianfar A (2020). "A new model of the contact force for the collision between two solid bodies". Multibody System Dynamics, 50, 233-257.

Oestersötebier F, Wang P, Trächtler A(2014). "A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces". In Proceedings 10th International Modelica Conference, Lund, March (pp. 10-12).

Hofmann A, Mikelsons L, Gubsch I, et al(2014). "Simulating collisions within the Modelica multibody library".

Otter M, Elmqvist H, López J D(2005). "Collision handling for the Modelica multibody library".In Proceedings of the 4th International Modelica Conference.

Machado M, Moreira P, Flores P, Lankarani H.M (2012): "Compliant contact force models in multibody dynamics: evolution of the Hertz contact theory". Mechanism and machine theory, 53, 99-121.

Sextro W, Poli C(2003). "Dynamical Contact Problems with Friction: Models, Methods, Experiments and Applications". Lecture Notes in Applied Mechanics, Vol 3[J]. Appl. Mech. Rev, 2003, 56(1): B2-B3.

Elmqvist H, Goteman A, Roxling V, et al(2015). "Generic modelica framework for MultiBody contacts and discrete element method". In Proceedings of the 11th International Modelica Conference, Versailles, France (Vol. 118, pp. 427-440).

Van Den Bergen, G. (2003). "Collision detection in interactive 3D environments". CRC Press.

Automatic Optimization of Energy Supply Systems in Buildings and City Quarters based on Modelica Models

Dipl.-Ing. Torsten Schwan[1] M. Eng. David Feige[2] Dipl.-Ing. Leonhard Wenzel[1] Dipl.-Ing. Charlotte Voelckner[1] M.Sc. Martin Leuschke[1]

[1]EA Systems Dresden GmbH, Germany, {torsten.schwan, leonhard.wenzel,charlotte.voelckner,martin.leuschke}@ea-energie.de

[2]HKL Ingenieurgesellschaft mbH, Germany, d.feige@hkl-ingenieure.de

Abstract

The evaluation and analysis of complex energy supply systems with Modelica models is more and more an integral part of the building design processes. Dynamic system modeling became there especially important regarding analyses of the use of storage and the integration of volatile renewable resources as well as intelligent control.

However, this still requires extensive engineering work and time-consuming modeling efforts, although the basic work steps are largely comparable and based on the same fundamentals. Therefore, the open interfaces to and from Modelica offer extensive possibilities for automation and generalization of these processes.

This paper describes such a new integrative and automated optimization framework for energy systems of buildings and districts, which uses Modelica models and FMUs iteratively for the identification of optimal system configurations.

Keywords: System Optimization, HVAC System Models, Python Automation

1 Introduction

Energy supply systems in buildings become more and more complex. Therefore, huge knowledge and a high number of professionals in different trades must be coordinated. This adds further challenges for architects, engineers and designers especially regarding the increased requirements on energy efficiency and availability.

These extensive engineering tasks can only be solved with adequate calculation tools. These tools must be able to deal with an increasing variety of solution options and degrees of freedom as well as influencing factors.

For example, the required heat of new buildings is nowadays provided by renewable energies and no longer by individual boiler systems. Renewable heat sources often require heat pumps to provide the necessary temperature level. Both the volatile heat sources (such as waste heat, solar heat, geothermal heat) and the heat pumps must be considered in detail.

An engineer can now no longer focus on a singular balancing of the necessary natural gas consumption of the boilers. Design decision now need an influence analysis of weather and site conditions as well as the necessary power requirements for the heat pumps. These design analyses must often include an additional coverage by local renewable power production (e.g. by photovoltaics or wind power) which is another important boundary condition and influencing factor. In case of an additional seasonal storage (e.g. ice storage) in a system, engineers need to solve a multi-valent and cross-domain design problem already only regarding the singular task of heating system design.

Keywords like multi-valent and cross-domain quickly bring simulation-savvy engineers to the versatile, multi-physics modeling language Modelica. Therefore, Modelica already provides a large number of well-proven library solutions, e.g. Buildings, BuildingSystems, IBPSA, AixLib, IDEAS, Green City, etc. (c.f. Wetter 2009, Müller et.al. 2016). Furthermore, the various Modelica simulation environments (like SimulationX, Modelon Impact, Dymola, Open Modelica, etc.) offer a variety of open interfaces for the simulation automation. They can help to automate the necessary extensive variant analysis. This saves a significant amount of engineering time. Additional export options, especially like the Functional Mockup Interface Standard (FMI), enable IP protection and tool-independent development of the necessary design tools.

Engineers can design the energy supply for each building individually. However, thinking on a district level often provides more efficient solutions. It allows to use synergy effects between different buildings and surrounding

DOI
10.3384/ecp204789

energy sources. This is the core topic of the Smood (Smart Neighborhood) network (Roselt, and Büttner, 2019).

The Smood joint project consists of a group of German engineering companies and local scientists. It tries to develop a holistic value chain of tools and processes for the decarbonization of the energy supply of residential neighborhoods. Another important research and development aspect is the market launch of the developed toolsets.

The whole project considers four core development goals. SmoodQIM represents a holistic neighborhood information model for data management (i.e. comparable to BIM for buildings). Another core piece, i.e. SmoodManage, includes automated process steps for the building retrofit planning. Besides these software and process components, the SmoodHardware part considers the development of novel storage systems in both the thermal and the electrical domain. For the Smood collaborators, all these components are necessary for a future sustainable energy supply to residential neighborhoods.

