THE AMERICAN MODELICA 2022 CONFERENCE

DALLAS

OCTOBER 26-28

Modelica

PROCEEDINGS OF THE AMERICAN MODELICA CONFERENCE 2022
Dallas, Texas, USA, October 26-28, 2022

EDITORS
Michael Tiller, Hubertus Tummescheit, Luigi Vanfretti, Christopher Laughman, and Michael Wetter

PUBLISHED BY
Modelica Association and Linköping University Electronic Press

Series info: Linköping Electronic Conference Proceedings 186
ISBN: 978-91-7929-218-8
ISSN: 1650-3686
eISSN: 1650-3740
DOI: 10.3384/ecp21186

ORGANIZED BY
North America Modelica Users' Group
na.modelica-users.org

in co-operation with:
Modelica Association
c/o PELAB, Linköpings Univ.
SE-581 83 Linköping
Sweden

CONFERENCE LOCATION
Davidson-Gundy Alumni Center
University of Texas at Dallas
800 West Campbell Road
Richardson, TX 75080
USA

SCOTT A. BORTOFF

Chief Scientist,
Mitsubishi Electric Research Laboratories (MERL)

BIO

Scott A. Bortoff is Chief Scientist at Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA, USA. His research interests include mathematical modeling and control of thermofluid systems and mechatronic systems. Prior to joining MERL in 2009, Scott was group leader of Control Technology at United Technologies Research Center (now Raytheon Technology Research Center), in East Hartford, CT, USA. During his 9.75 year tenure, he led several projects that used Modelica for system-level dynamic modeling and control design of fuel cell power plants, airborne power generation and distribution systems, and HVAC systems. He held positions of Assistant and Associate Professor of Electrical and Computer Engineering at the University of Toronto from 1992-2000, where he conducted research and taught courses in control. He received the B.S. and M.S. degrees from Syracuse University in 1985 and 1986, respectively, and the Ph.D. degree from the University of Illinois at Urbana-Champaign in 1992, all in Electrical Engineering. He is currently Associate Editor of the IEEE Control System Magazine and serves the Modelica community in various capacities.

ABSTRACT

Sustainable HVAC: Research Opportunities for Modelicans

What are the modeling and control research challenges that, if addressed, will drive meaningful innovation in sustainable building HVAC systems in the next 20 years? In particular, what innovations might help to address the considerable problems with the earth's climate that are caused by human activity, especially greenhouse gas production? Buildings, as end users of various forms of energy (mainly electricity, natural gas and oil), account for approximately one third of greenhouse gas emissions worldwide, and their HVAC systems are major energy consumers.

In this talk, I will present several ideas and opportunities in the HVAC area that might help to address this sustainability problem. First, at the equipment level, energy-efficient and low-GWP HVAC products can be engineered, but present challenges especially related to control. Building envelopes can be designed to produce lower heating and cooling loads, and also to provide integrated power production sufficient to meet HVAC demand, but issues related to storage and intermittent supply remain. HVAC operations can be monitored using ``digital twins'' that make visible what is otherwise invisible, helping to maintain efficient operation. Yet these technologies are immature and fragile, and challenges remain in designing and maintaining robust operation. Modelica, as a tool for modeling, simulation and analysis of multiphysical, heterogeneous systems, can and will continue to play a major role in developing and maturing new building HVAC technologies for both products and services. The Modelica community has made significant contributions to this area, such as through the development of open-source libraries, and should continue to view these urgent problems as opportunities.

DIRK ZIMMER

Head of Team for Aircraft Energy Systems at
German Aerospace Center (DLR)

BIO

Dr. Dirk Zimmer is head of a research team for the modeling and simulation aircraft energy systems at the German Aerospace Center (DLR). He received his PhD degree at the Department of Computer Science of the Swiss Federal Institute of Technology (ETH). Also, he is lecturer at the Institute of Computer Science at the Technical University of Munich (TUM) and member of the Modelica Association. Recently he developed a new computational scheme for complex thermal architectures and the corresponding library was awarded with the Modelica Library Award in 2021.

ABSTRACT

Mathematical Modeling in Modelica: The Art of Compressing Reality

When we enjoy the freedom of mathematical expression in Modelica, we build on the legacy of many great scientist who condensed the laws of nature into elegant equations. This makes differential-algebraic equations an extremely powerful modeling tool to compress the real system down to an intelligible, computable and hopefully controllable form.

But all too often, this compression is flawed and as simulation engineers we run into errors. For large modeling efforts, the freedom of expression needs to be rigorously constrained by a clear methodology. All experienced modelers know that the dull application of text-book equations will not bode well in a large, challenging simulation project. But how to do better? How to ensure that you reach a computational feasible form that is still effective? How to ensure that what works on component level will work on system level?

In this keynote, I revisit the principles of idealization that modelers often apply without being aware of them. I demonstrate how quickly idealization can lure you into problems rather than solving them. But better understanding the process of idealization also offers an approach leading to a robustly solvable form, even for large complex systems. The immediate practical value is demonstrated by the simulation of complex thermal architectures and the real-time simulation of stiff mechanical contact problems.

CONFERENCE CO-CHAIRS

Dr. Michael Tiller, Ricardo

Dr. Hubertus Tummescheit, Modelon

PROGRAM CO-CHAIRS

Prof. Luigi Vanfretti, RPI

Dr. Michael Wetter, Lawrence Berkeley National Laboratory

CONFERENCE EXECUTIVE COORDINATOR

Dr. Christopher Laughman, Mitsubishi Electric Research Laboratories

LOCAL CHAIR

Prof. Yaoyu Li, University of Texas at Dallas

Behnam Afsharpoya, Dassault Systèmes

PROGRAM COMMITTEE

Miguel Aguilera, ICE

Bernhard Bachmann, Fachhochschule Bielefeld

Christian Bertsch, Robert Bosch GmbH

Volker Beuter, VI-grade GmbH

David Blum, Lawrence Berkeley National Laboratory

Scott Bortoff, Mitsubishi Electric Research Laboratories

Timothy Bourke, INRIA

Daniel Bouskela, EDF

Robert Braun, Linköping University

Felix Bünning, Empa/ETH Zürich

Yan Chen, Pacific Northwest National Lab

Clément Coïc, Modelon Deutschland GmbH

Sergio A. Dorado-Rojas, Rensselaer Polytechnic Institute

Atiyah Elsheikh, Mathemodica.com

Olaf Enge-Rosenblatt, Fraunhofer

Gianni Ferretti, Politecnico di Milano

Virginie Galtier, CentraleSupélec

Valentin Gavan, ENGIE Lab

Anton Haumer, OTH Regensburg

Dan Henriksson, Dassault Systemes

Yutaka Hirano, Woven Planet Holdings, Inc.

Jianjun Hu, Lawrence Berkeley National Laboratory

Christian Kral, Electric Machines, Drives and Systems

Alessandro Maccarini, Aalborg University

Alexandra Mehlhase, TU Berlin

Thierry S Nouidui, The United African University of Tanzania

Hans Olsson, Dassault Systèmes

Martin Otter, DLR, Institute of System Dynamics and Control

Kaustubh Phalak, Ingersoll Rand

Meaghan Podlaski, Rensslaer Polytechnic Institute

Adrian Pop, Linköping University

Johan Rhodin, ModSimTech, LLC

Lisa Rivalin, Facebook

Clemens Schlegel, Schlegel Simulation GmbH

Michael Sielemann, Modelon Deutschland GmbH

Giorgio Simonini, EDF

Martin Sjölund, Linköping University

Wilhelm Tegethoff, TLK-Thermo GmbH

Matthis Thorade, Modelon

Jakub Tobolar, DLR - German Aerospace Center

Alfonso Urquia, Universidad Nacional de Educación a Distancia (UNED)

Volker Waurich, TU Dresden

Dietmar Winkler, University of South-Eastern Norway

Stefan Wischhusen, XRG Simulation GmbH

Dirk Zimmer, DLR

Marcelo de Castro Fernandes, Federal University of Juiz de Fora

CONTENTS

PAPERS

BESMod - A Modelica Library providing Building Energy System Modules

Fabian Wüllhorst[1] Laura Maier[1] David Jansen[1] Larissa Kühn[1] Dominik Hering[1] Dirk Müller[1]

[1]Institute for Energy Efficient Buildings and Indoor Climate, E.ON Energy Research Center, RWTH Aachen University, Germany, fabian.wuellhorst@eonerc.rwth-aachen.de

Abstract

Towards the analysis and optimization of a coupled building energy sector, various component model libraries for hydraulic, ventilation, electrical, control, and building domains exist. However, no uniform open-source framework to couple these domains in a holistic building energy system simulation exists. Thus, we present *BESMod*, an open-source Modelica library, providing a modular approach towards domain-coupled building energy system simulations. *BESMod* relies on existing component specialized model libraries for the underlying physics. For the analysis of complex system simulations, user-friendly parameterization, consistent model interfaces, precalculated KPIs and debugging options are applied. The library is available at www.github.com/RWTH-EBC/BESMod. This paper motivates the library, lays out the interaction with existing model libraries and the general modular approach. An exemplary use case demonstrates the applicability of *BESMod*. Concluding, we motivate future development options.

Keywords: Coupled simulations, HVAC, Building, Modular

1 Introduction

Towards the integration of renewable energy into a domain-coupled building sector, coupled simulation based analysis of the hydraulic, ventilation, electric, control, and building domain can aid the development of innovative design and control methods (Ramsebner et al. 2021). To this extent, various Modelica libraries exist with the focus on modeling specific components from the domains hydraulic, ventilation, electric, and building (Modelica Association 2022). All existing libraries deliver relevant component[1] models. Besides providing simple system examples (cf. section 2), none of the mentioned open-source libraries aim for the aggregation of different components into compound energy systems.

One possible reason is the heterogeneity and complexity of energy systems. Especially for buildings, energy systems tend to be unique and, hence, not easily transferable. However, re-occurring subsystems[2] exist (EN 15316-1 2017). Researchers and practitioners often combine the same components to different hydraulic, ventilation, and electrical subsystems. For instance, the combination of a heat pump and a heating rod (hydraulic), a ventilator and a heat recovery (ventilation), or a photovoltaic power plant (PV) and a battery energy storage system (electrical) are typical subsystems. To enable the aggregation of components into subsystems and subsystems into a unique building energy system (BES), a high level of modularity of each subsystem is required (Baldwin and Clark 2000).

As no such library exists, researchers need to aggregate existing component models into a BES each time they want to model interactions between hydraulic, ventilation, electric, control and the building envelope. However, towards the integration of renewable energy and the coupling of sectors, analysis of these interactions are more important than ever (Ramsebner et al. 2021). Following this need, we highlight and close two research gaps:

- *Gap 1:* Modelica enables the modeling and coupling of different domains. While models for different domains exist, no uniform approach for coupling relevant domains within one BES exists.
- *Gap 2:* Coupled system models encompass large equation systems and countless parameters. Thus, the analysis is time-consuming and error-prone (Remmen et al. 2018). While existing libraries are user-friendly on component level, no clear approach exists for system analysis (cf. section 2).

To close these gaps, we develop the open-source library *BESMod*, a Modelica library for Building Energy System Modules. Regarding *Gap 1*, we realize a modular subsystem structure and a straightforward aggregation of subsystems into a unique BES. Further, we supply user-friendly and consistent approaches for parameterization, system analysis, and debugging to address *Gap 2*.

The remainder of this publication is structured as follows: Section 2 discusses and compares existing Modelica libraries for BES simulations in detail. Based on deducted

[1]In the context of this paper, a *component* refers to a single device in an energy system, such as a mover, a pipe or a storage.

[2]In this paper, we refer to subsystems as *modules*.

gaps, *BESMod* is presented in section 3. Section 4 describes an exemplary use case for the annual simulation of two building models. Finally, section 5 concludes with further development options and furture use cases.

2 Existing Libraries

In recent years, a variety of open-source libraries for the modeling language Modelica have been developed. Following the idea of *BESMod* library, we focus our state of the art on open-source Modelica libraries that are relevant for BES modeling and simulations. As a basis for our literature review, we take the Modelica Association's library list (Modelica Association 2022). Table 1 qualitatively categorizes libraries which are considered relevant for further evaluation and comparison with *BESMod* regarding the following criteria: On the component level, we distinguish between the existence, level of detail, and comprehensiveness of hydraulic (H), ventilation (V), electrical (E), control (C), and building (B) envelope models. Following *Gap 1*, we introduce the existence of system (Sys) configurations as an additional category. For instance, system packages include a connected overall BES or aggregate subsystems in a similar form. Furthermore, the libraries are evaluated regarding their modularity (Mod), i.e., the interconnectability of the subsystems. Following *Gap 2*, the aggregation of subsystems to a fully connected BES results in a tremendous increase in complexity. Therefore, simple and consistent parameterization (Par) is a key feature that modeling libraries should offer. For this category, the focus lies on consistency and simplicity of parameterization. For example, we evaluate if parameters are organized (e.g. in records), if parameter naming is consistent, whether the parameters were propagated and if measures to aggregate or simplify system parameters are offered. Finally, open-source development suffers from poor maintenance (Main) in many cases. Maintenance increases both model and library quality and is thus defined as the last category.

Twelve libraries are classified as relevant for *BESMod*. The first five libraries, namely *IBPSA* (Wetter, Blum, et al. 2019), *AixLib* (Müller et al. n.d.), *Buildings* (Wetter, Zuo, et al. 2014), *BuildingSystems* (Nytsch-Geusen et al. 2013), and *IDEAS* (Jorissen et al. 2018) are part of the cooperation within *IBPSA* Project 1 and are the follow-up projects of the former IEA EBC Annex 60 (Wetter and Treeck 2017; Wetter 2022). The aim of Annex 60 and Project 1 is to develop a new generation of open-source Modelica-based computational tools for building and district energy systems. One of the project's outcomes is the joint modeling library Modelica *IBPSA* as well as the four additional separate modeling libraries which all incorporate the *IBPSA* models but supplement them with further modeling efforts. In Table 1, we evaluate the libraries as they are, neglecting the fact that *AixLib*, *Buildings*, *BuildingSystems*, and *IDEAS* partially consist of *IBPSA* models and vice versa. For example, *AixLib*'s high representation of hydraulic models is partially due to a solid representation in the core library *IBPSA*. For instance, the *BuildingSystems* and *IDEAS* introduce e.g. battery and photovoltaic models or the *Buildings* adds detailed Building and EnergyPlus coupled building models.

For building envelope, the *IBPSA* offers the basis to model simplified building envelope models based on reduced-order approaches. These models are enhanced by the *Buildings* library which offers an extensive interface for EnergyPlus models and by the *AixLib*, introducing high-order modeling approaches (Xanthopoulou et al. 2021). System aggregation possibilities and modularity are similar among the regarded *IBPSA*-based libraries. All of them introduce examples to increase comprehensiveness or even integrate tutorials on how to build your own BES model, e.g. *IDEAS* and *Buildings*. Yet, system aggregation is not based on replaceable subsystems with uniform interfaces, leaving room for improvement. In addition, the coupling between electrical and thermal models is impeded by inconsistency, i.e. different electrical ports, or non-existent interfaces, i.e. electrical power as component output.

In addition to *IBPSA*-related libraries, few Modelica-based libraries focusing on holistic building performance simulation have successfully been introduced over the past years. Among them, the *BuildingSysPro* (Plessis, Kaemmerlen, and Lindsay 2014) is the only library outside the Project 1 cooperation (Wetter 2022) that also includes the *IBPSA*, but does not use it within the models. Like the *IBPSA*-based libraries, the *BuildingSysPro* includes models of all relevant subsystems, too. However, system aggregation is not included and the examples which refer to system simulation tend to focus on specific components. In addition, parameterization lacks the organization in records and detailed documentation, resulting in less manageable parameter structures. Furthermore, modularity is impeded due to the omission of consistent interfaces (e.g. busses) and subsystems, respectively.

Another library providing models for building performance simulations is the *FastBuildings* (Coninck et al. 2014). It is developed for low-order and grey-box modeling of buildings and simplified heating, ventilation, and air conditioning (HVAC) components for model predictive control applications. These individual subsystems can be integrated in other frameworks. However, the modeling and parameterization effort is rather high. Further, the last changes have been committed 7 years ago, thus the library appears to be inactive.

The *ThermofluidStream* modeling library enables detailed simulations of complex thermofluid systems and refrigerants (Zimmer 2020). Again, the individual subsystems are suitable to be incorporated in external BES simulation frameworks, but the library itself does not aim at providing them. The usage of different ports further inhibits an integration of *IBPSA*-based models.

The remaining open-source modeling libraries focus on selected domains rather than providing holistic coupled

Table 1. Comparison of open-source Modelica libraries offering models relevant to BES. The following abbreviations are used: H:=hydraulic, V:=ventilation, E:=electrical, C:=control, B:=building, Sys:=system, Mod:=modularity, Par:=parameterization, Main:=maintenance. The evaluation metrics are qualitative measures ranging from - - (poor representation/not existent), 0 (intermediate representation) to ++ (very good representation).

	BES subsystems									
Name	**H**	**V**	**E**	**C**	**B**	**Sys**	**Mod**	**Par**	**Main**	**URL**
IBPSA	0	0	- -	-	0	0	+	0	++	`https://github.com/ibpsa/modelica-ibpsa`
AixLib	++	+	0	+	+	+	+	+	+	`https://github.com/RWTH-EBC/AixLib`
Buildings	++	+	+	+	++	+	+	+	++	`https://github.com/lbl-srg/modelica-buildings`
Building Systems	+	0	++	-	+	+	+	0	0	`https://github.com/UdK-VPT/BuildingSystems`
IDEAS	+	0	++	+	+	+	+	0	+	`https://github.com/open-ideas/IDEAS`
Building SysPro	0	0	+	+	+	-	- -	-	0	`https://github.com/EDF-Lab/BuildSysPro`
FastBuildings	- -	-	- -	- -	0	-	-	- -	- -	`https://github.com/open-ideas/FastBuildings`
Thermofluid-Stream	+	+	- -	+	- -	- -	0	+	+	`https://github.com/DLR-SR/ThermofluidStream`
dhcSim	0	- -	- -	-	- -	0	-	-	-	`https://github.com/mabachmann/dhcSim`
DisHeatLib	-	- -	- -	+	- -	+	0	0	0	`https://github.com/AIT-IES/DisHeatLib`
TransiEnt	-	- -	+	0	- -	+	+	-	+	`https://github.com/TransiEnt-official/transient-2.0.1`
Building-ControlLib	- -	- -	- -	+	- -	- -	- -	-	- -	`https://github.com/TechnicalBuildingSystems/BuildingControlLib`

building performance simulation. Nonetheless, they are potentially suitable to be integrated into the respective subsystem packages. For example, the *dhcSim* (Bachmann et al. 2021) and *DisHeatLib* (Leitner et al. 2019) both specialize on district heating and cooling simulations. Since they primarily aim is to enable large district simulations, the subsystems are simplified for low computational effort. *DisHeatLib* provides well documented and extensive system examples and basic control blocks for district use cases. However, the modularity is limited. *dhcSim* provides less extensive system examples and components. Another library that falls into the category of domain-specific libraries is the *TransiEnt* library (Andresen et al. 2015). Here, the focus lies on coupled energy systems on grid level. Consequently, the lower level BES is not part of the library. In addition, the library's documentation is limited and the parameterization effort is rather high. This impedes an integration into other frameworks. Nevertheless, the library provides extensive system examples. A satisfactory level of modularity is provided.

Finally, the *BuildingControlLib* is another domain-specific Modelica library (Schneider, Pessler, and Steiger 2017). The library aims at providing state-of-the-art control modules which occur in BES. Unfortunately, the library is not maintained anymore.

The comparison and categorization of different open-source Modelica libraries enabling building performance simulations reveals that the scientific community currently lacks a library that fulfills all of the following criteria:

- Selection of models for all relevant building subsystems, namely hydraulic, ventilation, electrical, control, and building envelope.
- Straightforward option to aggregate subsystems to coupled BES.
- Uniform interfaces and model structures among all subsystems to increase modularity and facilitate system aggregation.
- Consistent parameterization within and between the subsystems to reduce parameterization effort.

To bridge these gaps, we developed the modeling library *BESMod* which is presented in the following chapters. *BESMod* joins the rich pool of existing component models, providing models for full, domain-overarching BES simulations. Consequently, rather than being an alternative to the libraries discussed above, we enhance the current library portfolio and build upon the existing libraries by integrating them as dependencies.

3 Library Design

Based on the gaps identified in Section 1 and 2, we present the library design of *BESMod*. First, the integration of the existing component library pool is discussed (subsection 3.1). The realization of modularity is presented in subsection 3.2. The following subsections discuss the implementation of the subsystems hydraulic, ventilation, electrical, control, building, DHW and user profiles (subsections 3.3-3.9). After the description of the system aggregation in subsection 3.10, subsection 3.11 presents a keystone for high quality maintenance of *BESMod*, namely *Continuous Integration*.

Figure 1 illustrates *BESMod's* library structure. Besides the common `User's Guide`, the `Systems` package is the heart of *BESMod*. The `Utilities` package provides interfaces, icons and other. The `Examples` package contains the use case presented in section 4 as well as additional examples demonstrating the flexibility of *BESMod*. Last, the `Tutorial` package is an addition to the `User's Guide` to document how inheritance and replaceability of modules work.

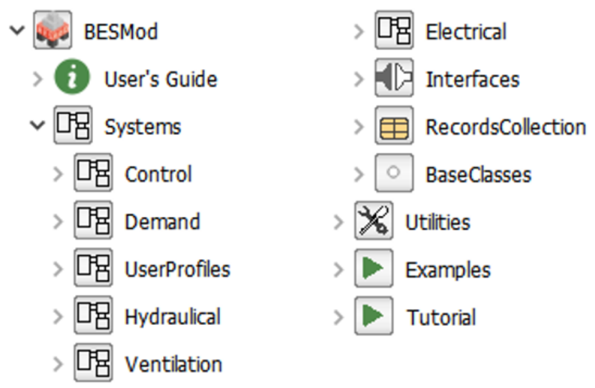


Figure 1. Package order of *BESMod*.

3.1 Dependencies

BESMod focuses on system models, not component models. As existing libraries already contain numerous component models of high quality, we build *BESMod* upon these libraries. Figure 2 depicts this setup. We distinguish between required and optional libraries.

The Modelica Standard Library (MSL) and the *IBPSA* library are required libraries. While the former is an obvious requirement, the choice of the *IBPSA* library is based on our state of the art. Out of all libraries listed in Table 1, the libraries derived from the *IBPSA* have the highest potential on component level for BES. Further, maintenance is applied, and the developer community is active. Thus, we choose the *IBPSA* library as the most suitable required library to integrate fluid models, media models, and other, which are compatible to the majority of component model libraries.

All other libraries are optional. At the current state of development, existing subsystems use the *AixLib*, the *Buildings*, and the *BuildingSystems* library.

The Python script `install_dependencies.py` installs all optional and required dependencies. For further information, the repositorie's `README.md` provides detailed documentation.
Regarding software dependency, *BESMod* is only tested using Dymola. As soon as the component libraries simulate in OpenModelica, *BESMod* will be tested for OpenModelica as well to enable the simulation of the open-source library in open-source software.

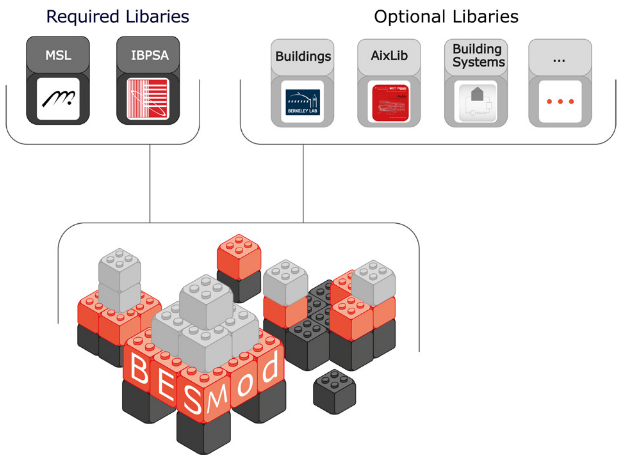


Figure 2. *BESMod* is built upon required and optional library dependencies.

3.2 Modular Subsystem Design

Baldwin and Clark (2000) define modules as “units in a larger system that are structurally independent of one another, but work together“ (Baldwin and Clark 2000, p. 63). To achieve modularity, a framework needs to allow “for both independence of structure and integration of function“ (Baldwin and Clark 2000, p. 63). Transfering these design rules to Modelica, we build each module with the same concept using expandable bus connectors, vector sized ports, and a uniform parameterization approach.

Expandable bus connectors: Each module is structurally independent. To exchange information with other modules, we implement expandable bus connectors (type `Real`, `Integer`, and `Boolean`). Five types of bus connectors arise. We distinguish between this variety of bus connectors by using color coding.

First, each module with HVAC components contains a bus to interact with the control module (white). As a naming convention, we apply the idea of the *AixLib*. For instance, a radiator transfer system receives the thermostatic valves set point `yValSet` as an input and sends the current valve position `yValMea` as an output. Using `Set` and `Mea` as indicators, users can always distinguish between control's and system's in- and outputs.

Second, each module contains a bus to pass on relevant key performance indicators (KPI) of the module to the system (gray). This facilitates a fast analysis of complex BES. For instance, a generation subsystem using a heat pump may always output the consumed electrical energy, the supplied heat, and the number of starts. As we define these KPIs on a module level and propagate them using expandable bus connectors, the user can directly access the most important simulated states by analyzing the `outputs` bus.

The existing weather bus of the *IBPSA* library is used to distribute the boundary conditions to all subsystems (yellow). For user set points, all control models have access to the `UseProBus`, which includes for instance user presence and temperature set points (green). Last, the `BuiMeaBus` distributes current building measurements to the control subsystems (red).

Vector sized ports: Besides the exchange of control signals and system inputs, a module may exchange heat, mass, and electricity with other modules. While the usage of bus connectors for that purpose is generally possible, existing libraries commonly use the `FluidPort` for mass, the `HeatPort` for heat, and the `Pin` for electricity from the *MSL*. Thus, we consider the usage of expandable busses in this context as non-intuitive. Furthermore, a bus also only serves for information exchange in reality.

The individual use of ports depends on the type of subsystem and is elaborated in the accompanying subsections. Besides DHW supply, we apply vector sized ports. Thus, any number of parallel thermal zones, hydraulic circuits, or air ducts may be simulated. Again, this guarantees modularity.

Parameterization: Beutlich and Winkler (2021) motivate the decoupling of the "behavioral implementation from its actual design parameters" to facilitate a robust parameterization of simulation models. Applying this general method to *BESMod*, four model parameter types result.

1. **Top-Down**: All parameters given by the parent system. For instance, the heat demand of the building is a top-down parameter.

2. **Bottom-Up**: All parameters used by the parent system, but are inherently defined by the components in the subsystem. For instance, the system's pressure drop is given by the component models.

3. **Component choices**: All replaceable component models or packages. For instance, the hydraulic control system contains a replaceable thermostatic valve control. Thus, users may switch between P- and PI-controlled valves directly.

4. **Component records**: Each component in the system is equipped with a replaceable parameter record. This decouples the physical parameters from the modeling as motivated in (Beutlich and Winkler 2021). Ideally, such records would already exist in the component library. As most libraries do not support this approach, we add records for several components, for instance for the *IBPSA* mover models.

To enable a simple parameterization, composing the component records as a function of top-down parameters is convenient. Currently, the hydraulic subsystems follow the approach presented in previous work (Wüllhorst et al. 2022). In future versions, the electrical and ventilation subsystems may apply similar approaches.

While all subsystems follow this modular layout, differences occur depending on the subsystem's type.

3.3 Hydraulic Subsystem

In the following subsections, we highlight the choices for the **system layout**, the **connector** choice, the **parameterization** approach, and present already **available subsystems**.

System layout: The layout of the hydraulic subsystem follows (EN 15316-1 2017), which separates the system in generation, distribution, and transfer subsystems. To decouple the physical components from the control, we separate the control subsystem. The resulting subsystem layout is depicted in Figure 3 on the lower left.

Connectors: Water-based heating systems have two tasks. Firstly, they supply heat to possibly multiple thermal zones. Secondly, they provide DHW. Thus, the hydraulic subsystem contains heat and fluid ports from the MSL.

Parameterization: Wüllhorst et al. (2022) present a uniform method towards minimal parameterization effort in BES simulations. Inhere, each subsystem contains the exact same parameters to be quantified. The method follows the analogy of thermal and electrical systems and uses nominal heat and temperature demands to calculate remaining parameters. For instance, the mass flow rate is given by the nominal heat flow rate divided by the nominal temperature difference and the specific heat capacity of the fluid. As the demands are given by the building and DHW models, these are top-down parameters. Using correlations between top-down parameters and the selected component records, all remaining parameters are quantified. Fine-tuning is always possible, as we use the `final` modifier for top-down parameters only.

Available Subsystems: *BESMod* was originally developed for hydraulic-based BES simulations. Thus, a rich pool of subsystems already exist. Direct electric heating, bivalent heat pumps, gas boilers, solar thermal systems, different storage options, and radiator as well as underfloor heating subsystems may be combined. As control system, several approaches based on Vering et al. (2021) and the corresponding state of the art for the control of bivalent heat pump systems are implemented, too.

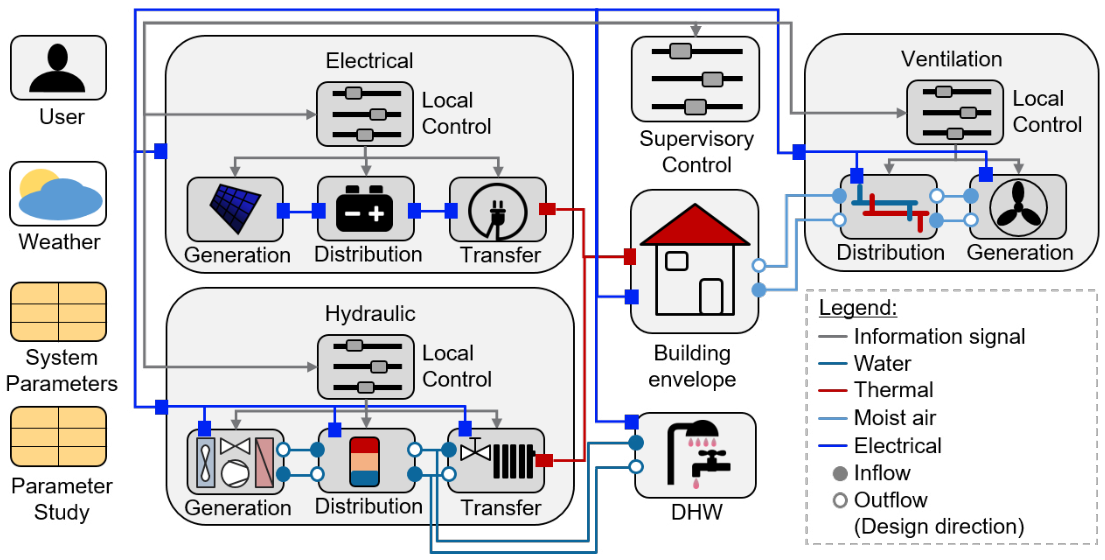


Figure 3. The aggregation of subsystems into the building energy system

3.4 Ventilation Subsystem

The ventilation system is used for heating, cooling, and air supply to control temperature, moisture and CO_2 concentrations. Especially towards the analysis and optimization of new buildings, simulation of ventilation coupled to hydraulics and electrics is vital.

System layout: As the hydraulic subsystem, the layout of the ventilation is based on the core idea of (EN 15316-1 2017). However, as the supply air directly flows into the thermal zones, no transfer system is required. The resulting subsystem layout is depicted in Figure 3 on the upper right.

Connectors: The ventilation system exchanges air with multiple thermal zones. Thus, fluid ports are used in addition to the usual bus connectors.

Parameterization: Similar to the hydraulic system, the parent system propagates temperature and heat demands top-down to each subsystem. Besides this propagation, no simplifying parameterization principles are applied. The supply flow rate has to be defined in each case. For future versions, we plan an automatic parameterization based on the net least area according to DIN 1946-6 (2019).

Available Subsystems: Currently, only a simple configuration exists in the open-source library. Inhere, a heat recovery system is simulated and equally distributed to all connected thermal zones. While only this example is open-source, more complex systems have been investigated in internal projects. For future versions, usage of air handling unit models, for instance, as presented in the *AixLib*, could be integrated.

3.5 Electrical Subsystem

The electrical subsystem in the *BESMod* library is used for the generation and distribution of electricity, as well as electricity-based space heating.

System layout: Based on this usage, the system layout is equal to the hydraulic system's one. Even though the decomposition into generation, distribution, and transfer derives from hydraulic systems, it is equally suitable for electrical systems.

Connectors: The existing *IBPSA*-related Modelica modeling libraries for BES focus on thermal and hydraulic subsystems and respective components. Even though the concept of electrical pins exists, e.g., in the MSL or the *Buildings*, they are not jointly used among the other libraries. In addition, most components which were especially modeled for thermal simulation assessments simplify electrical relations and solely calculate the electrical power output instead of detailed voltage or current relations. Hence, we introduce a simplified electrical connector which incorporates the flow variable power, only. Apart from that, the *IBPSA* weather bus and radiative and convective heat ports to the thermal zones are included, as well. The latter is an interface for electricity-based heat transfer by, e.g., electrical floor heating systems.

Parameterization: In contrast to the modeling approach by Wüllhorst et al. (2022), we parameterize each subsystem of the electrical system mostly independently. Nonetheless, the electrical subsystem predominantly relies on records and the definition of relations based on design parameters. For example, we introduce a design factor as a bottom-up parameter for system sizing relative to the building envelope's roof area, which is a top-down parameter.

Available Subsystems: The electrical subsystem incorporates the most commonly used electrical components in BES, namely a PV as generation subsystem and a battery energy storage subsystem to represent the distribution side. The PV subsystem is based on the *AixLib* model. Following the library's aim of simple parameterization, the PV model's parameters are organized in records and system specific information is propagated to the top level. Again, to achieve a high degree of modularity, the system is introduced as a vector, providing the possibility to consider PV systems of different orientations, tilts, etc. Future developments for the generation models will include wind power plants.

As an exemplary model for the electrical distribution models, a simplified battery energy system based on the *BuildingSystems* library and a direct grid connection are included (Nytsch-Geusen et al. 2013). Again, different battery types can be selected based on a quick record change.

3.6 Building Envelope

The heart of any BES simulation is the model for the building envelope. For this subsystem, *BESMod* enables the connection to all three domains (hydraulic, ventilation, and electric) using heat, fluid, and electrical ports. The building subsystem outputs all relevant measurements to the other control systems. Currently, only temperature control (`TZoneMea`) is enabled. Future versions may add, e.g., moisture measurements and set-points.

For parameterization, no top-down parameters are relevant, as the building itself defines the dimensions of the HVAC components. Thus, various bottom-up parameters are used to propagate the building geometry to the residual subsystems. Currently, the building's height, ground area, roof area, and the area and height of each thermal zone have to be specified by the subsystem.

The current version contains two building models. The reduced order model used in TEASER (Remmen et al. 2018), as well as the detailed building envelope of the *Buildings* Library. The example is using the BESTEST validation model `Case600FF`.

3.7 DHW Subsystem

The DHW subsystem models the tapping of water from the hydraulic system. Thus, if the hydraulic system is not considered, DHW is disabled as well. For ports, fluid ports are an obvious choice.
Currently, DHW tapping according to *EN 16147* (2017) and Jordan and Vajen (2005) is included.

3.8 User Profiles

Both building and DHW subsystems greatly depend on the user profiles and vice versa. As no uniform approach to model internal gains, DHW tapping, etc. exists, no uniform user profile system is given. Instead, a replaceable model using the `UseProBus` enables a fully modular implementation of any user profile approach.
For instance, the `Case600FF` building model assumes area specific heat gains for internal gains. The `TEASER` model calculates the internal gains with relative inputs between 0 and 1. Besides internal gains, different set-points (constant, night setback, etc.) may be implemented.

While DHW tapping is a user action, we model the tapping profiles in the DHW subsystem itself. This avoids numerous user profile combinations, which would arise if different DHW and building model approaches are combined. Furthermore, disabling the DHW subsystem all together is straight forward if the profiles are in the DHW subsystem.

3.9 Control Subsystem

Last, a supervisory control system enables the simulation of an overarching home energy management system. Even though the hydraulic, ventilation, and electric subsystem all have a local control, no coordination of the coupled domains is possible. We consequently introduce the possibility to include a supervisory control model on system level.

The local control decides whether to enable supervisory control or not. Extending the supervisory control implemented by Blum et al. (2021), the user can select if the control signal derives from (i) the local control model only, (ii) a Modelica internal supervisory control model, or (iii) an external supervisory control as in BOPTEST. As we follow the BOPTEST concept for supervisory control, *BESMod* could be used as a BOPTEST test case in future versions (Blum et al. 2021).

For example, surplus PV electricity can be used for load to overcharge the thermal energy storage. While the basic control of the hydraulic system is modelled in the hydraulic subsystem's local control package, storage temperature set points are given by the supervisory control model on aggregated system level. Thus, is enabled and the coupling of domains is also realized from a control point of view.

3.10 System Aggregation

Figure 3 depicts the resulting system aggregation. To compose a custom BES, it is sufficient to extend the `PartialBuildingEnergySystem`, replace all gray subsystem models, and adjust the parameterization of the subsystems. The subsystem parameterization encompasses component choices and records. Fine-tuning is possible though adjustment of non-final bottom-up parameters.

Besides the replaceable subsystems, specification of weather boundary conditions and system parameterization is required.

For weather boundary conditions, we use the `ReaderTMY3`, which is compatible with building models based on the *IBPSA*. In the context of this study, no use case for replaceable weather models arose. However, future versions may include this, for instance, if weather data is not convertable to the TMY3 format.

The replaceable record `systemParameters` aggregates all top-down parameters, such as nominal outdoor air temperature or nominal room set temperature. The replaceable record `parameterStudy` collects parameters users may want to perturb in a simulation study.

If the resulting system aggregation raises errors, we provide `Booleans` to disable the coupling of single subsystems. Besides debugging, these parameters may be used to discard single domains in an analysis. The only non-optional system is the building itself. Further, all subsystem packages contain dedicated testing models. With these tests, users can directly debug if an error is caused by the aggregation or a subsystem.

3.11 Continuous Integration

BESMod uses Continuous Integration (CI) to ensure the functionality and quality of the implemented models. The library is primarily hosted on GitHub and mirrored in the internal GitLab of RWTH Aachen University. Using the GitLab CI, a pipeline is triggered for each commit to the repository. The CI performs the following stages *Check* and *Simulate* using a Dymola Docker container.

The *Check* stage uses the built-in Dymola function to verify syntactic integrity and equal number of equations and variables. The *Simulate* stage searches for models in the library or the given Modelica package that extend `Modelica.Icons.Example` and simulates these models. Both stages are executed separately for the respective packages of the library. This procedure makes it easier to identify faulty models.

In the future, it is planned to include regression tests based on *BuildingyPy*[3] for individual integration tests of the overall systems.

4 Exemplary Use Case

As stated in section 1, user-friendly coupled BES simulations with integration of hydraulic, ventilation and electrical components are an open research gap. To illustrate the usability of *BESMod*, we aggregate an all-electric energy system for retrofit buildings.

4.1 System Layout

On the demand side, we demonstrate the flexibility and library independence of *BESMod* by using two different building model approaches. The first approach is the reduced order approach from the *AixLib* (Müller et al. n.d.). Second, the *Buildings* library `BESTEST` validation model `Case600FF` is used (Wetter, Zuo, et al. 2014). For DHW, we consider the tapping profile M according to *EN 16147* (2017). The applied weather conditions are taken from a test reference year of Aachen.

In the hydraulic subsystem, a bivalent heat pump system consisting of a pump, a heating rod, and a heat pump provides heat. In the distribution system, DHW is prioritized over space heating. To account for DHW loading phases, we consider a DHW and a space heating storage. For space heating, radiators transfer the heat to the thermal zones. Corresponding thermostats are controlled via PI controllers.

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

In the ventilation system, a heat exchanger recovers heat in the generation subsystem. This heat is distributed equally to all thermal zones.

In the electrical system, a PV system generates electricity. The PV area is designed based on the south facing roof area of the building and a design factor to specify the portion of the roof area to use for PV. Further, a lithium-ion battery based on manufacturer specifications maximizes self-consumption using the internal control logic presented by Nytsch-Geusen et al. (2013).

In the supervisory control system, a heuristic to overheat the DHW storage each day at 12 am for one hour is implemented. This control could increase the probability of an efficient heat pump operation using PV and higher ambient temperatures.

The resulting use case model is available in the librarie's Examples package as `UseCaseModelicaConferencePaper`. For specified parameters, records and structural parameter options, we refer to the source code. We highlight that further examples for different use cases based on previous work exist in the library as well (Vering et al. 2021; Wüllhorst et al. 2022).

4.2 Simulation Results

Performing an annual simulation[4] of the presented use case, we analyze the results for the two building models.

The focus of the analysis is not on a quantitative comparison of the two simulation models. Instead, we want to prove that models from different libraries can be coupled with little modeling effort and that the results can be analyzed quickly and easily on the basis of the integrated KPIs. For this purpose, excerpts of the KPIs calculated by *BESMod* for both building models (*AixLib* and *Buildings*) are shown in Table 2.

Table 2. Relevant KPIs for both building models.

KPI	*Unit*	*AixLib*	*Buildings*
$W_{\mathrm{el,HP}}$	kWh	8,969	2,221
$W_{\mathrm{el,HR}}$	kWh	36	440
$W_{\mathrm{el,PV}}$	kWh	8,836	3,931
$Q_{\mathrm{th,Tra}}$	kWh	25,456	5,011
$Q_{\mathrm{th,DHW}}$	kWh	2,209	2,222
Discomfort	Kh	0.7	1.7
CPU time	s	326	357

Further, Figure 4 illustrates trajectories for all domains simulated in the use case. As the two buildings have a different geometry, inertia and insulation, the results differ regarding absolute and relative values. For a detailed understanding of the results, we invite the reader to analyze all KPIs, parameters and trajectories in the repository.

At this point, we want to emphasize that these results are calculated without additional adjustments to the mod-

[4]600 s stepsize; Dassl Solver

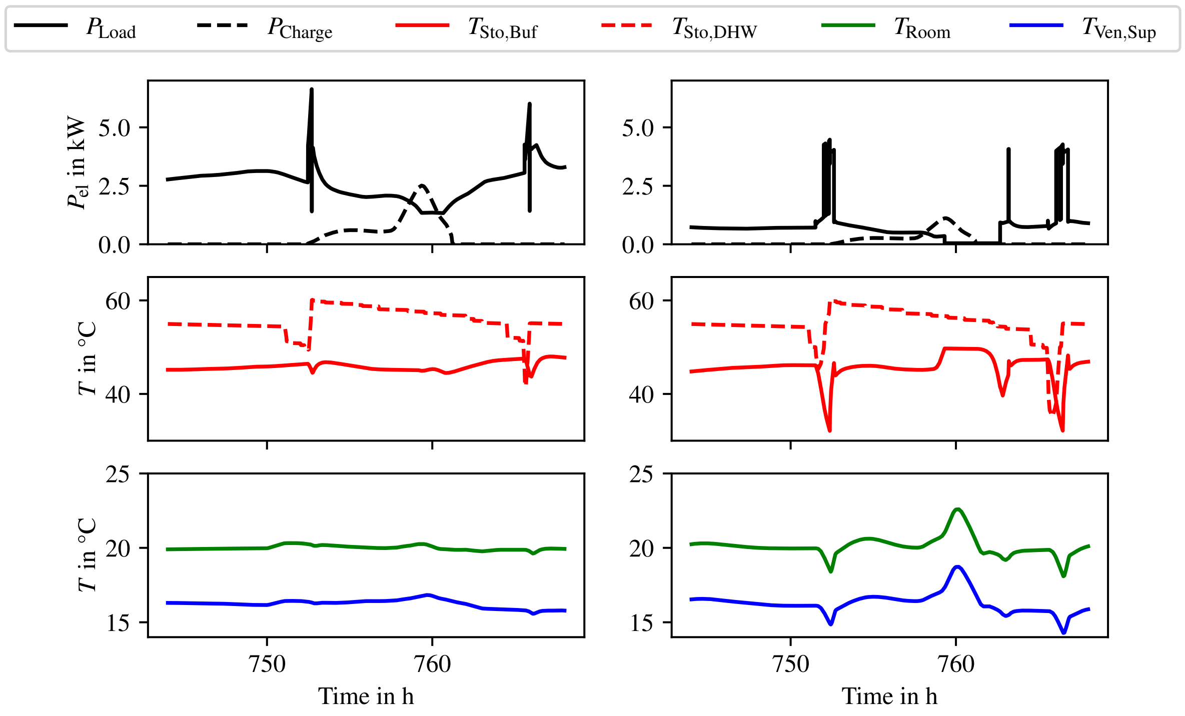


Figure 4. Relevant trajectories for the 1st of Feburary for the cases *AixLib* (left) and *Buildings* (right).

els. Further, only 184 lines of model code are necessary to build both examples using the newly presented *BESMod* library.

5 Conclusion

Closing the gap for a Modelica library which couples the domains hydraulic, ventilation, electrical, control, and building envelope in a modular, user-friendly manner, *BESMod* is presented. The use case demonstrates the ease of application, despite the high complexity of the domain-coupled building energy system simulation. Furthermore, in the face of the system's complexity, annual simulations take less than six minutes.

For the future library development, we hope modeling experts will contribute new modules and join the active developer community. As the current parameterization follows European guidelines, extensions towards international and American guidelines should be considered. Further, model validation is required as a next step. In here, *Continuous Integration* will integrate regression tests to ensure valid models for future developments. A shortcoming regarding modeling accuracy and the representation of real-world applications is that the electrical connectors are currently purely power-based. Future versions should add extensions towards current, voltage, and grid interactions. However, such development must be aligned with the underlying component libraries.

Regarding future use cases, the modular library approach lifts synergies for design and control domains.

For control, *BESMod* enables an efficient and realistic evaluation. Inhere, development of a test case for BOPTEST should be considered (Blum et al. 2021). Additionally, the modular system structure enables the efficient usage of ontologies such as Brick (Balaji et al. 2016). The modularity coupled to ontologies could enable the plug-and-play development of advanced control strategies.

For design, all domain-dependencies of renewable building energy systems are regarded in *BESMod*. Thus, simulation based design optimization methods as in (Vering et al. 2021) should be extended towards sustainable design strategies for building energy systems.

Acknowledgements

We gratefully acknowledge the financial support by the Federal Ministry for Economic Affairs and Climate Action (BMWK), promotional reference 03ET1495A.

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

Andresen, Lisa et al. (2015). "Status of the TransiEnt Library: Transient Simulation of Coupled Energy Networks with High Share of Renewable Energy". In: *Proceedings of the 11th International Modelica Conference, Versailles,* *France, September 21-23, 2015*. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 695–705. DOI: 10.3384/ecp15118695.

Bachmann, Max et al. (2021). "dhcSim — A Modelica library for simple modeling of complex DHC systems". In: *Energy Reports* 7, pp. 294–303. ISSN: 23524847. DOI: 10.1016/j.egyr.2021.08.143.

Balaji, Bharathan et al. (2016). "Brick: Towards a unified metadata schema for buildings". In: *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, pp. 41–50.

Baldwin, Carliss Young and Kim B Clark (2000). *Design rules: The power of modularity*. Vol. 1. MIT press. ISBN: 9780262267649.

Beutlich, Thomas and Dietmar Winkler (2021). "Efficient Parameterization of Modelica Models". In: *Modelica Conferences*, pp. 141–146. DOI: https://doi.org/10.3384/ecp21181141.

Blum, David et al. (2021). "Building optimization testing framework (BOPTEST) for simulation-based benchmarking of control strategies in buildings". In: *Journal of Building Performance Simulation* 14.5, pp. 586–610.

Coninck, Roel de et al. (2014). "Grey-box Building Models for Model Order Reduction and Control". In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 657–666. DOI: 10.3384/ECP14096657.

DIN 1946-6 (2019-12-01). *Ventilation and air conditioning - Part 6: Ventilation for residential buildings - General requirements, requirements for design, construction, commissioning and handover as well as maintenance*. Tech. rep. Bruxelles, Belgium: CEN/TC 113.

EN 15316-1 (2017-09-01). *Energy performance of buildings - Method for calculation of system energy requirements and system efficiencies - Part 1: General and energy performance expression, Module M3-1, M3-4, M3-9, M8-1, M8-4; German version EN 15316-1:2017*. Tech. rep. Bruxelles, Belgium: CEN/TC 113.

EN 16147 (2017-01). *EN 16147:2017, Heat pumps with electrically driven compressors - Testing, performance rating and requirements for marking of domestic hot water units*. Beuth Verlag GmbH.

Jordan, Ulrike and Klaus Vajen (2005). "DHWcalc: Program to generate domestic hot water profiles with statistical means for user defined conditions". In: *Proceedings of the ISES Solar World Congress, Orlando, FL, USA*, pp. 8–12.

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.

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: 0360-5442. DOI: 10.1016/j.energy.2019.06.016.

Modelica Association (2022). *Modelica Libraries*. Last accessed: 25.04.2022. URL: https://modelica.org/libraries.html.

Müller, Dirk et al. (n.d.). "AixLib - An Open-Source Modelica Library within the IEA-EBC Annex 60 Framework". In: *Proceedings of the BauSIM 2016*, pp. 3–9. URL: http://www.iea-annex60.org/downloads/2016-bausim-aixlib.pdf.

Nytsch-Geusen, Christoph 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.

Plessis, Gilles, Aurelie Kaemmerlen, and Amy Lindsay (2014). "BuildSysPro: a Modelica library for modelling buildings and energy systems". In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 1161–1169. DOI: 10.3384/ECP140961161.

Ramsebner, Jasmine et al. (2021). "The sector coupling concept: A critical review". In: *Wiley Interdisciplinary Reviews: Energy and Environment* 10.4, e396.

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.

Schneider, Georg Ferdinand, Georg Ambrosius Pessler, and Simone Steiger (2017). "Modelling and Simulation of Standardised Control Functions from Building Automation". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 209–218. DOI: 10.3384/ecp17132209.

Vering, Christian et al. (2021). "Towards an integrated design of heat pump systems: Application of process intensification using two-stage optimization". In: *Energy Conversion and Management* 250, p. 114888.

Wetter, Michael (2022). *IBPSA Project 1: BIM/GIS and Modelica Framework for building and community energy system design and operation*. Ed. by IBPSA. URL: https://ibpsa.github.io/project1/ (visited on 2022-04-26).

Wetter, Michael, David Blum, et al. (2019-05). *Modelica IBPSA Library v1*. DOI: 10.11578/dc.20190520.1. URL: https://www.osti.gov/biblio/1529269.

Wetter, Michael and Christoph van Treeck (2017-09). *IEA EBC Annex 60: New Generation Computing Tools for Building and Community Energy Systems*. ISBN: 978-0-692-89748-5. URL: https://www.iea-annex60.org/pubs.html.

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.

Wüllhorst, Fabian et al. (2022). "BEStPar: Towards Minimal Effort in Building Energy System Simulation Parameterization". In: *9th Conference of IBPSA Germany and Austria*.