Another key component of the developed tool chains is a new methodological simulation approach (i.e. SmoodSimulator) that evaluates buildings and HVAC systems together in an iterative optimization process. This automated process results in a holistic retrofit strategy including a therefore optimized HVAC system configuration. It includes three main components.

A building-focused analysis tool (i.e. Caala) provides basic retrofit options for the particular building envelope with respect to CO_2 savings and gray energy demand. Then, the automated tool chain derives a model matrix with different system configurations based on the building requirements. This matrix is linked to a huge set of pre-configured HVAC system models developed in Modelica. These models run automatically after an automated setup of the necessary components parameters. All automation steps are based on the versatile scripting language Python. This Python framework also includes an optimization algorithm which iteratively adjusts the HVAC system parameters depending on the chosen optimization goals.

Such iterative approaches which connects Python-based optimization algorithms and Modelica models are not new. Leimeister, 2019 describes in her paper a combined optimization framework that uses both components. However, this work primarily focuses on the optimization of a singular system component, a wind turbine in its operating environment. A more general link between Python-based machine learning libraries and the simulation tool EnergyPlus was introduced by Christiaanse et.al. 2021. Eckstädt et.al. 2020 investigated extensively the use of simulation-based methods and optimization approaches with respect to different application scenarios in the context of building design process.

The use of multi-criteria optimization approaches with focus on architectural design was introduced by Dan Hou et.al. 2019 among others. The BeDOT tool (c.f. Bergel et.al. 2019) uses some the scientific approach of holistic software tools for energy system optimization. An alternative view on building controls with a holistic framework is also part of Arroyo et.al. 2021.

Comparable approaches to Smood regarding a holistic view of all these topics on a neighborhood scale is also part of the research at the University of Innsbruck, Austria (c.f. Dermentzis et.al. 2019).

2 General Concept

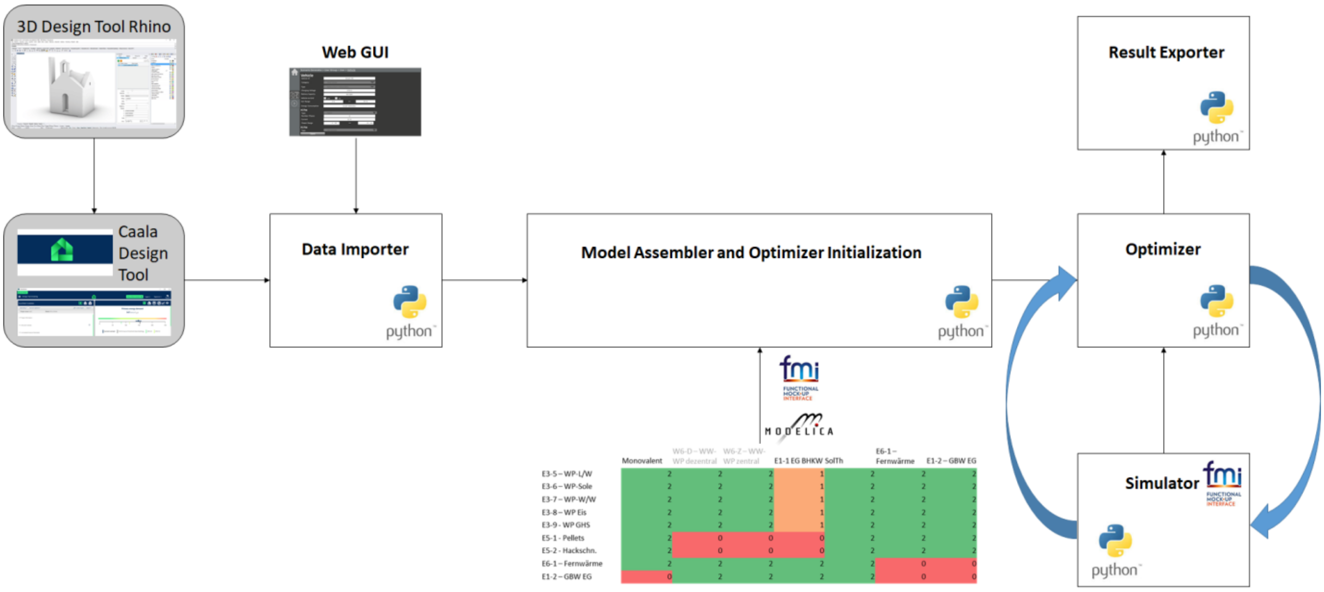
The entire Smood nucleus includes a variety of companies and research groups as well as many advanced technologies and tools. One key component of the master plan is a novel, holistic simulation environment for the automated generation of retrofit strategies and energy concepts for larger residential districts, i.e. SmoodSimulator.

This tool needs to take into account the great variety of requirements of the building structure and the energy supply as far as possible. The identified solutions have to look for the local and global optima with regard to the joint consideration of life cycle costs (LCC) and life cycle assessment (LCA). Therefore, it can consider retrofit measures for the building envelope as well as the use of novel (renewable) supply technologies. Very often, potentials solutions can also be mixed forms of both approaches. The main goal of the SmoodSimulator is to automatically find exactly these solutions on the basis of the existing data.

Designated users of the tool can also be city and district planners as well as architects. High engineering and simulation know-how can therefore not be assumed. Requirements resulting from the use of Modelica models have to be handled as automated as possible by the SmoodSimulator itself or corresponding simplifications.

Figure 1 shows the general structure of the SmoodSimulator workflow and its abstract software components. This also includes application-specific tools from Smood partners (i.e. Caala) and third-party providers (i.e. Rhino 3D).

The Rhino 3D tool enables architects to plan the structure of new buildings from the design point of view or to optimize existing buildings with the help of a graphical interface. It provides only information about the building envelope. The tool Caala (via Grasshopper plugin) evaluates exactly this structural data of the 2D and 3D



	Monovalent	W6-D – WW-WP dezentral	W6-Z – WW-WP zentral	E1-1 EG BHKW SolTh		E6-1 – Fernwärme	E1-2 – GBW EG
E3-5 – WP-L/W	2	2	2	1	2	2	2
E3-6 – WP-Sole	2	2	2	1	2	2	2
E3-7 – WP-W/W	2	2	2	1	2	2	2
E3-8 – WP Eis	2	2	2	1	2	2	2
E3-9 - WP GHS	2	2	2	1	2	2	2
E5-1 - Pellets	2	0	0	0	2	2	2
E5-2 - Hackschn.	2	0	0	0	2	2	2
E6-1 – Fernwärme	2	2	2	2	2	0	0
E1-2 – GBW EG	0	2	2	2	2	0	0

Figure 1: Overview of smoodSimulator workflow and abstract software components incl. the integration of Modelica models and FMUs (sources of figures: Constantino and Pepe 2021)

building representations in order to run a first analysis of LCC and LCA based on simple key figures.

Its strength is primarily in the underlying databases on the necessary building material costs and their LCA statistics. In this manner, it enables a balanced quantification of the so-called "gray energy". This is a measure of necessary CO_2 emissions regarding the required building materials and amounts (i.e. incl. insulation). However, the energy supply system in Caala only uses simple and constant efficiency factors. This is an essential limit of the HVAC and power supply system evaluation in Caala. Especially, volatile production of renewables and local storages with nonlinear efficiency characteristics and availability are thus hard to assess. A more specific modeling was necessary for appropriate engineering evaluation.

The SmoodSimulator was designed to fill this gap using the available interfaces of Caala as well as its results and outputs. The main idea is to add a seamless automated toolset which takes the available information and adds the necessary complexity in the HVAC and power supply system part of the evaluation strategy. Therefore, Modelica models seemed to be most promising in accuracy but lacked regarding handling and usability for non-professional simulation engineers, like architects and designers. Therefore, adequate automation, e.g. via Python, was necessary.

This process starts with the building-specific data, such as thermal and electrical energy requirements, as well as the initial results of the LCA/LCC analyzes in Caala. The SmoodSimulator reads them via a specific interface API using the Data Importer. This process defines the caalaConfig. Furthermore, it reads additional information regarding optimization process parameters and the stored simulation models from other configuration files (i.e. userConfig, optiConfig). In the future, a graphical user interface will be available for this specific import task.

The core piece of the Python framework is the Model Assembler and the Optimization Initialization. It has a variety of tasks, especially in the area of automatic selection, preparation and instantiation of the required models.

The Modelica models have to represent as accurately as possible the interdependencies of any dynamic energy system with its variety on possible details. The building(s) are considered as static load profiles. Their optimization with respect to possible retrofit steps for the building envelope was part of the prior optimization loop in the Caala tool. However, these Modelica models of HVAC systems require consumption curves with high temporal resolution that go beyond the available outputs of the Caala tool (i.e., monthly balances only). Therefore, the Model Assembler has to generate automatically suitable input data sets for the models, i.e. heating and power loads, weather data.

The Model Assembler has access to an extensive model library of predefined energy supply system configurations. The variants choice for the optimization process is based on a dynamic requirements matrix. This is matched with the requirements of the building(s) under consideration (e.g. temperature conditions).

Furthermore, the Model Assembler prepares the required set of model configurations for the optimization process. This is a working step which has been optimized regarding the total optimization speed for several times. One of these steps included the execution of exported FMUs (Functional Mockup Units) via Python instead of running Modelica models via remote control. This provides a very performant solution which enable a parallel execution of models with different configurations on different platforms (i.e. using PyFMI or FMPy interface in Python platform). In order to perform appropriate parameter optimization (i.e., equipment technology performance data) for each configuration, each FMU provides appropriate variable parameters.

Another important initialization step is the initial parameterization of all system variants which is based on the requirements of the generated building loads. The Model Assembler checks all generated load profiles regarding potential critical values, such as peak power (e.g. dynamic heating load). Then, it derives the necessary preset parameters of each FMU with respect to the requirements of the variant matrix and some specific heuristics.

The optimization process runs iteratively and parallelized. Each model configuration and parameter variant runs an annual simulation as an executable FMU in Python. Because of the necessary high number of system configurations and suitable parameter numbers, the requirements on the model performance are very high. Section 3 therefore shows the most relevant steps of model performance optimization.

Each system configuration is first simulated as a baseline with its initial parameter set. LCC and LCA are then calculated based on the energetic results of the executed FMUs in post-processing. With the Python hyperparameter optimization framework Optuna, the variable parameters (i.e., power classes) of all system variants are then optimized in parallel using an iterative loop and a high number of year simulations. Special attention is given to bivalent equipment configurations such as district heating grid and heat pump etc. during optimization.

If the optimizer reaches the previously defined termination conditions, the optimization process ends. The results of the analysis, i.e. a selection of the different optimized system configurations and the presentation of the best result, are displayed. This includes an export of suitable graphics and necessary data for post-processing.

3 Model Concept

The developed framework focuses on Modelica-based simulation models of technical equipment for heat and power supply of (residential) buildings. The building itself is only a simple look-up table based load profile because of highly accurate tool sets (i.e. Caala, Rhino 3D) before the Modelica models in the tool chain. However, these simple models also require suitable interfaces to the still necessary dynamic model components. The Modelica-based Green City library in SimulationX therefore provides some relevant interfaces and a suitable data integration (c.f. Schwan et.al. 2017).