Xanthopoulou, Konstantina et al. (2021). "Validation of a building model as part of the AixLib Modelica library for dynamic plant and building performance simulations". In: *Energy and Buildings* 250, p. 111248.

Zimmer, Dirk (2020). "Robust object-oriented formulation of directed thermofluid stream networks". In: *Mathematical and Computer Modelling of Dynamical Systems* 26.3, pp. 204–233. DOI: 10.1080/13873954.2020.1757726. eprint: https://doi.org/10.1080/13873954.2020.1757726.

Fan and Pump Efficiency in Modelica based on the Euler Number

Hongxiang Fu[1] David Blum[1] Michael Wetter[1]

[1]Building Technology and Urban Systems Division, Lawrence Berkeley National Laboratory, USA,
{hcasperfu,dhblum,mwetter}@lbl.gov

Abstract

Simulation programs often assume constant hydraulic efficiency for fan or pump models when performance curves are unavailable. This is inaccurate because the hydraulic efficiency varies with the operation condition. It therefore consistently underestimates the power draw at off-design conditions at which the hydraulic efficiency drops. Use of a modified Euler number allows computing the hydraulic efficiency and shaft power with limited data. Others showed the validity of the modified Euler number for fan efficiency calculations. We show that it is also applicable for pumps, and present its implementation in Modelica for a fan or pump model. The only input required from the user is one data point at which the hydraulic efficiency is at its maximum. The reported method is applicable regardless of the type, size, or operational region of the fan or pump. Across a sample of eighteen sets of pump data and seven sets of fan data, the errors of the computed power from interpolated data were within 15% for the range of 20% - 70% of maximum flow rate and 40% - 90% of maximum pressure rise, excluding outliers.

Keywords: fan efficiency, pump efficiency, component model

1 Introduction

Movers (fans and pumps) are important components in building energy systems and the accuracy of their component models are pertinent to the accuracy of the energy model as a whole. Often of particular interest is the calculation of power consumption, which in reality is a function of mover fluid volume flow rate $\dot{V}$, pressure rise Δp, hydraulic efficiency η_{hyd}, and motor efficiency η_{mot}, shown by

$$P = \frac{\dot{V}\,\Delta p}{\eta_{hyd}\,\eta_{mot}}. \tag{1}$$

Note that the flow work is simply

$$\dot{W}_{flo} = \dot{V}\,\Delta p \tag{2}$$

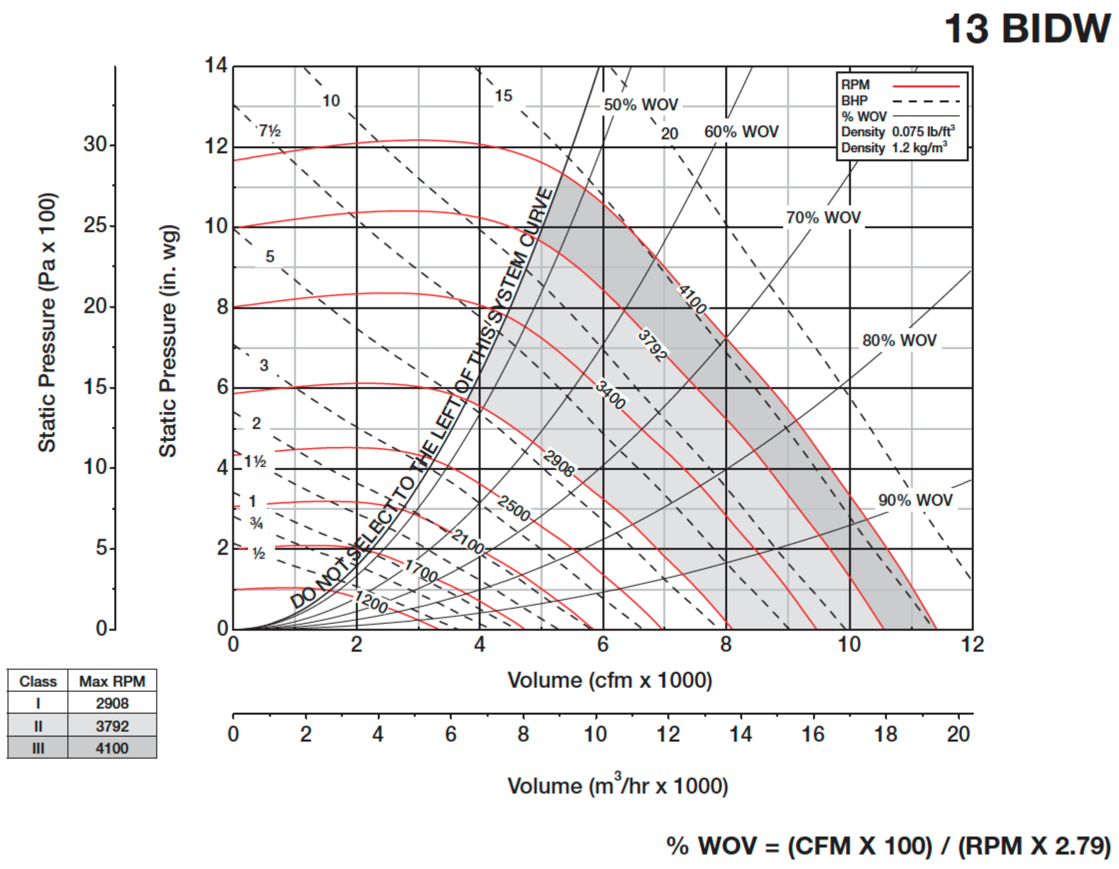
and the hydraulic work, $\dot{W}_{hyd}$, is the mechanical work transmitted to the shaft of the mover to provide such flow work with a hydraulic efficiency as

$$\dot{W}_{hyd} = \frac{\dot{W}_{flo}}{\eta_{hyd}}. \tag{3}$$

The hydraulic efficiency is itself a function of volume flow rate and pressure rise. Calculating power accurately for a broad range of operating conditions and system configurations, therefore, requires both explicit calculation of volume flow rate and pressure rise as well as correct characterisation of efficiency as their function.

Conventional building energy modeling programs do not explicitly calculate both volume flow rate and pressure rise. Therefore, they rely on similarity laws or polynomial regressions that calculate power as a function of flow rate only, implicitly making assumptions about system pressure characteristics and mover efficiency. One particular danger in this approach was pointed out by Englander and Norford (1992a) in the case of static pressure-controlled fans, a typical application for variable air volume (VAV) supply fans common in U.S. commercial HVAC systems. There, Englander and Norford (1992a) showed that, because a static pressure-controlled fan will maintain a pressure rise even at very low flow, power consumption does not trend to zero as the flow rate trends zero. The similarity laws and flow rate regression polynomials with this assumption therefore do not hold. Their data showed this could lead to underestimation of fan power consumption in cases without static pressure reset strategies. Consider also that in a typical fan performance map, shown in Figure 1, the power consumption (indicated by BHP) is non-zero at zero flow or non-zero pressure rise. Therefore, these authors proposed a revised polynomial formulation of power as a function of flow and static pressure set point in a separate paper (Englander and Norford 1992b). Such a correlation utilized an offset term determined by static pressure set point for when flow approaches zero. Similar correlations were suggested by Hydeman et al. (2003).

Such curves can be implemented by the user in simulation software such as EnergyPlus (U.S. DOE 2021), Trace (The Trane Company 2019), and IDA-ICE (EQUA Simulation AB 2013). However, because this still requires the user to have some information on the mover and the system, users often use the default curve provided by the program. One commonly used default curve comes from ASHRAE Standard 90.1 Table G3.1.3.15 (ASHRAE



Class	Max RPM
I	2908
II	3792
III	4100

Figure 1. An example of fan curves from the manufacturer. Greenheck Fan Corporation (2012), reproduced as is.

2020) which takes the form

$$P/P_d = 0.0013 + 0.1470\,PLR + 0.9506\,PLR^2 - 0.0998\,PLR^3, \quad (4)$$

where P_d is the fan power at design condition and PLR is the flow part load ratio

$$PLR = \dot{V}/\dot{V}_d, \quad (5)$$

where $\dot{V}_d$ is the design flow rate. With this correlation, the computed power still effectively trends to zero when the flow rate goes to zero.

While the polynomial regressions represent improvements over the basic similarity laws, they do not generalize to any mover or system and control configuration where more detailed performance and operating state may be known through specific mover performance maps and explicit simulation of the pressure-flow network and controls. Modelica-based modeling has enabled such explicit network and controls simulation. However, the ability to represent the whole mover performance map fully and conveniently is lacking. This includes accounting for the complex variations in efficiency, especially with low flows and significant pressure rise, as would be the operating region of movers operating to maintain a pressure set point with little or no reset based on load. Here, low efficiency likely accounts for significant non-zero power consumption as $\dot{V}$ reduces to zero in (1).

Therefore, in this paper, we report a Modelica implementation of a convenient method to represent mover performance with such accuracy. It only requires the user to provide one data point of η, $\dot{V}$, and Δp where the efficiency is at its maximum. The method then uses a dimensionless modified Euler number and a correlation to estimate the efficiency and power at any operation point. This method is applicable regardless of the type, size, or operational region (stall or non-stall) of the mover. This method is valuable because it provides the analyst with more accurate estimation of mover shaft power while requiring only limited information. It thus is applicable at early stages of design or post-retrofit assessment during which detailed mover performance data are generally not available.

2 Methodology

2.1 Efficiencies

Manufacturers often provide fan or pump performance curves describing how its hydraulic power $\dot{W}_{hyd}$ (often denoted as "shaft power" or "brake horsepower") depends on $\dot{V}$, Δp, and mover speed N. $\dot{W}_{hyd}$ differs from $\dot{W}_{flo}$ by a factor of hydraulic efficiency as shown in (3) and from the total electric power drawn by the mover P by motor efficiency

$$\eta_{mot} = \dot{W}_{hyd}/P. \tag{6}$$

The total efficiency η can then be expressed as the product of the two

$$\eta = \eta_{hyd}\,\eta_{mot}. \tag{7}$$

The implemented method is based on $\dot{W}_{hyd}$ and η_{hyd}. However, in this paper, we also applied this method for P and η as this was the type of pump data available to us. For simplicity, the remainder of this work will use η^* to denote either η or η_{hyd} and P^* to denote either P or $\dot{W}_{hyd}$. Through Figure 2 we will show that this assumption has worked well with our data. We note that U.S. DOE (2014) shows that η_{mot} is mostly constant for motors larger than about 3.5 kW (around 5 HP) except when the motor part load drops below around 40%. To accommodate applications with small motors or large operating regions, our implementation of the model allows the user to specify a separate function for the motor efficiency.

2.2 Modified Euler Number

The Euler number is defined for any medium as

$$Eu = \frac{\text{pressure forces}}{\text{inertial forces}}. \tag{8}$$

This can be written as

$$Eu = \frac{\Delta p A}{p_d A} = \frac{\Delta p}{p_d}, \tag{9}$$

where p_d is the dynamic pressure and A is a characteristic area. The dynamic pressure is

$$p_d = \frac{v^2 \rho}{2} = \frac{\dot{V}^2 \rho}{2A^2}, \tag{10}$$

where v is the velocity, which is proportional to the volumetric flow rate. Substituting (10) into (9) yields

$$Eu = \frac{2\Delta p A^2}{\rho \dot{V}^2}. \tag{11}$$

U.S. DOE (2021) and Haves et al. (2014) reported a model in EnergyPlus that describes these multidimensional relationships of fans. The model is based on two steps: First, it expresses the fan performance using a non-dimensional equation that is derived from the Euler number as

$$Eu^* = \frac{\Delta p D^4}{\rho \dot{V}^2}, \tag{12}$$

where Eu^* is the modified Euler number, D is the fan wheel outer diameter, and ρ is the medium density at the mover inlet. Because D is constant for the same fan and ρ is approximately constant across the operating region in HVAC applications, the ratio of the modified Euler number can be expressed as

$$\frac{Eu^*}{Eu^*_p} = \frac{\Delta p}{\dot{V}^2}\frac{\dot{V}_p^2}{\Delta p_p}, \tag{13}$$

where the subscript p denotes the peak operation point at which η_{hyd} attains its maximum, and the quantities without subscript are any operating point.

Second, it expresses the ratio of hydraulic efficiency $\eta_{hyd}/\eta_{hyd,p}$ using an exponential-conditioned skew-normal function that takes the ratio of the modified Euler number Eu^*/Eu^*_p as an argument. U.S. DOE (2021) shows that this relationship is remarkably similar across different fan sizes and types, both within the stall and non-stall regions. The normalized exponential-conditioned skew-normal function is

$$\frac{\eta_{hyd}}{\eta_{hyd,p}} = \frac{\exp(-0.5 Z_1^2)\left(1 + \frac{Z_2}{|Z_2|}\operatorname{erf}\left(\frac{|Z_2|}{\sqrt{2}}\right)\right)}{\exp(-0.5 Z_3^2)\left(1 + \frac{Z_3}{|Z_3|}\operatorname{erf}\left(\frac{Z_3}{\sqrt{2}}\right)\right)}, \tag{14}$$

where

$$Z_1 = (x-a)/b, \tag{15}$$
$$Z_2 = (\exp(cx)\,d\,x - a)/b, \tag{16}$$
$$Z_3 = -a/b, \tag{17}$$
$$x = \log_{10}(Eu^*/Eu^*_p), \tag{18}$$

and

$$a = -2.732094, \tag{19}$$
$$b = 2.273014, \tag{20}$$
$$c = 0.196344, \tag{21}$$
$$d = 5.267518. \tag{22}$$

Note that with this formulation, the user merely needs to provide values for $\dot{V}_p$, Δp_p and $\eta_{hyd,p}$, from which η_{hyd} can be solved for any operating condition.

U.S. DOE (2021) and Haves et al. (2014) discussed this similarity in the context of fan but not pumps. But from (11) and (12) it follows that

$$\frac{Eu}{Eu_p} = \frac{Eu^*}{Eu^*_p}. \tag{23}$$

From (23), we conclude that (13) is applicable regardless of medium, thus it is also applicable for pumps. Next, it remains to be shown that the empirical relation (14) is also applicable for pumps. In Figure 2 we overlaid operating points of fans and pumps to the empirical relation (14). As can be seen from the figure, pump operation points also match the empirical relation (14). To the best of our knowledge, this is the first report that shows the validity of (14) for pumps. Further validation based on the model that is described in Section 3 is given in Section 4.

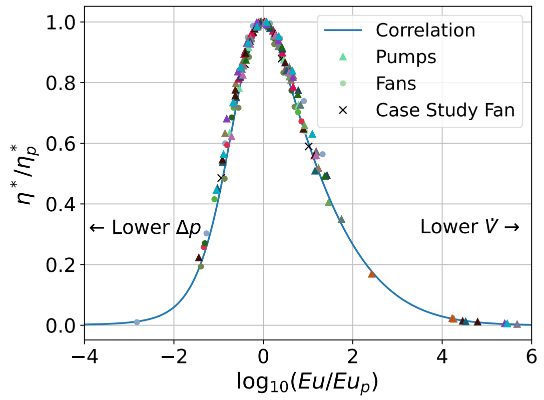


Figure 2. Normalized efficiency curves in dimensionless space and mover performance data. Each colour represents one dataset of seven fan models and eighteen pump models.

3 Modelica Implementation

The `Buildings.Fluid.Movers` package (Wetter 2013) of the Modelica Buildings Library (Wetter et al. 2014) was revised to implement the reported method. The revised model is available through commit 346f5a0 and will be released in future versions of the Modelica Buildings Library. In the previous model, the user could provide either data for $P = f(\dot{V})$, or data for $\eta_{hyd} = f(\dot{V})$ and $\eta_{mot} = f(\dot{V})$. The new implementation separated the computation of the three efficiency terms, η, η_{hyd}, and η_{mot}, allowing a user to specify two such that the third is computed by Equation 7. The efficiency η^* and the power P^* are pre-computed using the Euler number to construct two 2D look-up tables that are implemented using `Modelica.Blocks.Tables.CombiTable2Ds`. These two variables are then found through two-dimensional interpolation during the simulation. There are a number of rationales for this approach:

1. Storing pre-computed values avoids having to evaluate (14) at each time step. This is preferable especially because (14) is computationally expensive and not globally differentiable.

2. As can be seen from (18), both Δp and $\dot{V}$ must be bounded away from zero to avoid the logarithm at zero and the division by zero. This is more easily managed when the power and efficiency are pre-computed.

3. As can be seen in Figure 2, in (18) and in (13), $\eta_{hyd} \to 0$ when either $\Delta p \to 0$ or $\dot{V} \to 0$. This would cause the computed power P to approach infinity. Again, this is more easily avoided by using pre-computed tabulated values.

To construct the tables, the support points are computed from (14) in 10% equidistant increments as

$$\{\Delta p_i\}_{i=0}^{10} = \{\Delta p_{max}\, i/10\}_{i=0}^{10}, \tag{24}$$

where $\Delta p_{max} = \Delta p(\dot{V} = 0, n = 1)$ and n is the normalized speed, and similarly

$$\{\dot{V}_i\}_{i=0}^{10} = \{\dot{V}_{max}\, i/10\}_{i=0}^{10}, \tag{25}$$

where $\dot{V}_{max} = \dot{V}(\Delta p = 0, n = 1)$. The efficiency η^* at boundary points ($\Delta p = 0$ or $\dot{V} = 0$) is set to zero. The power P^* at these boundary points is extrapolated except at $\Delta p = 0$ and $\dot{V} = 0$ where it is set to zero.

4 Validation

We validated the implemented method by comparing the model output of η^* and P^* against values interpolated from performance maps. The peak performance data Δp_p, $\dot{V}_p$, and η_p were obtained from the mover curve at maximum speed. They are then used by the model to compute the look-up tables. The efficiency and power as computed by the model were then compared to the original performance curves. It is important to note that although full performance curves were used here to find the peak point for the purpose of this validation, the user only needs to provide the peak point to use the model.

4.1 Nominal Speed

Figure 3 shows the validation results using the performance curve at nominal speed $N = 4100$ rpm in Figure 1. The computed efficiency and power are compared against values interpolated from the performance map. Figure 3(e) shows that the fan reproduces the pressure curve at a constant speed. As can be seen from Figure 3(a) and Figure 3(b), the computed efficiency closely follows the values from the performance map for the full range of $\dot{V}$ and Δp. The computed power is also accurate for most of the range but the two curves diverged slightly at high $\dot{V}$ and more so at low Δp. Power values computed from constant efficiencies are plotted through the grey shaded region. When the efficiency was assumed constant at its peak value of $\eta_{hyd} = 0.68$, the computed power was always underestimated. As expected, when a lower value of $\eta_{hyd} = 0.46$ is used for the constant efficiency, the computed power was overestimated in some region and underestimated in some other, as reported by Englander and Norford (1992a). Either way, if a constant efficiency is used, the power goes to zero as the flow approaches zero, which is physically incorrect if the mover continues to create pressure rise.

Figure 4 shows the errors from a sample of eighteen sets of pump data and seven sets of fan data. First, the distributions of the errors are almost all skewed to the same direction in each subplot. The errors of η^* have tails on the negative side at high $\dot{V}$ and low Δp and the errors of P^* are skewed to the opposite direction. Second, with the outliers and boxes at extreme $\dot{V}$ or Δp values put aside, the errors are all roughly within 20%. If the threshold is tightened to 15%, this method provided satisfactory results for η in the range of 20% - 80% of $\dot{V}_{max}$ and 30% - 90% of Δp_{max}, and for P in the range of 20% - 70% of $\dot{V}_{max}$ and 40% - 90% of Δp_{max}, excluding outliers.

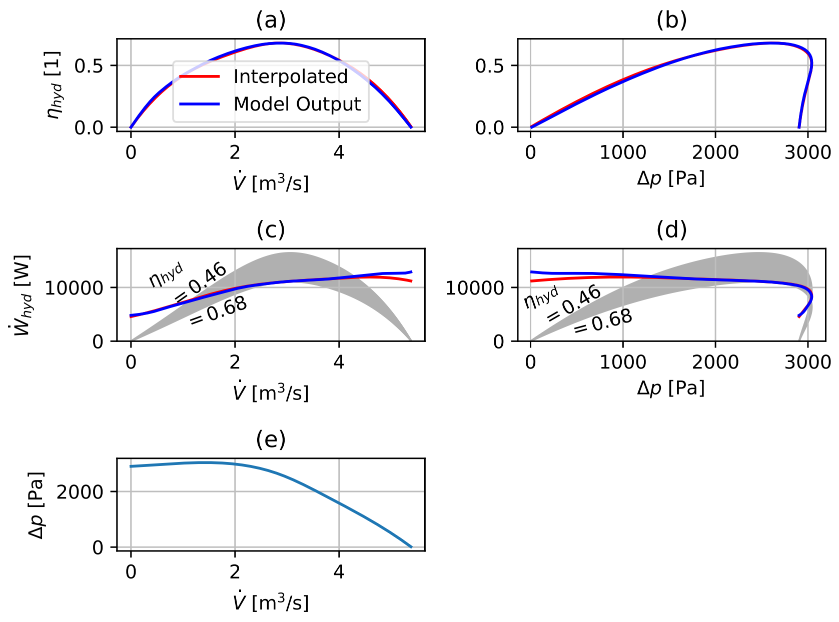


Figure 3. Hydraulic efficiency and power values that were interpolated from the performance map (red lines) and computed by the model (blue lines). The grey shaded area is bounded by the simplified assumption of a constant hydraulic efficiency between $\eta_{hyd} = 0.68$ (peak of this fan) and $\eta_{hyd} = 0.46$ and shown here to visualize the wrong results of this oversimplification.

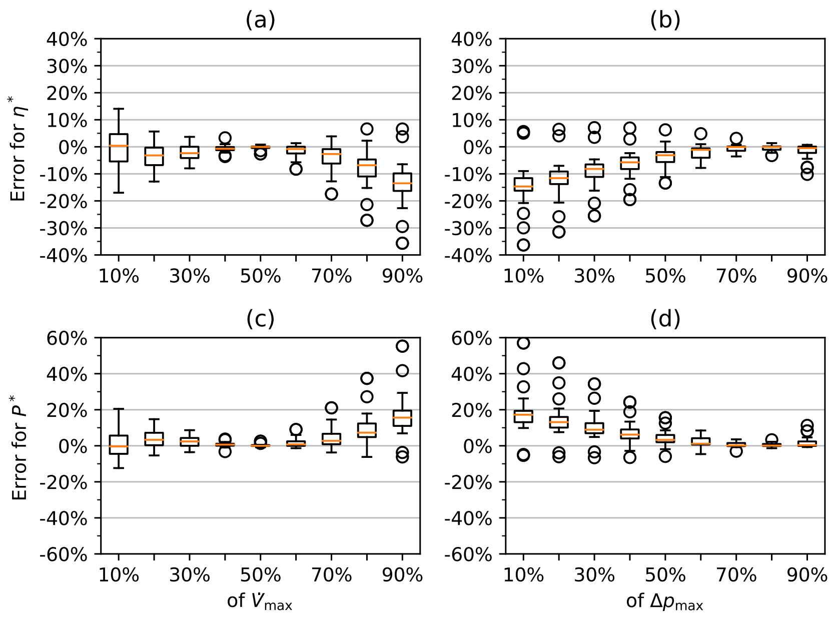


Figure 4. Box-whisker plots for errors of computed efficiency and power against values interpolated from performance maps. The sample consists of eighteen sets of pump data and seven sets of fan data. The middle lines in the boxes correspond to the medians. The outliers are defined as points more than 1.5 interquartile ranges away out from the maximum or the minimum.

4.2 Reduced Speed

We also validated the method for reduced speed. Figure 5 shows the power computed by the implemented method compared to interpolated values at three different speeds. The interpolation was done from Figure 1 at $N = 4100$ RPM ($n = 1$), 3400 RPM ($n = 0.83$), and 2500 RPM ($n = 0.61$). The figure shows that the error patterns are remarkably similar across the different speeds.

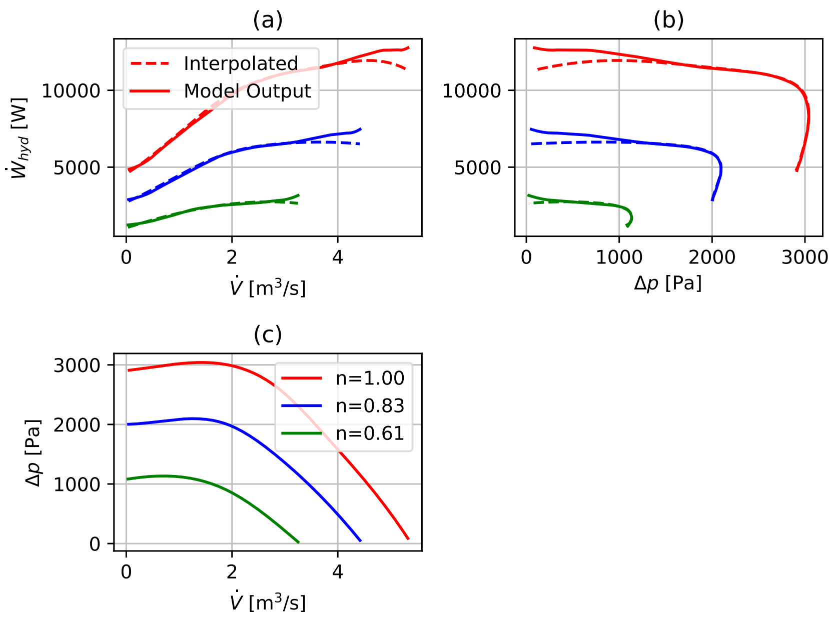


Figure 5. Hydraulic power values that were interpolated and that were computed by the model at nominal ($n = 1$) and reduced ($n < 1$) speeds.

5 Discussion

Figure 4 shows that the error distributions within each subplot are skewed to the same direction. The errors of η^* are consistently negative. This is likely caused by the fact that the global maximum of efficiency is above the curve of maximum speed, which leads to η_p^* being underestimated. Because η^* is computed based on η_p^*, η^* will also be consistently underestimated. Consequently, the errors of P^* are skewed to the positive side as $P^* \propto 1/\eta^*$.

Figure 5 suggests that, when compared to the manufacturer data, the increased discrepancies of $\dot{W}_{hyd}$ mostly occur at the lower right region of the fan performance map. A properly-sized fan or pump should operate somewhere near the midpoint on its nominal speed curve at full load. At reduced load, its operating point moves to the left because of reduced flow. A mover is therefore unlikely to operate in this region in which the discrepancy is the largest.

Besides η_{hyd}, the implementation allows the user to specify η_{mot} as well, although the part-load behaviour of η_{mot} is beyond the scope of this work.

We note that the Euler method does not reproduce efficiency degradation along constant system curves, e.g., along the curves $\Delta p = k\dot{V}^2$, for any constant $k \geq 0$. This limitation follows from (14), which has the functional form

$$\frac{\eta_{hyd}}{\eta_{hyd,p}} = f(x) = f\left(log_{10}(Eu^*/Eu_p^*)\right)$$
$$= f\left(\log\left(\frac{\Delta p}{\dot{V}^2}\frac{\dot{V}_p^2}{\Delta p_p}\right)\right). \qquad (26)$$

As $\dot{V}_p^2$ and Δp_p are constants, the functional dependency (26) can be further reduced to

$$\frac{\eta_{hyd}}{\eta_{hyd,p}} = g\left(\frac{\Delta p}{\dot{V}^2}\right). \qquad (27)$$

Therefore, the efficiency η_{hyd} is constant along any curve $\Delta p = k\dot{V}^2$, and it remains at its peak along the curve for which $k = \Delta p_p/\dot{V}_p^2$. This is in line with the simplification often used by fan manufacturers to generate fan curves at different speeds (Stein and Hydeman 2004) and this simplification is explicitly permitted by ASHRAE Standard 51-16 (ANSI/AMCA Standard 210-16) (ASHRAE 2016). For this reason, fan or pump performance maps whose reduced-speed performance is measured are difficult to find. Turbines share similar fluid-flow principals to fans and pumps and their performance maps often display contours of constant efficiency that suggest that there is no constant efficiency line along such a quadratic curve (e.g. Leylek (2012) and Garrett Motion (2019)).

This further indicates that the Euler number method is less accurate than using the measured data at reduced speed. However, such measured data are rare for fans or pumps. In this case the reported method has the same accuracy in theory and it would depend on whether the user finds it more convenient to only use one data point of peak operation instead of entering a whole curve into the model. Still, it is important to emphasize that the Euler number method is implemented to help with the situation where the mover data are unavailable.

6 Conclusion

This work describes the Modelica implementation of a method that computes fan or pump efficiency and power using the dimensionless Euler number. With this method, the only input required from the user is the flow rate, pressure rise, and hydraulic efficiency at the peak operation point where the hydraulic efficiency is at its maximum. This practice is valuable because it provides the user with an accurate computation that requires only limited data, and it does not suffer from the errors that occur if a constant efficiency is used.

The implemented method was validated with seven sets of fan and eighteen sets of pump data obtained from manufacturers. Across the sample and between the values computed from the reported method and from interpolation of manufacturer data, the errors of efficiency were within 15% in the range of 20% - 80% of $\dot{V}_{max}$ and 30% - 90% of Δp_{max}, excluding outliers. The errors in power were within 15% in the range of 20% - 70% of $\dot{V}_{max}$ and 40% - 90% of Δp_{max}, excluding outliers. The errors were larger when $\dot{V}$ was high or when Δp was low. These discrepancy patterns remained largely the same at reduced speeds. Because the increased discrepancies mostly occurred in a region in which a properly-sized mover is unlikely to operate, they have little effect on the accuracy of this method compared to using manufacturer data in common HVAC applications.

The paper shows how an underestimated peak efficiency introduces consistent and systematic errors. Improving the methodology for finding the peak efficiency on the manufacturer-provided power map would increase the accuracy of the model. Furthermore, more uncertainty analysis is needed to understand how the errors of the estimation of the peak point influence the errors of the computed efficiency and power. Understanding the uncertainty is important for this application because it is intended to be used when the user has limited information and must make reasoned estimations.

Acknowledgements

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

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

ASHRAE (2016). *ASHRAE Standard 51-16 (ANSI/AMCA Standard 210-16), Laboratory Methods Of Testing Fans For Certified Aerodynamic Performance Rating*. ASHRAE.

ASHRAE (2020). *ANSI/ASHRAE/IES Standard 90.1-2019: Energy Standard for Buildings Except Low-Rise Residential Buildings*. ASHRAE.

Englander, SL and LK Norford (1992a). "Saving fan energy in VAV systems- part 1: analysis of a variable-speed-drive retrofit." In: *ASHRAE Winter Meeting, Anaheim, CA, USA, 01/25-29/92*, pp. 3–18.

Englander, SL and LK Norford (1992b). "Variable Speed Drives: Improving Energy Consumption Modeling and Savings Analysis Techniques." In: *Proc. of the ACEEE Summer Study 1992*, pp. 3.61–3.78.

EQUA Simulation AB (2013-02). *User Manual: IDA Indoor Climate and Energy Version 4.5*. Date accessed: 31-Mar-2022. URL: http://www.equaonline.com/iceuser/pdf/ice45eng.pdf.

Garrett Motion (2019). *Turbo Tech 103 | Expert: Compressor Mapping*. Date accessed: 29-Aug-2022. URL: https://www.garrettmotion.com/wp-content/uploads/2019/10/GAM_Turbo-Tech-103_Expert-1.pdf.

Greenheck Fan Corporation (2012). *Double-Width Centrifugal Fan Performance Supplement*. Date accessed: 2-Dec-2021. URL: https://content.greenheck.com/public/DAMProd/Original/10002/CentrifugalDWPerfSuppl_catalog.pdf.

Haves, Philip et al. (2014-09). *Development of Diagnostic and Measurement and Verification Tools for Commercial Buildings*. Tech. rep. CEC-500-2015-001. Lawrence Berkeley National Laboratory.

Hydeman, Mark et al. (2003-10). *Advanced VAV System Design Guide*.

Leylek, Zafer (2012-10). *An Investigation into Performance Modelling of a Small Gas Turbine Engine*. Tech. rep. DSTO-TR-2757. Defence Science and Technology Organisation. URL: https://www.dst.defence.gov.au/sites/default/files/publications/documents/DSTO-TR-2757.pdf.

Stein, Jeff and Mark Hydeman (2004). "Development and Testing of the Characteristic Curve Fan Model." In: *ASHRAE transactions* 110.1.

The Trane Company (2019-02). *TRACE 600 Engineering Manual*. Date accessed: 31-Mar-2022. URL: https://tranecds.custhelp.com/ci/fattach/get/120633/0/filename/TRACE_600_Engineering_Manual%5C%5B1%5C%5D.pdf.

U.S. DOE (2014-04). *Determining Electric Motor Load and Efficiency*. Date accessed: 10-Mar-2022. URL: https://www.energy.gov/sites/prod/files/2014/04/f15/10097517.pdf.

U.S. DOE (2021-09). *EnergyPlus Version 9.6.0 Documentation: Engineering Reference*. Date accessed: 2-Dec-2021. URL: https://energyplus.net/assets/nrel_custom/pdfs/pdfs_v9.6.0/EngineeringReference.pdf.

Wetter, Michael (2013-08). "Fan and Pump Model That Has a Unique Solution for Any Pressure Boundary Condition and Control Signal". In: *Proc. of the 13th Conference of the International Building Performance Simulation Associationn*. Chambéry, France, pp. 3505–3512.

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

Transient Simulation of an Air-source Heat Pump under Cycling of Frosting and Reverse-cycle Defrosting

Jiacheng Ma[1] Donghun Kim[2] James E. Braun[1]

[1]School of Mechanical Engineering, Purdue University, USA, {ma516,jbraun}@purdue.edu
[2]Lawrence Berkeley National Laboratory, USA, donghunkim@lbl.gov

Abstract

Frost accumulation is a common but undesired phenomenon for air-source heat pump (ASHP) systems in winter operations. Reverse-cycle defrosting that applies heat to the outdoor coil by reversing the thermodynamic cycle, is one of the predominant means for periodic removal of the accumulated frost. This paper presents a dynamic modeling framework for ASHPs under cycling of frosting and reverse-cycle defrosting operations. A uniform model structure was applied to frost formation and melting models, which were incorporated into the evaporator model, without a need for reinitializing the system when the operating mode switches between heating and defrosting. The developed model was simulated to predict transients of a residential ASHP under multiple cycles of frosting and defrosting operations, and results yielded good agreement with measurements.

Keywords: dynamic modeling, heat pump, non-uniform frosting, reverse-cycle defrosting

1 Introduction

Frost accumulation on evaporator coil surfaces can significantly degrade the performance of air-source heat pump (ASHP) systems in winter operations. The continued buildup of frost eventually necessitates a defrosting mode to remove the accumulated frost and return the system to its normal operating characteristics. Among various approaches currently employed on ASHP systems, reverse-cycle defrosting (RCD) that applies heat to the outdoor coil by reversing the thermodynamic cycle, is one of the predominant means because of its easy implementation without additional heat sources. The nature of frosting-defrosting cycling operations imposes significant challenges for ASHP control designs, optimization, fault detection and diagnostics (FDD). Therefore, a simulation tool capable of capturing the system dynamics under mode-switching operations is extremely useful in development and evaluation of improved control and FDD algorithms.

Due to the complicated underlying physical behaviors of frosting and defrosting of ASHPs and significant computational complexities, a very limited number of system-level modeling efforts can be found in the open literature. Many studies solely focus on performance of heat exchangers either with frost formation or during RCD, e.g., (Dopazo et al. 2010; Breque and Nemer 2017; Padhmanabhan et al. 2011). Within the context of complete cycle modeling considering the dynamics on both the refrigerant side and air side, Qiao, Aute, and Radermacher (2017) integrated a one-dimensional frost growth model into an evaporator model coupled to other components of a two-stage flash tank vapor injection heat pump system. Transients of the system going through a start-up period, a stable frosting stage, followed by an unstable hunting stage due to the performance degradation by frost accumulation were simulated. Comparisons of the simulation results against experimental data indicate that the developed model can reasonably predict the system responses under frosting conditions. Steiner and Rieberer (2013) simulated the defrosting process of a CO2 heat pump system used for an electric vehicle by integrating a lumped frost melting model into the heat exchanger model. Predictions of the refrigerant dynamics and the compressor power showed reasonable agreements with the measurements. Then the model was used to conduct a parametric analysis on different expansion valve openings during defrosting. It was found that an optimal valve opening existed regarding defrost time and efficiency. However, only the system operation after the refrigerant pressure levels were equalized and the compressor was turned on again in defrost cycle, was simulated omitting the reverse flow transients. Han et al. (2022) carried out a simulation study of a residential heat pump system during defrosting cycles. A multistage frost melting model was coupled to a finite volume heat exchanger model. Results were compared with experimental data and yielded good agreement. Similar to Steiner and Rieberer (2013), both studies concentrated on system dynamics during a short time period of the defrosting cycle, and excluded the refrigerant pressure equalization stage triggering flow reversal. In a subsequent simulation study, Qiao, Aute, and Radermacher (2018) incorporated a multi-stage frost melting model in the evaporator model. With a reversing valve model, the system model was able to capture transients when the refrigerant flow was reversed during initiation and termination of RCD. No experimental validation was provided.

To address the need for a general simulation tool that is directly applicable to control and FDD, this paper presents a dynamic modeling framework of ASHP incorporated with frost formation and melting. The developed cycle model was validated using experimental data collected from a residential ASHP unit operating under multiple cycles of frosting and RCD. Section 2 presents major component models. Implementation of the developed models for simulating the system dynamics under cycling of frosting and defrosting is then described in Section 3. Section 4 reports simulation results and comparisons against the measurements, followed by conclusions of the present work summarized in Section 5.

2 Model development

2.1 Heat exchanger model

Much of the modeling efforts have focused on heat exchanger models since the dominant dynamics of a general vapor compression system reside in two-phase heat exchangers. In the present work, a heat exchanger model is integrated with detailed frost formation and melting models to investigate the overall impact of frost on the system performance. To describe dynamics of the refrigerant in heat exchanges, a number of assumptions are required to simplify the model construction of two-phase flow:

- The refrigerant flow is one-dimensional with uniform fluid properties at cross sections.
- Changes in kinetic energy and potential energy are negligible; viscous dissipation is negligible; axial heat conduction along the refrigerant flow direction is negligible.
- The liquid and vapor of the two-phase region are in thermodynamic equilibrium.
- Body forces are neglected in the momentum balance.

The finite volume approach, that segments a heat exchanger into an arbitrary number of equally sized control volumes, is utilized for discretization. The refrigerant pressure and enthalpy as a pair of independent thermodynamic properties are chosen as state variables to express the discretized governing equations. A staggered grid scheme is adopted to solve the balance equations in those control volumes (CVs) (Laughman et al. 2015). As shown in Figure 1, the mass and energy balances are solved in the upper grid, referred to as volume cells, where thermodynamic properties are determined, while the momentum balances are solved in the lower grid, referred to as flow cells, where dynamics of the mass flow rate are evaluated for neighboring volume cells.

With a discretization of n control volumes for a heat exchanger, $n-1$ momentum balances are formed, where each of the two boundary flow cells has an extended length of half the cell. In this way, the momentum balances solves interface mass flow rates on the $n-1$ inner edges of the volume cells, leaving mass flow rates on the outer edges $(\dot{m}_1, \dot{m}_{n+1})$ as boundary conditions to the grid. Meanwhile, the first and last volume cells expose the

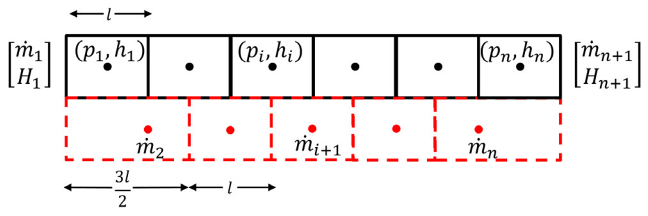


Figure 1. Staggered grid for discretization of balance equations.

thermodynamic states. As a result, connection with other flow devices avoids the solution of large nonlinear systems for algebraic pressures (Franke, Casella, Sielemann, et al. 2009). The discretized governing equations of refrigerant mass, momentum and energy balances are given respectively in (1)-(3),

$$V_i\left[\left(\frac{\partial\rho}{\partial p}\bigg|_h\right)_i\frac{dp_i}{dt}+\left(\frac{\partial\rho}{\partial h}\bigg|_p\right)_i\frac{dh_i}{dt}\right]=\dot{m}_i-\dot{m}_{i+1} \tag{1}$$

$$\begin{aligned}L_i\frac{d\dot{m}_i}{dt}&=\rho_{i-1}\bar{v}_{i-1}^2A_{i-1}-\rho_i\bar{v}_i^2A_i\\&+\frac{A_{i-1}+A_i}{2}(p_{i-1}-p_i)-F_{f,i}\end{aligned} \tag{2}$$

$$\begin{aligned}&V\left[\left(h_i\frac{\partial\rho}{\partial p}\bigg|_h-1\right)_i\frac{dp_i}{dt}+\left(h_i\frac{\partial\rho}{\partial h}\bigg|_p+\rho\right)_i\frac{dh_i}{dt}\right]\\&=H_i-H_{i+1}+\dot{Q}_i\end{aligned} \tag{3}$$

where V_i denotes the volume of each volume cell, L_i denotes the length of a flow cell, $\bar{v}_i$ denotes the average velocity of a volume cell, $\dot{Q}_i$ is the convective heat transfer rate with the metal wall, H_i denotes the enthalpy flow rate computed at the edge of a volume cell. The upwind difference scheme is used to approximate thermodynamic quantities at the volume cell interfaces. Therefore, the enthalpy flow rate under the nominal flow direction can be calculated as

$$\begin{cases}H_i=\dot{m}_ih_{i-1,\text{upstream}}\\H_{i+1}=\dot{m}_{i+1}h_i.\end{cases} \tag{4}$$

In determining quantities transported via convection such as the specific enthalpy, it is important to note that upstream and downstream are concerned with a certain flow direction. Since this work deals with reverse-flow modeling, flow dependent values need to be evaluated as the mass flow rate crossing zero. In Modelica-based modeling, this is handled by using *stream variables* that can describe the bi-directional flow in a numerically reliable way when singularity arises at zero mass flow (Franke, Casella, Otter, et al. 2009). Friction force across a flow cell is evaluated by the equivalent frictional pressure drop $\Delta p_{f,i}$:

$$F_{f,i}=A_{s,i}\Delta p_{f,i}\text{sign}(\dot{m}_i) \tag{5}$$

where $A_{s,i}$ is the surface area of a control volume. Note that a sign function of the mass flow rate is applied since the friction force is always in the direction opposite to the flow.

Assuming a uniform temperature of the tube wall and associated fins, conservation of energy for the coil metal structure can be derived as

$$\left(M_{fin}c_{p,fin}+M_t c_{p,t}\right)_i \frac{dT_{w,i}}{dt} = \alpha_{r,i}A_{s,i}(T_{r,i}-T_{w,i})+\dot{Q}_{f,i} \tag{6}$$

where $\dot{Q}_{f,i}$ denotes heat conduction with the frost layer. The air temperature and humidity profile can be derived from one-dimensional, quasi-steady-state mass and energy balances with a uniform surface temperature and inlet air conditions,

$$T_{ai,out} = T_{as,i} + (T_{a,in}-T_{as,i})e^{-Ntu} \tag{7}$$

$$\omega_{ai,out} = \omega_{a,in} - (1-e^{-\frac{Ntu}{Le^{2/3}}})\max\{0,\omega_{a,in}-\omega_{as,i}\} \tag{8}$$

where Ntu is the number of transfer units for sensible heat transfer,

$$Ntu = \frac{\alpha_{a,i}(A_t+\eta_{fin}A_{fin})_i}{\dot{m}_{a,i}c_{p,a}} \tag{9}$$

$\omega_{as,i}$ is the saturated humidity ratio evaluated at the surface temperature $T_{as,i}$. The heat and mass transfer analogy through the Lewis number $Le^{2/3}=0.9$ is adopted to correlate convection coefficients of heat and mass transfer, which is valid in both cases of condensation and sublimation (Bergman et al. 2011; Leoni et al. 2017). The total heat transfer consisting of sensible and latent parts can be obtained by

$$\dot{Q}_i = \dot{m}_{a,i}c_{p,a}(T_{ai,out}-T_{a,in}) + \dot{m}_{a,i}(\omega_{ai,out}-\omega_{a,in})\Delta h_{lat} \tag{10}$$

where Δh_{lat} represents the latent heat of condensation or sublimation according to the surface temperature. When the coil fan is off, heat transfer between the ambient air and coil through natural convection becomes dominant. The heat transfer rate is calculated assuming a uniform air temperature around the coil and neglecting latent heat transfer,

$$\dot{Q}_i = \alpha_{a,i}(A_t+\eta_{fin}A_{fin})_i(T_{as,i}-T_{a,in}). \tag{11}$$

The value of the Grashof number over the square of the Reynolds number Gr/Re^2 measures effects of buoyancy forces relative to inertial forces, which can be applied to indicate the dominant effect when the coil fan is energized and shut off. In this work, (7) - (10) are used when $Gr/Re^2 \leq 0.1$, otherwise the air-side heat transfer rate is calculated from (11).

The prediction of frost formation on the outdoor coil is essential to characterize the performance of an ASHP system under frosting conditions due to blockage of the air flow passage and additional thermal resistance. To make

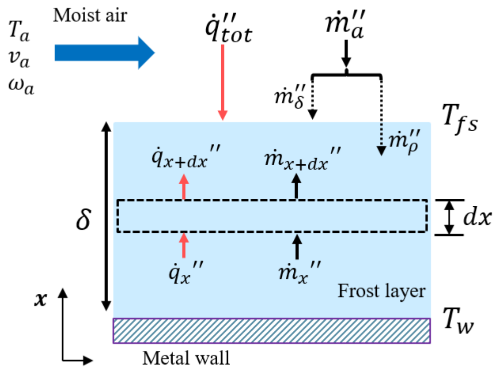


Figure 2. Schematic of one-dimensional frost growth.

a numerical model tractable, it is often assumed that frost growth normal to the cold surface is of primary interest. The frost layer growth and densification rates can be determined by solving the heat and mass diffusion equations. As shown in Figure 2, at the frost-air interface, the total heat flux from the air stream q''_{tot} is composed of sensible and latent heat due to the temperature and humidity differences,

$$q''_{tot} = \alpha_h(T_a-T_{fs}) + \alpha_m(\omega_a-\omega_{fs})\Delta h_{sg} \tag{12}$$

where α_h denotes the heat transfer coefficient, α_m denotes the mass transfer coefficient, T_{fs} denotes the frost surface temperature, ω_{fs} denotes the saturated humidity ratio at the frost surface, Δh_{sg} denotes the water sublimation heat. Note that a heat transfer rate obtained from (12) represents the same heat transfer rate as (10) under frosting conditions. Indeed, the heat transfer rate will be implemented as a connection variable of the air flow model and the frost formation model, such that a equation system can be formed to solve for intermediate variables (e.g., frost surface temperature). It also gives the amount of heat conducted to the metal structure ($\dot{Q}_{f,i}$) in (6), assuming that the frost growth process is quasi-steady-state. When the frost layer is of thickness $\delta_{f,i}$ and lumped density $\rho_{f,i}$, mass conservation of the frost layer can be formed as

$$\frac{d}{dt}(\rho_{f,i}\delta_{f,i}) = \alpha_{m,i}(\omega_{ai,in}-\omega_{fs,i}) = \dot{m}''_{a,i} \tag{13}$$

which can rewritten as

$$\rho_{f,i}\frac{d\delta_{f,i}}{dt} + \delta_{f,i}\frac{d\rho_{f,i}}{dt} = \dot{m}''_{\delta,i} + \dot{m}''_{\rho,i} = \dot{m}''_{a,i}. \tag{14}$$

(14) indicates that the total water mass flux $\dot{m}''_{a,i}$ is divided into two components: $\dot{m}''_\delta$ contributes to an increment of the frost thickness and $\dot{m}''_\rho$, which is diffused into the frost layer, contributes to its densification. Consider a differential control volume of thickness dx within the frost layer, the heat and mass diffusion can be formulated and then solved associated with boundary conditions at

the wall surface and the frost-air interface. For the sake of brevity, the solution procedure is not included in the present paper. Refer to (Qiao, Aute, and Radermacher 2017) for derivations to obtain the mass fluxes in (14).

Since the model is derived following a quasi-steady-state assumption, the frost thickness and density are updated at a fixed time step Δt

$$\rho_{f,i}(t+\Delta t) = \rho_{f,i}(t) + \frac{\dot{m}''_{\rho,i}}{\delta_{f,i}}\Delta t \tag{15}$$

$$\delta_{f,i}(t+\Delta t) = \delta_{f,i}(t) + \frac{\dot{m}''_{\delta,i}}{\rho_{f,i}}\Delta t \tag{16}$$

which enables the overall frost growth process to remain transient over time.

The rather stochastic frost melting process is idealized such that the defrost process progresses through several predictable stages. The present work applies a five-stage defrost model subdividing the overall process into preheating, melting start, melting, vaporizing, and dry heating (Qiao, Aute, and Radermacher 2018). The governing equations of each stage are presented in Appendix. The model is basically established by a lumped-capacitance analysis along with energy and mass conservation at interfaces, and switches between different stages according to the current temperature profile and phase presence. The overall melting process is illustrated in Figure 3.

A switching algorithm based on the Fuzzy logic is implemented (Kim et al. 2021). Since transitions between different stages are triggered by values of the wall temperature $T_{w,i}$, frost and a water film thickness $\delta_{f,i}$ and $\delta_{\text{water},i}$, three linguistic variables $T_{w,i} - T_{\text{melt}}$, $\delta_{\text{water},i}/\delta_{\text{water,max}}$ and $\delta_{f,i}$ are defined. T_{melt} is the melting temperature of water (e.g., 273.15 K). $\delta_{\text{water,max}}$ is a constant maximum thickness of melted water retained on the coil due to surface tension. Fuzzy numbers and their associated membership functions are constructed for each linguistic variable. For example, Figure 4 shows Fuzzy numbers N, P and membership functions for $T_{w,i} - T_{\text{melt}}$. Denote the governing equations of each stage as $f_j(\cdot)$ $(j = \{1,2,3,4,5\})$. The system dynamics can be determined based on linguistic descriptions of IF-THEN rules (**Rule i**, i={1,2,3,4,5}), which are subsequently converted into numerical outputs (known as the *defuzzification* process). Take the rule for the first stage (preheating) as an example,

Rule 1: IF $T_{w,i} - T_{melt}$ is N THEN $\dot{x} = f_1(x)$

where x is a state vector consisting of temperatures and thicknesses of different phase presences. Then a numerical output associated with the rule can be obtained

$$\omega_1 = \mu_N(T_{w,i} - T_{\text{melt}}) \tag{17}$$

which is used to compute a weighted combination of dynamics from all the stages as described in (18).

$$\dot{x} = \frac{\sum_{j=1}^{5} \omega_j f_j(x)}{\sum_{j=1}^{5} \omega_j} \tag{18}$$

In this way, the IF-THEN conditional rules can be eliminated, and the overall dynamics are evaluated by weighting the dynamics of all stages. The proposed switching algorithm systemically covers all possible switches and avoids discontinuities in switches. Consequently, the frost melting model can be used to evaluate dynamics of the frost density and thickness during RCD,

$$\frac{d\rho_{f,i}}{dt} = f_{\rho,\text{melt}}(\rho_{f,i}, \delta_{f,i}) \tag{19}$$

$$\frac{d\delta_{f,i}}{dt} = f_{\delta,\text{melt}}(\rho_{f,i}, \delta_{f,i}) \tag{20}$$

where $f_{\rho,\text{melt}}(\cdot)$ and $f_{\delta,\text{melt}}(\cdot)$ are the corresponding governing equations.