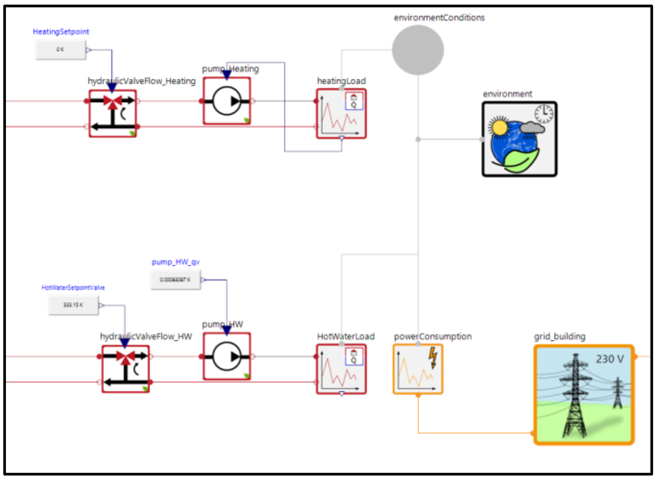


Figure 2: Simple modelling approach of building load curves with Green City (i.e. Modelica-based library) components

Regarding the co-simulation of HVAC systems with independent building models, Nicolai and Paepcke 2017 have already shown a first adequate solution with the Green City library. The presented framework uses the same premises as model base.

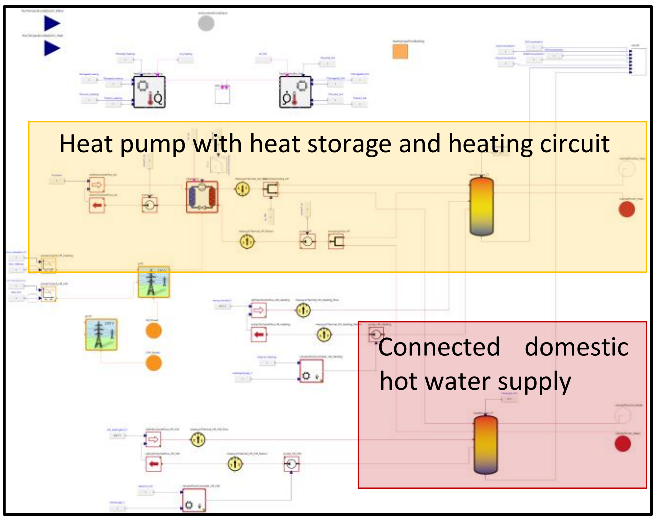

Heat pump with heat storage and heating circuit
Connected domestic hot water supply

Figure 3: Detailed HVAC system model component

The chosen modeling concept is based on the main task to enable simulations of each possible system configuration of the model matrix (cf. section 2). A first approach used detailed single Modelica models. There were saved in specific library as a model template for the Model Assembler. In these templates all model-internal parameters (e.g., pump parameters, storage tank sizes, etc.) depend on a reduced set of relevant parameters, like performance categories. If these performance values are changed during the optimization process, the model internally adapts automatically. This is important for both

DOI
10.3384/ecp204789

numerical stability and a realistic representation of real-world behavior.

Figure 3 shows an example model of a simple heat pump system with geothermal collectors as renewable heat source based on Green City components. The model only has a unified interface to the heating circuits and the local power grid which are compatible to the general load profile model (c.f. Figure 2). However, there are strong mathematical dependencies between the different temperature levels of both heating circuits (i.e. heating and domestic hot water) due to the availability of only one heat supply component (i.e. one heat pump). This model is comparatively accurate as individual controls define the heat pump output depending on individual storage temperatures and the respective dynamic loads. However, this significantly lowers the simulation performance. Simulation of one entire year thus requires about 10 to 15 min for each variant and system configuration.

In an optimization process with 10++ different system configurations and various options of discrete system parameterization (especially in bivalent system configuration), 1,000s to 10,000s simulation runs may be necessary. However, the optimization period should not exceed a frame of hours to a few days.

The first important performance optimization step considered the decoupling of dependencies between the two heat supply tasks (i.e. heating and hot water production). This approach required to focus on each template model of SmoodSimulator's system matrix. Obviously, it results in a higher deviation between the simulated system behavior and exact simulation results or real-world measurements. However, this loss of accuracy is acceptable because the intended field of toolset application are preliminary design phases of existing neighborhood districts. The level of detail of all assumptions is there quantified with +/-40% and higher (c.f. Kochendörfer et.al. 2010). Deviations from different efficiencies of the simulated systems (e.g. temperature-related COP of heat pumps) or limited availability due to simultaneity will be significantly lower.

The presented approach already reduced the average simulation times of an entire year to 2 to 3 min. However, this was still not fast enough. Further optimization steps were necessary. The model concept update still considered a coupled forward-backward modelling approach. Heat and power load characteristics defined a backward model of the considered building(s). The energy system template models still represented a forward model which operated depending on internal control on temperature levels of the decoupling storage tanks (i.e. internal system capacities).

The next optimization step consisted of a complete conversion of the Modelica model approach to a backward model. This required a redesign of the individual system components of the HVAC system models in the templates. Now, these models only represent a nonlinear, dynamic efficiency characteristic and are thus only conversion models (e.g. heat pump - heat/electric energy).