2.2 Component models

A quasi-static model is developed to describe performance of a variable-speed compressor using efficiency maps. The refrigerant mass flow rate is determined by

$$\dot{m} = \rho_{\text{suc}}\eta_v V_s \frac{N}{60} \tag{21}$$

where V_s is a fixed displacement, N is the number of rotations per minute. The volumetric efficiency η_v is regressed by a polynomial of pressure ratio and compressor speed,

$$\eta_v = a_0\left(\frac{p_{\text{dis}}}{p_{\text{suc}}}\right) + a_1\left(\frac{p_{\text{dis}}}{p_{\text{suc}}}\right)^2 + a_2\left(\frac{N}{60}\right) + a_3 \tag{22}$$

where $a_0 - a_3$ are constant coefficients. The power consumption is estimated using an isentropic efficiency

$$\eta_{is} = \frac{\dot{m}(h_{\text{dis,is}} - h_{\text{suc}})}{\dot{W}} \tag{23}$$

which is fitted by

$$\eta_{is} = b_0\left(\frac{p_{\text{dis}}}{p_{\text{suc}}}\right) + b_1\left(\frac{p_{\text{dis}}}{p_{\text{suc}}}\right)^2 + b_2\left(\frac{N}{60}\right) + b_3\left(\frac{p_{\text{dis}}}{p_{\text{suc}}}\right)\left(\frac{N}{60}\right) + b_4. \tag{24}$$

The refrigerant discharge state can be determined by forming an energy balance for the compressor,

$$\dot{W} = \dot{m}(h_{\text{dis}} - h_{\text{suc}}) + f_q\dot{W} \tag{25}$$

where the heat loss ratio has a strong dependence on the operating conditions and the ambient temperature T_{amb},

$$f_q = c_0\dot{m} + c_1 T_{\text{amb}} + c_2\left(\frac{N}{60}\right) + c_3. \tag{26}$$

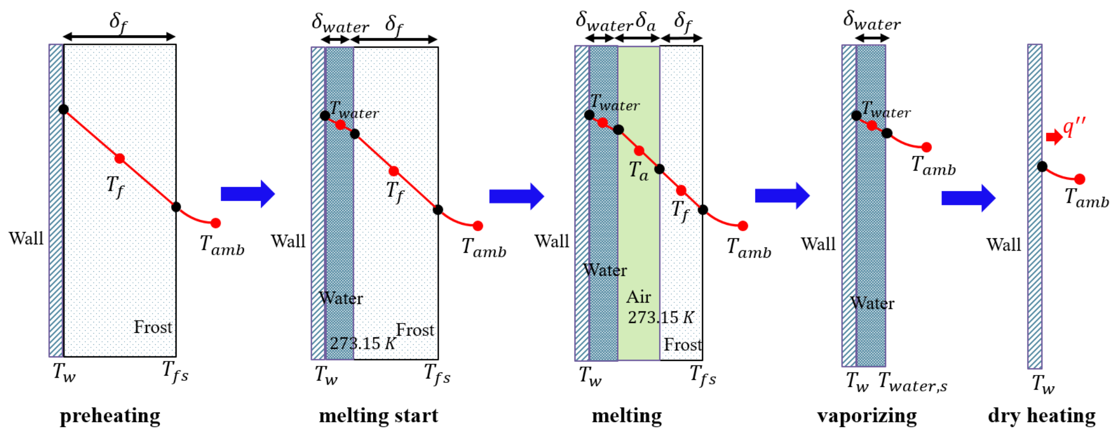


Figure 3. Five-stage frost melting process.

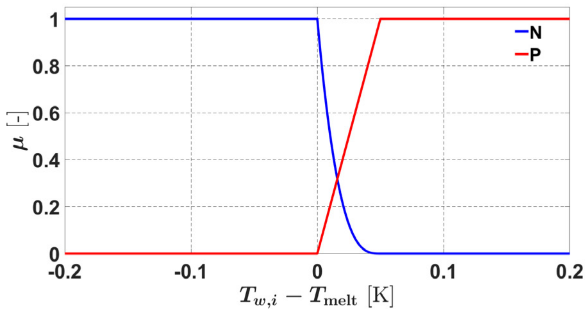


Figure 4. Fuzzy numbers and membership functions for linguistic variable $T_{w,i} - T_{\mathrm{melt}}$.

The expansion process is assumed to be isenthalpic. The mass flow rate is determined by

$$\dot{m} = C_d A_v \sqrt{2\rho_{\mathrm{in}}(p_{\mathrm{in}} - p_{\mathrm{out}})} \tag{27}$$

where A_v is the varying valve opening area, and C_d is the discharge coefficient that accounts for corrections to the mass flow rate at different operating conditions. Note that the valve is modeled as a bi-directional device that the inlet and outlet can switch when flow is reversed. The opening area is adjusted based on superheat control. An empirical correlation based on a power law is utilized to estimate the discharge coefficient (Liu, Cai, and Kim 2022):

$$C_d = d_0 \phi^{d_1} \left(\frac{T_{\mathrm{sc}}}{T_c}\right)^{d_2} \tag{28}$$

where T_{sc} is the subcooling at the valve inlet, T_c is the critical temperature of the refrigerant, and ϕ is the normalized valve opening.

An accumulator is modeled assuming that vapor and liquid inside are saturated and in thermal equilibrium, which lead to the mass and energy conservation equations as shown in (29)-(30),

$$\frac{d}{dt}(V_g\rho_g + V_f\rho_f) = \dot{m}_{\mathrm{in}} - \dot{m}_{\mathrm{out}} \tag{29}$$

$$\frac{d}{dt}(V_g\rho_g u_g + V_f\rho_f u_f) = \dot{m}_{\mathrm{in}} h_{\mathrm{in}} - \dot{m}_{\mathrm{out}} h_{\mathrm{out}}. \tag{30}$$

Note that these saturated properties are solely dependent on the refrigerant pressure, thus the lumped pressure can be selected as a state variable. Given the constituent relation that the total volume of the tank is occupied by the saturated liquid and vapor volumes $V_f + V_g = V_{\mathrm{acc}}$, either one of the volumes can be selected as another state variable. The exit flow is assumed to be saturated vapor to account for the fact that the superheated vapor from the evaporator mixes with the liquid refrigerant stored inside the accumulator.

The coil fan is described via a polynomial characteristic curve:

$$\Delta p_{\mathrm{rise}} = a_0 + a_1\dot{V} + a_2\dot{V}^2 + a_3\dot{V}^3. \tag{31}$$

where $\dot{V}$ is the total volume flow rate. Since the coil pressure drop is assumed to be solely a result of friction, a hydraulic equilibrium is established between the fan and the coil, which leads to the system of equations below:

$$\Delta p_{\mathrm{rise}} = \Delta p_{a,i} \quad i = 1,\ldots,n \tag{32}$$

where $\Delta p_{a,i}$ is the air side pressure drop of the ith control volume. Along with the mass balance

$$\rho_a\dot{V} = \sum_{i=1}^{n} \dot{m}_{a,i} \tag{33}$$

where $\dot{m}_{a,i}$ represents the air flow distribution of each control volume, a non-linear algebraic equation system is formed to solve the air flow maldistribution under non-uniform frost formation.

A reversing valve model is essential to switch the heat pump system model between heating and defrosting modes. It is challenging to model the exact physical process of a reversing valve, since it mainly involves

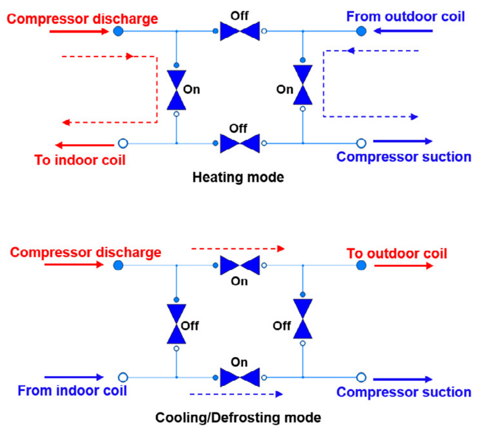


Figure 5. Model representation of a reversing valve.

mechanical movements. In the current work, two pairs of check valves are used to represent the reversing valve model (Qiao, Aute, and Radermacher 2015). As shown in Figure 5, each port of the reversing valve has a pair of check valves to control the flow direction. One is kept fully opened, while the other is closed. When the on-off statuses of them are switched, the refrigerant flow direction is reversed. A check valve is modeled as

$$\dot{m} = \begin{cases} \phi A_v \sqrt{2\rho_{\text{in}}(p_{\text{in}} - p_{\text{out}})} & p_{\text{in}} > p_{\text{out}} \\ 0 & p_{\text{in}} \le p_{\text{out}} \end{cases} \tag{34}$$

where A_v is the fully opened area, ϕ is valve opening degree normalized between 0 and 1. When reverse flow needs to be triggered, opening and closing a check valve is achieved by changing the valve opening. However, this process is not instantaneous during mode switching. For numerical stability concerns, a first-order filter is applied to smooth the transition,

$$\frac{d\phi}{dt} = \frac{1}{\tau}(\phi_{SP} - \phi) \tag{35}$$

where ϕ_{SP} is the opening setpoint, which is either 1 or 0 depending on the on/off status of each check valve. The heat loss from the high pressure side is primarily driven by the temperature difference between the discharge gas and the suction gas, and can be characterized by heat transfer loss/gain coefficients (Damasceno, Rooke, and Goldschmidt 1991).

3 Model implementation

To simulate the cycling behavior of non-uniform frost growth and melting on the outdoor coil, the frost formation and melting models are integrated into each discretized control volume of the heat exchanger model. Figure 6 depicts the cross-flow heat exchanger model composed of flow and heat&mass transfer elements. The equivalent refrigerant flow length is divided into n equally sized control volumes (CVs). Of each discretized length, four elements can be identified across the predominant heat flow direction, which is perpendicular to the refrigerant flow direction: the refrigerant flow, the coil metal structure, the frost layer, and the air flow. Note that the frost layer in Figure 6 represents simultaneously running the frost growth and melting models. These frost models are implemented as generic control volumes to maintain a consistent model structure, thus the frost layer dynamics coupled to the heat pump system dynamics can switch between frost growth and melting, according to the system operating mode. The standard *HeatPort* extended from *Modelica.Thermal.HeatTransfer* is employed at the interface between the frost and air flow. However, at the interface between the frost and metal wall, a heat source is utilized for the metal wall model whose input switches between computed heat flow rates from the frost growth and melting models. Furthermore, the lumped metal wall temperature is considered as an input to the frost models. Since the frost density and thickness characterize its dynamics, they are selected as internal states. That means, the current states of frost density and thickness at each time step are treated as known and inputs to the frost models, while their dynamics are obtained as outputs for predictions over the next time interval.

After that, the overall frost dynamics are evaluated as a weighted combination of dynamics obtained from these two models. Nonetheless, due to different assumptions applied for derivations, it is not straightforward to implement the scheme before having a uniform structure among them. Since the frost formation model is derived in a quasi-steady-state fashion, and updates the frost density and thickness in a fixed time step as shown in (15, 16), the progress made over time can be considered as reference values. Then a first-order filter is applied to track the reference values

$$\frac{d\rho_{f,i}}{dt} = \frac{1}{\tau}(\rho_{\text{ref},i} - \rho_{f,i}) \tag{36}$$

$$\frac{d\delta_{f,i}}{dt} = \frac{1}{\tau}(\delta_{\text{ref},i} - \delta_{f,i}) \tag{37}$$

where $\rho_{\text{ref},i}$ and $\delta_{\text{ref},i}$ are quantities evaluated in (15, 16), $\rho_{f,i}$ and $\delta_{f,i}$ are states representing the frost behavior, τ is a time constant, which should be selected such that the signal tracking is faster than update of the reference values. In this way, dynamics of the frost formation can be described in a continuous-time domain. Note that these reference values evaluated in the frost formation model are non-decreasing. This is because throughout the derivation the mass flux is assumed to be non-negative, meaning that no melting behavior is considered in the frost formation model. Therefore, in case of a mode switch from defrosting to frosting (heating), the reference values should be reset to the states at the termination of frost melting as initial conditions for the next frosting period. The overall

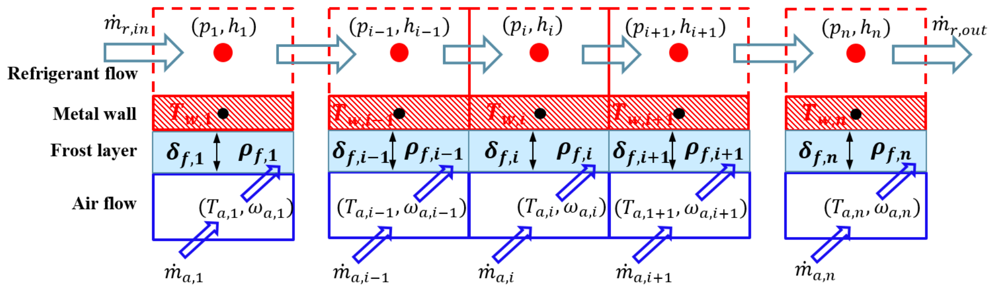


Figure 6. Segments of finite-volume heat exchanger model with frost incorporated.

frost dynamics are evaluated by

$$\frac{d\rho_{f,i}}{dt} = \phi\Big(\frac{1}{\tau}(\rho_{\text{ref},i} - \rho_{f,i})\Big) + (1-\phi) f_{\rho,\text{melt}}(\rho_{f,i}, \delta_{f,i}) \tag{38}$$

$$\frac{d\delta_{f,i}}{dt} = \phi\Big(\frac{1}{\tau}(\delta_{\text{ref},i} - \delta_{f,i})\Big) + (1-\phi) f_{\delta,\text{melt}}(\rho_{f,i}, \delta_{f,i}) \tag{39}$$

where $f_{\rho,\text{melt}}(\rho_{f,i}, \delta_{f,i})$ and $f_{\delta,\text{melt}}(\rho_{f,i}, \delta_{f,i})$ are the dynamics obtained from the frost melting model, ϕ is a variable that switches the frost dynamics between frost formation and melting. The reversing valve opening in (35) can be used for this purpose. The scheme allows the frost formation and melting models to run simultaneously during online simulations, which enables modeling the cycling of frosting and defrosting without interruptions.

Consequently, component models are interconnected through the standard fluid connectors to form a cycle model. Refrigerant properties are evaluated using a regression approach based on Artificial Neural Networks (Ma, Kim, and Braun 2018).

4 Simulation results and validation

This section presents simulation results and experimental validations of the developed cycle model for capturing transient characteristics of a residential heat pump system under multiple cycles of frosting and reverse-cycle defrosting (RCD). The heat pump unit is a commercial variable-speed system having a nominal capacity of 2 tons that utilizes R410A as the working fluid. The split system was installed in a pair of independently controlled psychrometric chambers, where the indoor unit was ducted to a nozzle box for air flow rate measurements. The experimental setup was fully instrumented with T-type thermocouples, pressure transducers, and a coriolis flow meter on the refrigerant loop. T-type thermocouple grids and chilled mirror dew point meters were installed for air-side measurements. Power meters were installed to measure power of the indoor and outdoor units as well as the outdoor fan. The temperature set points of the indoor and outdoor conditions were 291.5 K and 271 K, respectively. The relative humidity was set to 40% for the indoor and 85% for the outdoor. The unit was turned on in heating mode after these conditions were met. At low ambient temperature, a system built-in controller determined operating modes between heating and defrosting based on a fixed-interval defrost logic. During defrost cycles, the compressor is operated at a lower speed than the heating mode but never shut off. The EXV opening is regulated to control the evaporator outlet superheat in the heating mode but is fully opened in the defrosting as well as cooling modes. The indoor and outdoor coil fan speeds are controlled according to the compressor speed, however, the outdoor fan is off in defrost cycles. The defrost interval, representing the compressor run time between two defrost cycles, was set to 120 minutes as the longest. However, upon startup of the unit, the first defrost interval was defaulted to 30 minutes. Regarding this defrost control pattern, a test that lasted three hours ran through a start-up with minor frost accumulation for 30 minutes, followed by the first defrost cycle, then normal heating operation for 120 minutes with considerable frost buildup, followed by a second defrost cycle, and then another heating operation period of about 10 minutes. Even though the inlet air conditions were kept constant in both chambers, various durations of each frosting cycle and different defrost cycle durations yielded different cycling transients of the system, since frost buildup and removal are inherently transient processes.

Each heat exchanger was discretized into 30 control volumes based on a preliminary sensitivity analysis that the number of control volumes is determined until the solution is invariant to further increases of control volumes. The heat pump cycle model was initialized using the pressure and temperature measurements across various components. The initial refrigerant pressure of each component was almost at the same level, though the temperature differed depending on the room temperature. As a result, initial conditions of most components reflect gas phase refrigerant presence, while the refrigerant residing in the outdoor coil was initialized as two-phase. Without access to the liquid level of the suction accumulator, steady-state initialization was performed using the cycle model to determine the vapor and liquid volumes. Simulations were carried out to predict the complete cycling operations of frosting and reverse-cycle defrosting lasting three hours.

Figure 7 reports validations of the refrigerant discharge

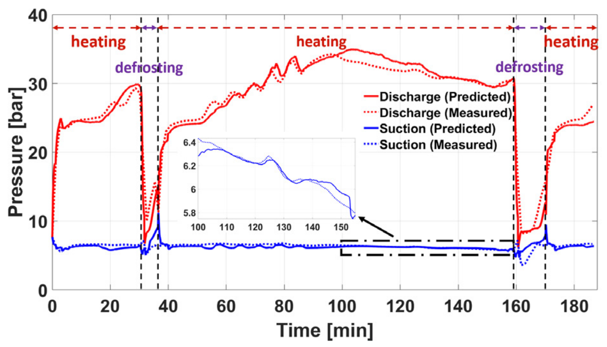


Figure 7. Validations of the refrigerant discharge and suction pressures.

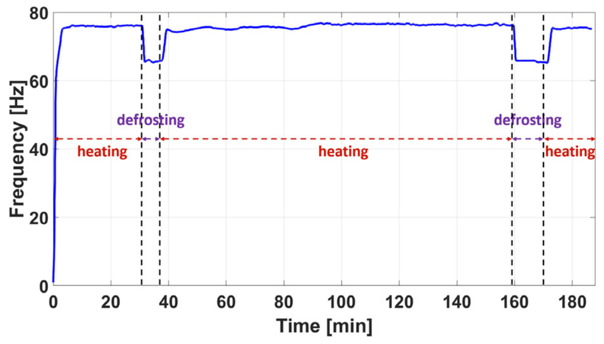


Figure 8. Compressor speed.

and suction pressures. As the compressor is powered on, its speed increases to the desired setting regulated by the control board in 3 minutes as shown in Figure 8, which results in the rapid rise of the discharge pressure. The EXV is fixed to a preset position during this period before the superheat control is enabled. After that, rise of the discharge pressure slows down since the compressor is running at a pre-defined speed. As the EXV is closing to achieve the superheat set point temperature, the resulting mass flow imbalance between the compressor and EXV leads to the continuing pressure increase. As mentioned before, the first defrost cycle is defaulted to 30 minutes after the initial power-up of the compressor. When the system initiates defrost operation, the EXV is fully opened, which leads to a dramatic pressure change due to the pressure equalization across the EXV. Shortly the reversing valve is energized, which further brings down the discharge pressure, and slightly increases the suction pressure due to the connection to the high pressure side when the valve is switching port positions. After the flow is reversed, the discharge gas is pumped into the outdoor coil, which operates as a condenser, and the liquid refrigerant residing in the indoor coil, which operates as an evaporator, vaporizes due to the abrupt pressure drop and flows towards the vapor line. As a consequence, the discharge and suction pressures increase until termination of the defrost mode. The first defrost cycle lasts about 5 minutes. Then the reversing valve is energized again to switch the flow directions, that reduces the pressure difference between the high pressure side and the low pressure side. Meanwhile, the EXV opening is reset to an initial position, and then adjusted following superheat control. A similar trend of the discharge pressure can be observed as during the start-up, a rapid rise due to the compressor speed regulation, followed by a gradual increase from 40 min to around 100 min as the refrigerant mass flow is being balanced between the vapor-line and liquid-line. Though frost starts to form shortly after the system switches back to heating mode, the refrigerant dynamics during this period are dominated by the mass flow imbalance without noticeable impact from the frost accumulation. However, performance degradation due to frost formation becomes more evident since 100 min until initiation of the next defrost cycle (160 min). The reduced air flow due to frost blockage results in a slight decline of the evaporating pressure (see zoom-in plot in Figure 7), yet a significant decline of the discharge pressure which is more sensitive to changes on the suction side. The second defrost cycle is initiated as the accumulated compressor run time reaches the preset defrost interval. The refrigerant pressures exhibit a similar trend as the previous defrost cycle, triggered by the same consecutive actuation of the EXV and reversing valve, although the pressure rises are relatively smooth because a large amount of heat is taken to melt frost that slows down the refrigerant energy storage. This defrost cycle lasts 10 minutes, after which the system returns to heating mode. The simulation results agree sufficiently well with the measurements, which demonstrate that the developed cycle model is able to capture the complicated system dynamics under the frosting-defrosting cycling operations. The discrepancy between the predictions and measurements during defrost cycles is due to model simplification and unknown response times of actuators including EXV and reversing valve that trigger the mode switches. For instance, rises in the predicted suction pressure around 35 min and 170 min during mode switches are not observed in the measurements. This is due to implementation of the reversing valve model using a system of check valves. When check valves are closed to switch flow direction, the rapid mass flow decrease almost results in a pressure equalization of the discharge and suction pressures.

Figure 9 shows the refrigerant mass flow rate prediction in the indoor liquid-line. It is important to note that the refrigerant flow can be bi-directional in most of the components depending on the operating mode. Following the sign convention that inflow quantities are positive and outflow quantities are negative, a positive mass flow rate at the inlet of a component indicates the nominal flow direction (e.g., heating mode), while a negative flow rate indicates reverse flow (e.g., defrosting mode). The sign convention applies to both mass flow and heat flow in the following context. In heating mode, the refrigerant mass flow rate undergoes abrupt changes when the compressor

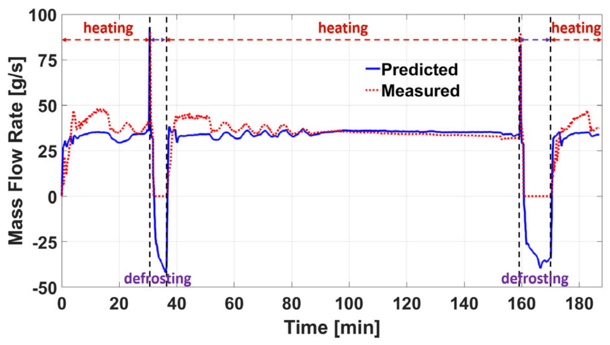


Figure 9. Liquid-line refrigerant mass flow rate.

speed is regulated, otherwise fluctuates according to the discharge and suction properties. From 87 min until initiation of the second defrost cycle at 160 min, the mass flow slightly declines during the stable heating operation due to an insignificant decline of the pressure ratio. When the system switches to defrost mode at 30 min and at 160 min, a spike of the liquid-line flow rate occurs as the EXV is fully opened. The reversing valve is then engaged to trigger the flow reversal. The mass flow rate steadily increases during the defrost cycle since expansion valves are kept fully open as the reversed refrigerant flow develops, until the flow is reversed at the end of each cycle. Note that the output signal of the mass flow meter saturates to zero when the flow direction is opposite to the configured direction in the test. That means, the flow rate measurement during defrost cycle is unavailable. The comparison between the simulated and measured mass flow shows that the model can capture major flow transients in heating operations and the flow reversal between mode switches. The discrepancy during each start-up in heating mode can be attributed to prediction errors of quasi-static models of the compressor and EXV that were correlated using steady-state data. As can be seen from 80 min to 160 min, the nearly steady-state mass flow after start-up transients is well captured.

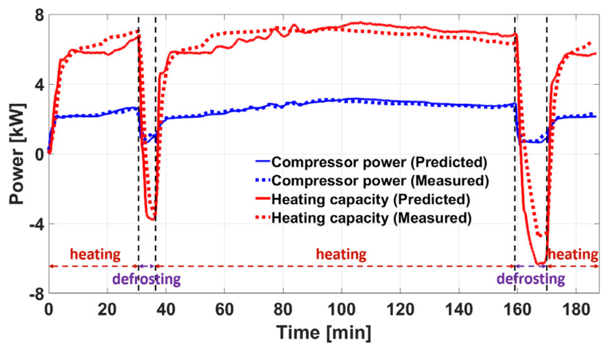


Figure 10. Validations of the indoor unit air-side capacity and compressor power.

Simulation results of the indoor unit air-side capacity and the compressor power are revealed in Figure 10. The indoor unit fan is continuously running throughout the test. The air flow rate climbs after the fan motor is powered, and reaches a constant value based on the measurements. The air-side capacity evolves corresponding to the refrigerant dynamics. Since the compressor operates at a nearly fixed speed, a decline of the heating capacity can be observed during 100 min to 160 min, when the refrigerant condensing pressure drops due to the frosting operation. In defrost mode, the indoor coil operates as an evaporator and the heat flow direction is reversed compared to heating mode. During this period, heat is removed from the indoor space, which can potentially result in thermal discomfort. The model captures dynamics of the indoor capacities under both condenser and evaporator modes well.

The change of compressor power consumption follows regulation of the compressor speed, while also varying due to the refrigerant dynamics. During heating cycles, the power consumption changes mainly due to transients of the discharge pressure, which in turn dominates the pressure ratio. The lower power consumption of defrost cycles can be attributed to the lower compressor speed (Figure 8) and the smaller pressure ratio. Predictions of the compressor power correspond well with the measurements since the model accurately captures refrigerant pressures.

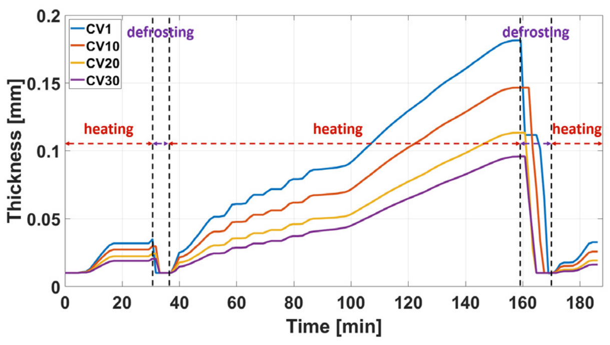


Figure 11. Simulated frost thickness of CVs 1, 10, 20, 30.

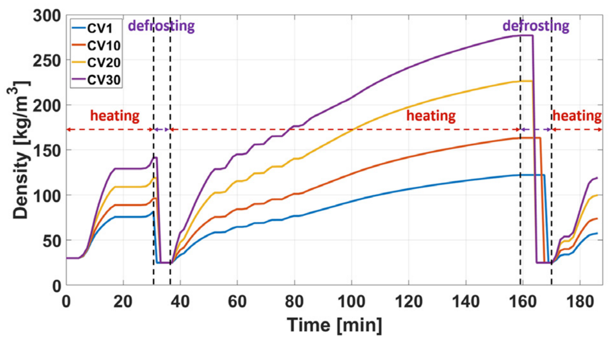


Figure 12. Simulated frost density of CVs 1, 10, 20, 30.

Figure 11 and Figure 12 show the simulated frost thickness and density trajectories associated with selected control volumes. Note that the CVs are indexed following the

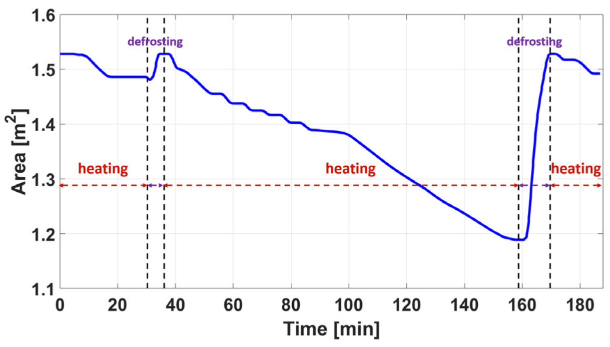


Figure 13. Simulated free airflow passage of the outdoor coil.

flow direction in heating mode, with CV1 representing the inlet connected to the EXV and CV30 representing the exit connected to the reversing valve. A significant amount of frost is formed during the 2-hour frosting cycle. Clearly more frost accumulates close to the liquid-line of the coil, where the refrigerant temperature is lower at the beginning of each frosting cycle. Correspondingly, the frost tends to be denser close to the vapor-line. During defrost cycles, the frost is considered to be completely melted when the thickness is smaller than 0.01 mm. Then the frost density is reset to 25 kg/m^3 to provide a reasonable initial condition for the next frosting cycle. Figure 13 shows the simulated free airflow passage of the outdoor coil due to frost blockage. It can be noted that 23% of the total flow passage is blocked after two hours of frosting operation.

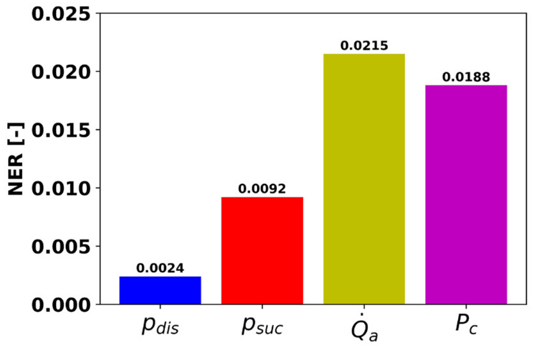


Figure 14. Normalized error residuals (NER) for: discharge pressure p_{dis}, suction pressure p_{suc}, indoor air-side capacity $\dot{Q}_a$ and compressor power P_c.

5 Conclusions

This paper presents a complete heat pump cycle model under cycling of frosting and reverse-cycle defrosting operations. Comparisons between the model predictions and measurements demonstrate capabilities of the developed model to capture system characteristics with non-uniform frost formation and melting. The overall simulation results yield good agreements with the measurements. Figure 14 shows quantification of transient prediction errors using normalized error residuals (NER). Relatively large values of NER in the indoor air-side capacity and compressor power can be attributed to prediction errors in the refrigerant mass flow rate associated with performance maps. However, results of NER all lie below 0.025, which are sufficiently good for transient simulations. Additionally, predictions of the evolution of frost properties over time provides insights into the non-uniform frost formation and melting phenomena, which are typically difficult to measure and characterize experimentally at a system level. The proposed models are attractive for development and evaluation of improved control and fault detection designs for ASHPs with frosting and defrosting taken into considerations.

Nomenclature

α	Convective heat transfer coefficient $[\mathrm{W\,m^{-2}K^{-1}}]$
δ	Thickness [m]
$\dot{m}$	Mass flow rate $[\mathrm{kg\,s^{-1}}]$
$\dot{Q}$	Heat flow rate [W]
$\dot{V}$	Volume flow rate $[\mathrm{m^3\,s^{-1}}]$
$\dot{W}$	Power [W]
ω	Humidity ratio $[\mathrm{kg\,kg^{-1}dry\,air}]$
ρ	Density $[\mathrm{kg\,m^{-3}}]$
A	Area $[\mathrm{m^2}]$
c_p	Specific heat capacity $[\mathrm{J\,kg^{-1}K^{-1}}]$
F	Force [N]
H	Enthalpy flow rate $[\mathrm{J\,s^{-1}}]$
h	Specific enthalpy $[\mathrm{J\,kg^{-1}}]$
L	Length [m]
M	Mass [kg]
N	Rotations per minute $[\mathrm{min^{-1}}]$
p	Pressure [Pa]
T	Temperature [K]
V	Volume $[\mathrm{m^3}]$
v	Velocity $[\mathrm{m\,s^{-1}}]$

References

Bergman, Theodore L, Frank P Incropera, David P DeWitt, and Adrienne S Lavine (2011). *Fundamentals of heat and mass transfer*. John Wiley & Sons.

Breque, Florent and Maroun Nemer (2017). “Modeling of a fan-supplied flat-tube heat exchanger exposed to non-uniform frost growth”. In: *International Journal of Refrigeration* 75, pp. 129–140.

Damasceno, G, S Rooke, and VW Goldschmidt (1991). “Effects of reversing valves on heat pump performance”. In: *International journal of refrigeration* 14.2, pp. 93–97.

Dopazo, J Alberto, Jose Fernandez-Seara, Francisco J Uhıa, and Ruben Diz (2010). “Modelling and experimental validation of the hot-gas defrost process of an air-cooled evaporator”. In: *International journal of refrigeration* 33.4, pp. 829–839.

Franke, Rüdiger, Francesco Casella, Martin Otter, et al. (2009). “Stream connectors-an extension of Modelica for device-oriented modeling of convective transport phenomena”. In: *Proceedings of the 7th International Modelica Conference*. Linköping University Electronic Press, pp. 108–121.

Franke, Rüdiger, Francesco Casella, Michael Sielemann, Katrin Proelss, and Martin Otter (2009). “Standardization of thermo-fluid modeling in Modelica. Fluid”. In: *Proceedings of the 7th International Modelica Conference*. Linköping University Electronic Press, pp. 122–131.

Han, Binglong, Tong Xiong, Shijie Xu, Guoqiang Liu, and Gang Yan (2022). “Parametric study of a room air conditioner during defrosting cycle based on a modified defrosting model”. In: *Energy* 238, p. 121658.

Kim, Donghun, Jiacheng Ma, James E Braun, and Eckhard A Groll (2021). “Fuzzy modeling approach for transient vapor compression and expansion cycle simulation”. In: *International Journal of Refrigeration* 121, pp. 114–125.

Laughman, Christopher R, Hongtao Qiao, Vikrant Aute, and Reinhard Radermacher (2015). “A comparison of transient heat pump cycle models using alternative flow descriptions”. In: *Science and Technology for the Built Environment* 21.5, pp. 666–680.

Leoni, Aurélia, Michèle Mondot, François Durier, Rémi Revellin, and Philippe Haberschill (2017). “Frost formation and development on flat plate: Experimental investigation and comparison to predictive methods”. In: *Experimental Thermal and Fluid Science* 88, pp. 220–233.

Liu, Haopeng, Jie Cai, and Donghun Kim (2022). “A hierarchical gray-box dynamic modeling methodology for direct-expansion cooling systems to support control stability analysis”. In: *International Journal of Refrigeration* 133, pp. 191–200.

Ma, Jiacheng, Donghun Kim, and James E Braun (2018). “Development Of A Fast Method For Retrieving Thermodynamic Properties To Accelerate Transient Vapor Compression Cycle Simulation”. In: *17th International Refrigeration and Air Conditioning Conference at Purdue*.

Padhmanabhan, SK, DE Fisher, L Cremaschi, and E Moallem (2011). “Modeling non-uniform frost growth on a fin-and-tube heat exchanger”. In: *International journal of refrigeration* 34.8, pp. 2018–2030.

Qiao, Hongtao, Vikrant Aute, and Reinhard Radermacher (2015). “Transient modeling of a flash tank vapor injection heat pump system–Part I: Model development”. In: *International journal of refrigeration* 49, pp. 169–182.

Qiao, Hongtao, Vikrant Aute, and Reinhard Radermacher (2017). “Dynamic modeling and characteristic analysis of a two-stage vapor injection heat pump system under frosting conditions”. In: *International Journal of Refrigeration* 84, pp. 181–197.

Qiao, Hongtao, Vikrant Aute, and Reinhard Radermacher (2018). “Modeling of transient characteristics of an air source heat pump with vapor injection during reverse-cycle defrosting”. In: *International journal of Refrigeration* 88, pp. 24–34.

Steiner, Alois and René Rieberer (2013). “Parametric analysis of the defrosting process of a reversible heat pump system for electric vehicles”. In: *Applied thermal engineering* 61.2, pp. 393–400.

Appendix

The governing equations of the preheating stage are:

$$\rho_{f,i}c_{p,f,i}\delta_{f,i}\frac{dT_{f,i}}{dt} = k_{f,i}\frac{T_{w,i}+T_{fs,i}-2T_{f,i}}{\delta_{f,i}/2} \tag{40}$$

$$k_{f,i}\frac{T_{f,i}-T_{fs,i}}{\delta_{f,i}/2} = q''_{a,i} \tag{41}$$

where $T_{fs,i}$ is the temperature of the frost-air interface, $q''_{a,i}$ is the heat flux transfer to the surrounding air. At this stage the wall surface temperature is still below the freezing point of water (273.15 K in the following equations), which indicates that no phase change occurs.

The governing equations of the melting start stage are (41), (42)-(45)

$$\rho_{water,i}c_{p,water,i}\delta_{water,i}\frac{dT_{water,i}}{dt} = k_{water,i}\frac{T_{w,i}+273.15-2T_{water,i}}{\delta_{water,i}/2} \tag{42}$$

$$\rho_{f,i}c_{p,f,i}\delta_{f,i}\frac{dT_{f,i}}{dt} = k_{f,i}\frac{273.15+T_{fs,i}-2T_{f,i}}{\delta_{f,i}/2} \tag{43}$$

$$-\rho_{f,i}\Delta h_{sf}\frac{d\delta_{f,i}}{dt} = k_{water,i}\frac{T_{water,i}-273.15}{\delta_{water,i}/2} - k_{f,i}\frac{273.15-T_{f,i}}{\delta_{f,i}/2} \tag{44}$$

$$\rho_{water,i}\frac{d\delta_{water,i}}{dt} = -\rho_{f,i}\frac{d\delta_{f,i}}{dt} \tag{45}$$

where the latent heat of fusion Δh_{sf} is 334 kJ/kg.

As the frost starts to thaw, a water film appears between the wall and the frost layer. It is assumed that temperature of the water-frost interface remains at 273.15 K. This stage terminates until the maximum amount of water that can be retained on the coil surface is reached. The maximum water film thickness is set to $\delta_{water,max} = 0.05$ mm.

The frost continues to thaw during the melting stage, but the melted water is assumed to be drained immediately, causing the development of an air gap between the water film and the frost. The temperature of the air-frost interface remains at 273.15 K. The governing equations of this stage are (41), (43), (46)-(50)

$$\rho_{water,i} c_{p,water,i} \delta_{water,i} \frac{dT_{water,i}}{dt} = k_{water,i} \frac{T_{w,i} - T_{water,i}}{\delta_{water,i}/2} - \frac{T_{water,i} - T_{a,i}}{\frac{\delta_{water,i}}{2k_{water,i}} + \frac{\delta_{a,i}}{2k_{a,i}}} \tag{46}$$

$$\rho_{a,i} c_{p,a,i} \delta_{a,i} \frac{dT_{a,i}}{dt} = \frac{T_{water,i} - T_{a,i}}{\frac{\delta_{water,i}}{2k_{water,i}} + \frac{\delta_{a,i}}{2k_{a,i}}} - k_{a,i} \frac{T_{a,i} - 273.15}{\delta_{a,i}/2} \tag{47}$$

$$-\rho_{f,i} \Delta h_{sf} \frac{d\delta_{f,i}}{dt} = k_{a,i} \frac{T_{a,i} - 273.15}{\delta_{a,i}/2} - k_{f,i} \frac{273.15 - T_{f,i}}{\delta_{f,i}/2} \tag{48}$$

$$\frac{d\delta_{a,i}}{dt} = -\frac{d\delta_{f,i}}{dt} \tag{49}$$

$$\frac{d\delta_{water,i}}{dt} = 0. \tag{50}$$

The governing equations of the vaporizing stage are

$$\rho_{water,i} c_{p,water,i} \delta_{water,i} \frac{dT_{water,i}}{dt} = k_{water,i} \frac{T_{w,i} + T_{water,s,i} - 2T_{water,i}}{\delta_{water,i}/2} \tag{51}$$

$$\rho_{water,i} \frac{d\delta_{water,i}}{dt} = c_v(\rho_{amb} - \rho_{water,s,i}) \tag{52}$$

$$k_{water,i} \frac{T_{water,i} - T_{water,s,i}}{\delta_{water,i}/2} = q''_{a,i} \tag{53}$$

$$\frac{\delta_{f,i}}{dt} = 0 \tag{54}$$

$$\frac{\delta_{a,i}}{dt} = 0 \tag{55}$$

where $T_{water,s,i}$ is the water layer surface temperature that appears in this stage as an intermediate variable to quantify heat dissipation to the ambient, ρ_{amb} and $\rho_{water,s,i}$ are water vapor densities evaluated at the ambient temperature and the water layer surface temperature, respectively. c_v is the vaporization coefficient (Qiao, Aute, and Radermacher 2018).

In the dry heating stage, heat is directly dissipated from the metal wall to the ambient air. Variables of the thicknesses and temperatures in the frost melting model are inactive.

Simulation of the on-orbit construction of structural variable modular spacecraft by robots

Matthias J. Reiner[1]

[1]German Aerospace Center (DLR), Institute of System Dynamics and Control (SR), Wessling, Germany,
Matthias.Reiner@dlr.de

Abstract

This paper gives an overview on the simulation of the on-orbit construction of structural variable modular spacecraft by robots using Modelica. For this purpose, a new concept using so-called tensor bodies was developed which enables the fast and continuous simulation of complex scenarios even during structural changes. This research was part of two ESA and EU projects. An overview of the modeling and simulation approach will be given. The scenarios include the on-orbit re-configuration of a modular satellite with a walking robot manipulator and the construction of a modular space antenna array platform with a walking robot with two arms and a torso.
Keywords: Modelica, variable structure, walking robot, on-orbit servicing

1 Introduction

Future advanced spacecraft and orbital platforms can require on-orbit assembly either because of the size of the structures (e.g. large antennas, solar facilities or telescopes) or because of a desired modularity. Since on-orbit human labor is extremely expensive and dangerous, on-orbit assembly and servicing tasks should be done (mostly) autonomously by robots. Because of the complexity and cost of these types of missions, simulation and demonstrator studies on earth should be performed before committing necessary resources for a real mission. Recent research projects by ESA and the EU are working on this topic and DLR-SR was involved in the modeling and simulation of these complex scenarios.

Within the Horizon 2020 EU-funded project MOSAR (Modular and Re-Configurable Spacecraft, see (Letier, Yang, et al. 2019)) a ground demonstrator for on-orbit modular and re-configurable satellites as well as the supporting software and control system was developed.

The basic idea of the project is to use a set of re-usable spacecraft modules as part of a global eco-system. Each individual module can be dedicated to a specific function such as control, power, thermal management or sensors. Once assembled, they will allow the full functionality of the spacecraft. A symmetric walking robotic manipulator (WM, see (Deremetz, Letier, et al. 2020)) allows to capture, manipulate and position the spacecraft modules, while being able to reposition itself on the standard interfaces (abbreviated as SI or Hotdock, see (Letier, Siedel, et al. 2020)) of the spacecraft or on the modules. These SIs provide mechanical, data, power and thermal transfer for interconnection between the modules, spacecraft and the walking manipulator. They are actuated to ensure a safe connection when closed and are containing tubes for the heat transfer as well as connector ports for data and electricity. Within MOSAR, DLR-SR was responsible for the development of a Functional Engineering Simulation (FES) environment and design tool, offering assistance for module design, system configuration and operation planning, with the support of a multi-physics engine.

The FES as well as the design tool were developed in Modelica with an additional MATLAB interface for the project partners. Fig. 1 shows an example for the graphical output of the FES using DLR SimVis (see (T. Bellmann 2009), (Hellerer, Tobias Bellmann, and Schlegel 2014) and (Kümper, Hellerer, and Tobias Bellmann 2021)) for one of the simulated on-orbit scenarios, which is generated directly during the simulation by the Modelica model. The visualization displays a simplified representation of the most important components of the MOSAR scenario. The colored box blocks represent the individual modules. The pipes within the modules represent the (possible) flow of power and heat within the modules. The spheres within the modules are used to represent the powered state of the module (symbolized by a green or red star in the middle) and the color of the sphere represents the current temperature. The seven-axis robot WM is shown in grey The visualization was imported from CAD data provided by SpaceApplications.

In a similar (still ongoing) ESA project call MIRROR (Multi-arm Installation Robot for Readying ORUS and Reflectors), the on-orbit assembly of a large reflector consisting of hexagonal shaped modules is investigated using a ground equivalent laboratory demonstrator. For this purpose, the novel Multi-Arm Robot (MAR) is used which is a modular robot composed of three robotic subsystems: a torso and two symmetrical 7-degree of freedom (DOF) anthropomorphic arms. The arms are based on the WM design from the MOSAR project. The torso has one additional DOF and is the main body of the robot. This mechanical hub can equip three other appendages (limbs or payloads) or can be attached directly to the spacecraft structure (see (Deremetz, Grunwald, et al. 2021)) using

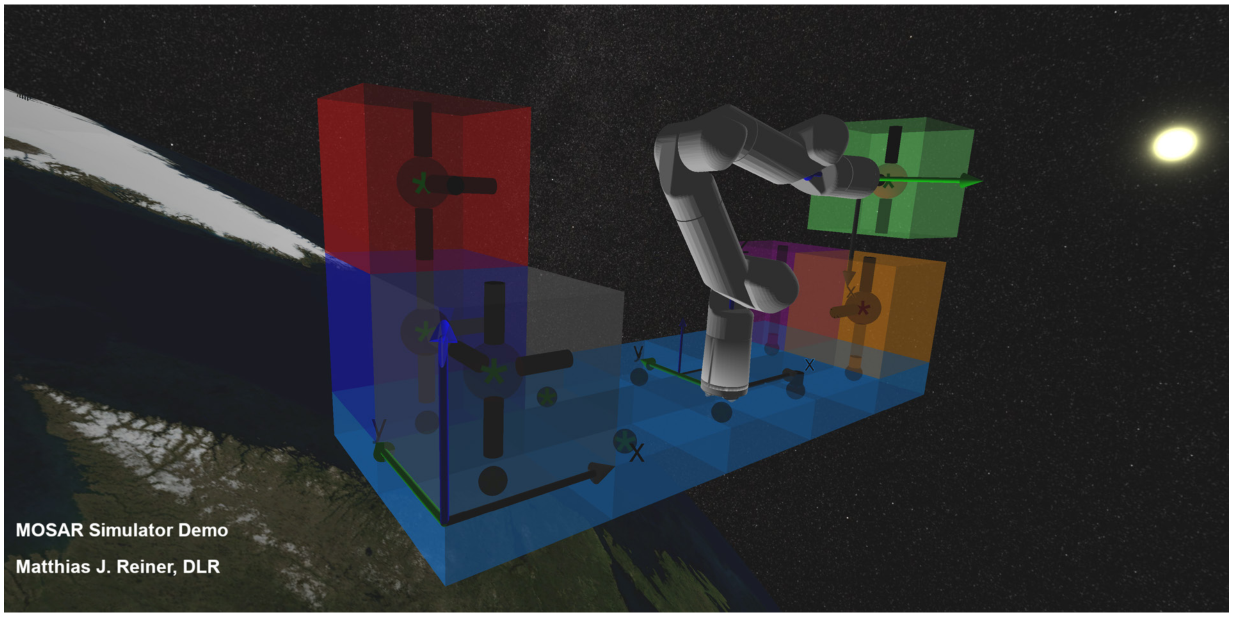


Figure 1. Visualization generated by the FES for an on-orbit assembly scenario investigated in the MOSAR project.

SIs. In this project, DLR-SR is responsible for the simulation of the scenario as well. A simulation tool called MIRROR Kinematic Dynamic & Graphical Simulator (abbreviated in the following as KDG) was developed, which extends the FES from the MOSAR project and is also developed in Modelica with an additional MATLAB interface. Fig. 2 shows the visualization generated by the KDG (also using SimVis). The hexagonal elements, which are used to build up the array, are taken from a stack on the servicer satellite (simplified represented by the yellow bar connector) by the MAR. In the configuration shown in the figure, the MAR is connected to a module with its SI at one of its WM arms and places another module at the edge of the array. In this configuration it has a very long reach. Alternatively, the MAR can also be connected to the array with its torso to enable handling with two arms simultaneously.

The focus of both projects is different, but from a simulation standpoint they share a lot of similarities. The focus for both simulation tools is to get insight in the dynamics of the systems at an early development stage and to test control algorithms for the robots and the logic and handling of the modular components and the standard interfaces, which are used to connect the elements as well as to allow the robots to move over the modules. To enable this the robots have the same SI connectors at the tip of the arms and torso. This allows the MOSAR symmetric walking manipulator (WM) to move by connecting one end of the robot to an SI and then with the other end to another (and switching the working base in the process). The MAR with its two arms and torso has even more possibilities to move in a similar manner.

The SI connectors can not only provide a mechanical connection, but can also be used to transfer power and exchange heat, what also has to be considered for the simulation.

The modular and changing nature of the MIRROR and MOSAR scenarios make the modeling and simulation a challenging task. Key challenges are the following:

- Complex structural variable systems consisting of modular satellite or array components. Modelica (and most other multi-body systems) do not directly support structural variable systems. The number of states must be constant during the simulation.
- Hybrid and stiff systems with discrete and continuous parts. The discrete elements result from the switching behavior of the SIs and algorithms for the control logic.
- Many disciplines involved: e.g. power and thermal management, robot control, and orbital mechanics
- Dynamic robot arm docking and pick & place operations with a high number of potential connection points. The term pick & place operation is used here for the following sequence of operations:
  1. The robot moves to the location of the module, ready to grasp the module
  2. The module is unfixed from the spacecraft and connected to the robot using the SIs
  3. The robot moves the module to the desired location
  4. The module is fixed at its new position via one or more SIs and disconnected from the robot SI.