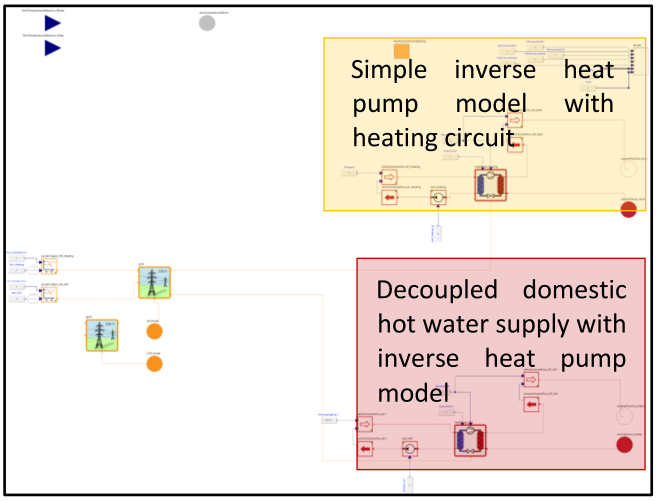

Simple inverse heat pump model with heating circuit
Decoupled domestic hot water supply with inverse heat pump model

Figure 4: Simplified HVAC system model component

Model dependencies on external control functions were removed as far as possible. However, since the interfaces of the model templates stay the same, reuse of the more detailed model options for detailed analyses within the framework will be possible at any time.

In this way, the average simulation time for a year simulation per system model template could be reduced to less than 30 s (i.e. using CVODE solver with common settings).

Now an acceptable performance range was reached. Further simplifications in the models themselves were almost no longer possible. However, further performance potential could still be identified during model execution.

On the one hand, the direct execution of Modelica models requires the use of a suitable simulation environment (in this case SimulationX). Communication with and automatic execution of models in this environment also requires a certain time period. The faster the models, the higher is the share of these communication time periods on execution total time. On the other hand, the execution of simulation models in the simulation environment takes place exclusively on one computational core of the respective computer/server. Parallelization is difficult to realize and also requires extended licensing costs, especially in the area of optimization for the simultaneous execution of a large number of models.

Therefore, the use of FMUs in Python with the help of the PyFMI framework provides an alternative solution. This allows the execution of parameterizable FMUs directly in Python without an additional running simulation environment. By exporting the models with solvers (FMI 4 co-simulation), similar or even better performance can be expected without the communication overhead.

However, the biggest potential advantage is the parallelizability of model execution. By using multi-threading approaches, the Python framework supports parallel execution of models within an optimization loops.

This last optimization step again provided another significant reduction of the necessary simulation time periods per model (FMU). With about 10 to 20 s per annual simulation to be performed, the framework now has a sufficient computing speed even for large problems. Common optimization tasks with about 5,000 model iterations can usually be performed in 2 to 3 hours.

The developed approach now represents a powerful simulation and optimization framework. However, due to the necessary simplifications, the model accuracy is now somewhat reduced. However, it is still in an acceptable range regarding the available degree of accuracy of necessary assumptions in an early building design phase (i.e. +/-40%).

Furthermore, the developed model concept represents another huge advantage regarding its upward compatibility. Because of the consistent interface definition, the models are always available in different accuracy levels (c.f. Figure 3).

If a more detailed consideration of some system variants with the help of the optimization framework is required in a later design phase (e.g. detailed planning), a more accurate simulation model with the same interfaces can simply be generated and analyzed with the help of the identified system configurations. These adapted models can then also be edited manually, refined and developed during the entire design period.

4 Examples of Optimization and Validation Results

The developed SmoodSimulator approach is predestinated for all scalable tasks of an automated design regarding retrofit and energy concepts of buildings up to urban quarters. However, the current version focuses primarily on residential quarters and thus on the use of buildings as living space.

The toolset is currently used in a comparatively limited field. The required input data sets for the simulations of buildings are thus often similar.

As a starting point of any optimization run, the Caala tool performs a simplified energy analysis of a given building structure and type. The results are monthly energy demand characteristics of the considered building. However, this is only the basis of input data set generation for the HVAC and power supply system models.

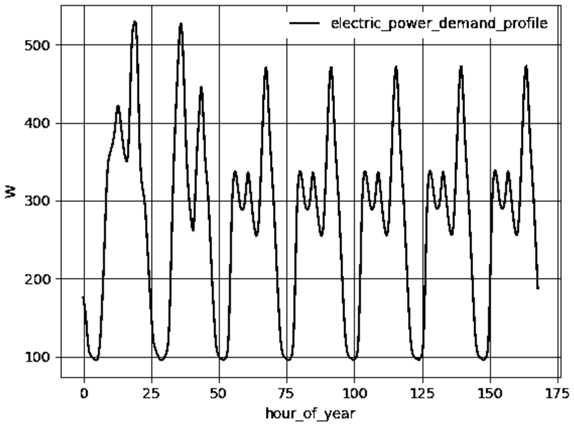


Figure 5: Example power consumption characteristic (1-week detail) of H0 standard load profile

Because of the similarity of energy consumption patterns in all potential households, generalized time series (i.e. standard load profiles – c.f. Figure 5) are the base to create load curves with higher temporal resolution. The Caala results only provide scaling factors, such as the cumulative energy demand values. This approach is used in the smoodSimulator for both electricity consumption and hot water demand.