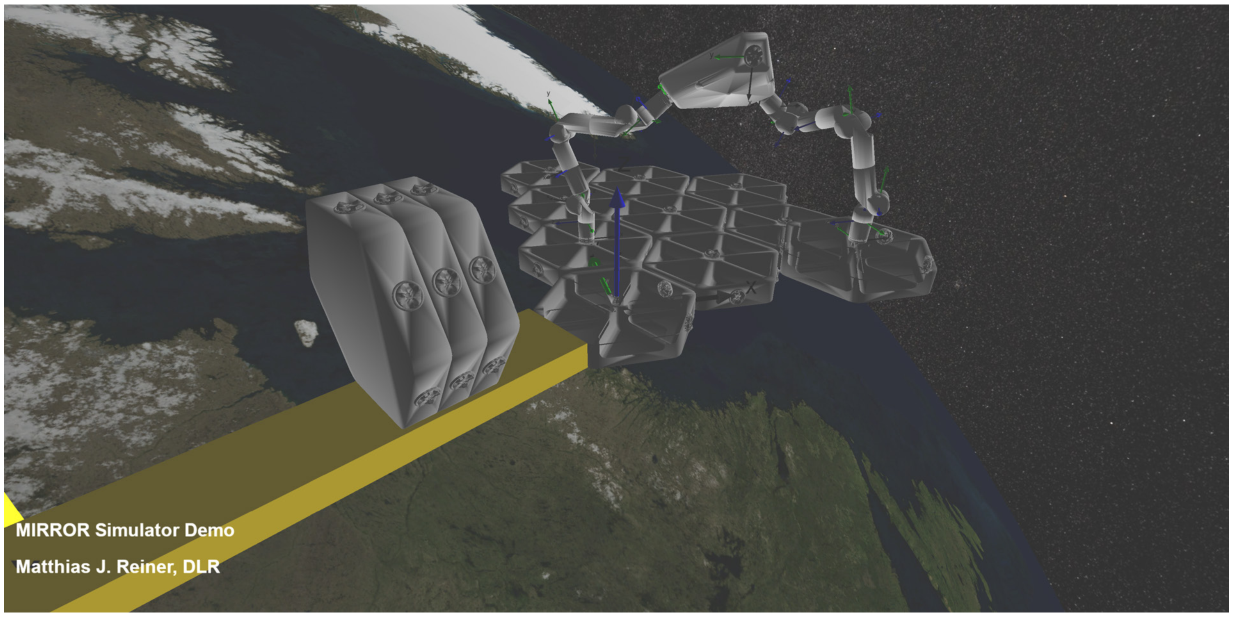


Figure 2. Visualization generated by the MIRROR Kinematic Dynamic & Graphical Simulator for an assembly scenario using the Multi-Arm Robot.

The model development and simulation of the scenarios with a focus on the Modelica aspects and unique design decisions will be described in the following sections. Since the models are quite complex only a brief overview can be given.

2 Model Development

The MOSAR Functional Engineering Simulator and the MIRROR Kinematic Dynamic & Graphical Simulator (both are abbreviated together as simulator, or individual as FES or KDG in the following) are built in the multi-physics modeling language Modelica to simulate the corresponding scenarios. The focus of the simulator is the walking robotic manipulator (WM and MAR) and the satellite platform with its modules and connectors (SIs, Hotdocks). The models were built up object-oriented, and allow to exchange individual components to generate different models for example to enable simulations with faster computational performance or more detailed models including additional dynamical effects.

The space environment of the MOSAR and MIRROR scenarios are implemented using the DLR SpaceSystems (Reiner and Bals 2014) and DLR Environment (Briese, Klöckner, and Reiner 2017) libraries, the robots working on these modules are modeled using the DLR Robot and RobotDynamics libraries (see (Reiner 2011) and (Tobias Bellmann, Seefried, and Thiele 2020)), the visualization is implemented directly in Modelica using the DLR Visualization library . A short summary of the content of these main libraries are listed in the following:

DLR SpaceSystems and Environment Libraries

The Modelica SpaceSystems library (SSL) has been developed at DLR over several years, to model space systems in a realistic space environment, ranging from satellites to launch vehicles, including their subsystems, components, and physical behavior, such as structural dynamics of solar panels and launcher stages. The SSL enables advanced controller design and verification, trajectory optimization, as well as development of path planning and other algorithms for new modular satellites, on-orbit servicing and reusable launcher concepts. It is a state-of-the-art enabling tool for future space dynamics and control activities. Various environmental effects on space systems can be represented with the co-developed environment models inside the DLR Environment library. In particular, an extendable and replaceable so-called World model is provided by the DLR Environment library. The World model defines coordinate systems, manages time and date, calculates sun and planet positions and provides state-of-the-art gravity models like EGM96. Optionally, models to calculate atmospheric conditions depending on the geodetic height of space vehicles, atmospheric drag, wind or other physical environmental effects can be activated within the analyses to provide a more realistic approximation of the relevant environmental conditions for space systems during all flight phases.

DLR Robots and RobotDynamics Libraries

The Modelica Robots and RobotDynamics libraries were designed to model serial kinematic robots. The libraries consist of components for the mechanical design of robots, including flexible elements and powertrains as well as models for different robot control structures. They were developed and refined over many years and are used in various projects. The libraries focus on the efficient and exchangeable implementation of robot kinematics and dynamics. They also provide algorithms to solve forward and inverse kinematics problems. In addition, the libraries provide tools for the visualization of robots.

DLR Visualization Library

The DLR Visualization library provides an advanced, model-integrated visualization tool for Modelica models. The visualizer elements are directly part of the Modelica model using mechanical connectors and the visualization is generated directly at runtime. The library contains visualizers for basic shapes, CAD files, flexible bodies, surfaces, textures, light, energy and mass flow visualizers, analogue instruments and weather effects. A virtual camera system can be used to define the point of view manually or controlled by simulation. For space applications with a large difference in distances, a logarithmic Z-buffer has been implemented to be able to simulate the environment with an exact scale. SimVis is the software tool which displays the output generated by the Visualization Library.

Modeling Approach

A main simulation challenge for both MOSAR and MIRROR is the appropriate modeling of the assembly and reconfiguration of the modules. In both cases, a standard module is removed from its original position at the spacecraft, attached to an SI at a robot end effector, and integrated at a new position. Each of these transitions causes discontinuous changes in the mass and inertia properties of the involved subsystems which has to be adequately considered in the description of the dynamic system and selected modeling technique dealing with re-configurable systems.

The preferred solution for systems with a limited total number of configurations and with a limited number of configuration switches is the preparation of independent models for each configuration which are run sequentially. At a configuration switch from A to B, the model end state of configuration A will be used as the initial state of configuration B. In the MOSAR and MIRROR scenarios, both the number of possible configurations and the number of configuration switches exceeds a reasonable number for predefined model architectures. For a moderate number of configurations, the preferred method is working with controllable constraints. If the condition for a configuration switch is fulfilled, e.g. a body A has reached its final plug-in pose relative to body B, an event is triggered that causes a change in the connector constraint settings. In this example the constraint will switch from "free motion" to "rigidly connected".

Mechanically, the constraint conditions are switching from zero forces/torques at plug and socket to identical position, velocity, and acceleration. The implementation is based on a method used at DLR-SR previously for rocket separation simulation (Acquatella and Reiner 2014) using force constraints with a variant of the Baumgarte stabilization. The basic idea is given in the simplified pseudo Modelica code in listing 1.

Listing 1. Simplified pseudo Modelica code to calculate switchable mechanical force constraint

```
// generalized position constraint
g_con = constraintForceAndTorque.frame_a.
    r_0 - constraintForceAndTorque.frame_b.
    r_0;
G_con = Frames.relativeRotation(
    constraintForceAndTorque.frame_a.R,
    constraintForceAndTorque.frame_b.R);

// generalized velocity constraint
g_con_dot = der(g_con);
G_con_dot = Frames.angularVelocity2(G_con);

// generalized acceleration constraint
g_con_ddot = der(g_con_dot);
G_con_ddot = der(G_con_dot);

// enable constraint switch
if not constrained then
 f_c = {0,0,0};
 tau_c = {0,0,0};
else
 g_con_ddot + 2*eta*g_con_dot + eta*eta*(
     g_con - pos_offset) = {0,0,0};
 G_con_ddot + 2*eta*G_con_dot
 + eta*eta*
 (Frames.Orientation.equalityConstraint(
 constraintForceAndTorque.frame_a.R,
     constraintForceAndTorque.frame_b.R) -
     angle_offset) = {0,0,0};
end if;
```

The code in listing 1 shows how the switchable connector is working: when the connector is open, the constraint force and torque are set to zero, so they can move freely. When the connector is closed, the force and torque between the two connector frames are computed such that the resulting relative position, velocity and acceleration are zero. The additional (Baumgarte) damping term (eta) is used to compensate numerical drift, caused by small integration errors. The parameter eta is used to define the stiffness and damping of the compensation.

However, extrapolating this technique to systems with a high number of switchable connectors would end up in huge stiff differential equation systems with an unacceptably high computational load. For a large number of potential configurations, a modeling technique was developed at the DLR during the MOSAR project that specifies only a general system architecture topology: the manipulator connected to the spacecraft with its end effectors using switchable constraints. Pick and place operations are then dynamically equivalent to adapting mass and inertia properties of the arrays to be assembled or the module stack to be reconfigured and of the modules at the robot end effectors.

The Tensor Body Concept

For this purpose, a new and innovative way to implement variable structure models was developed: individual configurations of the reflector or module stack are realized in the simulator by so-called tensor body models. These are, in essence, modifiable rigid bodies, which are consisting of individual sub-components and can change their inertia, mass and geometric shape. To clarify, since the word tensor is used very widely and differently in multiple fields, in this paper the term tensor is used to describe multidimensional arrays. These adaptable tensor bodies allow for activating and deactivating grid elements in terms of mass and inertia and also connectors depending on the actual operations. Thus, the simulator consists of a relatively simple model of the system dynamics and a logic component that adapts the tensor body model of the reflector or module stack at reconfiguration events as well as the corresponding tensor body located at the robot end effector. After each transition, the tensor bodies are re-computed based on the new configuration. This leads to an extreme reduction in the number of necessary states (depending on the number of modules) and improved numerical stability and computational speed. The properties of the components are dynamically re-calculated when the stack changes. The components of the body are defined via a tensor definition/syntax. They can also consider orientation and connectors of each individual component. Using the tensor body concept allows a simulation without restart/recompilation, because the number of states remains the same for the whole simulation duration. Reaction forces between the robot and platform/modules are fully considered. In addition, a special connection algorithm can check if an electrical connection can exist between the modules, considering the connectors of all involved components. It is implemented as a path finding search. The thermal balance is also computed in a similar way: a full grid of all possible module locations is dynamically updated when a reconfiguration occurs. This means the thermal capacity and thermal resistance of the connectors is updated depending on the current module type and the state of the connectors (including their current orientation). To make this possible, the thermal SI connectors are approximated by thermal resistors, which can switch from a very high resistance to a very low resistance value to simulate open or closed connectors. This results in a certain loss of accuracy, but leads to very fast simulation times and constant number of states for the thermal submodel of the tensor body. Since at an early mission state, most thermal data values are only rough estimates, this loss of accuracy can be accepted.

The current configuration of the modular satellite or mirror array is defined by the variable called shapeTensor with three dimensions for each spatial direction (tensor_nx, tensor_ny and tensor_nz). This maximum tensor size cannot be changed without a re-compilation of the Modelica model, however the content of shapeTensor can be modified during the simulation to account for the reconfiguration of modules.

Different types of modules have been implemented, they are given a number to differentiate them, while 0 stands for no module at that tensor location. The properties of these individual module types can be changed via the corresponding parameters (e.g. mass, inertia, thermal capacity, SI configuration etc.). The initial configuration of the arrangement of these modules is defined by a parameter shapeTensorInit.

Whenever the robot moves a module from A to B, the tensor body is re-calculated automatically (by triggering a recomputeTensor event) internally and the required connectors are automatically opened and closed as needed to realize the new configuration. In addition, tensor body elements are also located at each SI of the robot, such that the handling of all types of modules is possible. If no component is connected to a robot SI, the tensor body at the SI is just a dummy mass with a very small inertia and mass (called epsMass) to avoid a division by zero and to enable a constant number of states.

To be able to identify which exact module is located at each position within the tensor grid, each existing component is also given a unique ID in addition to its general module type number.

Each individual module within the tensor also has an individual orientation (relative to the tensor body base frame), and up to seven connectors for the hexagonal elements (and six for the cubic modules).

The following simplified pseudo Modelica code listing 2 shows the base algorithm for the tensor body, where some details were omitted for brevity.

Listing 2. Simplified pseudo Modelica code for the main algorithm of the tensor body

```
when {initial(),edge(recomputeTensor)} then
 //check possible connections with path
     finding algorithm
 (poweredTensor,connectionTensor,...) :=
 Components.Electrical.
     VarTensorBodyConnectionAlgorithm(..)
 //initialize mechanical properties
 m := 0;
 r_CM := {0,0,0};
 //iterate over spatial tensor dimensions
 for ii_x in 1:tensor_nx loop
  for ii_y in 1:tensor_ny loop
   for ii_z in 1:tensor_nz loop
   // check if element exists
   if shapeTensor[ii_x, ii_y, ii_z] > 0
       then
    //add center of mass (not normalized
        yet)
    //massPerEle is a vector which contains
         the individual mass of the
        specific module type
    //r_ele is a tensor with relative
        position vectors for each possible
        element location in the 3D tensor
        grid.
    r_CM := r_CM + massPerEle[shapeTensor[
        ii_x, ii_y, ii_z]]*r_ele[ii_x, ii_y
        , ii_z, :];
      //add total mass
      m := m + massPerEle[shapeTensor[ii_x,
        ii_y, ii_z]];
    end if;
    end for;
   end for;
  end for;
  //divide by total mass, avoid div/0
  if m < epsMass then
   m := epsMass;
  end if;
  r_CM := r_CM/m;
  //compute inertia tensor w.r.t. previously
       computed CM
  //init
  I := zeros(3, 3);
  for ii_x in 1:tensor_nx loop
   for ii_y in 1:tensor_ny loop
    for ii_z in 1:tensor_nz loop
    //inertia solid cuboid with Steiner term
    // check if element exists
     if shapeTensor[ii_x, ii_y, ii_z] > 0
         then
      // current rotation for the element
      R_ele := rotMatTensor[ii_x, ii_y, ii_z
          , :, :];
      //inertia for specific element
      I_ele := inertiaPerEle[shapeTensor[
          ii_x, ii_y, ii_z],:,:];
      I := I + R_ele*I_ele*transpose(R_ele);
      // helper vars -> distance to CM
      rx := r_ele[ii_x, ii_y, ii_z, 1] -
          r_CM[1];
      ry := r_ele[ii_x, ii_y, ii_z, 2] -
          r_CM[2];
      rz := r_ele[ii_x, ii_y, ii_z, 3] -
          r_CM[3];
      // Steiner tensor
      I := I + massPerEle[shapeTensor[ii_x,
          ii_y, ii_z]]*[ry^2 + rz^2,-rx*ry,-
          rx*rz; -rx*ry,rz^2 + rx^2,-ry*rz;
          -rx*rz,-ry*rz,rx^2 + ry^2];
     end if;
    end for;
   end for;
  end for;
  //avoid div/0
  if I[1, 1] + I[2, 2] + I[3, 3] < epsMass
      then
   I[1, 1] := epsMass;
   I[2, 2] := epsMass;
   I[3, 3] := epsMass;
  end if;
   //update of heat grid model and power
       state of SI connections omitted
  (...)
end when;
```

The code listing 2 shows the main mechanical algorithm to re-compute a tensor body when it is changed. A logical algorithm in the model (not shown) triggers the recomputeTensor event and checks which tensor bodies are involved and how they should be changed. For a pick & place operation this leads to the re-computation of the properties of the main tensor body, as well as the corresponding tensor body at an end effector on one of the robot arms. For each involved tensor body, first the new Center of Mass (CM) is computed based on the change of the stack of modules or array elements represented in the tensor variable shapeTensor, which is modified by the logical algorithm. Using the newly computed CM as the new center, an updated inertia tensor is computed for the tensor body which considers all individual orientations of the individual tensor body elements as well as the individual inertia tensors for each module. Since after a pick & place operation, no module can be left at an end effector of a robot, a very small (dummy) mass and inertia are used to approximate this. Since all involved tensor bodies are updated and recalculated with the same event at the same time, no discontinuities can occur, and the total mass always stays constant.

Modeling Overview

The base coordinate system for the KDG and FES simulation is an Earth Centered Inertial (ECI) system. The orbit of the robot and satellite or antenna array can be defined by using appropriate start position and velocity vectors, together with the simulation date. The date is used to compute the current rotation of the earth and the position of the moon and sun. Other planets are neglected because only the LEO (Low Earth Orbit) or GEO (Geosynchronous Equatorial Orbit) are considered here. This allows for a very generic simulation of nearly every LEO and GEO orbit and can also be used to consider lighting and heating conditions from the resulting sun position. For the simulation it is considered, that the servicer spacecraft and client satellite or platform are already docked and that the possible locations of all modules are known beforehand. For the cubic shape of the modules this is quite straight forward and leads to three-dimensional tensor shape, for the hexagonal modules of the MIRROR project, the possible locations are stored in a table (called r_ele in listing 2) which is used within the tensor body calculation.

The spacecraft modules (SM): The spacecraft modules are modeled as tensor bodies. It is generally a full SM stack with individually and dynamically activatable and de-activatable SM for implementing the modular reconfiguration, depending in the SIs' respective lock status. The SMs can have individual inertia properties, depending on their respective internal equipment. Each cubic module has up to 6 connectors (around all the sides). These can be used to grasp the modules, and to walk the robot over them. For the hexagonal modules a similar approach was implemented in the KDG simulator where each module can have up to seven SI connectors along the sides of the hexagonal modules and one on top. The simplified models of the electric and thermal domain are integrated in the tensor models.

The robot models consist of kinematic and dynamic models. For MOSAR they also include powertrain and sensor models. For the detailed model variants also the joint torque and position controllers are implemented very close to their actual implementation using the DLR Robot-Dynamics and Modelica Standard library. The robots are equipped with SI interfaces to enable walking and module grasping capabilities. The SI modules are modeled as switchable force/torque constraints, as described previously. The robot model includes the joint motor and friction as well as joint nonlinear elasticity for the complex simulator variants. For the simulation the full dynamic coupling of the robot's interaction with the satellite is considered.

The simulators contain many additional features and aspects which can only be briefly mentioned and referenced here. The following list gives an overview over the main features of the simulator:

- Dynamic satellite platform and modules based on the tensor body approach: After each transition, the tensor bodies are re-computed depending on the new configuration. This allows simulation without restart/recompilation for the complete simulation scenario (number of states remains constant). Reaction forces between the robot and platform are fully considered and the heat flow between components can be modeled. A path finding connection algorithm can check if an electrical connection to a power source module exists.

- Visualization of components and important properties.

- Detailed WM and MAR robot model using data from Unified Robot Description Format (URDF). For the WM also powertrain models with motor friction brake, nonlinear flexible joint models for gearbox and nonlinear friction as well as joint position and torque controllers are included.

- Variable module design with individual mass and inertia as well as individual SM configuration including heat and power properties.

- Detailed orbit model: orbital dynamics, including sun and moon.

- Simplified surface heat model including solar, albedo, deep space and planet infrared radiation. The models consider the position of the sun, moon and earth for the shadows as well as the spacecraft attitude. The implementation is based on (Posielek 2018) but is extended to be compatible with the tensor body concept.

- Surface contact model with friction: Contact forces between robot TCPs, modules and platform are considered and are dynamically updated depending on the shape of the tensor body. The implementation is based on previous work at DLR-SR, see (Reiser 2021).

- To stabilize the platforms and satellites, when the robots are moving, a Quaternion attitude controller is implemented, as a simplified Guidance, navigation and control (GNC) system.

- To allow the simulator to obtain signals from other software components and high-level control systems a UDP interface was implemented. This allows for the communication with an external high-level WM controller (developed by the Institute of Robotics and Mechatronics DLR-RM). The implementation is based on (Thiele et al. 2017).

3 Demonstrator Overview

Since the MIRROR project is still ongoing, this section will focus on the end results for the MOSAR project incorporating the FES. The FES is only one small part of the overall demonstrator setup of the MOSAR project, which involved many project partners: SpaceApps, GMV, MAG-SOAR, Thales Alenia Space (France and UK), SITAEL, Elidiss Technologies, University of Strathclyde, Glasgow and DLR (SR and RM) (see (Letier, Yang, et al. 2019)). Figure 3 shows pictures of some results of the MOSAR project.

The demonstrator setup consists of a physical assembly of prototype modules and the WM as well as multiple software components. The WM was designed to be able to work under the earth's gravity conditions. For the much larger MAR of the MIRROR project a weight compensation construction is being developed for the demonstrator on earth.

For the final demonstration of the MOSAR project multiple scenarios were developed and executed. For each scenario the starting and desired end configurations of the module stacks and the WM pose were defined with the help of the design tool, which was based on the FES, and a static simulation analysis of the configurations, to check the module SI connector compatibility and power source configuration.

A planning algorithm (developed by GMV) uses this configuration information to plan a sequence of operations for the WM and the SI connectors to move and reorient the modules collision free from the starting to desired end configuration. Because of the limited reach of the WM, this can also include relocating the WM between different steps of the operational plan.

The planning module communicates with a high-level robot controller developed by DLR-RM. This controller can communicate with the FES as well as the real hardware setup, since the interfaces were defined analogously. To ensure the safe operation of the WM and to validate the operation plan. The complete sequence is simulated using the FES first, before moving the real hardware.

The high-level WM controller can move the robot both in position mode (Cartesian and joint command possible), as well as in a compliance mode, using the robots joint torque sensors. For the simulation the high-level WM

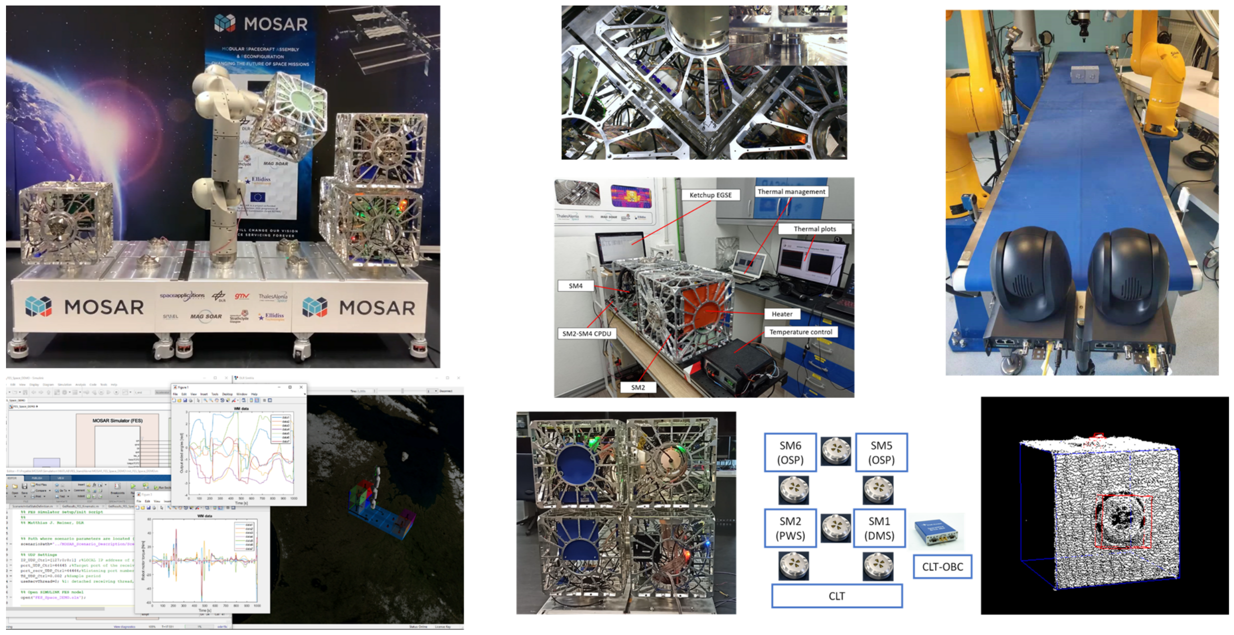


Figure 3. The mosaic overview taken from an internal MOSAR report shows some results obtained within the MOSAR project from the involved project partners SpaceApps, GMV, MAGSOAR, Thales Alenia Space (France and UK), SITAEL, Elidiss Technologies, University of Strathclyde, Glasgow and DLR (SR and RM). The top left shows the physical demonstrator setup with the WM relocating an SM. The top middle gives a closer look on the SIs and a thermal testing setup. The top right shows a setup for the visual inspection of the modules. The lower left shows a screenshot of the MATLAB interface of the FES with plotted simulation results. In the lower middle a closer view of an SM configuration and its corresponding software setup can be seen. The lower right corner shows the result of a camera-based damage inspection of an SM.

controller sends the desired joint position or torque commands as well as SI commands to the FES using the UDP interface. The low-level joint axis controllers for both the torque and position control mode are implemented within the FES. The FES uses this input data to compute the resulting motion and sends simulated sensor information back to the WM controller. The communication is synchronized by using a blocking UDP implementation, which waits for the corresponding resulting data package before executing the next step. The high-level WM controller also communicates with the planning module, to enable a re-planning in case something goes wrong or a plan is not feasible and needs to be re-planned.

After a completed simulation run, the resulting simulation data generated by the FES can be analyzed by an expert team before starting the same procedure with the real hardware setup to ensure a safe operation. For example, joint limitations and safety distances to objects can be checked there.

Since most project partners did not use Modelica directly, the Modelica model used within the FES was exported as an S-function, using a tool provided by Dassault Systems Dymola. This S-function can then be used and configured in MATLAB/Simulink. The visualization can also work together with the generated S-functions.

Since the definition of the scenarios and parameters is quite complex, the parameter settings, as well as the output processing were done using MATLAB scripts (.m files) which also enables the use of algorithms to simplify the definition and output generation.

4 Conclusions and outlook

The simulation of the complex scenarios in Modelica with the FES was possible because of the newly developed features, such as the tensor body concept for the structural variable SM stack, as well as the integration of many previously developed Modelica libraries at DLR-SR.

Using the complete demonstrator setup and all developed software components from all project partners, all demonstration scenarios could be successfully performed at the end of the MOSAR project.

The concept was used in the MIRROR project which enabled a very fast development time for the simulator there, and with small adaptations could also be used for many similar applications both on-orbit as well as on earth (e.g. building construction).

However, the tensor body concept and switchable force constraints also lead to a significant increase in the overall model complexity (especially the required logical parts) and some approximations and simplifications are necessary and restrict what is possible to simulate.

An extension of the Modelica language (and simulation tools) to directly handle structural variable systems or new developments within similar languages such as the modeling language Modia (Elmqvist et al. 2021) could improve this aspect in the future and would enable the direct implementation of structural variable systems in a more classical object-oriented way.

Acknowledgements

The work presented here would not have been possible without the great support of colleges at the DLR-SR who provided Modelica libraries that were part of the simulation as well as the good collaboration within the MOSAR and MIRROR project with the involved partners.

The MOSAR part of this work was funded by the European Commission Horizon 2020 Space Strategic Research Clusters on Space Robotics and Electric Propulsion programme under grant number 821966 (MOSAR: Modular Spacecraft Assembly and Reconfiguration). The following companies and institutions were involved in the MOSAR project: SpaceApps, GMV, MAGSOAR, Thales Alenia Space (France and UK), SITAEL, Elidiss Technologies, University of Strathclyde, Glasgow and DLR (SR and RM).

The MIRROR part of this work is still ongoing and funded by the European Space Agency (ESA) in the framework of the Technology Research Program (contract No. 4000132220/20/NL/RA) entitled Multi-arm Installation Robot for Readying ORUS and Reflectors (MIRROR). The following companies and institutions were involved in the MIRROR project: SpaceApps, Thales Alenia Space, LKE, Leonardo, Frentech Aerospace and DLR (SR and RM).

References

Acquatella, P. and M. Reiner (2014). “Modelica Stage Separation Dynamics Modeling for End-to-End Launch Vehicle Trajectory Simulations”. In: *Proceedings of the 10th International Modelica Conference*, pp. 589–598.

Bellmann, T. (2009). “Interactive Simulations and advanced Visualization with Modelica”. In: *Proceedings of the 7th Modelica Conference*, pp. 541–550. URL: https://www.modelica.org/publications.

Bellmann, Tobias, Andreas Seefried, and Bernhard Thiele (2020). “The DLR Robots library - Using replaceable packages to simulate various serial robots”. In: *Asian Modelica Conference 2020*. Linköping Electronic Conference Proceedings 174. Linköping University Press, pp. 153–161. URL: https://elib.dlr.de/138327/.

Briese, L., A. Klöckner, and M. Reiner (2017). “The DLR Environment Library for Multi-Disciplinary Aerospace Applications”. In: *12th International Modelica Conference*. DOI: 10.3384/ecp17132929.

Deremetz, Mathieu, Gerhard Grunwald, et al. (2021-12). “Concept of Operations and Preliminary Design of a Modular Multi-Arm Robot using Standard Interconnects for On-Orbit Large Assembly”. In: *72nd International Astronautical Congress (IAC),* URL: https://elib.dlr.de/147153/.

Deremetz, Mathieu, Pierre Letier, et al. (2020-10). “MOSAR-WM: A relocatable robotic arm demonstrator for future on-orbit applications”. In: *International Astronautical Congress, IAC 2020*. IAF. URL: https://elib.dlr.de/139962/.

Elmqvist, Hilding et al. (2021-09). “Modia - Equation Based Modeling and Domain Specific Algorithms”. In: *14th International Modelica Conference*. Ed. by Martin Sjölund et al. Linköping Electronic Conference Proceedings 181. Linköping University Electronic Press, pp. 73–86. URL: https://elib.dlr.de/144872/.

Hellerer, Matthias, Tobias Bellmann, and Florian Schlegel (2014-03). “The DLR Visualization Library - Recent development and applications”. In: *The 10th International Modelica Conference 2014*. Linköping Electronic Conference Proceedings. LiU Electronic Press, pp. 899–911. URL: https://elib.dlr.de/92153/.

Kümper, Sebastian, Matthias Hellerer, and Tobias Bellmann (2021-09). “DLR Visualization 2 Library - Real-Time Graphical Environments for Virtual Commissioning”. In: *14th Modelica Conference*. Ed. by Martin Sjölund et al. Modelica Association and Linköping University Electronic Press, pp. 197–204. URL: https://elib.dlr.de/144780/.

Letier, Pierre, Torsten Siedel, et al. (2020-10). “HOTDOCK: Design and Validation of a New Generation of Standard Robotic Interface for On-Orbit Servicing”. In: *International Astronautical Congress, IAC 2020*. IAF. URL: https://elib.dlr.de/139963/.

Letier, Pierre, Xiu Yang, et al. (2019-05). “MOSAR: Modular Spacecraft Assembly and Reconfiguration Demonstrator”. In: *15th Symposium on Advanced Space Technologies in Robotics and Automation*. URL: https://elib.dlr.de/129392/.

Posielek, Tobias (2018). “A Modelica Library for Spacecraft Thermal Analysis”. In: *The American Modelica Conference 2018*. URL: https://elib.dlr.de/123229/.

Reiner, M. (2011). “Modellierung und Steuerung eines strukturelastischen Roboters”. PhD thesis. Technische Universität München.

Reiner, M. and J. Bals (2014). “Nonlinear inverse models for the control of satellites with flexible structures”. In: *Proceedings of the 10th International Modelica Conference*, pp. 577–587.

Reiser, Robert (2021-09). “Object Manipulation and Assembly in Modelica”. In: *14th International Modelica Conference*. Ed. by Martin Sjölund et al. Modelica Association and Linköping University Electronic Press, pp. 433–441. URL: https://elib.dlr.de/144864/.

Thiele, Bernhard et al. (2017-05). “Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library”. In: *12th International Modelica Conference*. Ed. by Jiri Kofranek and Francesco Casella. Linköping University Electronic Press, 2017, pp. 713–723. URL: https://elib.dlr.de/118601/.

Extending a Multicopter Analysis Tool using Modelica and FMI for Integrated eVTOL Aerodynamic and Electrical Drivetrain Design

Meaghan Podlaski[1] Luigi Vanfretti[1] Robert Niemiec[2] Farhan Gandhi[2]

[1]Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY, USA
{podlam,vanfrl}@rpi.edu
[2]Center for Mobility with Vertical Lift (MOVE), Rensselaer Polytechnic Institute, Troy, NY, USA
{niemir2,gandhf}@rpi.edu

Abstract

This paper describes how the Functional Mock-up Interface (FMI) standard for model-based systems engineering can be used to expand the capabilities of aerodynamic multicopter analysis tools to perform integrated design of electric vertical take-off and landing (eVTOL) systems. The proposed eVTOL system, which consists of a drivetrain and battery, was developed using Modelica and exported to MATLAB/SIMULINK using model exchange to interact with a domain-specific tool specializing in calculating the aerodynamics of the aircraft. This shows the value of FMI to extend the capabilities of tools for multicopter aerodynamic analysis to design eVTOL using integrated multi-domain system simulation.

Keywords: eVTOL, multi-domain modeling, FMI

Glossary

BLDCL	Brushless DC Drives Library
eVTOL	Electric Vertical Take-Off and Landing
FMI	Functional mock-up interface
FMIT	FMI Toolbox for MATLAB and SIMULINK
FMU	Functional mock-up unit
PWM	Pulse width modulation
RMAC	Rensselaer Multicopter Analysis Code

1 Introduction

1.1 Motivation

Distributed electric propulsion has enabled the development of electric vertical take-off and landing (eVTOL) systems, such as NASA's Advanced Air Mobility (NASA 2020 (retrieved Nov 3, 2020)) and Uber Elevate (Holden and Goel 2016). The design of such systems requires to engineer sub-systems of multiple engineering domains, where simulation studies of each sub-system can provide insight on which components and designs provide the greatest benefit prior to building a physical prototype. However, specialized design tools tend to focus on a specific domain only, which creates difficulties for integrated system desing.

The development of distributed electric propulsion systems would greatly benefit from well-defined multi-engineering models at various levels of modeling fidelity to understand system behavior. However, expanding domain specific tools to encompass all domains poses a tremendous development challenge. On the other hand, the Functional Mock-Up Interface (FMI) standard can enable the interaction of models that do not exist in domain specific tools, using commercial as well as open-source Modelica models, to further expand their capabilities and understand the overall integrated system. This would only require the domain specific tool to implement the FMI's import specification, providing a faster route to expand the capabilities of existing domain specific tools. The biggest benefit for this approach to modeling is that it allows for collaboration with researchers and developers that are not familiar with Modelica but have created domain specific tools, just by adding support to the FMI Standard to their software. This allows us to fully integrate tools created for previous research and development, enriching simulation studies with relatively low effort.

In this paper, the Rensselaer Multicopter Analysis Code (RMAC) (Niemiec and Gandhi 2019) developed within the MATLAB/SIMULINK environment, is extended to support the FMI standard using the FMI Toolbox (FMIT)(Modelon 2018; Henningson, Akesson, and Tummescheit n.d.). This allows to import electrified drivetrain models developed in Modelica, which once imported into RMAC, can be used for integrated analysis of eVTOL vehicles. To illustrate the new capabilities of RMAC, its aerodynamic vehicle model is coupled with a electrified drivetrain modeled in Modelica and exported as an FMU to MATLAB/SIMULINK to study the interaction between the the aerodynamics and electrified drivetrain. The RMAC tool contains mathematical models for the aircraft rotor dynamics, which are coupled to the FMU, so that the drivetrain can be studied with accurate aerodynamics used within the multicopter domain. The use of the FMI standard creates a flexible environment that can be easily interfaced with RMAC's code, expanding analysis capability for multi-domain studies.

1.2 Related Works

The development of eVTOL systems has been of interest recently, especially focused on applications to Urban Air Mobility for passenger transport operations. Many of these systems, such as Uber Elevate (Holden and Goel 2016), do not have large-scale physical prototypes, making model-based systems engineering an attractive approach for development and analysis of new architectures.

Novel electrified aircraft concepts, such as fixed-wing aircraft, have been studied using Modelica. The More Electric Aircraft Systems (MOET) project utilized Modelica to develop and study novel aircraft concepts to understand system behavior prior to building physical prototypes (Bals et al. 2009). In addition, distributed electric propulsion concepts have been modeled using Modelica to better study the electrical architecture as other disciplines improve their specific components, for example electrical energy storage devices increasing capacity (Zhou et al. 2018). However, these projects have not coupled their electrical drivetrain models to other programs for integrated multi-domain system design, which could be beneficial in designing the entire vehicle.

A unique example of multi-domain system design is presented in (Velden and Casalino 2021), where multiple tools have been integrated to conduct studies on eVTOL systems for flight and noise assessment using multi-fidelity models. This study shows how FMI can be applied to utilize the features of other tools in the Dassault 3DEXPERIENCE suite for analysis of the eVTOL system. However, the electrified drivetrain models were not included in the system design.

1.3 Paper Contributions

This paper contributes a framework for eVTOL design through integrated multi-domain model development where the electrified drivetrain can be defined with multiple levels of modeling fidelity using Modelica. These models are coupled to a domain specific multicopter aerodynamics tool, RMAC, developed in the MATLAB/SIMULINK environment using the FMI Toolbox to study the eVTOL drivetrain dynamics. This application highlights the benefits expanding the capabilities of pre-exisitng tools by incorporating models developed using Modelica, which is possible thanks to the FMI standard.

1.4 Paper Organization

The paper is organized as follows. Section 2 outlines the development of the drivetrain models using Modelica. Section 3 discusses FMU development and interfacing with the RMAC toolbox in MATLAB/SIMULINK for integrated multi-domain dynamic simulation. Section 4 shows results of the drivetrain studied at various levels of modeling fidelity. Section 5 describes the conclusions of this work.

2 eVTOL Model Development

The aircraft modeled in this work is a 300 lb quadcopter used in (Walter et al. 2020). The rotors are assumed to be linearly twisted and tapered and have a 10% R tip clearance. The motor parameters are based on the Hacker Q150-45-4 (*Hacker Q150-45-4 Series Datasheet* 2021). The drivetrain models are described in further detail with their behavior validated in (Podlaski et al. 2021).

The aircraft was configured using two different power system architectures: (1) with a centralized battery (one battery powering all four drivetrains) and (2) with individual batteries powering each of the drivetrains (four batteries in total). These two architectures allow to study the performance of the battery and electrical system configuration on the eVTOL system aerodynamics. The schematic of the system with a centralized battery is shown in Figure 1, and the schematic of the system with a distributed battery is shown in Figure 2. Each drivetrain has a speed input to determine the duty cycle of the inverter and a torque that is applied to the machine's rotor. The torque and speed of each motor is used as an output to adjust the controller as necessary and interact with the RMAC rotor aerodynamics.

2.1 Drivetrain Model

The drivetrain consists of four components: a controller, pulse width modulation (PWM) of the converter, a DC/DC converter, and a DC machine. The component models are from Dassault System's Brushless DC Drives (BLDCL) (Dassault Systemes 2022) and the Modelica Standard Library (MSL). The drivetrain was developed in Modelica using the Dymola software, and is shown in Figure 3. The system consists of multiple domains, with the electrical, mechanical, and control domains represented.

The components of the drivetrain system in Figure 3 are labeled as follows:

A. FMU inputs
B. FMU outputs
C. Controller (`replaceable` model)
D. Modulation method (`replaceable` model)
E. Inverter (`replaceable` model)
F. Machine (`replaceable` model)
G. Electrical connection to battery
H. Rotor inertia

The controller, modulation method, inverter, and machine models are all modeled as `replaceable` components, meaning that components with different levels of detail can be easily replaced using different model *variants* for various dynamic simulation studies. For example, the machine was modeled with different losses and dynamic behaviors included, resulting in four different machine models to be considered in the studies. By using `replaceable` models, these cases can be easily changed to observe different dynamic behaviors in the model.

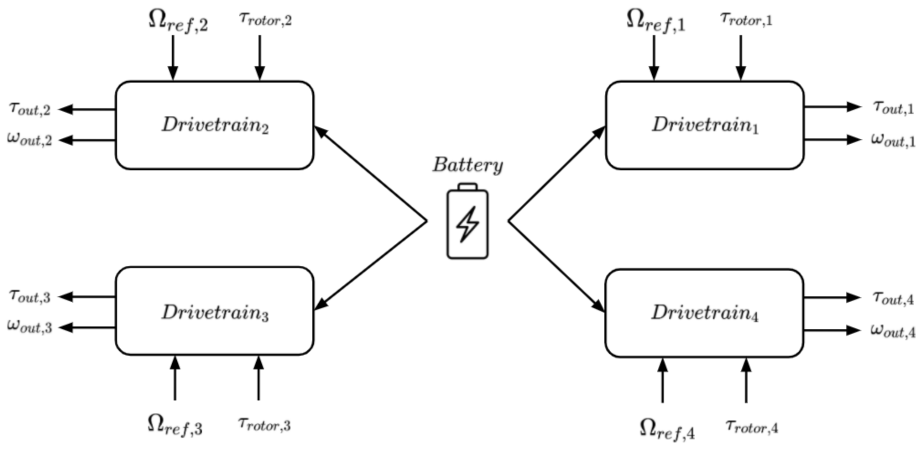


Figure 1. Multi-rotor aircraft model with a centralized battery.

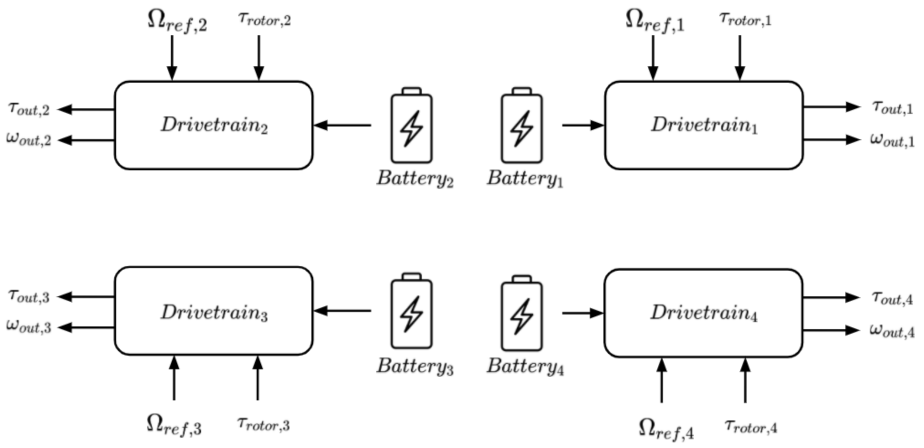


Figure 2. Multi-rotor aircraft model with a distributed battery.

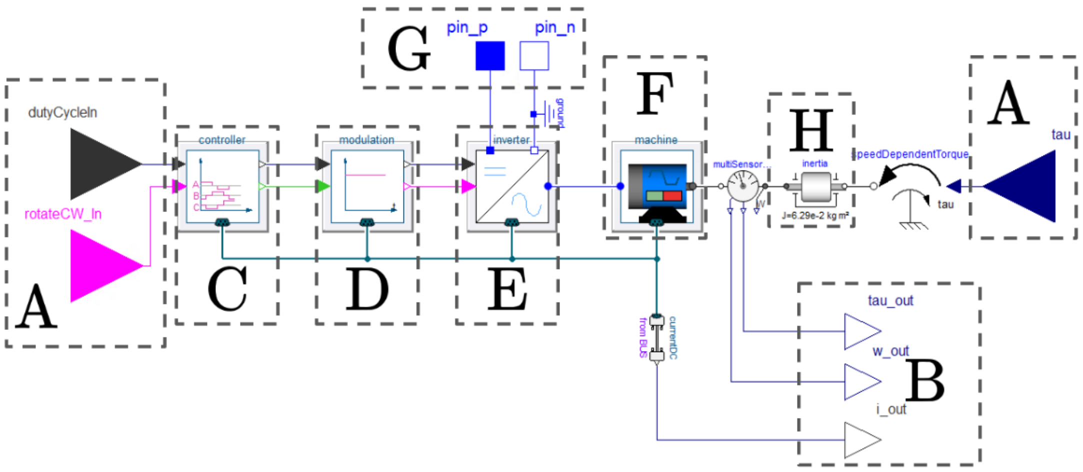


Figure 3. Drivetrain in Modelica using BLDCL and MSL.

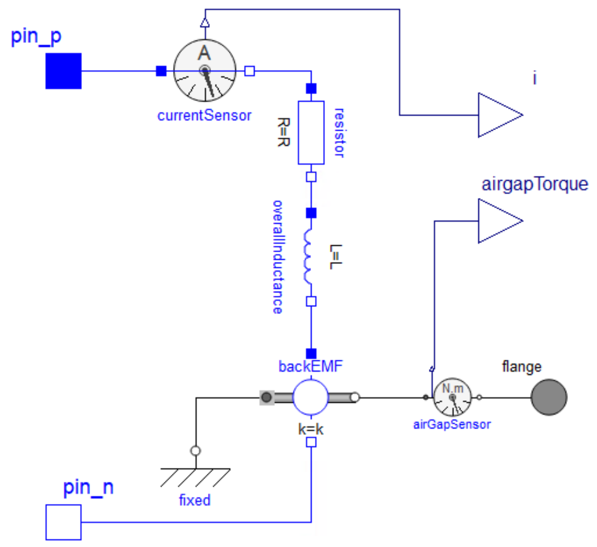


Figure 4. Implemented simplified variant of the motor model.

2.1.1 Machine Models

The machine model is shown as block F in Figure 3, which can be modeled at multiple levels of modeling fidelity. A simple motor model was developed as shown in Figure 4. This model is typically used to represent entire electrified power trains in the eVTOL community (Podlaski et al. 2021). While useful for preliminary studies, it limits the ability to perform integrated design of both aerodynamic and electrical sub-systems concurrently. In contrast, the BLDCL (Dassault Systemes 2022) contains machine models with averaged and trapezoidal back-EMF (see Figure 5, which enables studies to consider various non-ideal behavior and the electrical switching effects from the inverter. The model's architecture allows to use all three variants for different analysis, with low effort, for example, the averaged model helps to better represent frictional losses while the trapezoidal model allows to capture electrical and mechanical heat of the machine.

The speed of the motor in Figure 4 is calculated by Equation 1, where I is the rotor inertia, and the right-hand side of the equation is the net moment applied to the drivetrain. The motor torque is proportional to the current as calculated in Equation 2. Equation 3 gives the current drawn by the motor as a function of the resistance, inductance, and voltage applied to the motor. The motor can also be further simplified by setting the inductance $L = 0$, which eliminates any current delay in the electrical dynamics.

$$I\frac{d\Omega}{dt} = Q_{\text{motor}} - Q_{\text{aero}} \tag{1}$$

$$Q_{\text{motor}} = K_e i \tag{2}$$

$$L\frac{di}{dt} = V - Ri - K_e\Omega \tag{3}$$

The trapezoidal motor is the most complex model considered in the study, which is shown in Figure 5. The trapezoidal motor model uses a three phase input connector, as designated by `plug_sn` and `plug_sp`. This is due to the three-phase input needed to produce the trapezoidal back-EMF waveform that is produced by the switching of the inverter.

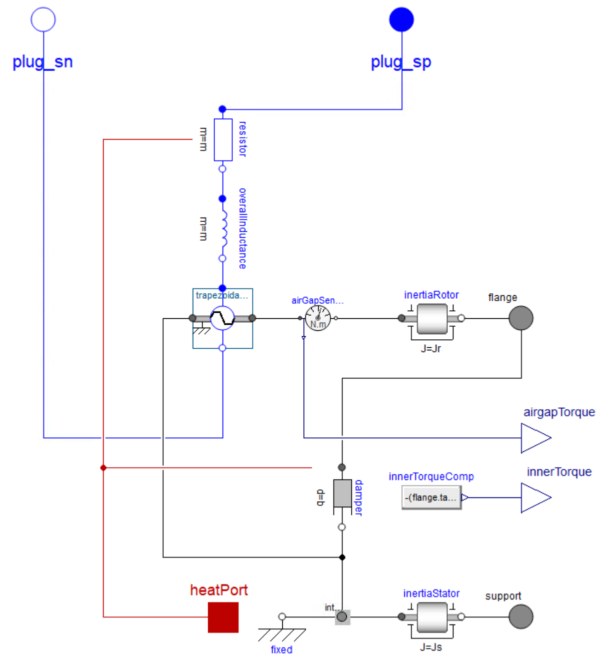


Figure 5. Trapezoidal variant of the machine model from the BLDCL

2.1.2 Controller, Inverter, and Modulation Models

The selection of the machine model defines which controller, inverter, and modulation models are used. When the averaged back-EMF and simplified motor models like the one in Figure 4 are used, the inverter model is an ideal buck DC-DC converter with a feed-through controller and modulation method as the duty cycle is a function of the actual and desired speed of the motor. The ideal buck converter steps down the battery voltage to the motor voltage as a function of the duty cycle as denoted by Equations 4 and 5.

$$V_{motor} = V_{battery} * \text{dutyCycle} \tag{4}$$

$$i_{battery} = -i_{motor} * \text{dutyCycle} \tag{5}$$

In the case of the trapezoidal motor, more complex power electronics converters, controllers, and modulation methods must be considered. The inverter now includes diodes and switches to created a three-phase signal to be applied to the motor, resulting in a trapezoidal back-EMF. The inverter is shown in Figure 6, which includes the same buck converter calculations in Equations 4 and 5. A three-phase star connection from the ideal buck component creates the three-phase voltage applied to the switches, `upperSwitch` and `lowerSwitch`, and

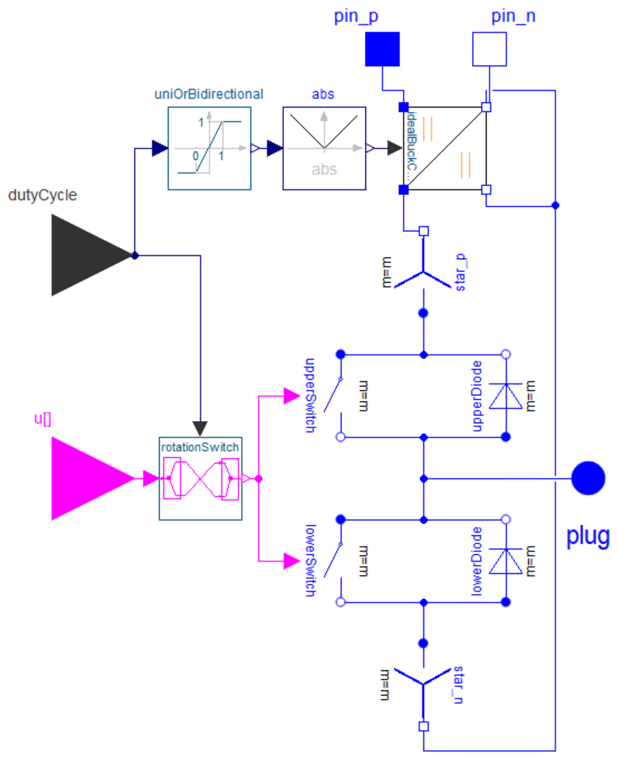


Figure 6. Inverter model with switching components from the BLDCL.

diodes, `upperDiode` and `lowerDiode`. Input `u[]` controls the switching on and off of the `upperSwitch` and `lowerSwitch` to produce a trapezoidal signal to apply to the motor via three-phase output `plug`.

The input `u[]` in Figure 6 comes from the a six-step control command generated by Hall sensor outputs to control each of the half-bridges in the inverter model. This is exemplified in Figure 7, where for the instant denoted by the black line the PWM signals denote that the upper switch for the third phase and the lower switch for the first phase are closed. This means that V_a is connected directly to ground, V_b is falling linearly between $V_{battery}$ and ground, and V_c is equal to $V_{battery}$. The `Modulation` block converts the `boolean` states from the six-step controller into switching signals to control the half-bridges, effectively linking the different domains in the model.

2.2 Battery Model

The battery model is a look-up table-based open-circuit voltage (OCV) model from the Dassault Systems Battery Library (Dassault Systems 2022). The model considers both electrical and thermal behavior, and utilizes data collected from experiments to populate look-up tables. These tables are then used to determine the parameters of the battery's electrical components for various operating conditions. The electrical schematic of the OCV battery is shown in Figure 8, where each cell of the battery is powered by an ideal voltage source. The values of the resistors and capacitors in the circuit in Figure 8 are determined from the operational state and values from the look-up tables.

The battery model in Dymola is shown in Figure 9 and shows both the thermal and electrical domains modeled. The components are modeled as follows:

A. Electrical connections to the drivetrain
B. Thermal housing model and connection to outside thermal models
C. Electrical scaling component
D. Thermal scaling component
E. Battery cell electrical model
F. Data connections for analysis of the battery

The battery cell model in Figure 9, block E consists of the electrical circuit shown in Figure 8. The `electricalScaling` in block C of Figure 9 scales the number of cells by m cells in parallel and n cells in series, as shown in Figure 8. Every cell produces a voltage given by Equation 6. The impedance in each cell is given by Equation 7, where the values of $R1$, $C1$, $R2$, and $C2$ are determined from the look-up tables as a function of battery state of charge and operating temperature.

$$V_{battery,ij} = OCV_{ij} - Z_{battery,ij} i_{ij} \tag{6}$$

$$Z_{battery,ij} = (R1_{ij}||C1_{ij}) + (R2_{ij}||C2_{ij}) + R_{ij} \tag{7}$$

For the vehicle studied in this paper, the battery in a centralized configuration in Figure 1 has 15 cells in series and 20 cells in parallel for a 60 V with a capacity of 43 Ah. In the distributed battery configuration in Figure 2, the capacity of each battery is a quarter of the centralized battery: 15 cells in series and 5 cells in parallel. This results in a 60 V battery with a 10.75 Ah capacity.

3 Coupling FMUs to RMAC

The drivetrain model in Figure 3 was coupled to the battery model in Figure 9, then exported as an FMU using the model exchange specification supported in Dymola. The FMU is imported into Simulink through a FMU for model exchange provided by the FMI toolbox (Modelon 2018) to be simulated with RMAC. The interfaces between the FMU and RMAC are shown in Figure 10. The FMU uses an input of a desired speed command, rotor torque, and a rotation direction of the motor (clockwise/counter-clockwise). The speed command is derived from the vehicle's attitude and heave control; the rotor torque is produced from RMAC's aerodynamic model. The FMU outputs the speed of the motor to interact with RMAC, as well as machine current and torque for monitoring and analysis. The motor speed output from the FMU is used to model the aerodynamic forces and moments about the rotor hub. These forces and moments are then coupled with the vehicle dynamics model.

Because the model in Figure 3 requires the duty cycle as an input, RMAC must also provide a controller. The

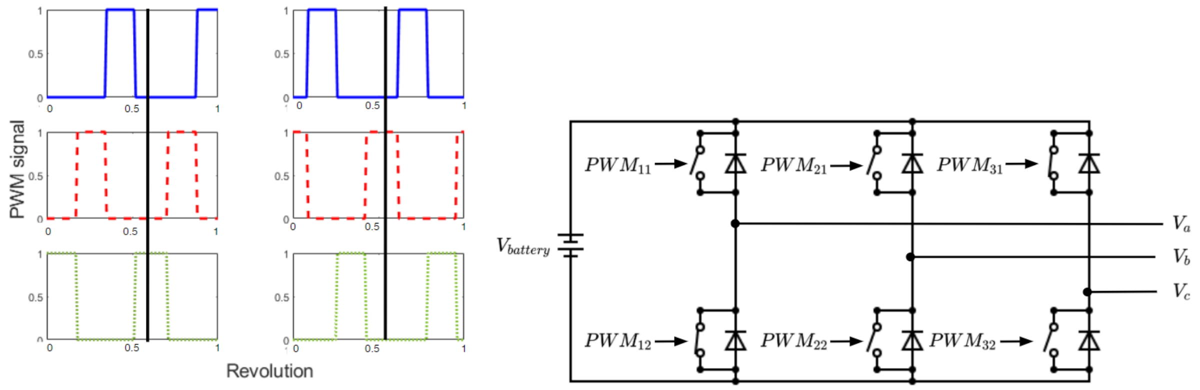


Figure 7. Switched three-phase converter with averaged input voltage.

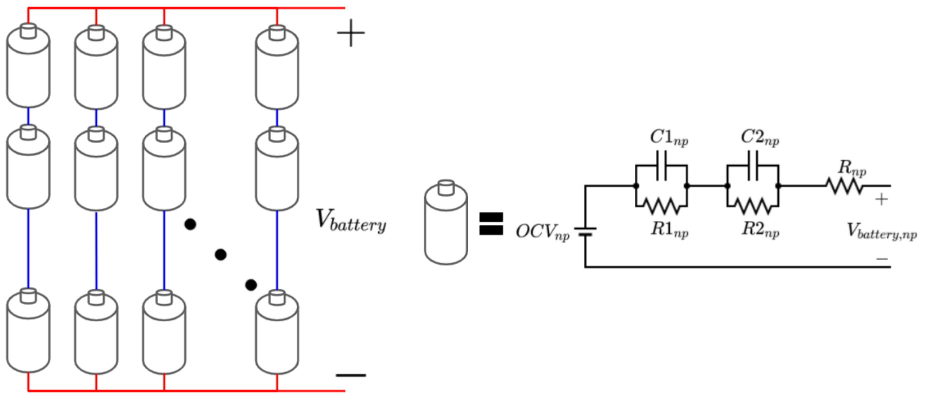


Figure 8. Electrical schematic of battery.

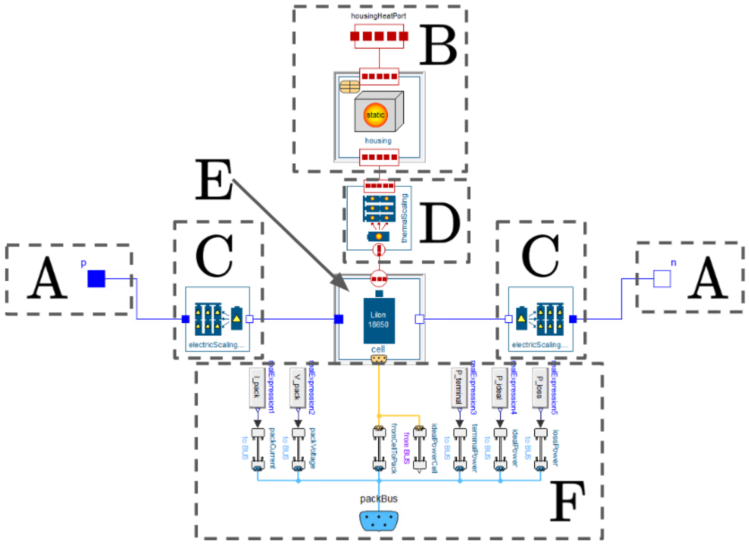


Figure 9. Battery model in Dymola using the Dassault Battery Library.

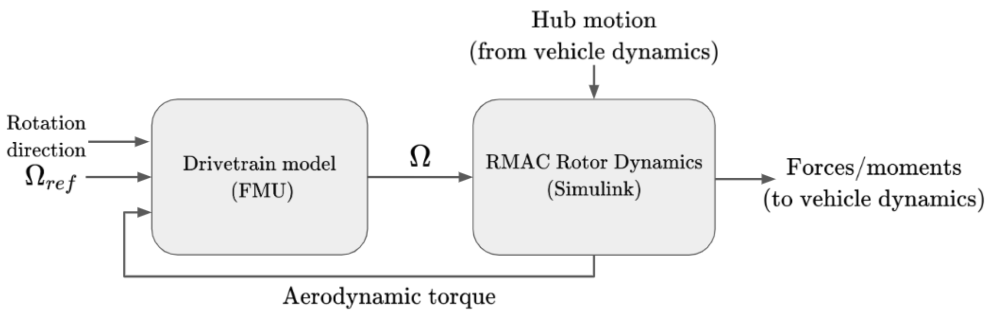


Figure 10. Interfaces between the electric drivetrain FMU and the RMAC rotor model in MATLAB/SIMULINK.

controller calculates the duty cycle of the drivetrain as a function of the rotor speed and the commanded speed. The controller is an explicit-model-following controller that can be tuned based on handling qualities requirements such as those in (Niemiec, Gandhi, et al. 2020), (Walter et al. 2020), and (Bahr et al. 2020).

Different electrified drivetrains can be used with the proposed interfacing approach, as long as the generated FMU can provide the same inputs and outputs for each variant, which are simple to define in Modelica. In this way, different model variants and architectures can be simulated by simply loading the desired FMU into the `FMIme` block from the FMI Toolbox (Modelon 2018). Thus, the integrated aerodynamic of RMAC and the electrical drivetrain developed in Modelica are simulated in MATLAB/SIMULINK.

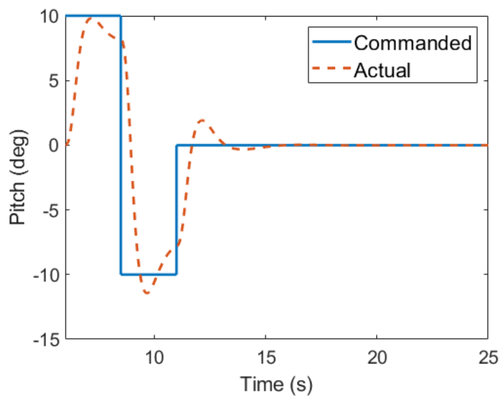


Figure 11. Pitch command and vehicle response.

4 Results

Using the proposed integration of RMAC with the FMI standard using the FMI Toolbox, both electrical architecture configurations of the eVTOL system in Figures 1 and 2 are analyzed next. All case studies simulate a heave command that is applied to study the interplay between the electrical drivetrain configuration and the aircraft dynamics. Six different cases were considered:

1. Centralized battery modeled using an ideal 60V voltage source.
2. Distributed (individual) battery modeled using an ideal 60V voltage source.
3. Centralized battery starting at 100% state of charge.
4. Distributed (individual) starting at 100% state of charge.
5. Centralized battery starting at 30% state of charge.
6. Distributed (individual) starting at 30% state of charge.

For the ideal voltage source cases, the battery is assumed to stay at a constant 60 V and would be able to supply power to the multicopter indefinitely, which is unrealistic. Thus, to highlight the importance of adequately modeling the battery, the model in Figure 9 is used for cases 3-6, where we apply the maneuver to the aircraft at the beginning of a flight (100% state of charge) and at the end of flight (30% state of charge).

To observe the closed-loop dynamic behavior of the vehicle subject to pitch, the command in Figure 11 is applied to the electrified drivetrains. The system is subject to a 10 degree pitch command for 5 seconds and a -10 degree pitch command for 5 seconds, as denoted by the blue line. The actual response of the controller is denoted by the red line in Figure 11. The command model in RMAC for pitch is a second-order model ($\zeta = 0.7, \omega_n = 3.46$ rad/s).

The front and rear rotors receive opposite commands to achieve the pitch behavior, as shown in Figure 12. The current drawn at each of the motors is shown in Figure 13, where the opposite speed commands are also reflected in the current draw. Since all of the motors are connected to one central battery, the current spikes in the motors cancel each other out when observing the total current draw from the battery (Figure 14).

Next, the distributed battery system in Figure 2 is subjected to the same pitch command. The speed response and current draw of the front and rear motors are identical to the centralized battery case, as shown in Figure 12. Since the spikes in the current draw from the front and rear motors cannot cancel each other out in this configuration, this ripple is observed in the battery voltage shown in Figure 15.

By exporting these mutlicopter models as FMUs to interact with RMAC, we can produce commands to show

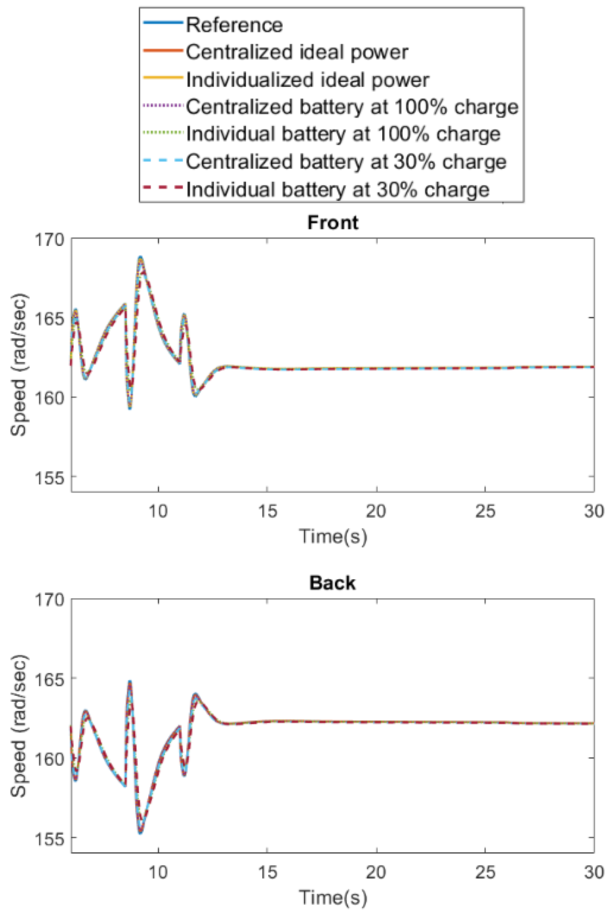


Figure 12. Speed response of multi-rotor system to pitch command.

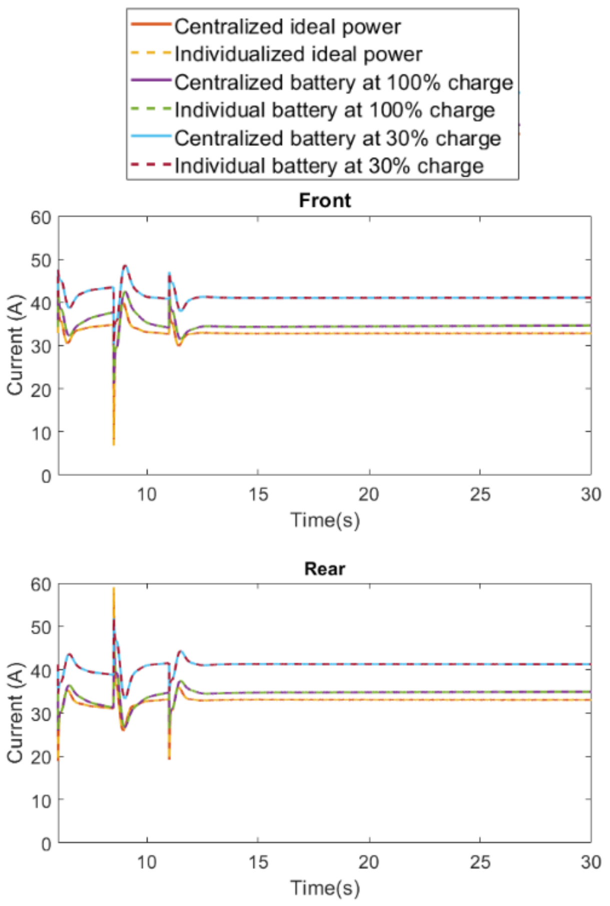


Figure 13. Current response of front and rear motors of multi-rotor system to pitch command.

that the system must be sized to accommodate the system architecture and desired commands. This modeling method allows us to compare both a centralized and distributed battery architecture, where both architectures produce the same speed response for a pitch command and thus have the same current draw per motor. This behavior is not observed in the electrical dynamics, where the current spikes are cancelled out when observed from the battery for a centralized architecture. If a centralized battery architecture is selected for the vehicle, the pitch command will not be the limiting factor that the battery must be sized to complete. When a distributed architecture is considered for a vehicle, the battery must be sized to accommodate for the current spikes produced by cases such as those of the pitch command.

5 Conclusions

The premise of the FMI standard is to enable model portability and re-usability, i.e. the usage of one model in many tools. This provides tremendous opportunities to extend the modeling capabilities of existing domain-specific tools. In the case of the emerging field of eVTOL, existing multicopter aerodynamic analysis tools can be expanded with relatively low effort to incorporate electric power train modeling capabilities.

In this multi-domain electrified drivetrain study for eVTOL, we showed how the FMI standard allowed us to integrate Modelica models with an existing specialized multicopter aerodynamic research tool (RMAC). To couple both modeling domains, RMAC provided aerodynamic inputs and feedback for the aircraft model in MATLAB/SIMULINK. Then, using the FMI Toolbox from Modelon, RMAC was extended to support electrified drivetrain models with specific input/output interfaces. Meanwhile, the FMI standard enabled us to utilize a multi-domain eVTOL drivetrain model developed in Modelica by using Dymola's export support for the Model Exchange specification. Thus, to couple the electrified drivetrains with RMAC it was possible to simply import the different model variants into MATLAB/SIMULINK to interact with RMAC.

The proposed coupling approach helped to obtain simulation results that enable a new understanding of the trade-offs between different types of propulsion architectures for new eVTOL vehicles. A centralized battery architecture can take advantage of canceling effects in the required

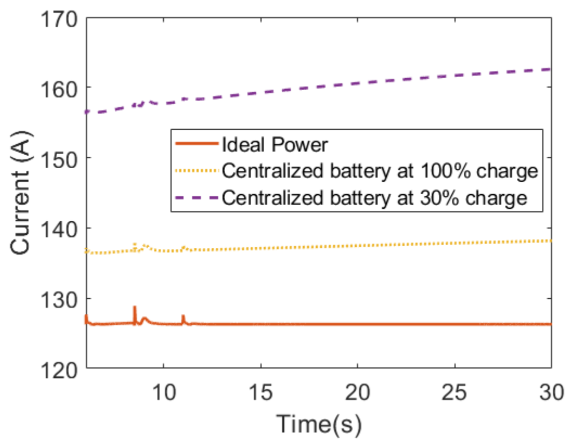


Figure 14. Current response of centralized battery of multi-rotor system to pitch command.

voltage/current, while a distributed architecture will need the battery to be sized by considering the load requirements resulting from heave commands.

Acknowledgements

This work was supported in whole or in part by the National Aeronautics and Space Administration under award number 80NSSC19M0125 as part of the Center for High-Efficiency Electrical Technologies for Aircraft (CHEETA), and in part by The Boeing Company through its Charitable Partnership with Rensselaer Polytechnic Institute.

The first author is 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.

References

Bahr, Matthew et al. (2020-10). “Handling Qualities Assessment of Large Variable-RPM Multi-Rotor Aircraft for Urban Air Mobility”. In: *76th Annual VFS Forum*. Virtual.

Bals, Johann et al. (2009). “Model based design and integration of more electric aircraft systems using modelica”. In: *Moet forum at European power electronics conference and exhibition*.

Dassault Systemes (2022-05). *CATIA Systems Brushless DC Drives Library*. https : / / www . 3ds . com / fileadmin / PRODUCTS / CATIA / DYMOLA / PDF / 3DS _ 2017 _ CAT _ BrushlessDCDrives_Flyer_A4.pdf.

Dassault Systems (2022). *CATIA Systems Engineering: Battery Library*. URL: https://www.3ds.com/fileadmin/PRODUCTS/CATIA/DYMOLA/PDF/3DS_2015_CATIA_BTY_Battery_ _Flyer_A4_WEB.pdf.

Hacker Q150-45-4 Series Datasheet (2021). https : / / HackerMotorUSA . com. URL: https : / / HackerMotorUSA . com.

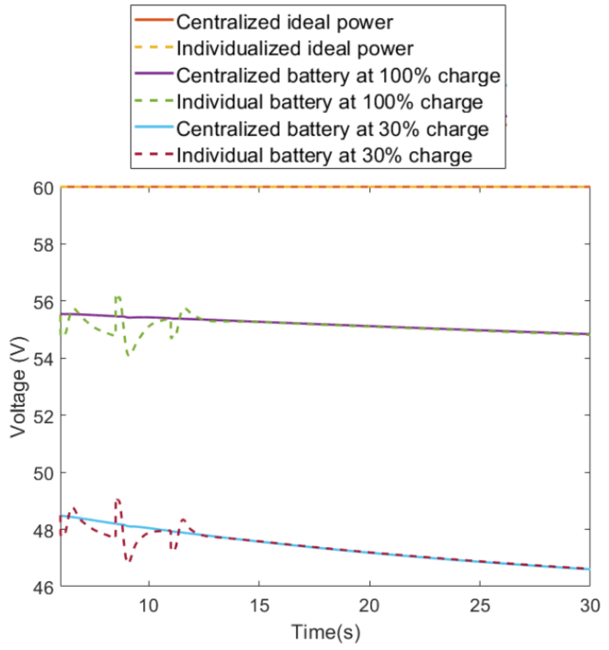


Figure 15. Voltage response of front and rear motors of multi-rotor system to pitch command.

Henningson, Maria, Johan Akesson, and Hubertus Tummescheit (n.d.). “An FMI-Based Tool for Robust Design of Dynamical Systems”. In: *Proceedings of the 10th International Modelica Conference;*

Holden, Jeff and Nikhil Goel (2016-10). *Fast-Forwarding to a Future of On-Demand Urban Air Transportation*. Available at: https://www.uber.com/elevate.pdf [Accessed 10 Oct. 2019].

Modelon (2018-07). *FMI Toolbox User’s Guide 2.6.4*. Tech. rep. Modelon AB.

NASA (2020 (retrieved Nov 3, 2020)). *AAM Overview | NASA*. https://www.nasa.gov/aeroresearch/aam/description/.

Niemiec, Robert and Farhan Gandhi (2019). “Development and Validation of the Rensselaer Multicopter Analysis Code (RMAC): A Physics-Based Comprehensive Modeling Tool”. In: *Proceedings of the 75th Vertical Flight Society Annual Forum*.

Niemiec, Robert, Farhan Gandhi, et al. (2020-10). “System Identification and Handling Qualities Predictions of an eVTOL Urban Air Mobility Aircraft Using Modern Flight Control Methods”. In: *76th Annual VFS Forum*. Virtual.

Podlaski, Meaghan et al. (2021). “Multi-Domain Electric Drivetrain Modeling for UAM-Scale eVTOL Aircraft”. In: *Proceedings of the 77th Vertical Flight Society Annual Forum*.

Velden, Wouter van der and Damiano Casalino (2021). “Flight and Noise Assessment of an eVTOL Vehicle using a Multi-Fidelity Model-Based System Engineering Methodology”. In: *DICUAM 2021, Delft International Conference on Urban Air-Mobility*.

Walter, Ariel et al. (2020-10). “Hover Handling Qualities of Fixed-Pitch, Variable-RPM Quadcopters with Increasing Rotor Diameter”. In: *76th Annual VFS Forum*. Virtual.

Zhou, Yan et al. (2018). “Modeling and simulation of a distributed electric propulsion aircraft by modelica”. In: *CSAA/IET International Conference on Aircraft Utility Systems (AUS 2018)*. IET.

Multirotor drone sizing and trajectory optimization within Modelon Impact

Clément Coïc[1] Marc Budinger[2] Scott Delbecq[3]

[1]Modelon, Germany, clement.coic@modelon.com

[2]*Institut Clément Ader (ICA), Université de Toulouse, CNRS-INSA-ISAE-Mines Albi-UPS, Toulouse, France*

[3]ISAE-SUPAERO, Université de Toulouse, France

Abstract

The design of multirotor drones often relies on optimizing its performance in terms of maximum speed requirements and hover time. This is well suited to undefined tasks. In the case of repetitive tasks, the drone trajectory can be added as a third degree of freedom. This paper focuses on the use of Modelon Impact and its dynamic optimization capabilities to reach a multirotor drone design and 1-D trajectory optimization. In comparison to other options investigated by the authors in a separate publication, Modelon Impact based optimization proved to be much simpler, more robust, and faster – for this use case.

Keywords: Multirotor, Drone, Dynamic optimization, Trajectory optimization, Sizing, Modelon Impact, Optimica,

1 Introduction

Multirotor drones are often associated with toys that are fun to pilot. The drone designer does not know in advance who will use the drone and how it will be used. Therefore, these drones are typically designed based on performance requirements. For a toy drone to be fun, the user expects it to be fast and to have a satisfying autonomy. The toy drone designer often takes as requirement a maximum speed and a given hover time.

On the other side, multirotor drones are also being developed for some industry applications for a variety of roles – from packages delivery to military assistance or payload lifting in substitution to cranes. Contrarily to the toy drones, industry drones typically have well defined missions often expressed as:

- Start point: initial elevation, hover time
- End point: final elevation, horizontal distance from the initial point and hover time

Missions including more points can be described as sequences of start and end points.

Getting back to the sole mission of the drone – in comparison to optimizing for performance requirements – relaxes an entire degree of freedom: the drone trajectory. Solving both the drone sizing and trajectory optimizations allows focusing on optimization criteria such as minimizing energy consumption or the cost function, if willing to associate a cost to the parts that compose the drone and its utilization (time of utilization – including potential operator – and energy consumption).

This paper presents how easy it is to perform sizing and trajectory optimization of a system within Modelon Impact. A drone is selected as example system. A selected case study – payload lifting – is introduced in section 2. The drone model is discussed in section 3 of this paper, with a particular focus on the propeller. In section 4, we present the optimization problem, the simplicity of its implementation in Modelon Impact and the associated results. Finally, section 5 is a collection of advantages that come with optimizing using OPTIMICA and Modelon Impact.

2 Case Study – Payload lifting

2.1 Use Case

It is typical on construction sites – mostly when approaching the end of the construction – to either keep a crane operating for a longer period time to lift some minor equipment or material, or to carry this payload by human strength. While the former solution is often expensive, the latter requires manpower and can affect physical health. The use of drones to lift these small payloads (Draganfly, 2022) is an economically attractive solution which has a low impact on physical health.

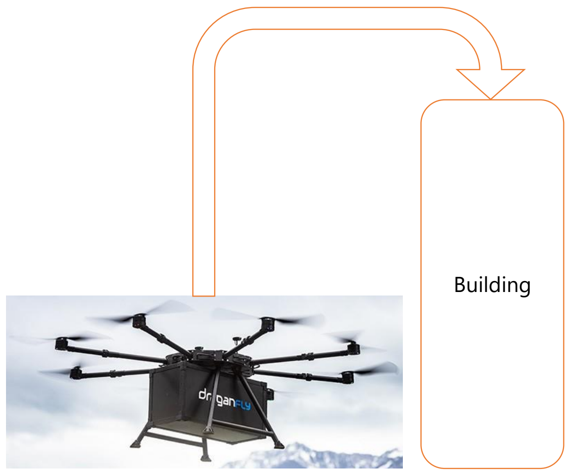


Figure 1. Illustration of the load lifting use case.

This paper focuses on a drone design aimed at a repetitive task: small payload lifting – below 25 kg – from ground to the top of a building.

2.2 Main Requirements

To match the use case presented in section 2.1, the main requirements for the drone designs are related to the lifting operation in terms of payload and endurance:

- Payload: The drone shall lift masses up to 25 kg during its full operation.
- Endurance: The drone shall operate at least 150 climbs of at least 10 m height with 5 seconds hovering – for the handling of the payload.

For this use case, the drone returns to ground without any payload and thus with very little energy consumption. Should the drone carry payload on the descent, the number of climbs would be reduced inevitably.

3 From Drone Architecture to Model

This section details the drone architecture, technological choices, the sizing problem formulation and discusses the associated model.

3.1 Drone Architecture

Different architectural choices can be made depending on the purpose and the mission. For multirotor drones, the main choices are the number of arms and the number of propellers per arms but also the materials and technologies of the components.

As the main usage of such a drone is in urban area, it was decided to select a fully electric drone design. Indeed, electric motor emit less pollution – in terms of emissions, smell, and noise – than the combustion ones. In the presented work, a single architecture is considered for the multirotor drone. The architecture is presented in Figure **2** and is composed of:

1. Four (4) fixed pitch propellers
2. Four (4) out-runner brushless motors
3. Four (4) electronic speed controllers (ESC) mainly made from MOSFET inverters
4. One (1) battery based on Li-Ion cells
5. One (1) mechanical structure (frame) consisting of four (4) arms and one (1) central body

Investigating variable architecture designs is let as perspective to this work.

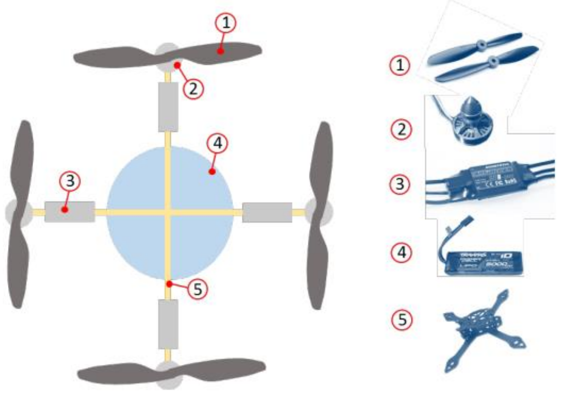


Figure 2. Multirotor drone architecture and components.

3.2 Discussion on the Drone Model Fidelity

As for physical systems, a simulation model is developed for a given purpose. This purpose guides technological choices on the model development, such as general assumptions, the physical effects modeled, the level of details, the smoothing, etc. These are all gathered within the so-called model fidelity.

The purpose of the multirotor drone model for this paper is to perform a component sizing and simultaneously optimize the 1-D trajectory of the drone to perform a well-defined task. From these two purposes, we can extract a few technological choices.

<u>Sizing purpose</u>

Implications on model causality

When it comes to sizing a system based on a desired trajectory – here, imposed by the optimization algorithm – , it is often necessary to reverse the power chain. This is also known as bicausality.

A performance simulation of the drone model would typically require a known load at the propellers and would compute the resulting speed of the drone, for a given command. On the contrary, a sizing scenario would provide both the load at the rotors and the drone speed. These two variables define the required power output, which can be propagated upstream on the power chain to compute the require power at the power generation or storage – here, the battery.

As the design validation would require a performance simulation, the acausality of the Modelica language is clearly a benefit to solve sizing problems. The sizing solving requires propagating the power variables through the component ports in one way or another – e.g. by having *flow* and *non-flow* which product would lead to the power or by adding the power to a connector.

Selection of model complexity

For cost reasons, we assume the drone will be assembled using off-the-shelf components. The sizing problem thus consists mainly of finding a good order of magnitude for each component size. Therefore, physical effects to be modeled should only be the dominant ones and their implementation complexity could be quite low. Consequently, scaling laws are used to design the components and efficiencies are often used to assume losses – instead of overloading the model with multiple separate physical losses, e.g. each separate friction.

Optimization purpose

Most optimization algorithms rely on gradients to define on which direction they should perform the next step. This means that the variables of the model should, at least, be continuous and derivable. A special effort is made in the Modelica code to respect this constrain.

In addition, minimum, maximum and nominal values of the design variables are provided to allow respectively to bound and normalize the variables and equations – key for optimization convergence.

Finally, dynamic optimization benefits from having both an initial and a nominal trajectory that match the specified requirements – without necessarily be optimal. This is easily achieved by simulating first the drone behavior with a smooth trajectory command. Here, the acausality of the Modelica language becomes once more convenient.

3.3 Drone Model – Focus on Propeller

The drone Modelica model serves two purposes:

1. It includes the scaling laws for all components to allow their sizing.
2. It encodes the physics equations to model the flight performance and power consumption.

The component sizing models used are scaling laws, linear regressions of data sheet and surrogate models – detailed by Budinger (2020). The physics equations are well known equations from components and are presented by Delbecq (2021). It is however relevant to present here the propeller model, as a representative component of the drone.

The propeller represents a key component in the drone propulsion chain. Its performance can be expressed as a function of two coefficients C_T and C_P, respectively expressing thrust (1) and mechanical power (2) equations:

$$Thrust = C_T \rho_{air} n^2 D^4 \qquad (1)$$

$$Power = C_P \rho_{air} n^3 D^5 \qquad (2)$$

where ρ_{air} represents the air density in [kg/m^3], n the rotational speed in [rev/s] and D the propeller diameter in [m].

While C_T and C_P are dimensionless, they are not constant. These depend on further variables such as the blade pitch p, the air Bulk Modulus K, the relative airspeed V (normal to the rotor plane). It is here assumed that:

$$Thrust = f(D, p, \rho, K, n, V) \qquad (3)$$

$$\rightarrow C_T = f(D, p, \rho, K, n, V)/(\rho_{air} n^2 D^4) \qquad (4)$$

Performing a dimensional analysis on these equations allows identifying a reduced set of dimensionless variables that can define this equation. Note that Buckingham's theorem gives us the insight that these dimensionless variables are in number of 3 (6 variables and 3 units dimensions). The detailed dimensionless analysis is available on request – please email the authors. This gives us the following three dimensionless numbers:

- The pitch to diameter ratio: $\beta = pitch/D$.
- The advance ratio: $J = V/(nD)$.
- The air compressibility indicator: $B = K/(\rho n^2 D^2)$

The analysis on C_P reveals the same dimensionless numbers. As both the thrust and power coefficients are surface responses, these are better fitted with polynomial regression rather than with power regression – as discussed in (Sanchez 2017).

A sensitivity study was conducted in (Budinger 2020) on the three dimensionless numbers within the domain of usage of the drone. It showed that both C_T and C_P are quite insensitive to the air compressibility indicator, within this domain. Finally, the fitting revealed the following equations:

$$\begin{aligned} C_T \approx\ & 0.02791 - 0.06543J - 0.23504J^2 \\ & + 0.02104J^3 + 0.11867\beta \\ & + 0.27334\beta^2 - 0.28852\beta^3 \\ & + 0.18677\beta J^2 \end{aligned} \qquad (5)$$

$$\begin{aligned} C_P \approx\ & 0.01813 - 0.00343J - 0.12350J^2 \\ & + 0.06218\beta + 0.35712\beta^2 \\ & - 0.23774\beta^3 + 0.07549\beta J \end{aligned} \qquad (6)$$

The surface responses and the corresponding datasets are presented below for both coefficients.

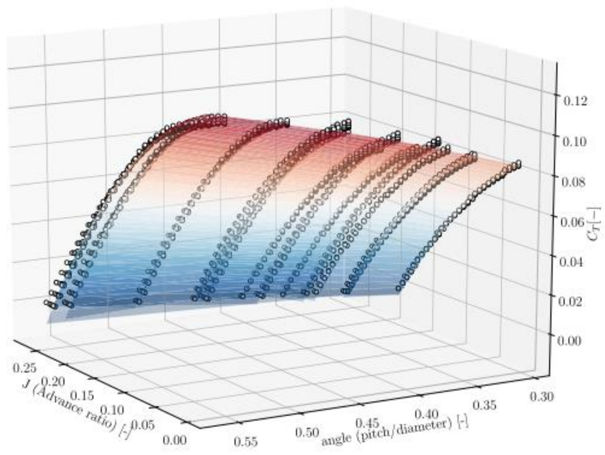


Figure 3. Surface response of C_T and reference dataset (dots).

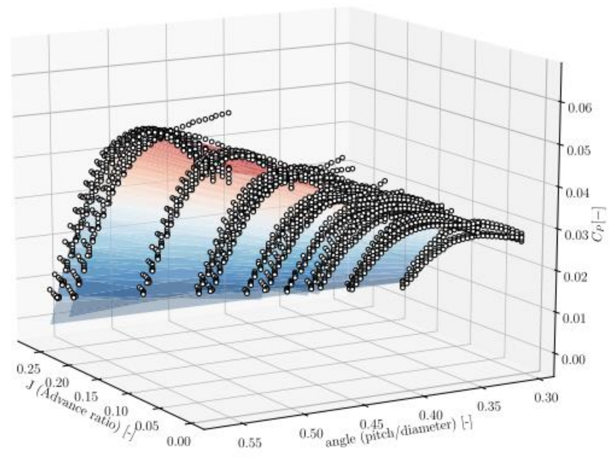


Figure 4. Surface response of C_P and reference dataset (dots).

This gives us the performance model of the propeller, that needs to be completed by its sizing model. For the propeller, that parameters of interest are the mass and the inertia. These are computed based on scaling laws. The propeller mass M_{prop} is found to be proportional to the cube of its diameter. From the mass, the inertia I_{prop} can be computed. Note that scaling laws require similar reference data to scale on to.

$$M_{prop} = \mathrm{M_{ref}}\left(D_{prop}/D_{ref}\right)^2 \tag{7}$$

$$\mathrm{I_{prop}} = \mathrm{M_{prop}}\left(D_{prop}/2\right)^3/3 \tag{8}$$

From this complete set of equations, we get the propeller sizing and associated performance – impacted by the sizing. Similar models are developed for all components and the following parts of the paper will focus on the general sizing problem formulation and optimization problem, rather than detailing each component. As a reminder, these equations are available in (Budinger 2020) and the propeller model fidelity is representative of the rest of the model. If of interest, a more detailed Modelica performance model is presented by Podlaski (2020).

3.4 Sizing Problem Formulation

The main sizing scenarios, design drivers and models for vertical flight applications of multirotor drones are summarized in Figure 5. Such applications consist of three sizing scenarios to be considered in the design problem that are:

1. the hovering flight with the advance ratio of the propeller *J = 0* – as the air speed *V* is null.

2. the takeoff phase which requires maximum power to accelerate the drone with an increasing *J* – increasing air speed *V*.

3. the climb phase with a constant vertical speed and thus constant *J*.

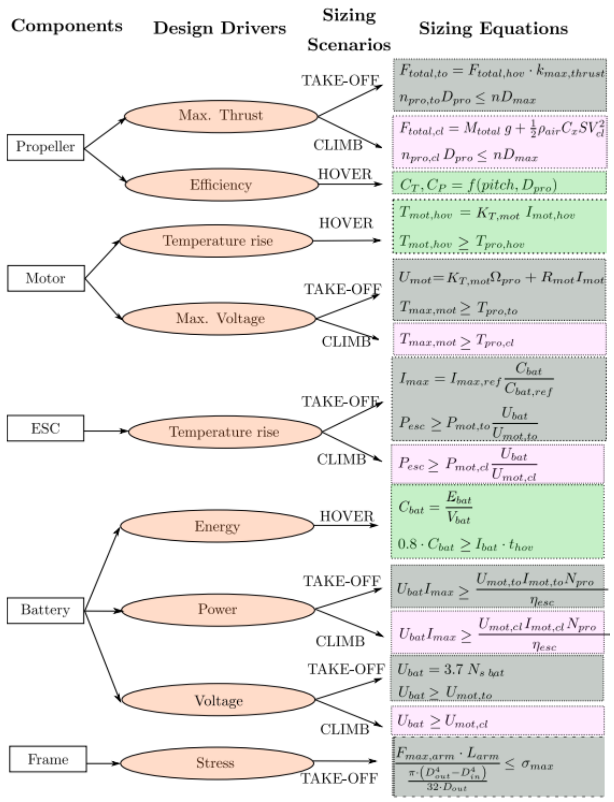

Components
Design Drivers
Sizing Scenarios
Sizing Equations
Propeller
Max. Thrust
Efficiency
Motor
Temperature rise
Max. Voltage
ESC
Temperature rise
Battery
Energy
Power
Voltage
Frame
Stress
TAKE-OFF
CLIMB
HOVER
$F_{total,to} = F_{total,hov} \cdot k_{max,thrust}$
$n_{pro,to} D_{pro} \leq nD_{max}$
$F_{total,cl} = M_{total}\, g + \frac{1}{2}\rho_{air} C_x S V_{cl}^2$
$n_{pro,cl}\, D_{pro} \leq nD_{max}$
$C_T, C_P = f(pitch, D_{pro})$
$T_{mot,hov} = K_{T,mot}\; I_{mot,hov}$
$T_{mot,hov} \geq T_{pro,hov}$
$U_{mot} = K_{T,mot}\Omega_{pro} + R_{mot} I_{mot}$
$T_{max,mot} \geq T_{pro,to}$
$T_{max,mot} \geq T_{pro,cl}$
$I_{max} = I_{max,ref}\frac{C_{bat}}{C_{bat,ref}}$
$P_{esc} \geq P_{mot,to}\frac{U_{bat}}{U_{mot,to}}$
$P_{esc} \geq P_{mot,cl}\frac{U_{bat}}{U_{mot,cl}}$
$C_{bat} = \frac{E_{bat}}{V_{bat}}$
$0.8 \cdot C_{bat} \geq I_{bat} \cdot t_{hov}$
$U_{bat} I_{max} \geq \frac{U_{mot,to} I_{mot,to} N_{pro}}{\eta_{esc}}$
$U_{bat} I_{max} \geq \frac{U_{mot,cl} I_{mot,cl} N_{pro}}{\eta_{esc}}$
$U_{bat} = 3.7\; N_{s\; bat}$
$U_{bat} \geq U_{mot,to}$
$U_{bat} \geq U_{mot,cl}$
$\frac{F_{max,arm} \cdot L_{arm}}{\frac{\pi \cdot (D_{out}^4 - D_{in}^4)}{32 \cdot D_{out}}} \leq \sigma_{max}$

Figure 5. Design drivers and equations of the multirotor drone.

The overall sizing model has been adapted from (Delbecq 2021) which is tailored for vertical flight applications of multirotor drones, and should be consulted for a more detailed sizing formulation.

4 Optimization Problem

4.1 Optimization Problem Formulation

The simultaneous trajectory and design optimization problem can be formulated as a mass optimization problem including the trajectory variables (motor torque command and final time):

$$\begin{aligned}
&\textbf{minimize } MTOW \\
&\textbf{with respect to } \beta_{prop}, k_{MTOW}, k_{ND}, k_{mot}, k_{mot,speed} \\
&\qquad k_{bat,mass}, k_{bat,voltage}, k_{arm}, T_{mot}(t), a_{t0}, t_f \\
&\textbf{subject to } T_{prop,t0} - T_{mot,max} \leq 0 \\
&\qquad E_{mission} - E_{bat} \leq 0 \\
&\qquad U_{mot} - U_{ESC} \leq 0 \\
&\qquad U_{ESC} - U_{bat} \leq 0 \\
&\qquad MTOW_f - MTOW \leq 0 \\
&\qquad T_{mot}(t) - T_{mot,max} \leq 0 \\
&\qquad h - z(t_f) \leq 0 \\
&\qquad \dot{z}(t_f) = 0
\end{aligned} \tag{9}$$

The objective is to minimize the weight of the vehicle for the defined mission with respect to design variables such as the propeller pitch β_{pro} or the nominal motor torque T_{mot}. The design has to respect some constraints to respect the technological constraints of components such as motor maximum electromagnetic torque $T_{mot,max}$ as well as others to respect voltages consistency in the power train ($U_{mot} \leq U_{ESC}$ and $U_{ESC} \leq U_{bat}$). Some consistency constraints are used to solve multidisciplinary couplings as suggested by Delbecq (2020) ($E_{mission} \leq E_{bat}$ and $MTOW_f \leq MTOW$). This also requests to add normalized design variables such as k_{MTOW} and $k_{bat,mass}$.

Instead of minimizing the drone weight, minimizing the energy could have been used as objective. Unfortunately, within the time given to investigate this solution, this seemed to be a less robust option. Minimizing the mass is a conscious problem simplification – a lower mass means less energy to carry it. The authors acknowledge that minimizing the energy might lead to a slightly different optimum to this problem.

4.2 Optimica Implementation

Modelon Impact (Coïc 2020-b) is a state of the art, cloud-based modeling and simulation environment, relying on open standards such as Modelica, FMI and Python. Modelica models can be developed within Modelon Impact by composition (drag and drop and connect) of existing models from available Modelica libraries, or by writing Modelica code within the code editor. Modelon Impact compiler, Optimica Compiler Toolkit (OCT), compiles the models – either in steady-state or dynamic simulation mode.

As many engineering problems can be cast as optimization problems – including optimal control, minimum time problems, optimal design, and model calibration – Modelon Impact compiler supports optimization of dynamic and steady state models. This is achieved relying on the extension of Modelica language for optimization: OPTIMICA (Åkesson 2008).

As the OPTIMICA language extends the Modelica language, it is convenient to opt for a similar approach when formulating an OPTIMICA optimization problem. Thus, the optimization model could be built as follow:

1. Create an *optimization* class and provide modifiers to set up the objective and time constraints.
2. Extend the Modelica model and provide modifiers to fix or relax parameters as well as minimum, maximum and nominal values.
3. Optionally add more variables and equations.
4. Add the constraints of the optimization problem.

The Optimica code of the drone optimization is listed in Listing 1.

Listing 1. OPTIMICA Code of the Drone Optimization

```
optimization SizingAndTrajectoryOptim (
  objective=M_total(startTime),
  finalTime(free=true, min=1, max=10, start=5))
// Minimize the total drone mass and relax the final simulation
time within bounds.

  import Modelica.Units.SI.DimensionlessRatio;

  extends Drone(
    x(start = 0, fixed=true),
    xp(start = 0, fixed=true),
    a(start = 0, fixed=true),
    beta(free=true, min=0.3, max=0.6, start=0.4),
    D(free=true, min=0, max=1),
    T_nom_mot(free=true, min=0),
    K_mot(free=true, min=0),
    M_bat(free=true, min=0, max=100),
    P_esc(free=true, min=0),
    k_D(free=true, min=0.01, max=1, start=0.05),
    D_out_arm(free=true, min=0.001, max=1));
// Inherit the Modelica drone model, fix initial conditions and relax
design parameters within bounds.

  Modelica.Blocks.Interfaces.RealInput Traj_in;
// Add input to the trajectory to optimize

  DimensionlessRatio n_norm(start=1, fixed=true)=n/n_hover;

  DimensionlessRatio N_norm(min=-1, max=1,
nominal=0.8)=ND/ND_max;

  DimensionlessRatio T_hov_norm(min=0, max=1,
nominal=0.6) = T_hover/T_nom_mot;

  DimensionlessRatio T_norm(min=-1, max=1, nominal=0.95) =
T/T_max_mot;

  DimensionlessRatio U_norm(min=0, max=1, nominal=0.5) =
U_mot/V_bat;

  DimensionlessRatio P_norm(min=0, max=1, nominal=0.5) =
P_mot/P_esc;

  DimensionlessRatio E_norm(min=0, max=1, nominal=0.25) =
E_drone/E_bat;

  DimensionlessRatio sigma_norm(min=-1, max=1,
nominal=0.15) = sigma/sigma_max;
// Create additional normalized variables with bounds as inequality
constraints

equation

  T=Traj_in; // Bind drone trajectory with optimization input

constraint

  x(finalTime) = 10;

  xp(finalTime) = 0;

  a(finalTime) = 0;
// Define end time constraints.

end SizingAndTrahjectoryOptim;
```

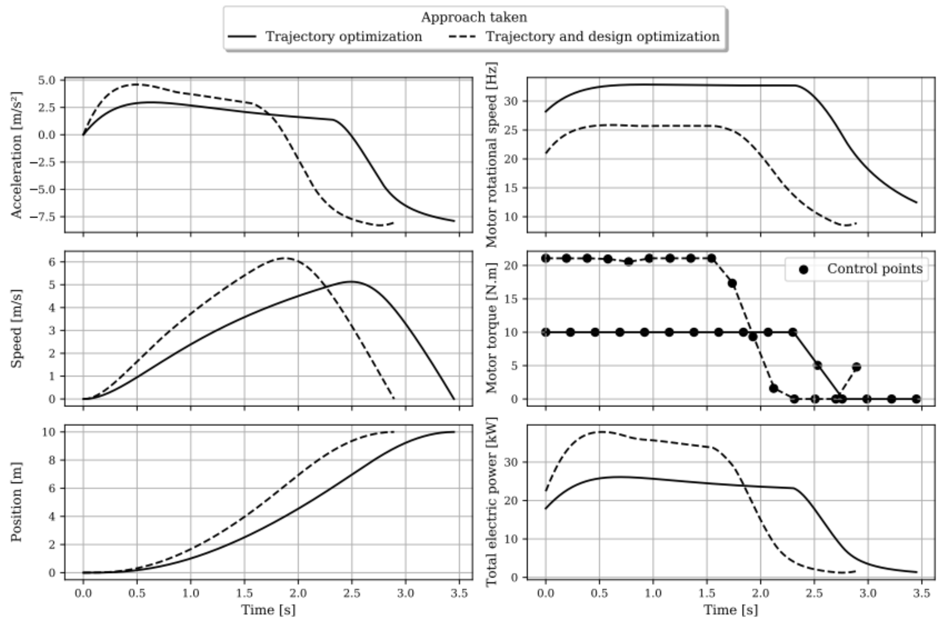


Figure 6. Comparison of resulting trajectories for the trajectory optimization with and without the sizing.

It appears that the OPTIMICA language is straightforward for a person used to the Modelica language – which itself is quite straightforward for many engineers. This way, the threshold to develop an optimization model is minimized.

4.3 Optimization Results

The result of the optimization problem – solving both the design (component pre-sizing) and trajectory optimization problems – are shown in dashed lines, for some variables, in **Figure 6**.

The solution of the sole trajectory optimization – with a separate sizing – is presented in full lines. This highlights the differences in trajectories when sizing is added as part of the optimization. When combining both, the optimizer could assess that it was more energy efficient to increase the size of the propeller, reducing its rotational speed, allow a smoother trajectory and compensating by a bigger battery – that can allow the relevant number of ascents.

After scaling to 150 climbs (endurance requirement), the optimum resulted in a drone weighting about 45 kg (without payload) with a battery contributing to more than half of the weight (about 27 kg). A drone designer might find interesting to investigate an easily replaceable battery pack in order to reduce the endurance requirement – leading to potentially more convenient (and safer) drones to operate.

These results were obtained with similar orders of magnitude with both Modelon Impact and a comparative solution based on FAST-OAD (David 2021).

5 The Benefits of Modelon Impact

There are several advantages in using Modelon Impact compared to a separate optimization with a Python package or in a dedicated optimization platform such as OpenMDAO, even if relying on a FMU for the plant model.

A key advantage of using Modelon Impact is that it relies on its OPTIMICA compiler and the OPTIMCA language, as mentioned previous, extends the Modelica language. Hence, a fair amount of the benefits listed below are Modelica features serving the optimization purposes.

5.1 Solving Initialization Problem

The Modelica language can deal with both Initial Value Problem (IVP) and Boundary Value Problem (BVP) for the model initialization. In the former case, the user provides the initial values for every state. In the latter, the number of independent initial values should match the number of states but is not necessarily their initial values.

Therefore, Modelon Impact compiler solves an initial problem – different from the dynamic problem – to resolve the BVP. In this process, the compiler can solve linear and non-linear systems, which is often not tolerated out of the box with a different solution. For example, in the Python optimization, it was necessary to adapt the code, define iteration variables (coefficient factors) and residuals (equations that should tend to zero) to solve these non-linear problems.

5.2 Acausality – One Model Several Purposes

A second advantage is that the Modelica language is acausal. This way, defining a Drone model based on the equations of the physics makes it useful for several use cases: position, speed or torque command. In our example, we want to optimize the torque trajectory while sizing the drone. Nevertheless, it is convenient to provide the optimizer a start trajectory not too far from our constraints. This is easily achieved by simulating the same model, providing a position trajectory which not optimized at all but matching our requirements.

5.3 Normalization for Convergence

Modelica models and FMUs usually use variables expressed in SI units. Variable values may therefore differ by several orders in magnitude (Coïc 2020-a). A typical example is thermodynamic models containing pressures, temperatures and mass flows. Such large differences in scales may have a severe deteriorating effect on the performance of numerical algorithms and may in some cases even lead to the algorithm failing. In order to relieve the user from the burden of manually scaling variables, Modelica offers the nominal attribute, which can be used to automatically scale a model. Modelon Impact compiler can use these nominal attributes to scale the variables (and thus objective) of the optimization problem. In addition, it is possible to provide a reference trajectory for scaling at every time step of the simulation.