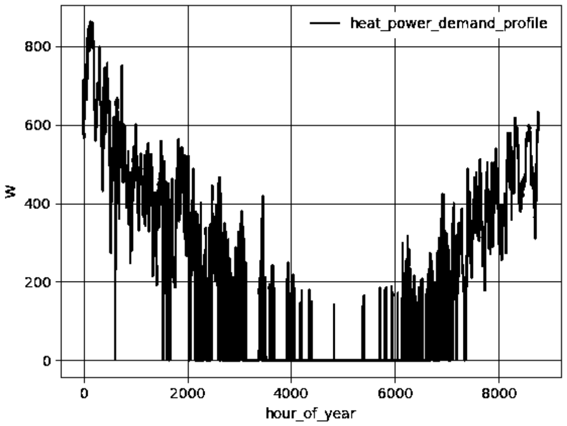


Figure 6: Example heat consumption profile based on TRY weather data

In addition to primarily occupancy-related energy consumption, buildings also consume climate-based heat. In order to be able to provide these with a higher temporal resolution, weather data (e.g. test reference year - TRY) with an hourly resolution are used as scaling base. The cumulative heat demands, again a result of the Caala tool, are weighted and distributed to a corresponding year profile with respect to the outdoor temperature and corresponding generic heat demands.

DOI
10.3384/ecp204789

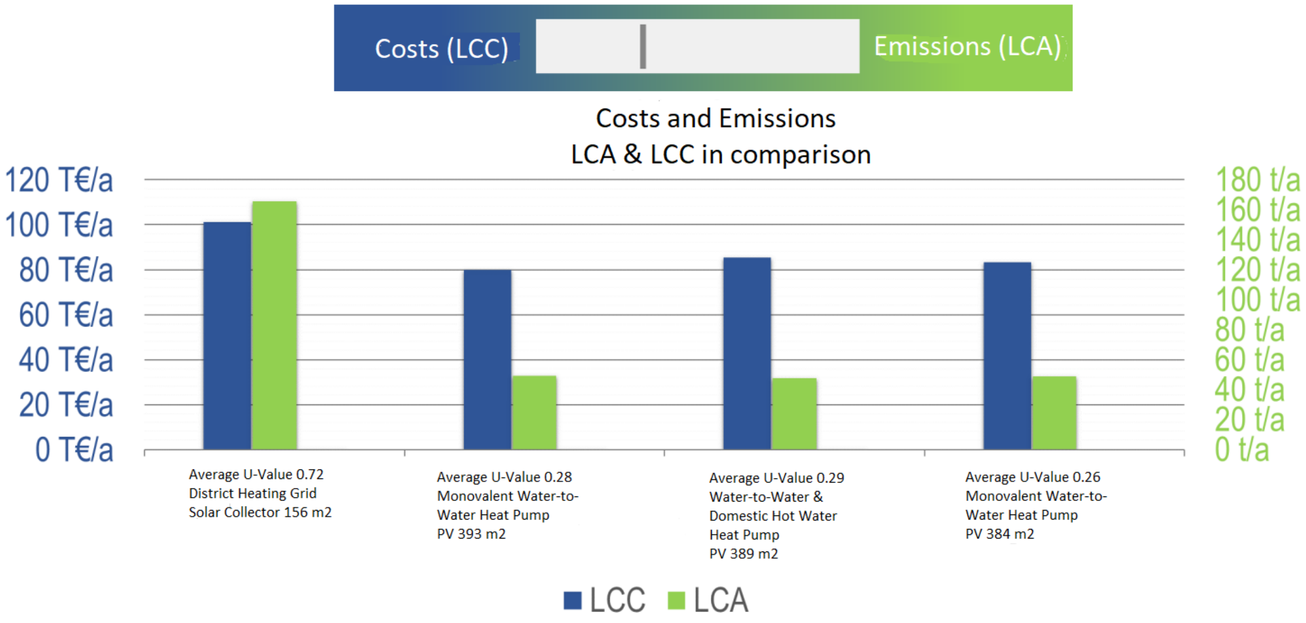


Figure 7: Example results of a holistic LCC/LCA analysis of an urban living quarter

However, the optimization framework does not only consider energy aspects. The goal of optimization is always a holistic system analysis, also in the direction of life cycle costs (LCC) and life cycle assessment (LCA). In this regard, adequate estimated values for the necessary investments are required in addition to suitable assessment factors.

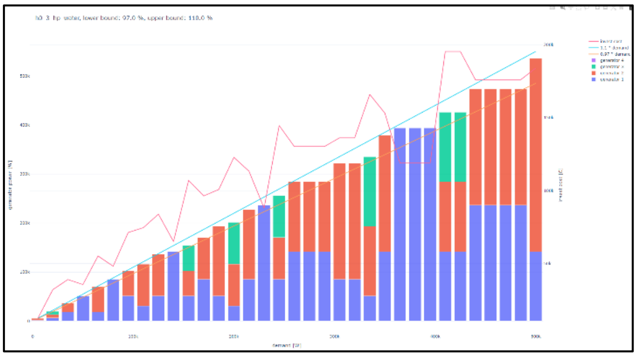


Figure 8: Cascading of heat supply systems depending on available system sizes and total power demands

The investment costs always depend primarily on the component size. However, not every system/unit size is available on the market as standard. Typically, the availability of the individual size follows a series development by power classes (e.g. 10 - 20 - 50 kW). Therefore, certain capacities can be achieved either with one unit or by cascading several units. Oversized units will cause additional costs. A cascade of several units is more expensive than one unit with the same total power. From this static optimization problem, an optimal component configuration can always be found for each (discrete) power class prior to any simulation run.

Figure 8 therefore shows an example configuration matrix of air-to-water heat pumps with different total (discrete) system sizes and individual optimal cascade solutions (i.e. blue / red / green – individual maximum power output of heat pumps).