5.4 Derivatives at Hand of the Optimization

Finally, the Optimica language, being an extension of Modelica, has access to all the equations of the model and can process these. Thus, the compiler automatically computes all the derivatives it requires to secure a fast and robust convergence to a global optimum of the problem.

5.5 Simpler, Faster and More Robust

Modelon Impact solution also proved to be much simpler than the python with FAST-OAD approach. The Modelica code wasn't written differently for performance simulation and for optimization purposes as it was the case for its Python version. Also, the optimization problem formulation in OPTIMICA language is very simply expressed, as mentioned above (see Listing 1), and does not diverge much from the Modelica language – which makes it really easy for a Modelica developer to ramp up on OPTIMICA

The same optimization problem was solved in Modelon Impact and using FAST-OAD. For this optimization problem that consists of 10 design variables, 7 inequality and 1 equality constraint, Modelon Impact could solve the problem in less than 30 seconds while it took more than 2 minutes to FAST-OAD. This can be explained by the different level of information on the model the optimizer has – by default FAST-OAD does not have access to the internal derivatives of the model.

Finally, and this might sound unfortunately qualitative, the Modelon Impact solution was more robust, more straightforward to converge. While it took several hours of debugging to get the Python code running and optimization solving with FAST-OAD, it appeared to work almost directly with Modelon Impact. In all transparency, the first attempt did not involve normalization of the added variables for inequality constraints, and it failed to converge. Normalizing solved the issue.

5.6 The Benefits of FAST-OAD

There are many applications for which higher model fidelities are required. FAST-OAD supports easy coupling with Computational Flow Dynamics or Finite Element models. FAST-OAD scales also very well with the number of models involved in the optimization process. Therefore, the authors value both technologies and, indeed, some authors are major contributors to FAST-OAD development.

6 Conclusion

This paper uses a multi-rotor drone (pre-)sizing and trajectory optimization to illustrate the needs for solving such a problem. The model fidelity is not necessarily the highest but constrains on the numerical aspect of the code are highlighted – e.g. acausality, smoothness, etc. The models shall be "optimization-friendly". The propeller model is detailed to emphasize the level of details sufficient for such a purpose.

In a second step, the optimization problem has been expressed, first analytically and then in OPTIMICA language – supported by Modelon Impact. The solving of the problem is achieved, and the benefit of this solution are discussed, in a generic way, and in comparison with another implementation using Python and FAST-OAD.

The results proved that solving the sizing problem in combination with the trajectory optimization leads to a better design, compared to solving both problems sequentially. All industries could benefit from optimizing systems – that are meant to exist – in a more complete manner, e.g. including dynamic optimization of trajectories.

As a perspective of work, we could show how the same model serve the purpose of Model Predictive Control of the multi-rotor drone to actually reach the optimum trajectory it was designed for. Another axis of improvement could be to use one of Modelon's 6 degrees of freedom drone model to be able to include environmental constraints – such as a wind field – in the overall design optimization problem.

References

Åkesson Johan (2008). "Optimica—An Extension of Modelica Supporting Dynamic Optimization". In *6th International Modelica Conference,* Bielefeld, Germany, 2008.

Budinger Marc, Aurélien Reysset, Aitor Ochotorena and Scott Delbecq (2020). "Scaling laws and similarity models for the preliminary design of multirotor drones". In *Aerospace Science and Technology*, 98. 1-15. ISSN 1270-9638.

Coïc Clément, Moritz Hübel and Matthis Thorade (2020). "Enhanced Steady-State in Modelon Jet Propulsion Library, an Enabler for Industrial Design Workflows". In *American Modelica Conference 2020*, Boulder, Colorado, USA.

Coïc Clément, Johan Andreasson, Anand Pitchaikani, Johan Åkesson and Hemanth Sattenapalli (2020). “Collaborative Development and Simulation of an Aircraft Hydraulic Actuator Model”. In *Asian Modelica Conference 2020*, Tokyo,Japan.

David Christophe, Scott Delbecq, Sébastien Defoort, Peter Schmollgruber, Emmanuel Benard, Valérie Pommier-Budinger (2021). “From FAST to FAST-OAD: An open source framework for rapid Overall Aircraft Design”. In *10th EASN Virtual International Conference on Innovation in Aviation & Space to the Satisfaction of the European Citizens*.

Delbecq Scott, Marc Budinger, Clément Coïc, and Nathalie Bartoli (2021). “Trajectory and design optimization of multirotor drones with system simulation”. In *American Institute of Aeronautics and Astronautics, Inc*, *SciTech*, DOI: 10.2514/6.2021-0211.

Delbecq Scott, Marc Budinger and Aurélien Reysset (2020). “Benchmarking of monolithic MDO formulations and derivative computation techniques using OpenMDAO”. In *Structural and Multidisciplinary Optimization*, Vol. 62, No. 2, 2020, pp. 645–666. DOI: 10.1007/s00158-020-02521-7.

Draganfly website – heavy lift, accessed in August 2022. https://draganfly.com/heavy-lift/

Podlaski Megan, Luigi Vanfretti, Hamed Nademi and Hao Chang (2020). “UAV Dynamics and Electric Power Systems Modeling and Visualization using Modelica and FMI”. In *American Modelica Conference 2020*

Sanchez Florian (2017). “Génération de modèles analytiques pour la conception préliminaire de systèmes multi-physiques : application à la thermique des actionneurs et des systèmes électriques embarqués”. Doctoral’s thesis. Université Toulouse 3 Paul Sabatier, France. URL: http://thesesups.ups-tlse.fr/3555/1/2017TOU30081.pdf.

Applying Design of Experiments Method for the Verification of a Hydropower System

Le Nam Hai Pham[1] Dietmar Winkler[2]

[1]University of South-Eastern Norway, Porsgrunn, Norway, Le.Pham@usn.no
[2]Department of Electrical Engineering, IT and Cybernetics, University of South-Eastern Norway, Porsgrunn, Norway, dietmar.winkler@usn.no

Abstract

Today, renewable energy plays a major role in the transition towards environment-friendly energy sources. Hydropower is one of the most important renewable energy sources leading to the high interest of research associated with the development of new technologies. These technologies aim to examine and predict the characteristics and behaviour of hydropower plants during different operating conditions and are often associated with simulation models. In the progress of creating accurate simulation models, it is necessary to have an organised and systematic method to verify and optimise the model with the help of available data. This is where the "Design of Experiments" (DoE) principles should be applied.

A simulation model of a reference hydropower plant located in Seljord municipality in the south-east of Norway was implemented using the modelling language Modelica. All parts of this hydropower plant model were tuned according to DoE procedure with the purpose of design verification and optimisation. The results of the experiments are a complete and optimised hydropower plant model that gives reliable simulation results.

Design of Experiments, DoE, hydropower, modelling, Modelica, OpenHPL

1 Introduction

In the contemporary society, there is no denying that the use of renewable energy is an absolute must when it comes to trying to reduce the greenhouse effect and slow down climate change. Among renewable energy sources, hydropower has existed for hundreds of years and, depending on the geographical location, represents a large amount of the current electricity supply. According to IEA (International Energy Agency 2022), hydropower is remaining the largest renewable source of electricity, generating more than all other renewable technologies combined. Because of the large capability of producing electricity along with the clean, reliable and flexible advantages, research of hydropower is of the highest interest and is often associated with the development of new technologies. In the era of modelling and simulation, a hydropower plant simulation model is often built with the purpose of examining and predicting the characteristic and behaviour of real plant during the different operating conditions.

In Norway, around 98 percent of all power generation comes from hydropower and Norway is one of the world's largest electricity producer per capita (Energy 2016).

In cooperation with the company Skagerak Kraft AS, based in Porsgrunn, Norway, a hydropower system model of one of their power plants located in the south-east of Norway, Grunnåi power plant, was implemented in order to study the dynamic characteristics of the plant. In the progress of building the simulation model, it is necessary to have an organised and systematic method to verify and optimise the model with the help of as much measurement data as possible.

In the recent past, many researchers have investigated the methodology of verification and validation of simulation models (Sargent 2008; Kleijnen 1995). Most of them introduced basic approaches to help research community grasp the concept, but lack realistic cases or focus in a particular simulation model. To overcome this shortcoming, this paper contributes a methodology of verification and validation, focusing to real-world hydropower plant simulation model. This is where the "Design of Experiments" (DoE) principles should be applied to perform a series of experiments on the simulation model.

To build such model and implement different experiments of model verification, the object-oriented modelling language Modelica (Modelica Association 2021) is used to model the complex physical power plant. The commercial modelling and simulation environment Dymola was used. In addition, OpenHPL (OpenSimHub 2022), an open-source hydropower library that consists of hydropower unit models, was used for building the complete hydropower system.

2 Design of Experiment

The "Design of Experiments" (DoE) is a systematic, efficient methodology that can be effective for general problem-solving, as well as for improving or optimising product design and manufacturing processes. DoE includes a series of applied statistics tools used to systematically classify and quantify cause-and effect relations between variables and outputs in the studied process or phenomenon, which may result the finding the settings and conditions under which the process becomes optimised (Jankovic, Chaudhary, and Goia 2021).

In DoE, a linear regression method, which is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables, is broadly applied (Brownlee 2016). A general multiple linear regression model with one response and *i* repressor variables is expressed as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_i x_i \tag{1}$$

where:

y Response variable

$x_{j,j=0,1,\ldots,i}$ Input variables or input factors

$\beta_{j,j=0,1,\ldots,i}$ Regression coefficients or parameters, represents expected change in response variable per unit change in input variable

2.1 Basic Principles of DoE

2.1.1 Randomisation

Randomisation is the practice of using chance methods to assign treatments to experimental units in a manner that protects against unintended influences on the assignments. A treatment, which is one specific combination of several factors at specific levels, are applied to a set of experimental units to plan an experiment to ensure valid statistical analysis is possible. Randomisation allows experimenters to safeguard against unforeseeable and uncontrollable variables which might have mask relationships between the factors and the response (Emily Divis et al. 2020).

2.1.2 Replication

Replication means repetitions of the entire basic experiment or a portion of it under one or more conditions. In other words, it is a process of running the experimental trials in a random sequence. Replication is very principal because it adds information about the reliability of the conclusions or estimates to be drawn from the data. Therefore, it has two important properties.

These are:

- Allowing the experimenter to gain the experimental error estimation
- Permitting the experimenter to gain a more precise estimate of the factor/interaction effect

It is noted that if the number of replicates is equal to one or unity, the conclusions of the effect of the factors or interactions cannot be given. Therefore, it is necessary to have a sufficient number of replicates. (Antony 2014; eMathZone 2014)

2.1.3 Blocking

Blocking is a method of eliminating the effects of extraneous variation according to noise factors, thereby improving the efficiency of experimental designs (Antony 2014). In the statistical theory of the design of experiments, the experimental units in groups or blocks, which are similar to one other, are arranged. Generally, a blocking factor is a source of variability that is not of primary interest of the experimental designs (eMathZone 2014). Experimenters can collect data under the same experimental conditions in the same block and determine the variability between blocks from the experimental error, which increases the precision of the experiments (Antony 2014).

2.2 DoE steps

This section describes the steps to perform experiments on a hydropower plant simulation model which then are later implemented. To gain good results of experiments, the key steps of DoE can put into categories.

These are:

1. **Objective recognition**: A clear statement of the problem or the objectives for an experiment can be given to gain the understanding of what needs to be done. The statement should contain a specific and measurable objective that can optimise the practical value. Clearly defined goals or objectives of the experiments are important and influence the later steps of experiments. (Antony 2014)
2. **Selection of response**: The selection of a suitable response for the experiment is important to the success of the experiment. The response, or output of experiment which are potentially influenced by the factors and their respective levels, should be certain to provide the useful information about the process under study. (MoreSteam 2022)
3. **Selection of process variables**: This stage is dedicated to consider the factors or the inputs to the process that may influence the performance of a process or system. It is crucial for the experimental procedure since if the important factors are left out of the experiment, then the response of the experiment will not be accurate and useful for any later improvement action. (Antony 2014)
4. **Performing the experiment**: In this stage, the planned experiments are carried out and conducted. When running the experiment, it is vital to monitor the process carefully to ensure that everything is being done correctly according to the sequence of experiments.
5. **Interpreting experimental results and conclusions**: After the experiment is completed, the data gathered are interpreted. The experimental results carry out the practical conclusions of the experiments and recommendation for the next actions.

3 Grunnåi Power Plant

In the year 2006, a hydropower station was completed in the Seljord municipality, Telemark county, in the south-east of Norway. At this time, there was only one turbine installed, turbine 1 (T1), with the capacity of 15.1 MW and an annual production of 55 GWh. In 2019, to reduce the loss of energy production due to unused flood waters (over-spill), an extra turbine, turbine 2 (T2), with the capacity of 10.2 MW was installed and increased the annual production to 66 GWh. The two turbines are five-nozzle Pelton turbines that are regulated by the water level of the intake reservoir and associated inflow. Table 1 shows the general information of these turbines.

Table 1. The turbines nominal operation values, Grunnåi Power Plant.

Turbine 1 (T1) Pelton type		
Property	*Value*	Unit
Number of Nozzles	5	-
Nominal Head	385	m
Nominal flow rate	4.42	m^3/s
Nominal Power	15	MW
Turbine Efficiency	90	%
Turbine 2 (T2) Pelton type		
Number of Nozzles	5	-
Nominal Head	389	m
Nominal flow rate	3.08	m^3/s
Nominal Power	10.76	MW
Turbine Efficiency	91	%

3.1 Geometry Data

The watercourse of hydropower station primarily runs through Seljord municipality and its outlet is from the east side of the valley at Vallaråi river in Flatdal. The water reservoir is Slåkåvatn lake on the Lifjell mountain. The reservoir is located 387 meters above the power plant and the rated discharged is at 7.5 m^3/s. A rough sketch of the structure of Grunnåi power plant is depicted in Figure 1.

The water from reservoir is conveyed to the power plant by the waterway system containing different geometry parts (lengths, slope, etc.) consisting of two blasted tunnels, “1” and “2” that then connected with a steel pipe conduit “3”. This conduit is branched into two separate paths “4+5” and “6+7” into two respective turbines, “T1” and “T2”. Water discharged from two turbines through the respective outlet pipe “8” and “9” and flows to the downstream, Vallaråi river through outlet system containing outlet tunnel “10” connected with culvert “11”. The flow rate of water transfer through these units are commonly influenced by roughness parameter, however, this parameter is neglected in this paper. The general information of the waterway system’s elements can be seen in Table 2.

Table 2. The waterway geometry

Element	*Index*	*Length[m]*	*Diameter [m]*
`Tunnel_1`	1	203	5.8
`Tunnel_2`	2	1455	5.8
`Conduit`	3	30	1.2
`IntakeT1_1`	4	20	1.2
`Intake T1_2`	5	1.5	0.8
`Intake T2_1`	6	25	1.2
`Intake T2_2`	7	1.5	0.6
`Outlet T1`	8	1.5	0.8
`Outlet T2`	9	1.5	0.6
`Tunnel_3`	10	460	3
`Culvert`	11	58	2

3.2 Measurement Data

The measurement data of the Grunnåi hydropower plant has been retrieved using various monitoring systems and are taken from a several sensors installed at the power plant. There are hundreds of measured quantities from monitoring systems such as temperature, water pressure, etc. It is noted that not all measured quantities are relevant for the creation of simulation model. Relevant quantities, which provide the information for simulation model experiments, are water pressure, flow rate, generated power of turbines and nozzles opening values. According to these quantity names, the measurement name of available sensors in the power plant are extracted. Table 3 shows the available signals that were used for the simulation model. It consists of two data sets, *dataset 1* and *dataset 2*, which have been recorded at different points in time at different operation conditions of the hydropower plant. This means they have to be handled individually and have no cross-correlations but at the same time are close enough in time to not be affected by any structural changes due to ageing or maintenance done to the system. The measurement positions based on the element number of Figure 1 are shown in Table 2.

Table 3. Measured quantities

Index	*Name*	*Unit*	*Dataset*
7	Water flow rate	m^3/s	1
7	Water pressure	bar	
T1	Generated power	MW	2
T1	5 Nozzles opening value	%	

For better understanding of the provided measurement data that will be used in simulation model, two datasets are shown in Figure 2 and Figure 3.

Note, due to the tolerances of the sensors used in the system being negligibly small compared to deviations caused by model inaccuracies, the measured values from Table 3 are considered as accurate and valid values and can serve as reference values against the results from sim-

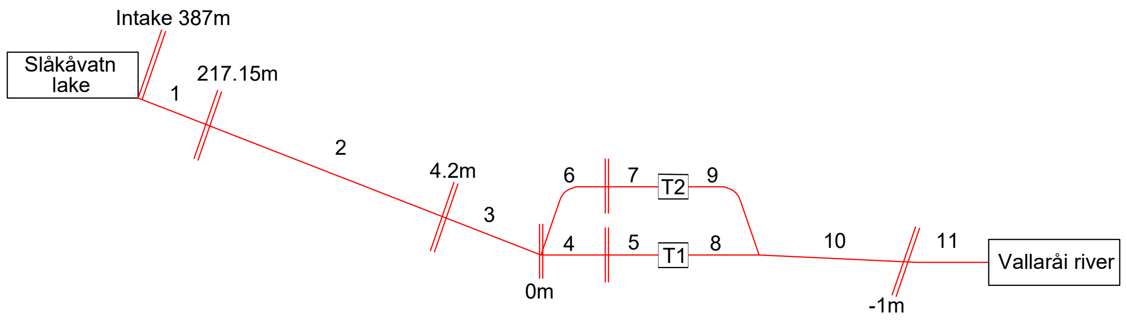


Figure 1. Overview of the structure of the Grunnåi Power Plant

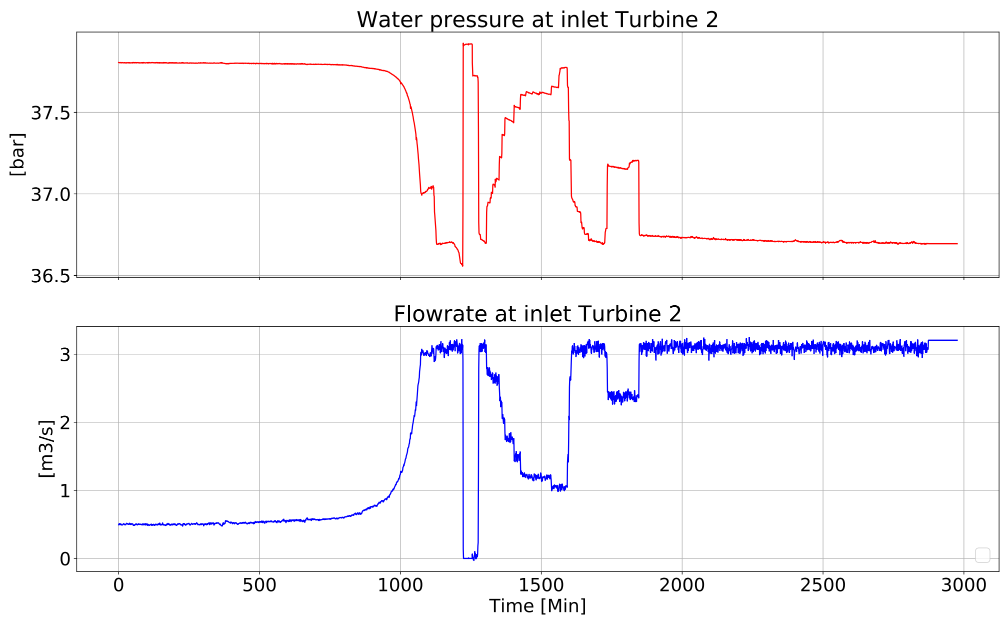


Figure 2. Measurement values of the dataset 1

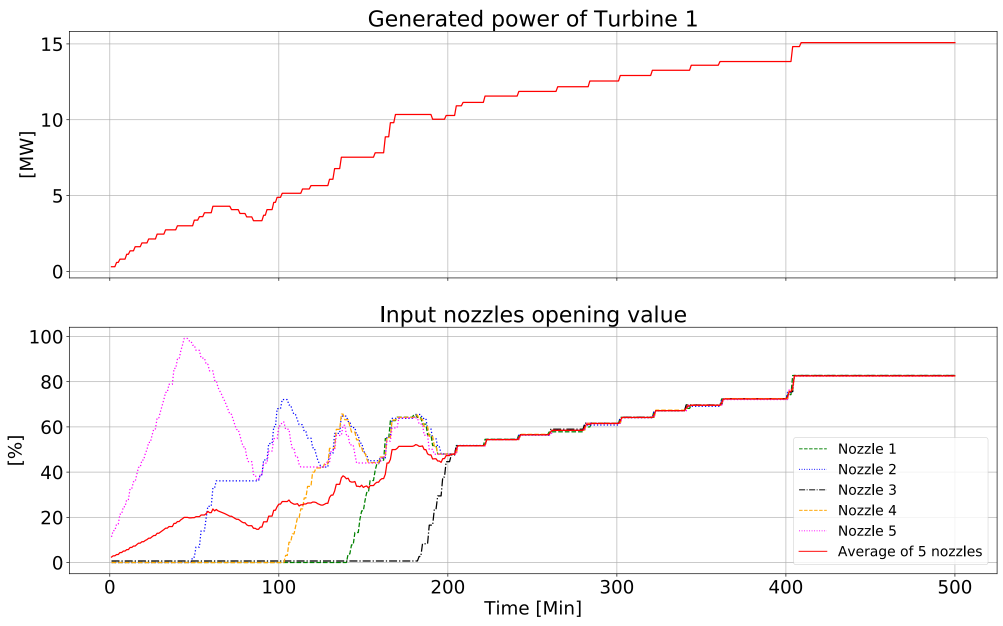


Figure 3. Measurement values of the dataset 2

ulation model.

4 Hydropower Modelling

In this section, a general description of the hydropower library OpenHPL is given as well the model of the Grunnåi power plant built based on this library is presented.

4.1 OpenHPL

OpenHPL is an open-source hydropower library that consists of hydropower unit models and is modelled using Modelica (Modelica Association 2021) and is available at (OpenSimHub 2022). This library is used to model hydropower systems of different complexity and connect them with models from other libraries, e.g., with models of the power system or other power generating sources. In this library, different waterway components of the hydropower system are described by both mass and momentum balance, and could include compressible/incompressible water. The mathematical models and methods used for the components in this library can be illustrated specifically in (Vytvytskyi 2019). An overview of the structure of the hydropower library, OpenHPL is shown in Figure 4. For this simulation model of the Grunnåi hydropower plant the version 1.5.0 of OpenHPL has been used.

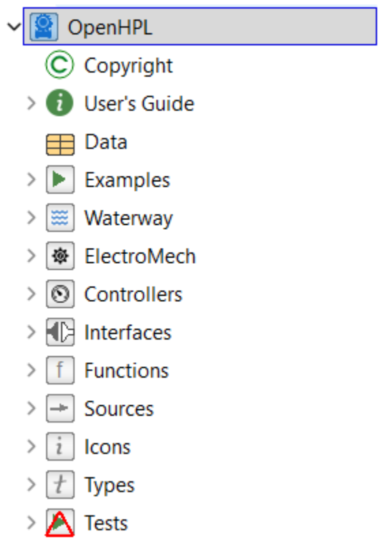


Figure 4. Screenshot of the structure of OpenHPL

4.2 Simple Model

According to general information of Grunnåi power plant given in Table 1, as well as the rough sketch of the structure as shown in Figure 1, a simple model of the hydropower plant was creating using the parameters settings from the Grunnåi power system. Figure 5 shows the simple model constructed for the simulation analysis.

5 Applying DoE for Simple Model Verification

This section describes the application of DoE to verify the simple model built based on the basic principles of DoE and the sequence of experiments is given.

5.1 Basic Principle

To verify the simple model, it is necessary to divide the simple model into parts and verify, optimise each part. This is where the blocking principle is applied. According to the structure of the hydropower plant (see Figure 1) and the available measurement data in Table 3, blocking divides the simple model into two main parts, the inlet system and the turbine system according to the two data sets. The inlet system consists of the elements from reservoir to water inlet pipe into the turbines (element “1” to “7” in Table 2) and the turbine system includes two turbine blocks with the main subject is “T1” according to the measurement data in Table 3 that is only available for “T1”. In the situation that there are several data sets, the principle of repetition and randomisation will be applied to iteratively divided parts according to the blocking principle under different operating scenarios of the hydropower plant. This is done in order to gain the experimental error estimation as well as confirm the final conclusion of experiments.

5.2 Sequence of Experiments

The verification experiments of the simple model have a sequence and cause-and-effect relationship. This means that the results of the previous part of model will directly affect the next part of the model. Therefore, setting up the experimental sequence plays an important role in the verification process of the entire simulation model. The inlet system will be simulated first to ensure the accuracy of flow rate into two turbines according to the relationship between flow rate and the generated power of the turbine. The parameters of the components in this part model are set according to the geometry data in Table 2, so these values are considered constant and cannot be adjusted. However, there is one parameter that deserves attention, which is the branching part of the water inlet (element “4” to “7”) shown in the simple model as the parallel connection between the inlet branch of “T1” and “T2”. The branching part will be experimented on using different connection components of the OpenHPL in order to find the most optimal component that represents the branching. After verifying the inlet system, the next step is to verify the turbine model. Since the turbine parameters are set up according to the provided general information in Table 3, the factors, five nozzles opening values that have main influence on the turbine model are investigated. At the beginning a *Trial Run* of the simple model is used where the average of the five nozzles vane opening values serves as an input signal to turbine model. This method needs to be verified again after having an effective inlet system. In case

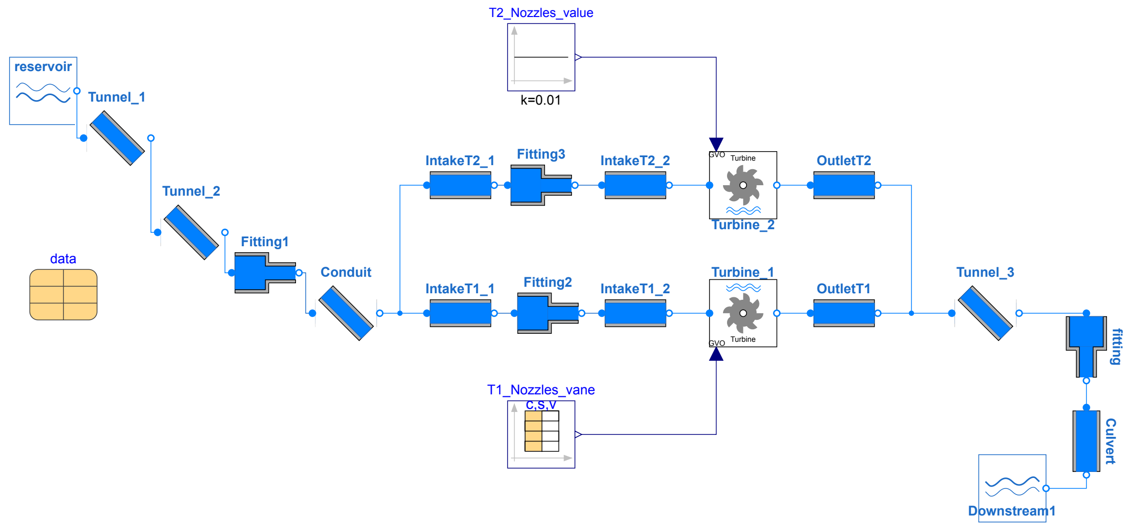


Figure 5. Simple model of hydropower plant modelled in Dymola using OpenHPL

that this method is not effective, a mathematical model between the generated power and the input signal, main vane opening value using Equation 1 will be constructed and the mathematical model will be verified by use of the simple model.

6 Experiments

The series of experiments on the simple model are implemented in order to verify and optimise the design. The sequence of experiments on the simple model is shown in Figure 6.

Trial Run First, a simulation trial was run to verify the accuracy of the built hydropower plant model by comparing the simulation results with the reference values in Table 3. This simulation trial is called *Trial Run* for which *dataset 2* was used. The input of the simulation model is the average opening value of five nozzles of T1 and the reference value of simulation results is the generated power of T1. Figure 7 shows the simulation results of the model comparing with the measurement values. T2 was deactivated for this *Trial Run*.

According to Figure 7, there is a difference between the simulation and measurement values which proves the inaccuracy of the simple model of the hydropower plant. This simple model needs to be verified and optimised which will now be shown following the DoE philosophy.

6.1 Experiment 1

Objectives recognition The objective of experiment is to determine the optimal design of branching part in the inlet system.

Description There are three different branching designs using components in OpenHPL library for the simulation model.

Design 1: Using basic connection same as the *Trial Run*.

This type of connection represents for water flow and contains the information about the pressure in the connector and mass flow rate that flows through the connector.

Design 2: Using "Fitting" component.

The "Fitting" component is modelled based on the functions defining the pressure drop due to different constrictions in the pipes. There are specify types of "Fitting" including: *Square*, *Tapered*, *Rounded*, *Sharp* , and *Thick*. These types require the diameter of the first and second pipes at the input and output of "Fitting", which are used to calculate the pressure drop in the various fitting.

Design 3: Using "BendPipe" component.

The "BendPipe" component means the bend in pipes. This bend causes a pressure drop in the water flow caused by the loss coefficient parameter which can be obtained from manufacture's information or guessed from the experimenters.

The information of the methodology and relevant equations modelling these components used in three designs is available in (Vytvytskyi 2019).

Selection of response The response of interest for the experiment is the water pressure at the inlet of T2.

Selection of process variables There are two variables in this experiment:

- The value of flow rate at inlet of T2 in *dataset 1*
- Three designs

Performing the experiment The experiment of Design 3 is implemented as Figure 8 and another designs, Design 1 and Design 2 are implemented similarly. The

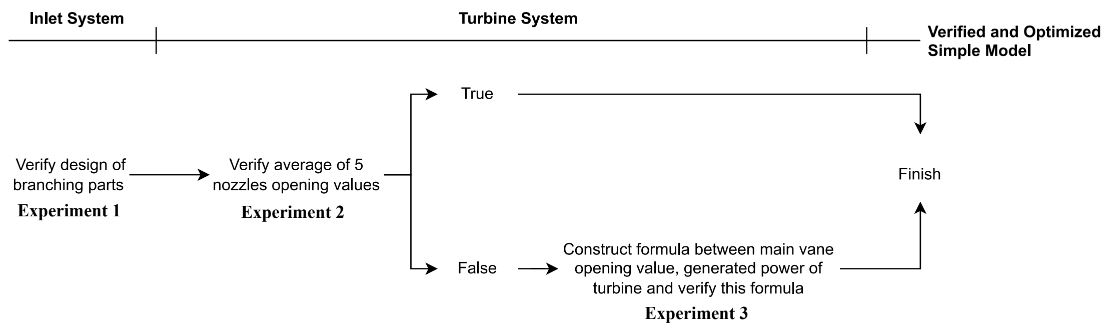

Inlet System
Turbine System
Verified and Optimized Simple Model
True
Verify design of branching parts
Experiment 1
Verify average of 5 nozzles opening values
Experiment 2
Finish
False
Construct formula between main vane opening value, generated power of turbine and verify this formula
Experiment 3

Figure 6. Sequence of experiments for simple model

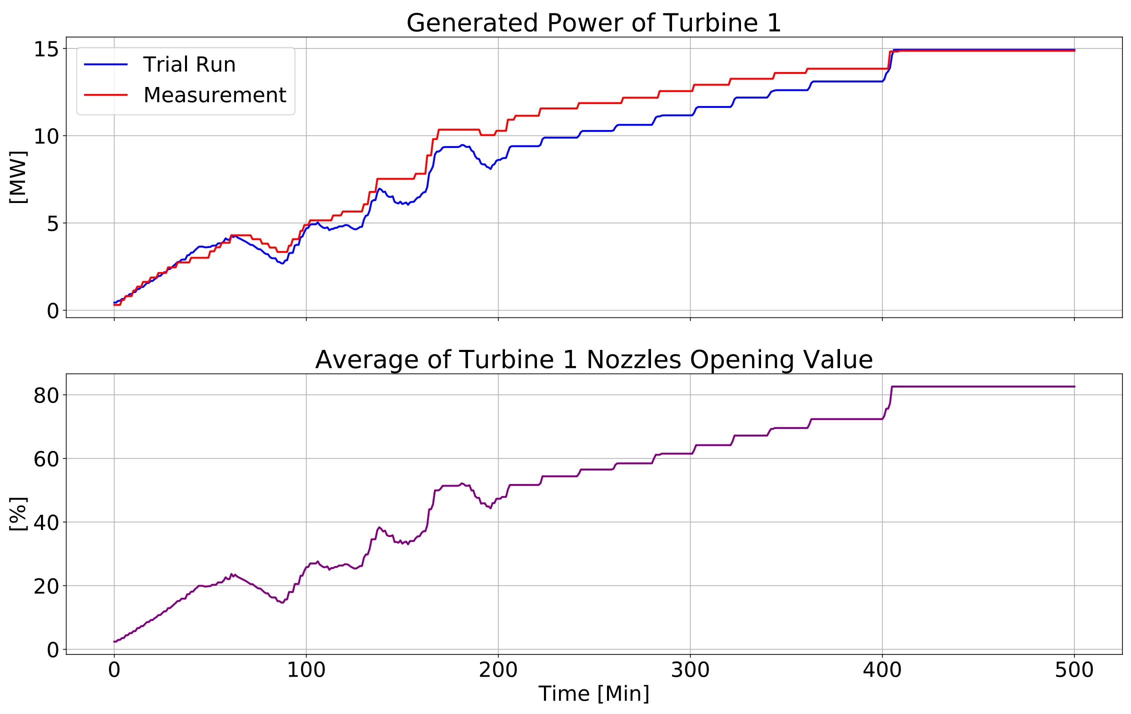


Figure 7. Comparison of the model simulation results and measurement values against change in averages of nozzle opening values

blocks named "T1" and "T2" are used to represent turbines without using turbine block models in the experiment since these models have not been verified. These blocks are used to input the measured flow values from the block "Data". In Design 2, the parameter of component "Fitting" representing for the branching point is set up according to the diameter of "Conduit" and "IntakeT1_1" that have the same diameter. The type of *Square* is used for this component. In Design 3, the parameter of component "BendPipe" is setup according to the diameter of "Conduit" and "IntakeT1_1". The authors in this paper recommended the loss coefficient parameter as 10 according to the lack of information from the manufacture.

Interpreting experimental results and conclusions The experimental results are collected and plotted in Figure 9.

According to Figure 9, the water pressure at inlet T2 varies due to the change of flow rate and it can be easily seen that the Design 3 shows the simulation results are nearly same as the measurement values as reference values in this experiment. Therefore, Design 3 with "bendPipe" component is considered the most optimal for branching part representation and also the model design of inlet system. The Design 3 will be used for the following experiments.

6.2 Experiment 2

Objectives recognition The objective of experiment is to verify the method of using average of five nozzles opening value as the input signal of the T1 model.

Selection of response The response of interest for the experiment is the generated power of T1.

Selection of process variables The process variables in this experiment are five nozzles opening values of T1 in

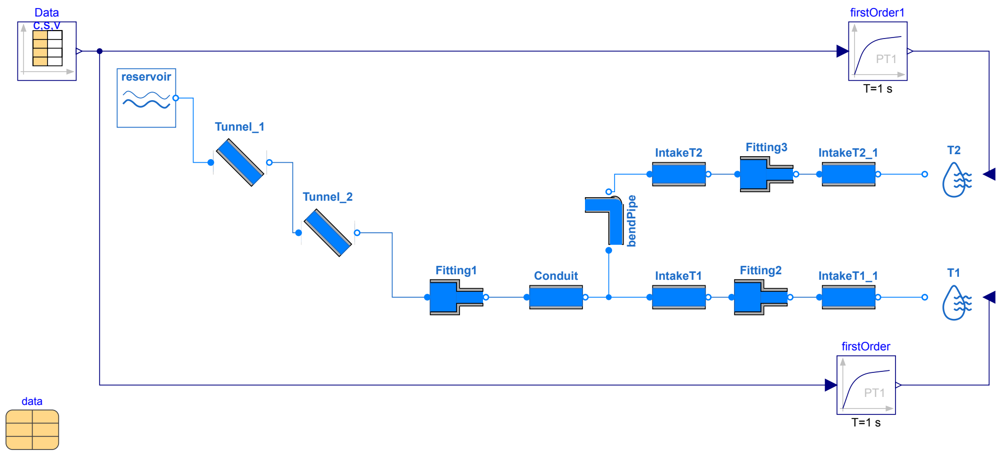


Figure 8. Experiment 1, simulation model

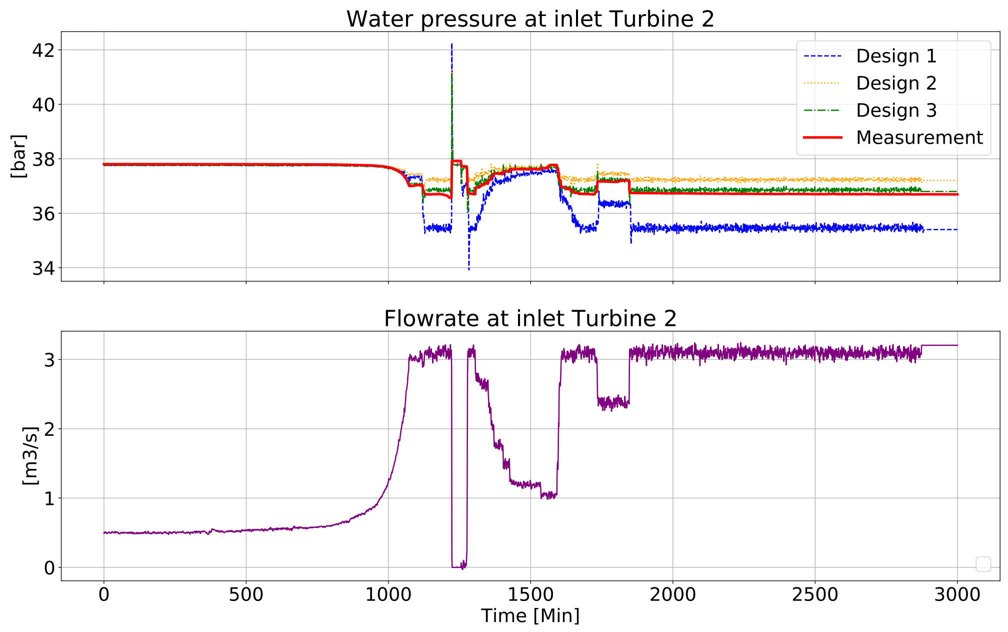


Figure 9. Experiment 1, comparison of the results of three model designs and measurement values against change in flow rate

dataset 2.

Performing the experiment The experiment is implemented same as Figure 5 with the inlet system as Design 3 in experiment 1 is used.

Interpreting experimental results and conclusions The experimental results are collected and plotted in Figure 10.

According to the results showing Figure 10, it illustrates the method of using average of five nozzles opening value as the input signal of the T1 model does not yield the desired improvement between the simulation results and the reference value for generated power of T1 when compared with the simulation results of the *Trial Run*. Therefore, the method of using average of 5 nozzles opening value is not suitable for the input values of the turbine model.

6.3 Experiment 3

Objectives recognition The following are the objectives of the experiment:

- to develop a mathematical model which relates generated power of turbine and the input signal value of turbine block, the main vane opening value
- to verify the mathematical model built on the simple model

Selection of response The response of interest for the experiment is the generated power of T1.

Selection of process variables The variables of this experiment are following

- The main vane opening value
- The generated power of T1

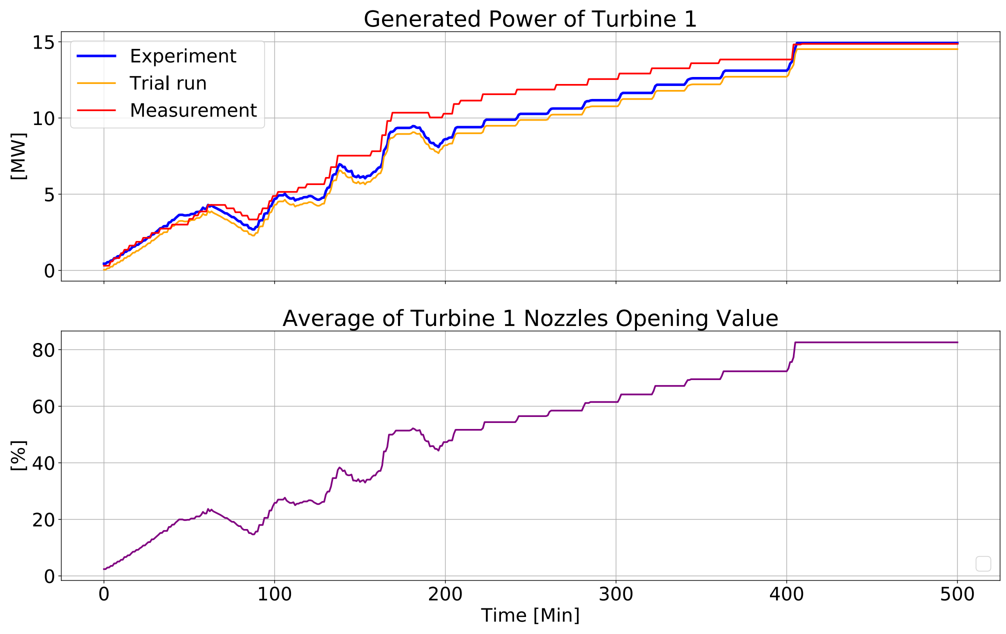


Figure 10. Experiment 2, comparison of the results of simulation model and measurement values against change in average of nozzles opening values

Performing the experiment The model to implement this experiment is same as that of Experiment 2. The implementation of building the equation showing the relationship between the input value of the turbine model, the main vane opening value, and the generated power of T1 can be performed in three steps:

Step 1: Run the simulation with the main vane opening value from 0% to 100% and obtain the generated power of T1 values.

Step 2: Export the data from simulation under CSV (Comma-seperated values) format and import this file into Python (*Welcome to Python.org* 2022) environment via pandas (*pandas - Python Data Analysis Library* 2022) to analysis the data through Python commands.

Step 3: Apply sklearn.linear_model.LinearRegression package (*sklearn.linear_model.LinearRegression* 2022) to create the mathematical model under the format of Equation 1.

According to these steps, the mathematical model (Equation 2) showing the relationship between generated power and main vane opening value is constructed. The generated power of T1 in *dataset 2* is used to calculate the opening value of main vane. Accordingly, this mathematical model is built in Dymola/Modelica as the component named "Uv_T1" and applied in the simple model as Figure 11.

Interpreting experimental results and conclusions Mathematical model:

$$y = -0.003 + 5.57 \cdot 10^{-8} x \tag{2}$$

Where:

y	Main vane opening value [%]
x	Generated power of turbine 1 [MW]

According to the results showing Figure 12, it can be easily seen that the experimental and measurement values of generated power of T1 are similar. Therefore, the mathematical equation of generated power and main vane opening value is accurate.

7 Conclusions

The paper presents a simple hydropower plant model set up on Dymola/Modelica based on OpenHPL which is a specialised library to model a real hydropower plant. In the process of setting up a simulation model according to the elements in the library, the design model needs to be verified and optimised to suit the actual structure of a hydropower plant. This makes it difficult to choose the right design elements to optimise for the model. DoE is an distinct method in order to simplify the optimal solution for simulation model design based on experiments for the model. The principles and experimental steps of DoE are outlined and applied to a typical hydropower plant simulation model. With the available measurement data each portion of the model was simulated in turn to verify and optimise the design as well as eliminate noise factors. The result of the series of experiments and the completed simulation model will be used for research and further study cases.

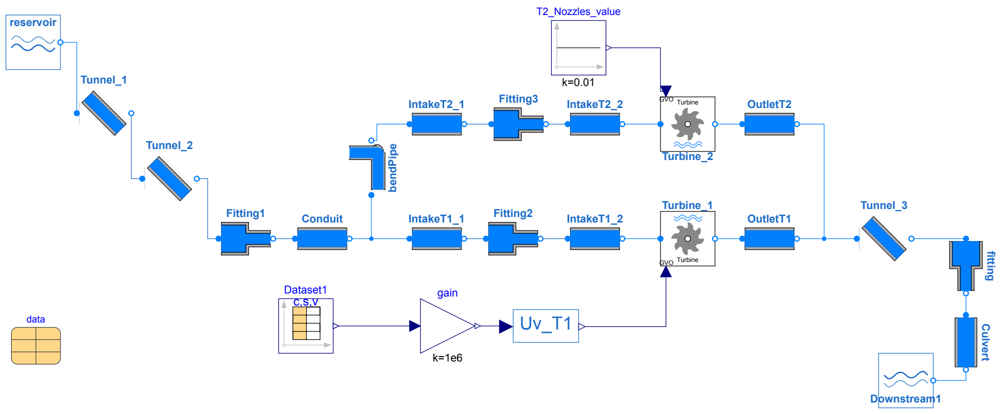


Figure 11. Experiment 3, simulation model

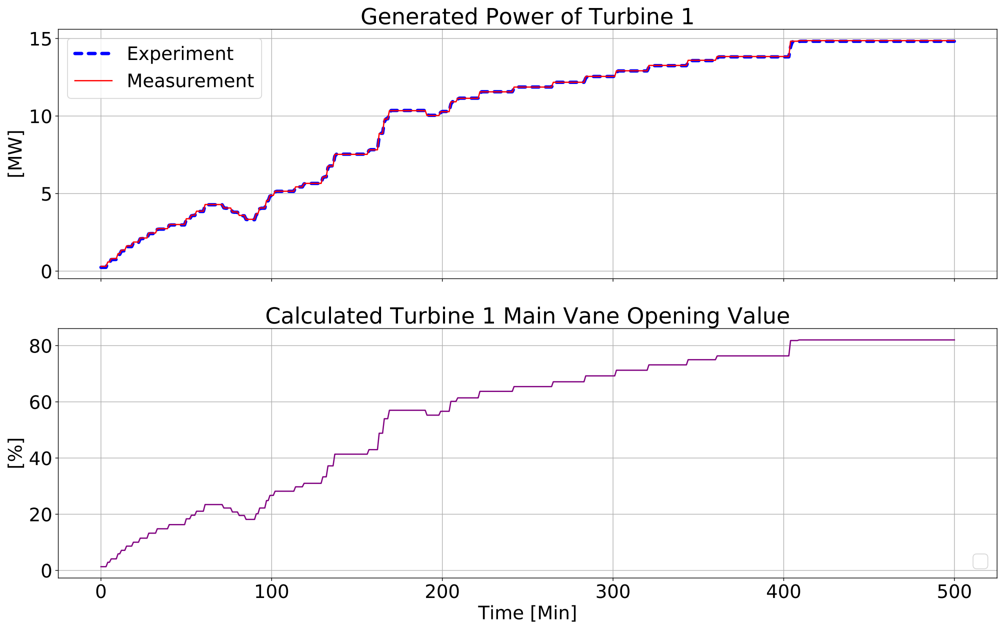


Figure 12. Experiment 3, comparison of the results of simulation model and measurement values against change in calculated main vane opening value

In practical situations of many complex systems with complicated chain of parts, these systems are commonly simulated by different simulation tools or software under vast amount data of operational data. However, the verification and optimisation of these simulation models always play an important role in studying characteristic of systems. This paper contributed to a simple solution to verify and optimise various type of simulation models in the future.

Acknowledgements

This paper is the result of a Master's Thesis of Le Nam Hai Pham, with the title “Application of Design of Experiments for the Verification of a Hydropower Plant”.

References

Antony, Jiju (2014). *Design of Experiments for Engineers and Scientists*. 2nd edition. Elsevier Insights. London: Elsevier. 208 pp. ISBN: 978-0-08-099417-8. URL: https://doi.org/10.1016/C2012-0-03558-2.

Brownlee, Jason (2016-03-24). *Linear Regression for Machine Learning*. Machine Learning Mastery. URL: https://machinelearningmastery.com/linear-regression-for-machine-learning/ (visited on 2022-04-12).

eMathZone (2014-11-02). *Basic Principles of Experimental Designs | eMathZone*. URL: https://www.emathzone.com/tutorials/basic-statistics/basic-principles-of-experimental-designs.html (visited on 2022-04-12).

Emily Divis et al. (2020-08-31). *Randomization: A Core Principle of DOE*. STAT COE-Report-03-2020. STAT Center of Excellence. URL: https://www.afit.edu/stat/statcoe_files/1001AFIT2020ENS09115%201001divi%202-2.pdf.

Energy, Ministry of Petroleum and (2016-05). *Renewable energy production in Norway*. en-GB. Redaksjonellartikkel. Publisher: regjeringen.no. URL: https://www.regjeringen.no/en/topics/energy/renewable-energy/renewable-energy-production-in-norway/id2343462/ (visited on 2022-08-08).

International Energy Agency (2022). *Hydropower Analysis*. IEA. URL: https://www.iea.org/reports/hydropower (visited on 2022-04-12).

Jankovic, Aleksandar, Gaurav Chaudhary, and Francesco Goia (2021-11). "Designing the Design of Experiments (DOE) An Investigation on the Influence of Different Factorial Designs on the Characterization of Complex Systems". In: *Energy and Buildings* 250, p. 111298. ISSN: 03787788. DOI: 10.1016/j.enbuild.2021.111298.

Kleijnen, Jack PC (1995). "Verification and validation of simulation models". In: *European journal of operational research* 82.1, pp. 145–162.

Modelica Association (2021-02). *Modelica a Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.5*. Linköping: Modelica Association. URL: https://specification.modelica.org.

MoreSteam (2022). *Design of Experiments (DOE) Tutorial*. URL: https://www.moresteam.com/toolbox/design-of-experiments.cfm (visited on 2022-04-12).

OpenSimHub (2022-04-01). *OpenHPL*. OpenSimHub. URL: https://github.com/OpenSimHub/OpenHPL (visited on 2022-04-12).

pandas - Python Data Analysis Library (2022). URL: https://pandas.pydata.org/ (visited on 2022-05-05).

Sargent, Robert G (2008). "Verification and validation of simulation models". In: *2008 Winter Simulation Conference*. IEEE, pp. 157–169.

sklearn.linear_model.LinearRegression (2022). en. URL: https://scikit-learn/stable/modules/generated/sklearn.linear_model.LinearRegression.html (visited on 2022-08-13).

Vytvytskyi, Liubomyr (2019). "Dynamics and model analysis of hydropower systems". In: *Doctoral dissertations at the University of South-Eastern Norway; 37*.

Welcome to Python.org (2022). en. URL: https://www.python.org/ (visited on 2022-07-28).

Using Multi-Physics Simulation to Estimate Energy Flexibility for Local Demand Response Strategies in a Microgrid

Iker Landa del Barrio[1,*] Julen Cestero[1] Marco Quartulli[1] Igor G. Olaizola[1] Naiara Aginako[2] Juan José Ugartemendia[3]

[1]Department of Data Intelligence for Energy and Industrial Processes, Vicomtech, Donostia-San Sebastián, Gipuzkoa, Spain
[2]Department of Computer Science and Artificial Intelligence, Faculty of Informatics, University of the Basque Country (UPV/EHU), Donostia-San Sebastián, Gipuzkoa, Spain
[3]Department of Systems Engineering and Control, Faculty of Engineering of Gipuzkoa, University of the Basque Country (UPV/EHU), Faculty of Informatics, Donostia-San Sebastián, Gipuzkoa, Spain
[*]`Corresponding author, email: ilanda@vicomtech.org`

Abstract

This work discusses the development of a multi-physics simulated model, in the frame of the decarbonization and energy efficiency objectives of the European Commission. Its central feature is the interconnection, through a micro-grid, of a distributed PV installation and of several electric dispatchable loads, thus powering a Collective Self-Consumption network. The simulator presented within this document aims to serve as a technological enabler for the design and testing of On-Site DR strategies, which actuate directly on the connection status of the loads, before their deployment on the target, real-world systems. The simulator supports the design and validation of such strategies by generating realistic simulated data of certain loads that present monitoring difficulties, taking into account online, real external weather conditions. All the elements described and modeled in the current work belong to a real-world installation, which is a university campus —ESTIA, Bidart, France— composed by several buildings with DER.

Keywords: Demand Response (DR), Distributed Energy Resources (DER), data generation, energy demand disaggregation, microgrids, multi-physics simulator

1 Introduction

The non-dispatchable nature of renewable sources usually leads to remarkable differences between generation and demand profiles in microgrids and general power grids. These differences must be solved somehow, as a uninterruptible balance between generation and demand must be ensured at all times, mainly by the EMS ruling these grids. Historically, the most common solution to this issue is to use storage systems in order to shift the time of consumption of the energy generated in the grid, giving the system a limited, yet dispatchable, energy source (Jurasz et al., 2020), (Angenendt et al., 2019), (Marańda, 2019).

The renewable generation cannot be largely modified, since these power generators are mostly dependant of weather conditions, which present a stochastic behaviour. The use of short-term energy buffering in storage systems is studied by Marańda *et al.* (Marańda, 2019), for its application into different scenarios. The figure 1 depicts the typical generation and demand profiles —$P_P(t)$ and $P_L(t)$ respectively— of residential PV installations. In the case depicted in (b) chart, the storage is considered to cause the buffering of the energy surplus for later use. Following the notation proposed by Marańda *et al.*, E_B^- is the amount of energy absorbed by the storage, which is consumed by the system when needed. Contrarily, the supply of previously stored energy into the system is referred as E_B^+.

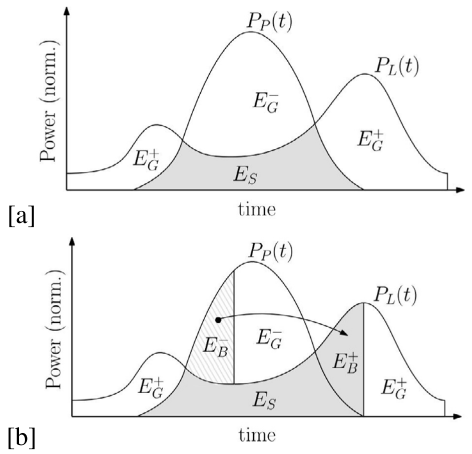


Figure 1. Typical generation and demand profiles of residential applications with (b) and without (a) energy storage ((Marańda, 2019))

It is previewed that the deployment of smart systems and the Internet of Things (IoT) will largely contribute to the success of DR programs, since this technology is previewed to allow the remote control of electrical loads (Pop et al., 2018). Duman *et al.* (Duman et al.) and Romanchenko *et al.* (Romanchenko et al., 2021) analyze the impact of DR actions achieved through deviations of the set-point of the so-called smart thermostats. Li *et al.* present an optimal management through DR actions of multi-stakeholder systems in low-carbon communities, considering the carbon emission restrictions through the carbon tax (Li and Yu, 2020).

However, all the mentioned techniques are designed to influence the demand on a macro-scale, or aggregating several end-users. Micro-scale is not the most common aim of the existing DR programs. To overcome the engagement issues of residential DR, some EMSs are designed to shape the performance of the loads, with previous agreement of the user, actuating directly over them to follow more precisely the actions given by the management strategies. These strategies usually generate a schedule for each day and for each load of the system, synchronizing them to reach certain energy and money saving objectives (Vahedipour-Dahraie et al., 2020). The EMS used for these cases computes certain parameters of the loads like ToU (Time of Use), power, etc. Also some user defined parameters, in order to respect the comfort of the users and evaluate the available energy resources to manage the demand levels, scheduling the loads' activation (Mohsenian-Rad et al.).

Nolan *et al.* point the main barriers and challenges for the deployment of DR programs in their study, giving a notorious importance to the modeling of the physical characteristics of such strategies (Nolan and O'Malley). Another main barrier in this study is the lack of data regarding the disaggregated consumption of some dispatchable loads. This is mainly due to the fact that the energy consumption is usually measured at the installations point of interconnection (POI) with the general grid, while each loads consumption is usually left unmonitored. This is a crucial information for EMS algorithms that need to evaluate at any time the amount of power available to whether increase or decrease the demand levels in order to produce the required DR actions. This is explored by Azari *et al.* focusing onto the data uncertainty of several DR programs (Azari et al.).Through the use of the simulator presented within this document, we aim to produce the missing data heuristically, thus overcoming the lack of data mentioned before, which is considered a barrier to deploy some DR programs successfully.

In summary, the state of the art in the available literature mainly refers to macro-scale DR strategies evaluated on off-line, fixed weather conditions. A clear gap in this state of the art refers therefore to generic simulation environments, taking into account online weather conditions, aimed at the benchmarking and optimization of generic algorithms for DR, focusing in particular on micro-scale, direct load management. The present contribution has the ambition to address this gap in the literature, by quantitatively validating the performance of the approach presented here, demonstrating the value of micro-scale, online-weather, simulation-based benchmarking and tuning environments for direct load management strategies in a specific real-world case. Furthermore, since the thermal devices are external conditions dependant, the present simulation environment allows to evaluate dynamically the energy flexibility that can be achieved through deviations of the temperature set-point.

2 Description of the Scenario

This study analyses the case of a certain university campus and its electric system interconnecting energy generators and loads. The *École d'ingénieurs ESTIA* is located in a technology campus called *Technopole Izarbel* (see figure 2), located on the outskirts of Bidart, near Biarritz and Bayonne, in South-West France.

Figure 2. *Technopole Izarbel* technology campus.

Considering the structure of a microgrid, we can conceive the campus electric system as a microgrid, since it has its own generation and consumption systems, both connected by a distribution network. There are several buildings (called ESTIA1, 2 and 3) that need to be supplied with electric power. These three buildings are foreseen to have their own generation too; so, at some point, the power generated by their generators could be shared between each other, when the EMS of any of the buildings measures a generation surplus, forming a CSC network in the campus. Following the economic constraint mentioned before, the energy is preferentially used by another building of the CSC network instead of injecting it to the general grid. However, we must note that the energy transactions between buildings is also taxed in the French system, since these transactions use the general grid's distribution network to share the energy between buildings. Thus, the energy should be used preferentially by the building where the energy has been generated. However, these taxes belong to a special category —conceived for CSC networks— with reduced taxes, avoiding to constrain the profitability of these CSC networks by its users.

2.1 Energy production

The described campus production relies on a local renewable energy source, as mentioned before, which is a PV installation. This installation is composed, at the moment of this study, by **32** PV modules with a maximum rated power of **175 W** per module. We could not be provided with further information regarding the characteristics of the certain model of the mentioned modules, inasmuch as this model is not available in the market anymore. Thus, we have used a commercially available PV model that is due to be installed in the future on the campus and whose technical characteristics are provided by the manufacturer. This model is **TARKA 120 VSMS 300** which presents a maximum power — on NOCT conditions — of **244.9 W**.

2.2 Energy demand

ESTIA does not have a very different energy demand profile from an office building, which is composed of typical electrical devices like space heating, lighting etc. The buildings' installation is not equipped with smart meters for each load type, just a general meter at the grid POI and another one dedicated to PV generation. Therefore, it is impossible to find, by measuring methods, which fraction of the total energy demand is requested from each load group (lighting, air conditioning, etc.).

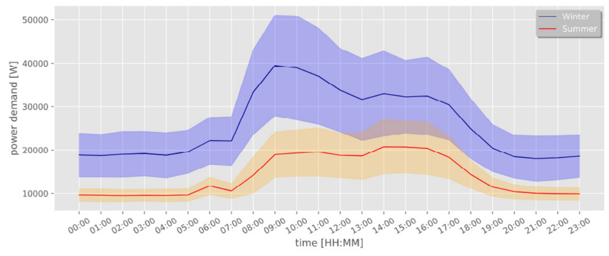


Figure 3. ESTIA's mean power demand profiles with shaded standard deviations' range.

It must be noted that the space heating and cooling of this campus are powered by electric air-source heat pumps and not by a natural gas-based system. Thus, as mentioned, both heating and cooling systems rely on electric energy to produce the required heat transfer rates to maintain the indoor air between certain comfort ranges.

The datasets used in this project were obtained from ESTIA1 building's general smart meter, regarding the general demand of the building during two separated periods, with a step time of 1 hour. More specifically, the winter dataset comprises data from January and February 2020, while the summer dataset regards July, August and September 2020. As seen in figure 3, we have computed the average value and standard deviation of each hour, making the distinction between summer and winter periods. Also, we must note that due to the limited available data we will only focus on the modeling of ESTIA1 building.

2.2.1 Dispatchable loads

From a DR perspective, we must identify the loads that could be disconnected from the supply at certain moments, in order to shape the behavior of the building's general demand without disrupting the users' comfort. The so-called dispatchable loads are those that, when required, can totally or partially reduce or increase their energy demand.

In the case of ESTIA1, two loads have been identified as the candidates to perform the DR actions, due to its large power demand and controllability. These loads are: HVAC, water heating system and an electric car battery (BT) charger.

Load Type	Power [kW]	ToU [h]
HVAC	15	Variable
Electric Car BT Charger	11	3
Water Heating System	1.2	Variable

Table 1. Dispatchable loads' power and ToU.

The ToU of the BT charger is variable between different car models, depending on their BT type and capacity. Also, the charging process of the BTs does not usually have a constant power profile, but again, a constant profile has been used for simplification matters. Instead, the BTs are usually charged with a constant current profile, followed by a constant voltage charge once the BT voltage has reached the set voltage. So, we have assumed an indicative ToU of **3 h** and a constant charging power of **11 kW**, this one provided by ESTIA.

As we can see in table 1, the ToU of the HVAC system is denoted as variable. This is because the behaviour of these systems depends on the thermal losses of the building, which also depend on the external temperature, a stochastic variable. The control of the heating systems is typically a thermostatic ON/OFF control system with hysteresis thresholding, whose actuation will also vary along with the thermal losses of each day. Furthermore, their thermal contribution varies along with external temperature too. Besides, the HVAC system relies on an air-source heat pump to whether heat or cool the space. Due to this heat recovery from the exterior, its electric consumption is external air temperature dependant, with a maximum consumption at 15 kW.

3 Modeling of the energy systems

In the present project, we have developed a simulated model of the energy system described in the previous section. The model has been generated using *OpenModelica*, an open source simulated environment, mainly powered by the *Modelica* modeling language, also using the libraries Buildings (Wetter, 2009), PhotoVoltaics (Brkic et al., 2019), and OpenModelica native PVSystems (OpenModelica, 2021). The model, once generated, is subjected to different external conditions (solar irradiation, external air temperature and wind speed), in order to design and test the DR strategies.

3.1 PV generation

The PV generation is estimated by its own model separated from the main model, which includes the major part of the energy system of the campus, with the aim of lightening the execution of the main model. This simulated model regarding the PV installation is shown in figure 4. In this model, we are simulating the production of a single module connected to the 400 V side microgrid of the campus via an ideal DC/DC converter, as seen in the diagram.

The PV module block computes two inputs to calculate the electrical power produced by them, which are solar irradiance and cell temperature. Solar irradiance is provided by local weather station, while cell temperature is determined computing both external air temperature and wind speed, also provided by local weather station. The cell temperature calculation follows the model presented by Duffie and Beckman in (John A. Duffie, 2013), which is the most accurate model of the analysis made by Yang *et al.* in (Yang et al., 2019) and shown in the equation 1:

$$T_c = T_{air} + \frac{G}{G_{NOCT}} \cdot \frac{9.5}{5.7 + 3.8 \cdot V_{wind}} \cdot \frac{T_{c_{NOCT}}}{T_{a_{NOCT}}} \cdot \left(1 - \frac{\eta_c}{\tau\alpha}\right) \quad (1)$$

where $\mathbf{T_c}$ is cell temperature [°C], $\mathbf{G}$ is irradiance [Wm^2], $\mathbf{G_{NOCT}}$ is irradiance at NOCT conditions —Nominal Operating Cell Temperature— [Wm^2], $\mathbf{V_{wind}}$ is wind speed [ms], $\mathbf{T_{c_{NOCT}}}$ and $\mathbf{T_{a_{NOCT}}}$ are cell and ambient temperatures respectively at NOCT conditions [ºC], η_c is the conversion efficiency of the PV module and τ_α is the product of transmittance-absorbance. The value of these parameters, for the PV module used in the simulator, is shown in table 3 of the Appendix.

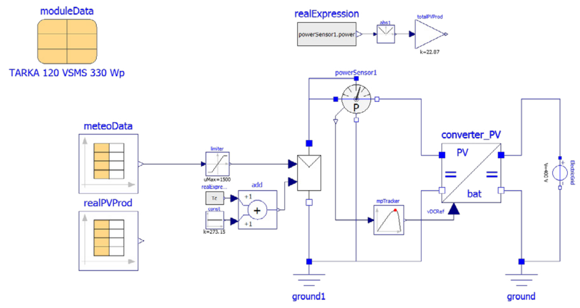


Figure 4. Block diagram of the PV generation model

Once T_c is determined, the PV module block is fed with this data —in Kelvin— and the irradiance from the *meteoData* dataset. The irradiance is limited to a defined range —[0, 1500]— previous to the connection with the PV module, in order to avoid measurement errors, which, in the case of negative values, could cause an error in the electric section. Then, the PV module block injects an electric current into the circuit depending not only on the mentioned inputs but on the voltage on its terminals too. Thus, we are using a *mpTracker*, along with a power sensor, aiming to find the Maximum Power Point (MPP) at any time, which changes along with the weather conditions. The MPP is a certain voltage value for each weather state, therefore the controller aims to track this value through the whole voltage range. Even though there are many ways to produce this tracking, the tracker used in this model is based on the widely used Perturb and Observe technique (Putri et al., 2015).

3.2 Microgrid structure

The electric section is the one in charge of simulating the main loads of the system and, additionally, the energy supply of these loads. This energy supply comes from two different and compatible sources. On the one hand, the microgrid has a distributed energy source with the PV modules and, on the other hand, the system's energy supply also relies on a POI with the general electric grid. We assume that the energy can flow in both directions of this POI, whether demanding energy from the grid or injecting the surplus of the PV generation, when necessary. Both sources and the whole electric section are displayed in detail on the figure 5.

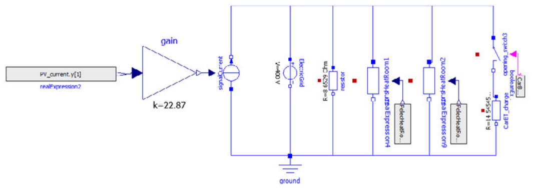


Figure 5. Block diagram of the electric section of the model

Due to computational matters PV production is simulated in a separated model, as explained in previous section. Therefore, we use the results obtained in the microgrid side of the PV installation model into the general model of the installation. Since we are using the same voltage —400 V— in the mentioned microgrid side of the PV model and the microgrid of the general model, we can just use a current source fed with the obtained data. This is achieved through a .txt file that is generated using Python and the *combiTimeTable* Modelica block, which is able to read such file.

The actual electric system uses 50 Hz alternated current (AC), and the power converters required for an AC system normally work at a rate of 22 kHZ, or higher. Also, the iteration frequency of the simulation is recommended to be set 10 times bigger than the highest frequency of the system, which would suppose to work with a time step of $T_{SIM} = 4.\overline{54} \cdot 10^{-6}$ s. This fact results in an unpractical high computational cost when simulating the system in the order of days, weeks or months, as desired for this study.

Since the analysis of the dynamics of the system is not the aim of this study, the simulated microgrid has been set as a direct current (DC) circuit, allowing to set a time step as high as 15 s. Thus, the results obtained by the simulator regard the steady state of the system, which is fully valid at the context of this study.

We have selected a voltage of **400 V**, being this value the typical RMS voltage value of the European three-phase grid connection, and assuming that ESTIA is connected to the general grid using this connection type. Based on the fixed voltage of the microgrid, each load has been modeled using a fixed resistor (equivalent to the nominal power consumption of the loads) connected to the

DC circuit and it is activated or deactivated using an opening or closing switch, depending on the case. The value of each equivalent resistor has been obtained combining the Ohm's Law and the electric power calculation, getting as result equation 2:

$$R = \frac{V}{I} = \frac{V^2}{P} \tag{2}$$

where **V** is constant voltage [V], **R** is resistance [Ω], **I** is constant current [A], and **P** is electric power [W].

We are representing the consumption of all the loads taken into account in this study, which are: the dispatchable loads shown in table 1 —with two HVAC systems due to its use in two different spaces—, and a fixed resistor representing resting fixed consumption, whose calculation is defined next.

As it might be seen in figure 3, the demand average curve never crosses a certain lower threshold. This can be clearly seen at nights, where the demand is closer to this threshold. This lower threshold in the demand can be due to certain parasitic loads that are not switched off during nights or weekends, when the building is empty. We included a fixed resistor to simulate this effect, equivalent to this fixed consumption, computing the average value of the time where the building is unused —from 19 to 6 on weekdays and the 24 hours of weekends and holidays—. We use the same expression as for the rest of the loads to calculate the equivalent resistor (equation 2).

3.3 Physical environment of the loads

As mentioned previously, we have considered three load types. Between them, the two heaters consume electric energy to convert it into thermal energy, with the difference that the HVAC relies on a heat recovery system to leverage the thermal energy contained in the external air. Contrarily, the BT charger consumes electric energy from the circuit to subsequently inject it into the car's BT, after converting and adapting the electric supply to certain conditions. We have considered the charger as a constant power consumption that can be switched on and off when needed, so, it can be modeled with a fixed resistor controlled by a switch to model the dispatchable consumption.

In order to model the control strategy of the heating systems, we have used the so-called **hysteresis thresholding**. This control technique switches the controlled thermal devices on and off, aiming to keep the indoor temperature inside a defined range that can be set according to the user's preferences. This type of control avoids a continuous intermittent switching on and off by setting a temperature range sufficiently wide that allows the temperature to increase or decrease before switching the system again. Thus, the temperature of the plant will cycle between the hysteresis upper and lower limits, ideally never surpassing them. This, in addition to avoiding a continuous and relatively fast switching, saves energy while it ensures the users comfort requirements.

3.3.1 HVAC System's environment

The HVAC system actuation depend on the external weather conditions, both in terms of thermal contribution and electric consumption. Therefore, we included the influence of the external air into the model. For the internal section, we can find *room1* and *room2* block simulating the internal air closed volume of two separated spaces of the building, along with a *heatCapacitor* for each of them, simulating the thermal capacitance of any object in contact with the internal air. Each *room* block has been set with a rough approximation of the volume of the spaces. This approximation has been made using satellite images to measure the side lengths of the building (65x32 m). Besides, assuming a 3 m floor height, we got a volume of **6 560 m^3** for each *room*.

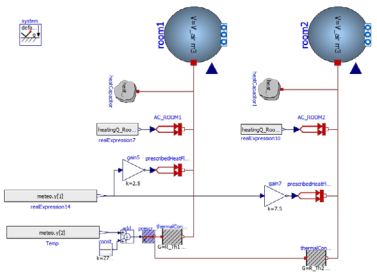


Figure 6. HVAC system's environment

There are three thermal sources associated to each *room*. The upper heat source —AC_{ROOM_x}— represents the main source, which is the HVAC contribution of each room, independent between them. The thermal contribution is interpolated through the data presented in the appendix tables 4 and 5, obtained from (Priarone et al., 2020), which represent the performance of a certain air-source heat pump. Thus, for each external temperature we get a different thermal contribution, along with a electric consumption that is represented in the electric circuit of the model, related to the Coefficient Of Performance (COP) of the HVAC system. The second thermal contribution is related to the irradiance coming from the sun, which is scaled for each room, aiming to simulate different orientations of the rooms. The third contribution comes from the thermal losses with the exterior. Here we compute the external temperature data in K, connecting it with the thermal circuit through a *thermalConductor* block that represents the thermal insulator of the building's walls. The value of the convection constant G has been assumed taking a fixed amount of heat power losses (1 000 W) at a certain temperature difference with the exterior (3 K).

4 Environment Programming Interface

The tool presented within this document aims to serve as a intermediate step for any DR strategy previous to its deployment into the actual installations. These strategies are more likely to be developed in a programming language different from Modelica, such as Python or similar. Therefore, we identified the need for a programming infrastructure interconnecting the simulator with the mentioned strategies, which in the context of this project is referred as the Environment Programming Interface, illustrated in the figure 7.

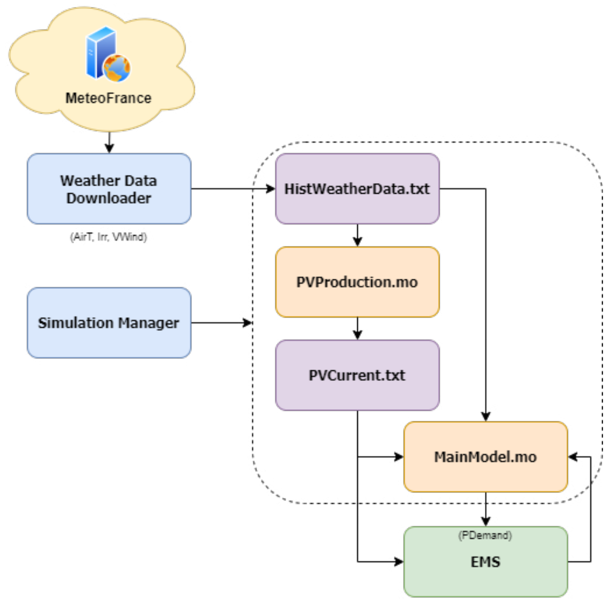


Figure 7. General flowchart of the Environment Programming Interface

As seen in the flowchart of the interface, the starting point is the online weather data obtained from local weather station, which could be historical or forecasted data, depending on the use of the simulator. The download of this data is made by the script *Weather Data Downloader*, which stores it into a text file to be read by the simulator through a *combiTimeTable* block. Once this data is stored into the text file, the script *Simulation Manager* controls the execution of the simulation models through the API conceived for such use called OMPython (Ganeson et al., 2012). Through this API we are able to send the parameters of the simulation to OpenModelica, execute the simulation, and store the results into a user defined CSV file. This file is subsequently read by the *Simulation Manager*, which extracts the estimated PV current to store it into a text file. Finally, the DR strategies contained in the EMS can be tested controlling the dispatchable loads contained in the *MainModel*.

5 Results and Validation

The simulator presented within this document aim to replicate the performance of the energy system analyzed in previous sections, in order to produce the missing data heuristically. Since we were provided with data regarding this installation, we can proceed to evaluate the correctness of this modeling, interpreting the results of the simulations while comparing them with real data of the energy installation.

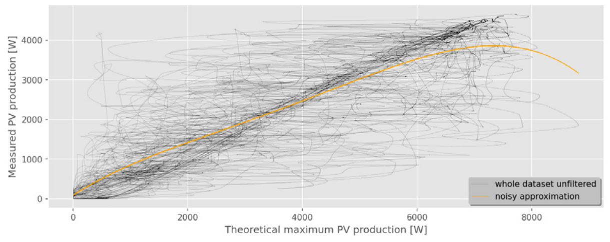
5.1 PV production

In the case of the PV production we are conscious about the differences between the estimation through simulation and the reality of the installation, since we are neglecting the losses due to: non-optimal orientation of the modules on the one hand, and electric inefficiencies on the other hand. Thus, a comparison with the real measured data is crucial to adapt our model more precisely to reality.

The data we are using to validate this model comprises PV production from the period between the 6^{th} April and 24^{th} May 2021. Having this dataset, we are executing a simulation of the PV model using weather data relative to the same period, to subsequently save the production data into a CSV file.

The principal aim of this process is to tune a direct relation between data obtained through simulation and real measured data. This is needed to complete the PV simulator since, as mentioned before, the simulator estimates the maximum theoretical production for given weather conditions, neglecting certain power losses. Thus, in order to produce a direct relation, the methodology applied was to tune a polynomial curve that relates the production obtained through simulation and the real data.

Since the simulations are close to be continuous processes, the data obtained from them are close to be continuous too. Thus, each line of the figure 8 represents the trajectory of the PV production power through a whole day. However, as it can be observed in the graph, the related data present high levels of noise. As is evident from figure 8a, sudden changes occur in the production, since there are many lines with horizontal and vertical sectors. This is due to changes in one variable that are not reflected in the other one, which produce the sudden changes in the trajectories. These changes in the production data, whether measured or simulated, must be due to differences in weather conditions too. The only weather variable able to present such high changes is the irradiation, which is highly affected by clouds. Furthermore, the used weather data does not belong to the exact location of the modules —it is measured by a station less than 10 km away—, thus, the clouds would not cover the sunlight the same way in both locations, causing the differences we are observing here. Finally, we tried to fit a polynomial curve to this noisy dataset, getting as result the orange curve of the figure 8a. This approximation can be visually discarded, since it does not follow the pattern of the major part of the trajectories.



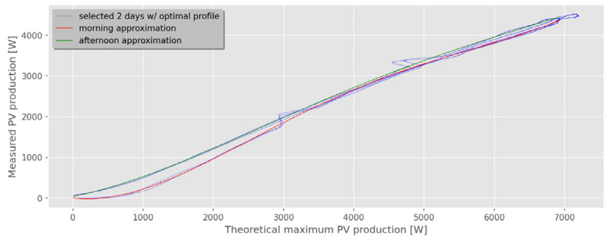


Figure 8. Correlation between estimated and measured PV production. a: using the whole dataset unfiltered, b: selecting two days with *optimal* production profiles

Considering the clouds a highly chaotic system, instead of taking its influence into account to tune the polynomial curve, we selected several *optimal* days where the production curves are not distorted by the clouds. These *optimal* production curves are shown in the figure 8b, where we can see that they follow the pattern of the unfiltered dataset mentioned before. Another fact to mention is that every *optimal* day used in this graph presented the same two distortion points —located around 3000 and 5000 W respectively in the horizontal axis—, where there is a sudden change on each of them too. Nevertheless, due to its repetition on different days' patterns, we concluded that they are the result of other objects' or buildings' shadows into the PV modules or the irradiance sensors. Splitting the dataset into morning (first half of each day) and afternoon, and using a 4^{th} degree polynomial fitting, we get the curves presented in the figure 8b, which follow precisely the mentioned pattern. These curves form a function defined in two parts, which are related to morning and afternoon periods and presented in the next expression:

$$\begin{aligned} f(x,h) &= A_P + B_P(C_P + D_P \cdot x) + E_P(C_P + D_P \cdot x)^2 \\ &+ F_P(C_P + D_P \cdot x)^3 + G_P(C_P + D_P \cdot x)^4 \end{aligned} \quad (3)$$

where x is the estimated theoretical maximum production data, h is hour, and the constant parameters (A_P, G_P) are defined in table 2.

It must be noted that the obtained function is tuned for the period with available data, which is April and May 2021. For a more precise function we should use a dataset comprising the production of the whole year, since the sunlight pattern changes along with the period of the year. However, the period used to tune this function is one of the most productive periods of the year in this certain climate zone, due to high irradiation and cool temperatures. Therefore, it is highly probable that any estimated value at any other time of the year belongs to the range of this function.

5.2 Dispatchable loads demand

One of the main objectives of the simulator was to reproduce the performance of certain electric loads. In this section we aim to validate this simulated performance through a comparison with real data. Anyway, as mentioned in previous sections, we cannot be provided with actual data of the modeled loads. Instead, we are comparing the data obtained through simulation with the general demand data. Having in mind that the HVAC systems usually require the major fraction of the energy consumed in buildings —around 67 % of the total energy demand of buildings in France (Enerdata, 2021)—, we can assume that its particular consumption is highly correlated with the general demand measurements.

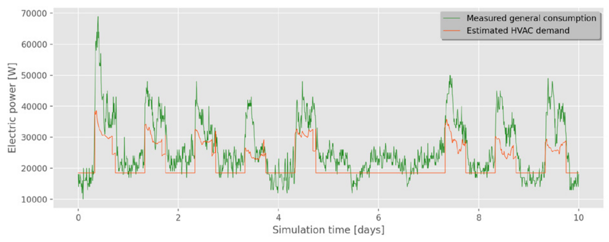


Figure 9. Estimated HVAC demand vs measured general demand comparison

In this case, we are executing a simulation of the general model of the installation, putting the focus onto HVAC demand. In this simulation we are using weather data of the same days as the general demand data of the actual installation, which is previewed to present a similar profile to the simulated data. Figure 9 displays this comparison, showing the data relative to 10 certain days between those with available data. These days belong to the period where the consumption is the highest, which is the winter period, as seen in figure 3. Furthermore, this period comprises several weekdays and a whole weekend, in order to simulate different scenarios. Also, we are adding a threshold to the HVAC consumption data, which is the average value of the consumption when the building is unused, as explained in section 3.2. As shown in figure 9, the general demand profile is highly influenced by the particular demand of the HVAC, as expected. We have computed the Pearson correlation coefficient between these two variables, giving as result $\rho_{\mathbf{Sim},\mathbf{Real}} = \mathbf{0.852}$, which represents a very strong correlation.

6 Future work

The simulation environment presented within this document aims to replicate the behavior of several loads of a campus. As mentioned, these loads are external conditions dependant as well as several constructive parameters such

P (Period)	A	B	C	D	E	F	G
Morning	2249.243	2818.728	-1.00297	0.000289	-926.235	-640.06	858.615
Afternoon	2332.426	2557.087	-1.00294	0.000288	-341.671	-353.593	258.558

Table 2. Parameters of the equation 3

as the thermal envelope resistance, and the building thermal capacitance. These parameters have been estimated for this implementation of the simulator. However, future work will deal with this issue, estimating these parameters through several on-site experiments which will involve Design of Experiments procedures along with analysis of the resulting data. This calibration process will be based on recent studies on this issue (Wüllhorst et al., 2022).

The results of the present study will be embedded in a OpenADR structure, estimating dynamically the energy flexibility offer for the local consumption controller, or Virtual End Note (VEN) using OpenADR terminology, with the constraint of respecting users comfort in such flexibility offer. The implementation of this structure is an ongoing study which is linked to the present work.

7 Acknowledgements

The authors render thanks to ESTIA and Enargia for providing the data used in this work regarding electric power demand and PV generation respectively.

The present work is part of EKATE, a project co-financed by the European Regional Development Fund (ERDF) through its Interreg Program of Spain-France-Andorra (POCTEFA 2014-2020). More information is available at `https://www.ekate.eu/`.

References

Georg Angenendt, Sebastian Zurmühlen, Fabian Rücker, Hendrik Axelsen, and Dirk Uwe Sauer. Optimization and operation of integrated homes with photovoltaic battery energy storage systems and power-to-heat coupling. *Energy Conversion and Management: X*, 1(February):100005, 2019. ISSN 25901745. doi:10.1016/j.ecmx.2019.100005. URL `https://doi.org/10.1016/j.ecmx.2019.100005`.

Delaram Azari, Shahab Shariat Torbaghan, Hans Cappon, Karel J. Keesman, Huub Rijnaarts, and Madeleine Gibescu. *Sustainable Energy, Grids and Networks*. ISSN 23524677. doi:10.1016/j.segan.2019.100262.

Jovan Brkic, Muaz Ceran, Mohamed Elmoghazy, Ramazan Kavlak, Anton Haumer, and Christian Kral. Open Source PhotoVoltaics Library for Systemic Investigations. *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, 157:41–50, 2019. doi:10.3384/ecp1915741.

A. Can Duman, Hamza Salih Erden, Ömer Gönül, and Önder Güler. *Sustainable Cities and Society*. ISSN 22106707. doi:10.1016/j.scs.2020.102639.

Enerdata. France energy efficiency & trends policies, 2021. URL `https://www.odyssee-mure.eu/publications/efficiency-trends-policies-profiles/france.html`.

Anand Kalaiarasi Ganeson, Peter Fritzon, Olena Rogovchenko, Adeel Asghar, Martin Sjölund, and Andreas Pfeiffer. An openmodelica python interface and its use in pysimulator. *Proceedings of the 9th International MODELICA Conference, September 3-5, 2012, Munich, Germany*, 76:537–548, 11 2012. doi:10.3384/ecp12076537.

William A. Beckman John A. Duffie. *System Thermal Calculations*, chapter 10, pages 422–446. John Wiley & Sons, Ltd, 2013. ISBN 9781118671603. doi:https://doi.org/10.1002/9781118671603.ch10. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118671603.ch10`.

J. Jurasz, F. A. Canales, A. Kies, M. Guezgouz, and A. Beluco. A review on the complementarity of renewable energy sources: Concept, metrics, application and future research directions. *Solar Energy*, 195(October 2019):703–724, 2020. ISSN 0038092X. doi:10.1016/j.solener.2019.11.087. URL `https://doi.org/10.1016/j.solener.2019.11.087`.

Longxi Li and Shiwei Yu. Optimal management of multi-stakeholder distributed energy systems in low-carbon communities considering demand response resources and carbon tax. *Sustainable Cities and Society*, 61(April):102230, 2020. ISSN 22106707. doi:10.1016/j.scs.2020.102230. URL `https://doi.org/10.1016/j.scs.2020.102230`.

Witold Marańda. Analysis of self-consumption of energy from grid-connected photovoltaic system for various load scenarios with short-term buffering. *SN Applied Sciences*, 1 (5):1–10, 2019. ISSN 2523-3963. doi:10.1007/s42452-019-0432-5. URL `https://doi.org/10.1007/s42452-019-0432-5`.

Amir Hamed Mohsenian-Rad, Vincent W.S. S Wong, Juri Jatskevich, Robert Schober, and Alberto Leon-Garcia. *IEEE Transactions on Smart Grid*. ISSN 19493053. doi:10.1109/TSG.2010.2089069.

Sheila Nolan and Mark O'Malley. ISSN 03062619.

OpenModelica. OpenModelica PVSystems, 2021. URL `https://build.openmodelica.org/Documentation/PVSystems.html`.

Claudia Pop, Tudor Cioara, Marcel Antal, Ionut Anghel, Ioan Salomie, and Massimo Bertoncini. Blockchain based decentralized management of demand response programs in smart energy grids. *Sensors (Switzerland)*, 18(1), 2018. ISSN 14248220. doi:10.3390/s18010162.

Antonella Priarone, Federico Silenzi, and Marco Fossa. Modelling heat pumps with variable EER and COP in energyplus: A case study applied to ground source and heat recovery heat pump systems. *Energies*, 13(4), 2020. ISSN 19961073. doi:10.3390/en13040794.

Ratna Ika Putri, Sapto Wibowo, and Muhamad Rifa'i. Maximum power point tracking for photovoltaic using incremental conductance method. *Energy Procedia*, 68:22–30, 2015. ISSN 18766102. doi:10.1016/j.egypro.2015.03.228.

Dmytro Romanchenko, Emil Nyholm, Mikael Odenberger, and Filip Johnsson. Impacts of demand response from buildings and centralized thermal energy storage on district heating systems. *Sustainable Cities and Society*, 64(July 2020):102510, 2021. ISSN 22106707. doi:10.1016/j.scs.2020.102510. URL `https://doi.org/10.1016/j.scs.2020.102510`.

Mostafa Vahedipour-Dahraie, Homa Rashidizadeh-Kermani, Amjad Anvari-Moghaddam, and Josep M. Guerrero. Stochastic risk-constrained scheduling of renewable-powered autonomous microgrids with demand response actions: Reliability and economic implications. *IEEE Transactions on Industry Applications*, 56(2):1882–1895, 2020. ISSN 19399367. doi:10.1109/TIA.2019.2959549.

Michael Wetter. A modelica-based model library for building energy and control systems. *IBPSA 2009 - International Building Performance Simulation Association 2009*, (July): 652–659, 2009.

Fabian Wüllhorst, Thomas Storek, Philipp Mehrfeld, and Dirk Müller. AixCaliBuHA: Automated calibration of building and HVAC systems. *Journal of Open Source Software*, 7(72): 3861, 2022. ISSN 2475-9066. doi:10.21105/joss.03861.

Renata Lautert Yang, Gerson Máximo Tiepolo, Édwin Augusto Tonolo, Jair Urbanetz, and Muriele Bester de Souza. Photovoltaic cell temperature estimation for a grid-connect photovoltaic systems in Curitiba. *Brazilian Archives of Biology and Technology*, 62(specialissue):1–9, 2019. ISSN 16784324. doi:10.1590/1678-4324-SMART-2019190016.

8 Appendix

Parameter	Value	Unit
G_{NOCT}	800	Wm^2
$T_{c_{NOCT}}$	45	$^\circ$C
$T_{a_{NOCT}}$	20	$^\circ$C
$\eta_\mathbf{c}$	0.4	-
τ_α	0.9	-

Table 3. *TARKA 120 VSMS* PV module's characteristics.

External Air Temperature [ºC]	Cooling Capacity [W]	Electric Consumption [W]
40	41 900	16 115
35	38 700	13 345
32	34 000	10 000
30	29 100	6929
28	23 600	4917
25	8100	2132

Table 4. *Zephir CPAN-XHE3* HVAC system's performance data in cooling mode.

External Air Temperature [ºC]	Heating Capacity [W]	Electric Consumption [W]
-5	49 700	11 044
0	49 500	12 375
2	46 200	11 268
7	37 100	8065
12	28 400	5462

Table 5. *Zephir CPAN-XHE3* HVAC system's performance data in heating mode.

Power System Real-Time Simulation using Modelica and the FMI

Marcelo de Castro[1] Giuseppe Laera[1] Fernando Fachini[1] Sergio A. Dorado-Rojas[1] L. Vanfretti[1]
Shehab Ahmed[2] Chetan Mishra[3] Kevin D. Jones[3] R. Matthew Gardner[3]

[1]ECSE Department, Rensselaer Polytechnic Institute, USA, {decasm3, laerag, fachif, dorads, vanfrl}@rpi.edu
[2]King Abdullah University of Science and Technology, Saudi Arabia, shehab.ahmed@kaust.edu.sa
[3]Dominion Energy, USA, {chetan.mishra,kevin.d.jones,matthew.gardner}@dominionenergy.com

Abstract

Real-time digital simulation of power systems is incredibly important for the testing of appropriate control and protection strategies in the power system industry. However, the case in which one single model can be used in offline simulations and then for testing in real-time studies is rare, if existing at all, due to the lack of adequate standard development in the power industry or the adoption of successful standards elsewhere. A direct consequence of this lack of portability is the large amount of time and resources spent in re-implementation and validation of models for real-time simulation of power grids. The present study proposes the usage of Modelica and the FMI standard in order to address this issue. To test the proposed approach, power system models are built offline using the OpenIPSL library and are exported as FMUs. Real-time simulations of two typical power system models are performed using dSPACE SCALEXIO™, proving that the proposed framework using Modelica and the FMI can greatly contribute to the enhancement of today's current practice in the power industry by providing portability and tractability between offline and real-time power system models.

Keywords: Power Systems, Real-Time Digital Simulation, Modelica, Functional Mockup Interface, FMI

1 Introduction

Modern power systems depend heavily on the ability of engineers to anticipate outcomes that can harm the grid's safe operation (Kundur 2007; Chow and Sanchez-Gasca 2020). Hence, it is obvious that Modeling and Simulation (M&S) plays a crucial role in power system studies. Consequently, engineers have developed several strategies through the decades in order to be able to better represent the electric grid in all its intrinsic complexity. Real-time simulation is, of course, one of those strategies and it is used as a last step before deploying one piece of equipment in real-world conditions. In order to understand the context that motivates this present study, it is first necessary to understand the background on simulation of power systems.

1.1 Background

Before digital simulation became possible, hardware-based test beds in laboratories emulating equivalent models of power apparatus were used since the late 1920s for representing the dynamic behavior of the bulk power system in a smaller scale (Evans and Bergvall 1924). It was not until the 1950s, though, that larger portions of the grid started being represented by analog circuits, composed by amp-ops, capacitors, resistors and inductors (Baldini and Fugill 1952). These transient network analyzers were so important for the simulation of power systems that even today their causality-orientation principle used for modeling is the basis for most conventional model development in the power industry. Although the analog simulators required a myriad of different solutions to properly mimic a real system, they allowed to actively test the prototypes of actual controllers using real measurements (Isaacs 2017).

With the development of modern computers, digital simulation was born and it rapidly became a very attractive approach for the analysis of power systems (Brown and Tinney 1957; Stott 1979). After many years of important advances in simulation technology, different parties could program Dommel's (1969) method for electromagnetic transient (EMT) solutions into a DSP (Throckmorton and Wozniak 1994), starting what is known today as real-time digital simulation (Isaacs 2017). If compared to their analog counterparts, these digital network analyzers were cheaper to assemble, had greater flexibility in terms of operation, allowed for tests to be prepared and conducted more rapidly, and enabled better reproducibility of experiments (Watson and Arrillaga 2003). Due to these notable features, real-time simulators became very common in the assessment of power systems' dynamic operation, control and protection testing.

Simulation under real-time, however, has several limitations with respect to the solution method applied to the system of equations. Simulation of a model in real-time is done with fixed time step solvers due to easier implementation when compared to variable time step, especially considering the strict requirements related to the tasks executed in each time-step. In the interval of one time step, the simulator needs to read inputs, if available, perform the numeric calculations and make outputs available for measurement, as it is depicted in

Figure 1. When this is not achieved under one time step, there is an overrun. As one may note, the time step shown in Figure 1 might not be of importance for human interaction. However, the purpose of real-time simulation is to be able to prototype real-world hardware-based controllers and protection devices and to test their dynamic performance within a power system model (Watson and Arrillaga 2003).

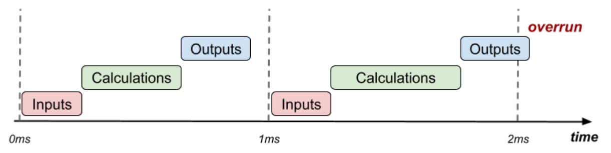


Figure 1. Tasks performed by real-time simulator within the range of a time step.

1.2 Motivation and Objectives

After years of development, real-time digital simulators have become a broadly adopted approach in the testing and prototyping of solutions for power systems over the last decade (Faruque et al. 2015). This is mainly due to the increase in their capabilities for power network representation and the possibility to conduct real-time hardware-in-the-loop simulations. With exception of solutions that adopt MATLAB/SIMULINK™ tools, such as eMEGAsim™ from Opal-RT (OPAL-RT Technology Inc. 2022), the traditional approach requires different proprietary M&S software tools to be used for offline M&S, control design, and real-time simulation. Due to that, engineers have to spend a great amount of time, effort, and resources being allocated in the re-implementation and verification of models, having great impacts on project's costs and the reliability of simulation results. Therefore, this clear lack of portability and tractability can be a tremendous bottleneck that should be urgently solved.

This paper, then, addresses this problem by proposing the usage of standardized dynamical models using the Modelica language and exploiting the Functional Mockup Interface (FMI) standard (Modelica Association Project 2021) to deploy these models into different platforms. Offline power system models can be developed using the Modelica language (Modelica Association 2022) and the Open-Instance Power System Library (OpenIPSL) (Baudette et al. 2018), an open-source library of power system models for stability studies using the "phasor" representation. The OpenIPSL has a set of models that are present in traditional proprietary software tools used in the study of power systems, such as PSS/E™. Moreover, as most Modelica tools implement the FMI standard, the language becomes a strong candidate along with Modelica-compliant software being the M&S tools in which real-time models are configured. Furthermore, the dSPACE SCALEXIO™ (dSPACE GmbH 2022) real-time simulator usually employed in aviation and automobile industries, has adopted the FMI standard, enabling it to natively execute models exported as Functional Mockup Units (FMUs).

1.3 Contributions

The contributions of this paper are twofold.

- The introduction and the assessment of a proposed framework using Modelica language and the FMI standard as means to address the portability and tractability challenges that are common in power system studies involving real-time simulation;
- The assessment of power system models assembled with the OpenIPSL library, which were made originally developed for offline simulation, in a real-time execution environment, with minimal modifications.

1.4 Paper Organization

The remainder of this paper is organized as follows: Section 2 presents the methodology for offline to real-time simulation proposed in this paper; Section 3 presents the example power systems used in this study; Section 4 describes the laboratory setup for the real-time simulations; Section 5 shows the measured results for the different simulations; Section 6 provides the concluding remarks of this paper.

2 Model Configuration Framework

The framework adopted in this paper to perform power system modeling, offline and real-time simulations, can be summarized into three steps. They are depicted in Figure 2 as a workflow, starting with the offline Modelica model and resulting in the real-time simulation on a dSPACE SCALEXIO™ real-time simulator.

2.1 Modelica Model Configuration

The first step, as shown in Figure 2, concerns the offline power system network model. It is assembled in Modelica and, in this study, the Dymola™ 2019 software is used to make modifications on the model. The network model is built using the OpenIPSL (Baudette et al. 2018), a library that has been under development during the last decade (Bogodorova et al. 2013) and that has shown potential for the study of power systems by the means of offline simulations for dynamic performance assessment(Baudette et al. 2018; Winkler 2017). For real-time simulation, the original Modelica model needs to be slightly modified to have outputs added, to specify the quantities to be measured during simulation. In addition, simulation parameters, such as solver tolerance and total simulated time, also need to be specified. These tasks can be easily done by using Modelica Standard Library's `Blocks.Interfaces.RealOutput` and by the addition of experiment `annotations`, respectively.

After performing these minor modifications in the original models, the FMU export procedure can be

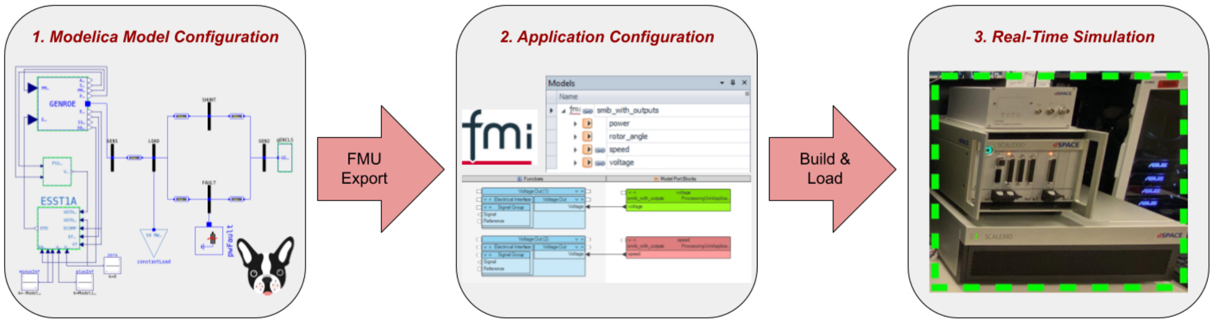


Figure 2. Three-step workflow adopted for simulation of FMUs in a real-time processing unit.

performed. The export should be configured in such way that a FMI v.2.0 Co-Simulation FMU with CVODE solver is created, with 64-bit binaries, including the model's source code. Although CVODE solvers are variable-order variable-step solvers, a fixed time-step variable-order CVODE solver is exported within the FMU. It is also necessary to highlight that one should check the compatibility between the versions of Dymola™ and dSPACE™'s software ConfigurationDesk® (the latter is the target in which the FMU will be loaded).

2.2 Application Configuration

After the export procedure, the FMU is loaded into dSPACE™'s software for real-time application configuration, ConfigurationDesk®. There, hardware resources such as the SCALEXIO™ and LabBox™ can be assigned to the project and configured. The former is the real-time processing unit and is responsible for the calculations that are executed during the real-time simulation. The latter is the unit responsible making outputs and inputs available. More information on these two pieces of hardware is presented in Section 4.

It is important to note that, once the FMU is loaded into ConfigurationDesk®, the outputs added on the first step are recognized and can be configured to be interfaced (via functions) to the real hardware outputs in the LabBox™ hardware, via the DS6101® board. The signals connected to these ports are converted into non-negative voltages of same magnitude and, therefore, special caution needs to be considered in output preparation. In fact, the DS6101® board supports outputs within the range of 0 to 10 V, justifying why some outputs of interest might need to be pre-configured to allow for measurements to be performed during the experiment.

2.3 Real-Time Simulation

After all the simulation configurations are set up and the outputs are properly configured, a "real-time application" is built. This procedure is executed automatically by ConfigurationDesk®. The real-time application can then be loaded into SCALEXIO™ and the simulation can be controlled by using another software tool made available by dSPACE: ControlDesk®. In this software tool, the FMU can be set up for simulation and the real-time processing unit will execute the necessary calculations. Although the quantities are physically measured in this paper using both USB-based and conventional oscilloscopes, the results from the real-time simulations can also be verified in ControlDesk®. More information about it can be found in Section 4.

3 Test Cases

In order to assess the framework, two typical test power system models are developed in Modelica using the OpenIPSL library and are configured for export using FMI, as described in Section 2. A brief description of these two system models is presented next.

3.1 Single-Machine Infinite-Bus

The Single-Machine Infinite-Bus (SMIB) is a very basic representation of a power network, with five buses, one machines and one infinite bus. The diagram representation of this power grid is presented on Figure 3. The infinite bus is connected at bus `GEN2` while the generator connected at bus `GEN1` is composed by a round rotor synchronous machine (`GENROE`), a fast static excitation system model (`ESST1A`) and a stabilizer (`PSS2B`). During the simulation, the system undergoes a three-phase-to-ground fault on bus `FAULT`. In addition, note that these models are also present in PSS/E™, a traditional software tool used in offline power system transient stability studies.

As mentioned in Section 2, the model needs minor additions, such as the placement of outputs to gather measurements during the real-time experiment. Four outputs are placed in the model and their relationship with model variables are determined via equations, as shown in Listing 1. Note that, because the output board does not allow for non-negative voltages to be measured, the frequency deviation, in Hz, is offset by 2.

In addition, although the SMIB system is quite simple, it can be helpful considerably helpful in basic experiments (Kundur 2007) and can be used in controller design (De Marco, Martins, and Rullo 2021; De Marco,

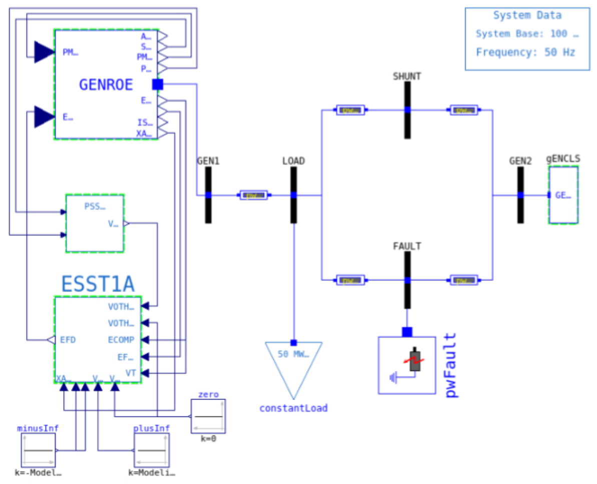


Figure 3. Implementation of the SMIB system in Modelica using the OpenIPSL.