The Model Assembler also organizes the optimal cascading of each analyzed system configuration automatically during the initial simulation period and the later iterative optimization process.

This also results in a necessary requirement for the parameterization of the models. No matter which models base is used, i.e. Modelica model or FMU, before the start of each simulation run all components parameters must be editable regarding new power categories and characteristics of the individual system configurations. However, this represents a huge challenge for the instantiation of the models, especially in the optimized FMU mode.

FMUs are not structurally changeable after export. Only constant parameters (unchangeable during simulation) can be adjusted. The modeling approach used in Green City is thus advantageous. It maps the cascadability of the individual component models by internal integer parameters.

Figure 8 shows exemplary results of one of the first test analyses. It considered a small urban living quarter in mid-Germany.

On the one hand, the analysis took into account the possible retrofit of the building envelope with different building materials (i.e. use of different U-values). On the other hand, different system configurations were analyzed

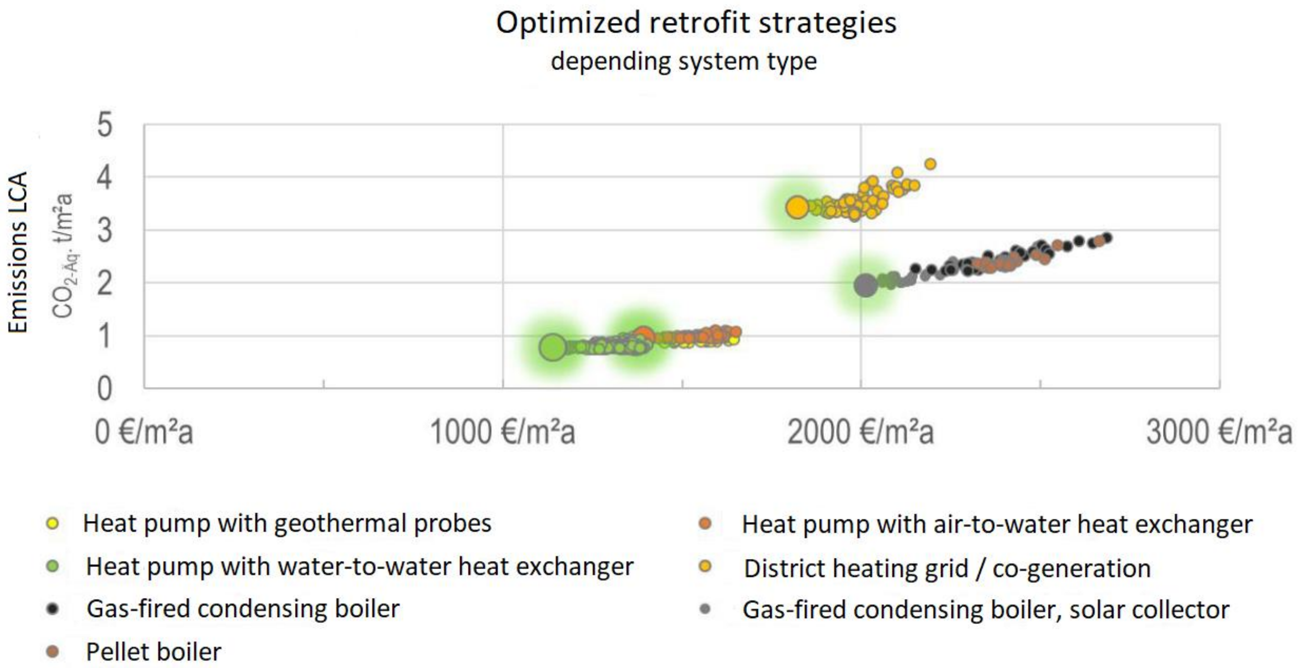


Figure 9: Full results of LCC/LCA optimization process of an urban living quarter

and their performance parameters were optimized with the help of the framework:

- District heating grid plus solar collectors
- Monovalent water-to-water heat pump and photovoltaics
- Bivalent heat pump system (air-to-water) for heating and domestic water supply

The existing system is district heating, which currently supplies heat monovalently only to buildings with simple insulation. In a simple optimization strategy, this is supplemented exclusively by thermal solar collectors. The retrofit carried out in this way generates the lowest investment costs. Over the lifetime, however, both the life cycle assessment (LCA) and the life cycle costs (LCC) are disproportionate to comparable heat pump systems.

Through iterative parameter optimization in the optimizer, the respective optimal system configuration with regard to LCAs and LCCs is found for all alternative supply concepts. The respective retrofit conditions of the building (i.e. the selected insulation standard) are also included in this optimization.

The example in Figure 8 shows the three best options, all describing different heat pump configurations. Depending on the efficiency and necessary costs of the selected technical systems, the energy standard of the building envelope can or must be adapted. Therefore, all variants describe slightly different insulation standards (i.e. U-values). The same applies to the use of local renewable generators (e.g. photovoltaics). Depending on the efficiency of the used heat pumps, slightly smaller or larger generator sets must be installed.

Figure 8 only shows the final results of the optimization process. It compares the baseline, i.e. the existing energetic standard and HVAC system, with the three best retrofit options of the optimization process.

Figure 9 adds to this plot a dot plot showing all parameter configurations simulated by the framework with the results of the LCC/LCA analysis for all evaluated system configurations.

It also shows different clusters of efficiencies and costs for different system configurations. The influence of parameter optimization is presented very precisely for each configuration from the deviations of the points of the same color. It is noticeable that the change of the HVAC system configuration has a significantly higher influence on efficiency and costs in each case than an optimal identification of the respective component parameters (e.g. power categories).

5 Conclusions

The presented approach of a holistic, automated optimization framework for energy systems of buildings and districts shows again the versatility of Modelica models.

On the one hand, multiphysical, noncausal modeling with Modelica is also widely used in the fields of energy supply and building services engineering. On the other hand, both the models themselves and their derivatives (especially

FMUs) can be integrated into automated software applications independently of tools.

Multiphysics simulation of power systems using Modelica allows very detailed evaluation of system behavior, especially in the presence of volatile generation and storage. However, these models are still very complex for use in optimization loops. Simplification to a minimum of necessary complexity is still required to ensure a reasonable time frame for optimization.

Further developments in the field of solver technologies are still necessary. This relates primarily to the computational speed of the models. In addition, knowledge-based potentials for increasing speed must continue to be tested and implemented. This mainly refers to the simplification of model equations by application-specific information. In this way, solvers can be relieved and high speed increases can be achieved. These approaches can be supported by the use of artificial intelligence and automated expert systems.

Another important next step is to extend the approach to non-residential buildings. These entail a significantly higher complexity and new dependencies for the optimization procedure and especially for the integrated simulation models. These arise primarily in the area of ventilation and air conditioning systems as well as refrigeration technology.

Since the approach of decoupling dependencies has already led to a significant acceleration of the existing optimization framework, this also seems to be a promising solution in this case. However, even such a solution can quickly reach its limits, especially due to increased complexity and deeper dependencies.

6 Acknowledgements

Supported by:

Federal Ministry
for Economic Affairs
and Energy

on the basis of a decision
by the German Bundestag

The presented research work and results were partly encouraged by subsidies of the German Federal Ministry of Economics and Technology.

References

Wetter, Michael (2009): Modelica-based Modeling and Simulation to Support Research and Development in Building Energy and Control Systems. In: *Journal of Building Performance Simulation 2(2), p. 143-161. DOI:10.1080/19401490902818259.*

Dirk Müller, Michael Lauster, Ana Constantin, Martin Fuchs, Peter Remmen (2016): "AixLib – an open-source modelica library within the iea-ebc annex 60 framework". In: *BauSim Conference 2016: 6th Conference of IBPSA-Germany and Austria. September 14-16, Dresden, Germany.*

Roselt, Kersten and Büttner, Christiane (2019). "Wachstumsmarkt energetischer Quartiersumbau". In: *DIB – Deutsches Ingenieursblatt 10-2019. October 2019, Germany.*

Leimeister, Mareike (2019). "Python-Modelica Framework for Automated Simulation and Optimization". In: *13th International Modelica Conference. March 04-06, 2019, Regensburg, Germany.*

Eckstädt, Elisabeth; Paepcke, Anne; Hentschel, Alexander; Schneider, André; Nicolai, Andreas; Schumann, Falk (2020). "Simulationsszenarien für Gebäudeenergiesimulation in frühen Planungsphasen". In: *BauSIM 2020, 8th Conference of IBPSA Germany and Austria, September 23-25, Graz, Austria.*

Dan Hou, Wei Yan, Gang Liu, Zhen Han (2019). "Primary-energy Based Optimization of a New Building District through Simulations on Flat, Building, Block and District Level". In: *16th International Building Simulation Conference, September 02-04, Rom, Italy.*

Ramón Bergel, Giovana Fantin do Amaral Silva, Max Tillberg, Angela Sasic Kalagasidis (2019). "Energy Performance Modelling: Introducing the Building Early-stage Design Optimization Tool (BeDOT)". In: *16th International Building Simulation Conference, September 02-04, Rom, Italy.*

Georgios Dermentzis, Fabian Ochs, Alexander Thuer (2019). "Primary-energy Based Optimization of a New Building District through Simulations on Flat, Building, Block and District Level": In: *16th International Building Simulation Conference, September 02-04, Rom, Italy.*

Theodor Victor Christiaanse, Paul Westermann, Will Beckett, Gaelle Faure, Ralph Evins (2021). "BESOS: a Python library that links EnergyPlus with optimization and machine learning tools". In: *17th International Building Simulation Conference, September 01-03, Brugges, Belgium.*

Javier Arroyo, Carlo Manna, Fred Spiessens, Lieve Helsen (2021). "An OpenAI-Gym environment for the Building Optimization Testing (BOPTEST) framework". In: *17th International Building Simulation Conference, September 01-03, Brugges, Belgium.*

Domenico Costantino and Massimiliano Pepe (2021): "Scan-to-HBIM for conservation and preservation of Cultural Heritage building: the case study of San Nicola in Montedoro church (Italy). In: *Journal of Applied Geomatics. DOI 10.1007/s12518-021-00359-2.*

Torsten Schwan and René Unger (2017): "Complex Large-Scale Heating, Cooling and Power Supply Models – Virtual Renewable Power Plant Simulation with Modelica". In: *15th International Building Simulation Conference, September 07-09, San Francisco, USA.*

Andreas Nicolai and Anne Paepcke (2017): "Co-Simulation between detailed building energy performance simulation and

Modelica HVAC component models". In: 12th International Modelica Conference. May 15-17, Prague, Czech Republic.

Bernd Kochendörfer, Jens H. Liebchen, Markus G. Viering (2010): "Bau-Projekt-Management: Grundlagen und Vorgehens-weisen. (Leitfaden des Baubetriebs und der Bauwirtschaft)". 4. Auflage. Vieweg & Teubner Verlag, ISBN 978-3-8348-0496-9.