Rullo, and Martins 2021).

Listing 1. Equations added to SMIB model for representing outputs of interest.

```
equation
  dfreq = 2 + gENROE.SPEED*SysData.fn
  "Frequency deviation with offset";
  rotor_angle = gENROE.ANGLE
  "Rotor angle measurement";
  voltage = gENROE.ETERM
  "Generator's terminal voltage magnitude";
  power = gENROE.PELEC
  "Generator's active power output";
```

Listing 2. Equations added to IEEE 9-bus model for representing outputs of interest.

```
equation
  statcom_out = sTATCOM.Q + 0.5
  "STATCOM's reactive output with offset";
  gen_Q = gen1.gen.Q + 0.5
  "Gen 1 reactive output with offset";
  freq_gen1 = gen1.gen.w*SysData.fn-59
  "Gen 1 frequency deviation with offset";
  freq_gen3 = gen3.gen.w*SysData.fn-59
  "Gen 3 frequency deviation with offset";
```

3.2 IEEE 9-Bus 3-Machine

The IEEE 9-bus 3-machine system is the second network used in this study. The one-line diagram of this test system, built using OpenIPSL, is presented in Figure 4. The machines are connected at buses `B1`, `B2` and `B3` and all of them are composed of a machine and an excitation system models verified against those of PSAT (Milano 2005). Moreover, a Static Compensator (STATCOM) is connected at bus `B8` and a `Fault` element, representing a three-phase-to-ground event is connected at bus `B9`.

Once again it is necessary to slightly modify the original network in order to introduce the outputs to be measured during the real-time experiment. The outputs are associated with the model's variables via the equations in Listing 2. Because the reactive power output of the STATCOM model and of machine 1 have negative values, at some time instants in the simulation, their values are offset by 0.5 per unit to conduct the experiment. The frequency deviation, in Hz, for generators 1 and 3 is also offset and centered at 1.

4 Simulator Setup

After the power system models are exported as FMUs, they are loaded into ConfigurationDesk® and are configured as described in Section 2. The real-time applications are then built and loaded into the assigned SCALEXIO™ real-time processing unit. Hardware specifications of the processing unit together with simulation parameters are summarized on Table 1. The processing unit is based on CPU architecture and from the 4 cores available, only one is used in all tests performed in this paper, indicating that larger power system models can be simulated if they are broken into different FMUs and assigned to different cores. Furthermore, the simulation time step used in all simulations performed in this paper is 1 ms, a common value for power system studies related to electromechanical transient stability. In addition, the maximum number of overruns is set to 150. After that, the real-time simulation stops.

Table 1. Hardware and simulation specifications.

Parameter	Description
Processor	Intel® Xeon® E3-1275 v3
Number of cores (used)	4 (1)
Clock Frequency	2.8 *GHz*
RAM size	8 Gb
Flash size	512 Mb
Allocated task stack size	1024 Kb
Task time step	1 *ms*
Overrun Count Max.	150

The outputs are configured in such way that measurements are made directly in the pins of the input/output (I/O) boards. The FMU's outputs are assigned to analog outputs and are measured using USB and conventional oscilloscopes, as displayed in Figure 5. Note that the behavior of determined variables is also available for display on ControlDesk®. The results displayed in Section 5 are obtained using the USB oscilloscope due to the convenience it offers to import results.

5 Real-Time Simulation Results

In this section, the results for the simulation of both systems is presented together with some discussion on the results and on the performance of the real-time processing

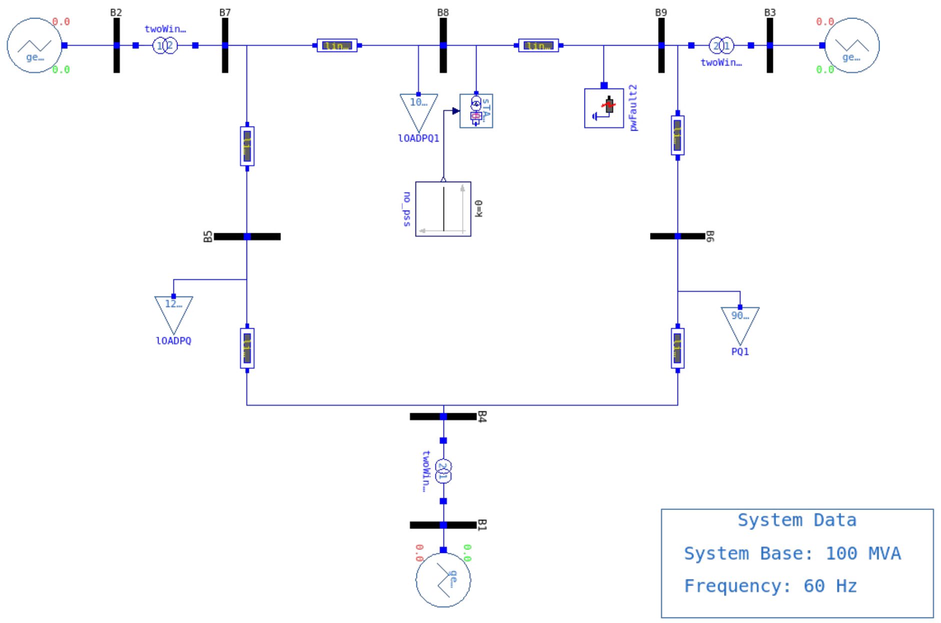
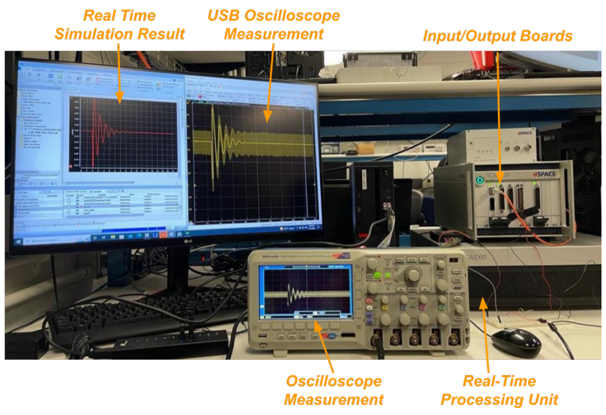


Figure 4. Implementation of the IEEE 9-bus 3-machine system in Modelica using the OpenIPSL.



Real Time Simulation Result
USB Oscilloscope Measurement
Input/Output Boards
Oscilloscope Measurement
Real-Time Processing Unit

Figure 5. Experiment set-up and comparison among real-time simulation result and observed measurements coming from two different devices.

unit.

5.1 Single-Machine Infinite-Bus

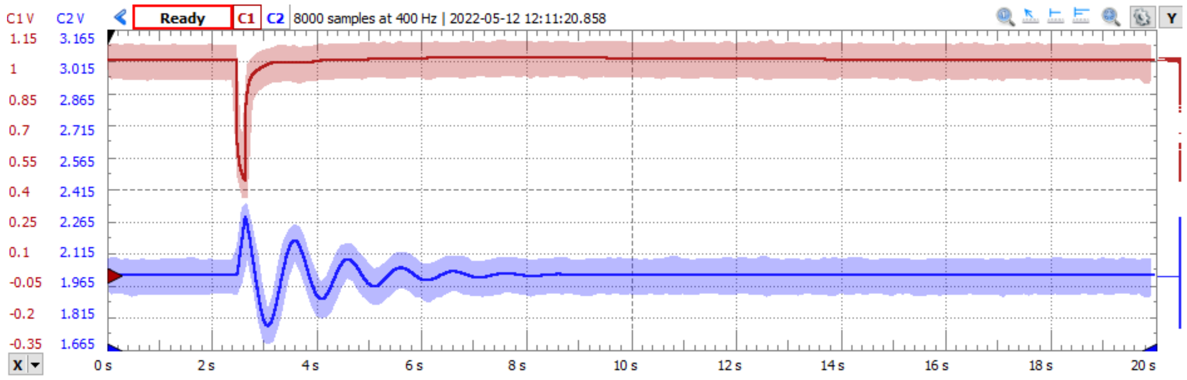
The results for the real-time simulation of the SMIB system are shown in Figure 6. The generator's terminal voltage is depicted in channel 1, in red, of Figure 6a while the generator's deviation from nominal frequency of 50 Hz is displayed in channel 2, in blue. Note that the latter is centered ≈ 2 V, as designated with Listing 1. In Figure 6b, channel 1, in red, represents the generator's electrical power output, in per unit, while channel 2, in blue represents the generator's rotor angle, in radians. All result windows present 20 seconds of measured real-time results captured at a sampling rate of 400 Hz. Also note that the solid lines on all channels represent the average value, while the shaded area around it is the measured noise. Furthermore, it is worth highlighting that the curves have different axes, as displayed in the left part of Figures 6a and 6b, and that they are measured in volts.

The averaged values behave as expected, and it is possible to clearly observe that the event occurs at approximately 2.5 s in Figure 6a and at 2.8 s in Figure 6b. This discrepancy between the event instant is due to a display time shift, since the measurement was done using two channels simultaneously. In addition, the simulation and measurement acquisition are not synchronized, which is why none of the curves present the result of the fault event at exactly at 2.0 s.

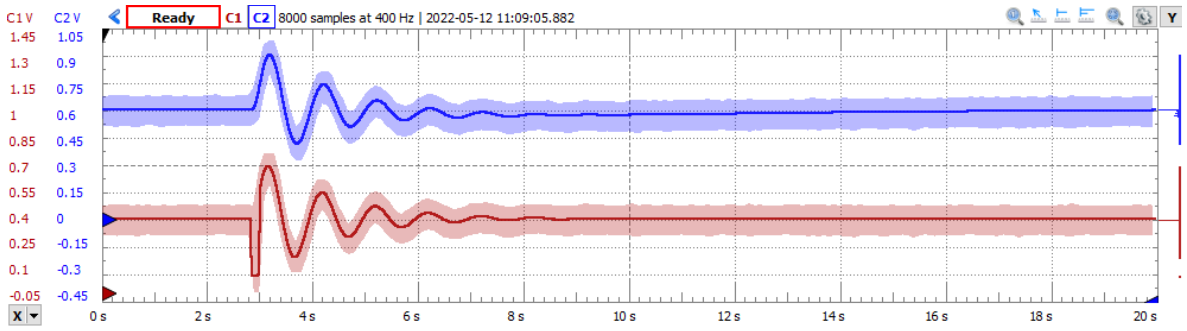
5.2 IEEE 9-Bus 3-Machine

The IEEE 9-bus 3-machine model is simulated in real-time in order to perform the tests using a slightly larger system. The measured results are depicted on Figure 7. The reactive power outputs of the STATCOM and generator 1 are represented, in per unit, by channels 1 and 2, respectively, in Figure 7a. Note that they are displayed with the same offset of 0.5 V and that the curves have different displacement in the *y*-axis, as shown in the left part of the plot. Note that, as the event occurs, both power system components peak in their reactive power generation.

The frequency deviation from the nominal value of 60 Hz, in generators 1 and 3, is displayed in Figure 7b. Channel 1, in red, depicts the quantity related to generator 1, while channel 2, in blue, pictures the quantity associated with generator 3. Note that both channels are, now, centered at the same value and they both have an offset of 1 V. The frequency deviations have opposing phase, suggesting that these two generators are not coherent. Moreover, observe that the event appears to occur at 3.05 s in Figure 7a and at 3.2 s in Figure 7b. The reason for this discrepancy is the same

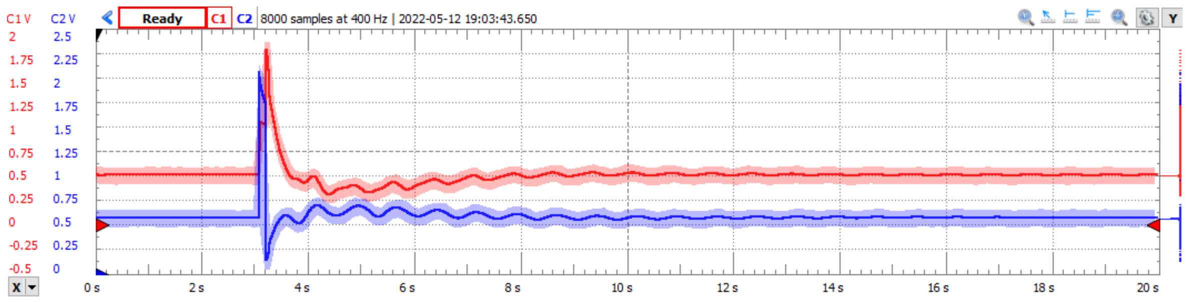


(a) Channel 1 represents the generator's output voltage, in per unit. Channel 2 represents the generator's deviation from nominal frequency, in hertz, and centered at 2 V.

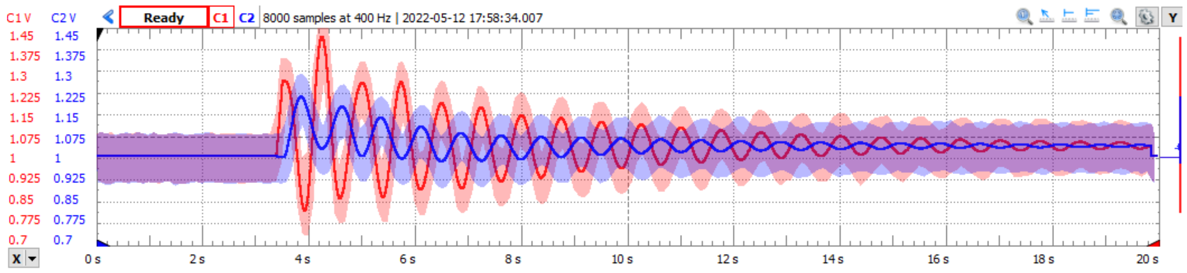


(b) Channel 1 represents the generator's electrical power output, in per unit. Channel 2 represents the generator's rotor angle, in radians.

Figure 6. SMIB system real-time simulation results measured with an USB oscilloscope.



(a) Channel 1 represents the reactive power output from the STATCOM, in per unit and centered at 0.5 V. Channel 2 represents reactive power output from generator 1, also in per unit and centered at 0.5 V.



(b) Channels 1 and 2 represent the deviation from nominal frequency, in hertz, on generator 1 and 3, respectively. Both values are centered at 1 V.

Figure 7. IEEE 9-bus 3-machine system real-time simulation results measured with an USB oscilloscope.

lack of synchronization between the measurement data acquisition and the real-time simulator, and the fact that both quantities are measured concurrently per experiment.

5.3 Discussion

Notable features observed in each simulation experiment are displayed in Table 2. Note that, although the IEEE 9-bus 3-machine model is $\approx 29\%$ larger than the SMIB in terms of the number of Differential and Algebraic Equations (DAEs), it has approximately $\times 6$ times more overruns during real-time execution. The IEEE 9-bus also has a task turnaround time 3.22 times higher, indicating that as the model increases in complexity, the time required for the simulator to execute each time step increases substantially. It is important to note that only one CPU from the processing unit was used in this study, meaning that for larger system models than the IEEE 9-bus, the model needs be separated into different FMUs for process parallelization, reducing the overruns. However, this is not assessed in this work.

Table 2. Comparison between studied systems and their performance in real-time simulation.

Studied System	SMIB	IEEE 9-bus
DAEs	369	476
Initialization Overruns	2	3
Simulation Overruns	16	106
Task Turnaround Time (μs)	≈ 83	≈ 268

In order to understand the reasons why these overruns are observed, it is possible to perform profiling of execution time using Dymola™, before generating the FMU. By setting some simulation flags, it is possible to enable the plotting of the execution time needed for each time step throughout simulation. Figure 8 depicts the execution time needed for Dymola™ to compute each time step during simulation marked as green $\times$, while the simulator time step is highlighted in a dashed red line. Note that the number of execution times that are larger than the time step adopted for the simulator is similar to the value found in rows representing Initialization and Simulation Overruns, on Table 2. Furthermore, it is possible to observe that a large number of overruns result from the event that occurs at 2 s and 2.15 s, which correspond to a three-phase fault being applied and its clearing, respectively. It can also be found that a state associated with one lead-lag block from the excitation system dominates the error and, therefore, should be the responsible for the majority of overruns. This is because the excitation system will act to provide synchronizing torque so to help the generator keep in synchronism when the fault is applied, and thus, the equations of the excitation system will be governing the dynamic response of the system.

It is also possible to conduct a similar analysis for the

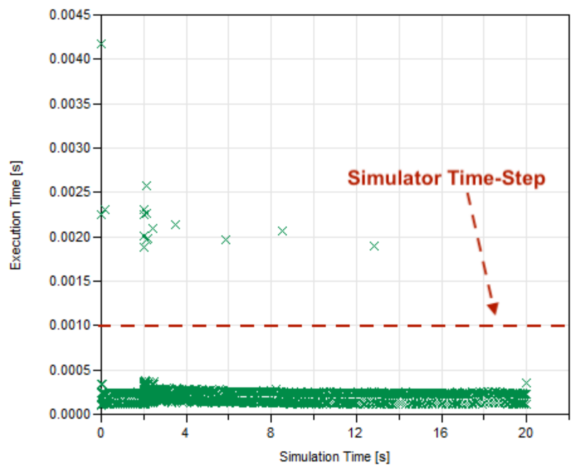


Figure 8. Comparison between execution time in Dymola™ and the simulator time step for the SMIB test system.

IEEE 9-bus 3-machine system and the execution time for Dymola™ , as shown in Figure 9 in green $\times$. Once again the simulator time step is highlighted in a dashed red line and it reveals when the overruns related on Table 2 that occur during simulation. Once again, the majority of the overruns occur during the fault being applied and its clearing. In this case, three states dominate the error; two related to first order blocks in the excitation systems of generators 1 and 2 and one related to a lead-lag control block used in the STATCOM. Similarly to the previous case, this is expected as both generators aid in stabilizing the system while the STATCOM aims to control its terminal voltage, thus engaging the equations that govern the dynamics of these three components.

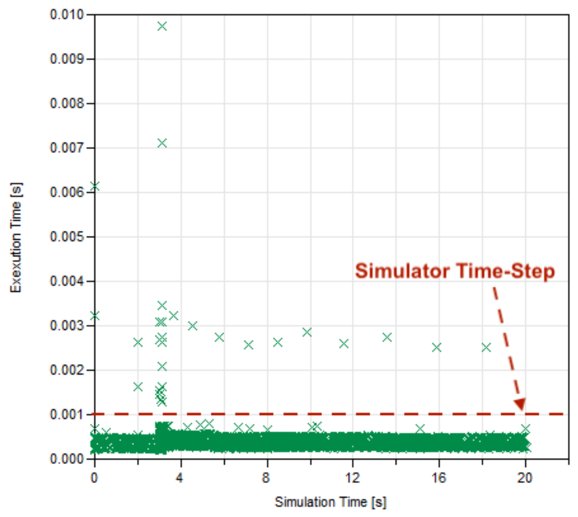


Figure 9. Comparison between execution time in Dymola™ and the simulator time-step in for IEEE 9-bus 3-machine test system.

6 Conclusion

This paper presented a framework that allows power system simulation studies to be performed using a power network model, built using Modelica and OpenIPSL, that is exported using the FMI standard for execution in a real-time simulator. The framework uses dSPACE hardware and software tools, since it natively runs FMUs, allowing for the exported Modelica-based power system models to be readily loaded, with minimal modifications. The results from the real-time simulation can be measured and verified by the usage of I/O boards that are connected to the processing unit.

Power system models built using Modelica, via the OpenIPSL, are usually used for offline simulation. However, in this study, their potential for real-time simulation and their real-time execution performance were assessed and determined to be satisfactory, i.e. with limited overruns. More importantly, the process of "going from off-line to real-time" involved very little additional effort. Therefore, by adopting this framework, the authors were able to demonstrate that the same model might be used in *both* offline and real-time simulation, eliminating the time and effort allocated for model re-implementation and verification, resulting from enhanced power system model portability and tractability. This work hopes to serve for both the Modelica and power engineering communities, as a proof of concept of the potential of the Modelica and FMI open access standards for model portability and interoperability across different simulation use cases, which are currently performed using disparate and siloed tools and hardware.

As future works, the authors aim at assessing the real-time simulation of larger power systems by exploring process parallelization and the connection of multiple FMUs. Furthermore, the authors aim to perform control-hardware-in-the-loop experiments using this framework, allowing for fast prototyping of power system controllers while providing maximum model portability and traceability.

Acknowledgements

This material is based upon work supported in whole or in part by Dominion Energy, by the Center of Excellence for NEOM Research at the King Abdullah University of Science and Technology under grant OSR-2019-CoE-NEOM-4178.12, and 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. The views expressed herein do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

Baldini, EA and AP Fugill (1952). "A power system analogue and network computer [includes discussion]". In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 71.1, pp. 291–297.

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.

Bogodorova, T. et al. (2013). "A Modelica power system library for phasor time-domain simulation". In: *IEEE PES ISGT Europe 2013*.

Brown, Rodney J and William F Tinney (1957). "Digital solutions for large power networks". In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 76.3, pp. 347–351.

Chow, Joe H and Juan J Sanchez-Gasca (2020). *Power system modeling, computation, and control*. John Wiley & Sons.

De Marco, Fernando, Nelson Martins, and Pablo Rullo (2021). "Using a SMIB Model to Analyse and Damp the Electromechanical Modes in Multigenerator Plants". In: *2021 IEEE URUCON*. IEEE, pp. 252–257.

De Marco, Fernando, Pablo Rullo, and Nelson Martins (2021). "Synthetic Power System Models for PSS Tuning and Performance Assessment". In: *2021 IEEE Electrical Power and Energy Conference (EPEC)*. IEEE, pp. 107–112.

Dommel, Hermann W (1969). "Digital computer solution of electromagnetic transients in single-and multiphase networks". In: *IEEE transactions on power apparatus and systems* 4, pp. 388–399.

dSPACE GmbH (2022-05). *Home-dSPACE*. https://www.dspace.com/.

Evans, RD and RC Bergvall (1924). "Experimental analysis of stability and power limitations". In: *Transactions of the American Institute of Electrical Engineers* 43, pp. 39–58.

Faruque, MD Omar et al. (2015). "Real-time simulation technologies for power systems design, testing, and analysis". In: *IEEE Power and Energy Technology Systems Journal* 2.2, pp. 63–73.

Isaacs, Andrew (2017). "Simulation technology: The evolution of the power system network [history]". In: *IEEE Power and Energy Magazine* 15.4, pp. 88–102.

Kundur, Prabha (2007). *Power System Stability and Control*. McGraw Hill, Inc.

Milano, Federico (2005). "An open source power system analysis toolbox". In: *IEEE Transactions on Power systems* 20.3, pp. 1199–1206.

Modelica Association (2022-05). *Modelica Language*. https://modelica.org/modelicalanguage.html.

Modelica Association Project (2021-07). *FMI Standard*. https://fmi-standard.org/.

OPAL-RT Technology Inc. (2022-05). *OPAL-RT website*. http://opal-rt.com.

Stott, Brian (1979). "Power system dynamic response calculations". In: *Proceedings of the IEEE* 67.2, pp. 219–241.

Throckmorton, Paul Jeffrey and Louis Wozniak (1994). "A generic DSP-based real-time simulator with application to hydrogenerator speed controller development". In: *IEEE Transactions on Energy Conversion* 9.2, pp. 238–242.

Watson, Neville and Jos Arrillaga (2003). *Power systems electromagnetic transients simulation*. Vol. 39. IET.

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 Playground for the Modelica Language

Michael M. Tiller[1]

[1]modelica.university, `michael.tiller@gmail.com`

Abstract

This paper introduces a Modelica playground which allows users to experiment with the Modelica language without having to install any specific Modelica tools. This web-based application also contains content and lessons that provide users with a guided tour of the language and the opportunity for advanced users to create domain specific content built on top of this same infrastructure. This paper will explain the various open source technologies employed in creating this application and discuss potential future work to further enhance the experience for the user as well as the reach for Modelica itself.

Keywords: Modelica, education, interactive, animation, playground, web

1 Introduction

1.1 Playgrounds

To help "onboard" users, many programming languages include a web-based environment that allows users to see working fragments of code in that language. What makes such an environment a *playground* is that it allows these code fragments to be edited and compiled as well. This enables users to explore the language and understand at least the basics of different syntactic constructs without having to install any of the normal tooling associated with the language.

These playgrounds are not only useful tools for users to "try out" a language before committing to installing all the tooling, they are also very useful as educational tools. Such playgrounds often include examples of specific features of the languages. In a sense, they are used to help "sell" users on the design of the language or help explain difficult concepts by giving the users running examples (created by language experts) to help users understand the particularly idiomatic ways of accomplishing various tasks in that language.

The reality is that Modelica lags behind many other language ecosystems. This is, in part, due to a lack of resources. Modelica is, after all, something of a niche language. Nevertheless, this application was developed in part because applications like VPython (Bruce Sherwood 2022) are being used in a classroom context to teach students about math and physics through the use of 3D visualization. But using VPython is quite tedious compared to Modelica because users must implement all the numerical methods themselves. By creating the Modelica Playground, we hope to provide a better platform for students.

1.2 Goals

This project was developed with several goals in mind:

- **Freely Available**: As with all other content at `https://modelica.university`, this content is made freely available. The goal here is to support, to the greatest extent possible, those interested in learning the Modelica language. There are many tools out there with greater commercial resources than those in the Modelica community which is why it is important that the unique value and capabilities inherent in Modelica are demonstrated by material that is as accessible as possible.

- **Collaboration**: When using the Modelica Playground, users create models (and post-processing reports). This can involve a significant amount of effort. As such, it should be possible for users to easily save, share and publish their work.

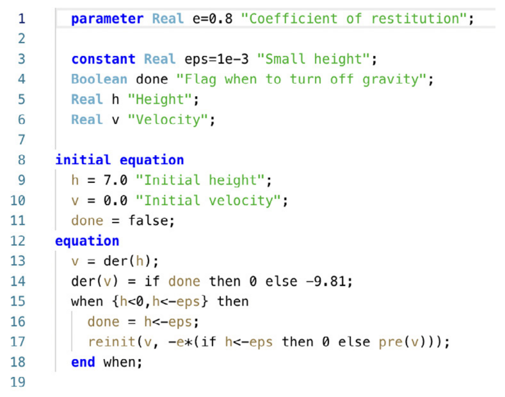
- **Visualization**: For most programming language playgrounds, it is sufficient to simply capture output from the running program and display that. But Modelica is a modeling language and the "output" of Modelica code is (generally) time-varying simulation results. So in order for the user to fully comprehend what their code "means" in a mathematical sense, it is essential that visualization tools are available to bring those simulation results to life. Although there are many ways to visualize data in a web browser, we don't want the user to be required to become a frontend web developer with full knowledge of Javascript, HTML and CSS in order to craft their visualizations. For this reason, a no/low-code approach was taken requiring minimal amounts of imperative code to be written.

- **Extensible**: This application is about more than just teaching people Modelica. It is to provide a means to communicate ideas via Modelica code. This platform has been designed explicitly to allow ordinary users to create content that can be organized into "lessons" such that these lessons can be shared among users *without the need to edit the source code of the playground application itself.*

- **Privacy**: Cookie consent popups have become ubiquitous since the rollout of GDPR. While it is useful to track the popularity of the tools hosted at `modelica.university`, there is no useful purpose in tracking users as individuals. As such, there is no cookie consent popup because there are no cookies being dropped in the user's browser. It simply isn't necessary to track users in order to accomplish this application's goals. Sorry Google.

2 Modelica Editor

The first aspect of the Modelica Playground that we shall discuss is the Modelica editor. The Modelica editor, shown in Figure 1, is built on top of Monaco (Microsoft 2022a), the code editor that powers Visual Studio Code (Microsoft 2022d), a widely used open source integrated development environment. The Monaco platform provides a playground of its own at: `https://microsoft.github.io/monaco-editor/playground.html`.



```
  parameter Real e=0.8 "Coefficient of restitution";

  constant Real eps=1e-3 "Small height";
  Boolean done "Flag when to turn off gravity";
  Real h "Height";
  Real v "Velocity";

initial equation
  h = 7.0 "Initial height";
  v = 0.0 "Initial velocity";
  done = false;
equation
  v = der(h);
  der(v) = if done then 0 else -9.81;
  when {h<0,h<-eps} then
    done = h<-eps;
    reinit(v, -e*(if h<-eps then 0 else pre(v)));
  end when;
```

Figure 1. Modelica code editor in Modelica Playground

Currently, the Modelica editor provides syntax highlighting as well as "error decoration" (both syntax errors and compilation errors). The implementation details of these features will be discussed shortly. Monaco itself is quite a powerful platform and hopefully other features that it provides will be incorporated in the future.

Before talking about syntax in more detail, it is important to point out one way that the Modelica Playground **deviates from most Modelica tools**. For most Modelica tools, the user compiles a `model`. The understanding is that this `model` will be instantiated implicitly by the compiler and that the `model` is, in some sense, the fundamental compilation unit.

As shown in Figure 1, the Modelica Playground doesn't take this approach because no root level restricted class is required. The reason for this is that by allowing the user to start with simple variables and equations, no previous knowledge about restricted classes or object oriented programming features are required. As a result, the typical code fragments found in the Modelica Playground read more like a program in an interpreted programming language like Python or Javascript. The goal in making this change is to lower the barrier of entry for new users and provide them with an initial context that is more familiar and intuitive.

2.1 Syntax Highlighting

Syntax highlighting is an essential requirement for any kind programming language renderer whether it simply be rendering source code on a page or implementation of a text editor. The Monaco system provides something called Monarch (Microsoft 2022b) for implementing syntax highlighting as a series of simple rules. The goal, with Monarch, is to avoid the need to implement a complete language parser and instead reduce the process down to one that can be accomplished with a collection of regular expressions. This can certainly be done with Modelica, but that isn't how syntax highlighting is implemented in the Modelica Playground.

Ultimately, syntax highlighting is simply about identifying the semantic significance of regions of text in the source code. While Monarch does this via regular expressions, the Modelica playground actually uses a full blown Modelica parser. Normally, this would probably be considered overkill. But there are two reasons this is reasonable in this case. First, the Modelica Playground doesn't deal with large quantities of code so the extra computational effort required to do a complete parsing of the code isn't really that significant and doesn't really impact responsiveness of the user interface. Second, the particular parser we are using is actually purpose built for these kinds of tasks.

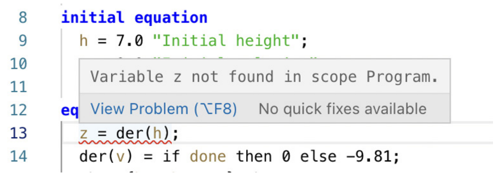
The parser we are using was created using Tree-sitter (Brunsfeld 2022). There are two aspects of Tree-sitter that make it well suited for our purposes. The first is that it was developed specifically as an incremental parser. What that means is that it is designed to parse source code that is constantly changing. The typical use case that an incremental parser would concern itself with is syntax highlighting source code in a text editor. The goal is to quickly re-parse the source code after text has been inserted or deleted in a certain range. The parser itself is designed to reuse as much of the effort from previous parsing passes as possible. Although in a playground context where the source code is small, this is of minimal benefit. But the parser itself could be reused in other contexts with larger files. The other aspect of Tree-sitter that makes it well suited for our purposes is the fact that it compiles down to WebAssembly (WebAssembly Community Group 2022). It does this by first compiling a C language implementation of the parser and then using Emscripten (Empscripten Contributors 2021) to compile that into Web Assembly. The result is near native performance in the browser.

The resulting parser has been open-sourced as Modelica-tree-sitter (Michael M. Tiller 2022b). Because tree-sitter is developed and used by Github, its existence will hopefully lead to a future where Modelica source code is natively highlighted on Github (and perhaps other platforms).

2.2 Error Handling

Error handling comes in two varieties. The first is basic syntax errors. These can be easily detected by the same parser that is used to generate the syntax highlighting. Unfortunately, one of the current limitations in Tree-sitter is a lack of good diagnostic messages from generated parsers. As a result, the decoration of syntax errors in Modelica simply identifies the text where a syntax error occurs but doesn't provide much useful information beyond that. There are several open issues related to this topic associated with the Tree-sitter project and the authors appear to recognize these limitations. Hopefully future versions of Tree-sitter will address these limitations which could translate into better syntax error diagnostics in Modelica Playground.

The other type of error that Modelica Playground handles are compilation errors. These errors are more semantic in nature and are reported back from the OpenModelica compiler (Open Modelica Consortium 2022) used by Modelica Playground to compile the Modelica source code. Fortunately, OpenModelica errors include information about the text range for each error. And, unlike the syntax errors, they include considerable information about the nature of the error. All of this is leveraged by the Monaco platform in providing text decorations over the regions of text that, when hovered over (see Figure 2), elaborate on the nature of the error contained there.



```
initial equation
  h = 7.0 "Initial height";

eq
  z = der(h);
  der(v) = if done then 0 else -9.81;
```

```
Variable z not found in scope Program.
```

Figure 2. Semantic Error Highlighting

3 Simulation

Of course, editing Modelica code is just the beginning of what is required in order to engage readers with the Modelica language. The next logical component is to enable simulation of those models. For many language playgrounds, code is compiled and then run and the textual output of the program is captured and relayed back to the user. But in this case, we need to compile the code, run a simulation and relay the simulation results back.

3.1 OpenModelica

Let's start with the compiler itself. As previously mentioned, the Modelica Playground runs the OpenModelica compiler to compile code. Architecturally, the Modelica Playground application makes a request to an HTTP API asking that the model be simulated. The model source code is *included in the request*. This kind of an approach admittedly does not scale for dealing with large code bases. But because this is simply a "playground" (dealing with small fragments of code), it is acceptable. The backend is implemented using the Go language (which features its own playground, (Google 2022)). The API writes the source code to a temporary directory, running the OpenModelica compiler and then bundling the simulation results up in the response. Rate limiting is implemented using a worker pool in each server responding to such requests. These servers are themselves deployed as Kubernetes `Deployment` resources and, being stateless, can be scaled up as needed. The default number of replicas is two but a horizontal autoscaler could easily be associated with such a deployment to handle high load situations.

3.2 Results

For the moment, simulation results are handled in a fairly simplistic way. The compilation step requests output in `csv` format and the API parses that output and identifies which signals are constant at every time interval and which ones are not. The results returned in the simulation response segregate the signals accordingly. A better approach would be to output results in a more "sophisticated" output format, like the `dsres` format, that was more space efficient (*e.g.,* leveraging things like alias elimination). But again, the requirements for a playground are not so demanding.

4 Report Editor

A basic proof of concept of the Modelica Playground providing a basic text editor and the ability to request simulation results for Modelica source was put together in a day or two. But providing a high quality user experience takes much more effort. Apart from the syntax highlighting and error handling already discussed, adding functionality that provides attractive visuals and extensibility takes a lot more effort. In this section, we'll talk about how post simulation reports are generated and all the various possibilities the Modelica Playground provides developers of such reports.

5 Report Rendering

If the only purpose of the Modelica Playground were to allow users to compile Modelica code without needing to install tools, then generating simple tables and plots (which is the default behavior when no post-processing report is specified) would be sufficient. But this type of approach limits the kind of narrative that can be associated with a given model.

Since one of the goals of this project was to build a platform for users to create content that told a story about various models (and to stitch them together with some degree of structure), a richer capability was required. Furthermore, previous work has demonstrated that if the platform itself requires the underlying source code for the application to be modified in order to add additional content, this will put considerable constraints on who can add new content and how it can be added.

For all of these reasons, the Modelica Playground was created with a post-processing report generation capability whose goal was to bring a no/low code approach to creating visual content. While MCP-0033 (Tidefelt and Tronarp 2020) proposes standard annotations for plots, the rendering functionality in the Modelica Playground goes well beyond (and could complement) that capability by opening up vastly more types of visualizations and interactivity. The following subsections will address the moving parts that make this possible.

As shown in Figure 3, the report tab in the Modelica Playground (found on the left) contains the purely textual source code for the post-processing report. On the right the Modelica Playground shows the rendered report (when simulation results are available).

5.1 Markdown

The heart of the report generation process is Markdown (John MacFarlane 2021). Markdown is widely used across the web as an easy to learn format for creating textual content. Various platforms have extended Markdown in different ways but Commonmark represents a fairly standard core which works reliably across different platforms.

Markdown brings standard markup support for text, paragraphs, images, font style, inline HTML, *etc*. This is the foundation for generating the reports, but it is simply the beginning of the transformations that occur. We have chosen the Remark (*Remark* 2021) and Rehype (*Rehype* 2022) tool chains because, as we shall see shortly, they can be quite easily extended via plugins.

These two tools by themselves allow the report textual description to be rendered on the fly in the right pane. This ability to immediately preview a report is not only useful for previewing how the text in the report will appear, it also works with all the extensions discussed in the remainder of this section which means the (report) content creator can preview mathematical equations, tables, plots and animations all in the context of simulation results. Every adjustment made to the report provides an instant preview.

5.2 Math

For rendering of mathematical equations, the Modelica Playground leverages the `remark-math` package (*remark-math* 2022). This provides both a `remark` and `rehype` plugin for parsing the mathematical markup in the Markdown content and rendering these equations using KaTeX (*KaTeX* 2022), respectively.

5.3 Custom Components

Another possibility with the `remark` rendering engine is to define custom components. As mentioned previously, Markdown allows HTML code to appears alongside Markdown syntax. But what the `remark` engine allows us to do is effectively "extend" HTML to introduce new element types and then gives us a hook by which we can render those custom elements.

Using this functionality, we define two additional specialized components. The first is the `<constants>` element. When rendered, the `<constants>` element will be replaced by a table that renders all constant variables found in the simulation results.

Another custom component provided by the renderer is the `<chart>` element. By default, the `<chart>` element will be rendered as a plot (using ECharts (Apache Software Foundation 2022)) containing all time varying variables in the simulation results. However, the `<chart>` element provides a `signals` attribute which, when supplied with a comma separated list of signal names, will display just the signals explicitly listed.

5.4 Templating

So far the rendering has been leveraging functionality that exists in `remark` plugins. But one challenge content creators may face is creating reports that leverage reuseable Markdown code fragments. Another challenge is the injection of information from the simulation into the report. The limitation of Markdown itself is that it doesn't actually provide any kind of templating functionality. So while it is excellent for describing content, it isn't designed at all for *managing* it.

This is where Nunjucks (Mozilla Software Foundation 2020) comes in. This is a templating engine written in Javascript and heavily inspired by the (Python based) Jinja (Pallets 2022) package. Nunjucks is a templating engine that allows us to define macros, variables, expressions and conditional constructs and in this way create reusable "units" of markdown as well as inject contextual information into the rendered report.

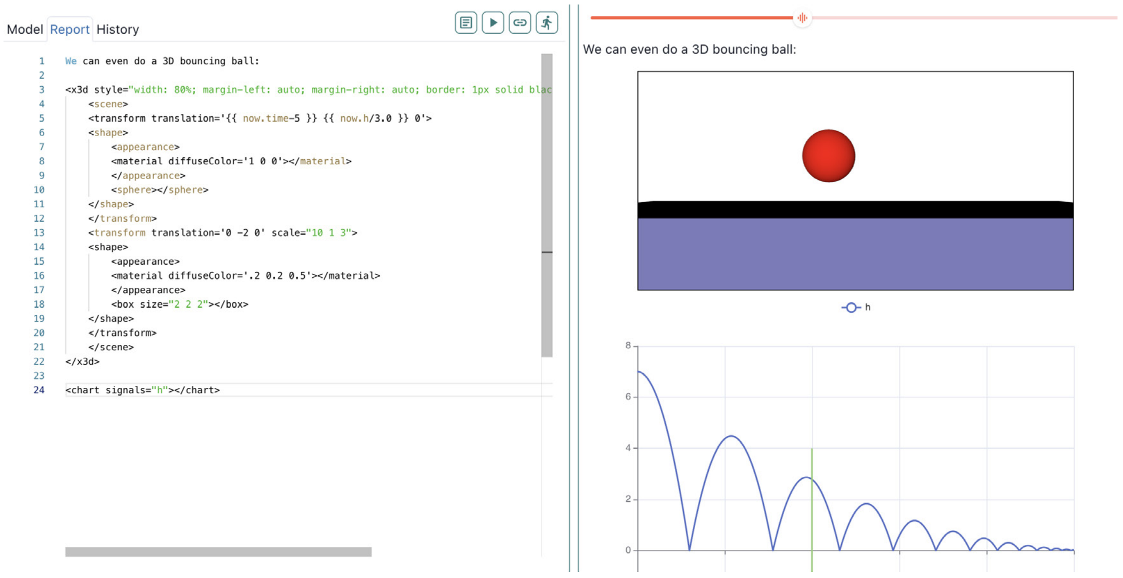
Note that the Modelica Playground is written in TypeScript (another language with its own playground, (Microsoft 2022c)) and leverages the React (Facebook 2022) framework. So one might wonder why is *another* system for creating reusable units of markdown required?

Why not simply use React components? This was certainly considered. For example, the MDX (MDX Community 2022) platform would have allowed us to mix React components into our Markdown code. But any solution that involves React *involves code*. Recall that one of the goals here is to have a no/low-code solution. Those creating content for this application should be able to do it easily without having to learn React or modify the source code of the application.

Nunjucks' learning curve was judged sufficiently easy to consider it for this purpose. It doesn't involve "linking" at all with the underlying application code and can be offered up and exposed to end users in a compartmentalized way that insulates them from the underlying application's architecture and technology stack.

Note that the template processing of Nunjucks is applied *before* the markdown processing. In this sense, we are using Nunjucks as a preprocessor.

In general terms, we are using Nunjucks to render a report. The report might make reference to constant values.



```
We can even do a 3D bouncing ball:

<x3d style="width: 80%; margin-left: auto; margin-right: auto; border: 1px solid blac
    <scene>
    <transform translation='{{ now.time-5 }} {{ now.h/3.0 }} 0'>
    <shape>
        <appearance>
        <material diffuseColor='1 0 0'></material>
        </appearance>
        <sphere></sphere>
    </shape>
    </transform>
    <transform translation='0 -2 0' scale="10 1 3">
    <shape>
        <appearance>
        <material diffuseColor='.2 0.2 0.5'></material>
        </appearance>
        <box size="2 2 2"></box>
    </shape>
    </transform>
    </scene>
</x3d>

<chart signals="h"></chart>
```

Figure 3. Report textual description vs. rendered report

In such cases, we can refer to those values in Nunjucks expressions by referencing the predefined `constants` object, *e.g.,* `constants.x`. Assuming x is an expression in our simulation results, `constants.x` will be replaced, during Nunjucks preprocessing, with the actual simulated value for x.

But what happens if the result we want to reference is time varying. Such results have different values at different times. Our Nunjucks preprocessor defines a special object referred to as `now`. So if y is a time varying variable, we can refer to the "current value" of y as `now.y`. Similarly, there is a built in function in Nunjucks called `at` and we can use that to refer to the value of a time varying signal *at* a particular time *e.g.,* `at(1.2).x`. Keep in mind the `now` objects relies on a notion of what the current time is. But how do we define "current value"?

The Modelica Playground application is equipped with a play/pause button and a scrubber controller. The application itself assumes, by default, that `now` represents the start time of the simulation. But by pressing "play" or dragging the scrubber control around, the value of `now` is automatically updated to the time associated with the current position of the scrubber. Any Nunjucks output that depends on the `now` variable is then automatically rerendered.

This templating is particularly useful when dealing with the potentially verbose constructs associated with the visualization languages described next.

5.5 2D Visualization

Markdown, like Modelica and HTML, is a declarative approach to rendering. It doesn't involve imperative commands for how to render. Instead, it focuses on a description of what to render and leaves it to the tooling and the platform to perform the rendering task according to the specifications. In order to promote a no/low-code approach for 2D visualization, a similar approach was required. Fortunately, browsers have built-in rendering capabilities for 2D (and 3D) visualizations. In fact, the browsers include *two* such approaches. The first is the Canvas API (*HTML Canvas 2D Context* 2011). The problem with the Canvas API is that it is *not* declarative. Fortunately, the other option, Scalable Vector Graphics (*Scalable Vector Graphics* 2018), also known as SVG, *is* declarative.

For our purposes, it is not sufficient to simply render SVG. A normal Markdown processor can already do that. What is required for the Modelica Playground is that the SVG be rendered *as a function of the simulation results*. In other words, where numerical literals would appear in SVG to describe positions, rotations, scaling and transformations, we require the ability to replace those numeric literals with *simulation results*. This is made possible thanks to the Nunjucks preprocessor described in subsection 5.3.

Note that these numeric literals might arise from constants in our simulation results. But more often than not, they arise from time varying variables in our simulation results. In the former case, we can use the `constants` variable described earlier and in the latter case, we can use the `now` variable. In this way, any SVG figure that references the `now` variable is effectively transformed automatically into an animation.

5.6 3D Visualization

Just as with 2D animation, our requirement for 3D animation depends on the ability to provide a declarative representation of the 3D scene we wish to render and then describing it via our templating capabilities in order to couple it to our simulation results for the purposes of visualization and animation. However, unlike the 2D case, browsers have no built-in analog to SVG for declarative specifications of 3D scenes.

But this isn't the end of the world. The first thing to note is that browsers *do* have built-in, hardware accelerated, 3D rendering capabilities in the form of WebGL (*WebGL* 2022). Also fortunate for us is the existence of a framework called X3D (*X3D* 2022) that *does* provide a declarative scheme for describing 3D scenes. Even thought X3D isn't built-in to browsers, it can be loaded into any standard compliant browser so it is the next best thing to a built-in capability.

So once again, we can leverage the Nunjucks rendering to inject numeric values into a declarative scene description. And once again, references to the `now` variable automatically translate into animations of our now three dimensional scene and thereby satisfying our requirement for a no/low-code approach to visualization.

5.7 Vega

As mentioned previously, the custom `<chart>` component relies on ECharts for rendering the chart. ECharts is one of many different visualizations libraries available for the browser. Another is called Vega (*Vega: A Visualization Grammar* 2022). An important property of the Vega approach is that it provides a rich visualization grammar. Through this grammar, users are able to describe a wide ranging set of data visualizations going well beyond simple plots as shown in Figure 4.

Just as with SVG and X3D, we have a declarative vocabulary for describing a nearly infinite set of rich visualizations. To support this, an additional custom component was added, the `<vega>` component. This component can be used to delimit a JSON object that conforms to the expected structure of a Vega visualization. In such cases, the custom component will be replaced, during rendering, by the actual Vega visualization.

5.7.1 Safe HTML

Allowing users to define their own markup brings with it some risks. A modern browser is actually quite a powerful platform and it is the platform that is used for lots of other important tasks besides visualizing Modelica models and their results. As such, we need to ensure that the Modelica Playground doesn't expose users to any security risks.

Fortunately, the rendering toolchain for `remark` includes the `rehype-sanitize` package (*reype-sanitize* 2021). While it might seem tempting to blacklist specific HTML elements (*e.g.,* the `<script>` element) in order avoid introducing opportunities for security exploits, it turns out that blacklisting is impractical. There are simply too many ways in a modern browser to give people unwanted access if you allow users to simply type in "code".

For this reason, `rehype-sanitize` employs a whitelisting approach. What this means, in practice, is that it is necessary to identify explicitly all legal elements *and attributes of those elements* and `rehype-sanitize` will remove any references to any non-whitelisted elements. This is a tedious process, but it is one that was followed in producing the Modelica Playground. The result is that the Modelica Playground should be a very safe "sandbox" in which to play around with Modelica code.

6 Content

The default mode for the Modelica Playground is to present the user with a "blank slate" where they can type in any Modelica code they wish and create any post processing report. There is a table of contents that can be accessed that provides a few simple examples as a means of getting started, but the base application is deliberately quite open ended.

However, there is a mechanism by which specific models and reports or even collections of models and reports can be "published" using the Modelica Playground. In this section, we'll discuss how this is accomplished.

6.1 Links for Sharing

As a user, if you develop a particular model (and associated post processing report) that you would like to share with other users, you can click on the "Copy Link to Clipboard" button. Doing so copies a URL to the clipboard (the same URL visible in the browser's address bar, in fact).

This URL can then be emailed to other users. The URL will include query parameters that encode the text of the model and report. As a result, anybody who follows the generated link will be placed in the Modelica Playground with the associated models and report already pre-loaded. The results themselves are too bulky to bundle in with the URL. But since they can be reconstituted simply by pressing the "Simulate" button, doing so means that the link recipient will then see exactly the same visualizations that the original author saw.

Such link sharing could even be the basis for collaboration since participants could each modify the models they receive and send them back to the original developer. Similarly, professors could assign homework to students and request the solutions be done in the Modelica Playground and the students could then copy the link to their solutions into an email and send them back to the professor.

6.2 Lesson Plans

In some cases, users may want to share more than just a single model. Instead, they may wish to share a collection of models that, progressively, tell a story or explain a topic. In the Modelica Playground application, a collection of models and their associated post processing reports is referred to as a *lesson plan*.

Each lesson in a lesson plan can consist of five distinct parts:

- **metadata**: This is used to specify a title for each lesson as well as an ordering for each lesson.

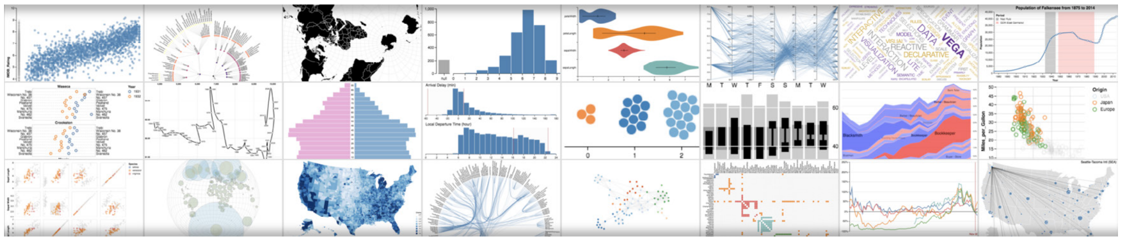


Figure 4. Samples of various Vega visualizations

- **a model**: The assumption is that each lesson will contain exactly one Modelica model.
- **explanation** (optional): The content of the explanation should be written in Markdown and will be rendered just above the model to provide some introductory context. Mathematical equations along with 2D and 3D figures may be used as part of this explanation but they may not reference simulation variables (since nothing has been simulated at this point).
- **a report** (optional): This report should be provided in the format described in Section 4. Unlike the explanation, it may reference simulation results since it is only rendered once simulation results are available.
- **macros** (optional): As mentioned previously, the model text and report text are encoded in each URL. This allows users to start from a lesson and potentially modify it for their own purposes. The creator of the lesson may have included predefined macros to be used in the post processing report. Since these are "static" (users aren't allowed to edit these), they are *not* contained in the URL. Instead, they are referenced as part of a templating "preamble" associated with the lesson itself.

Each lesson is composed of the various parts mentioned. These lessons are then bundled together into a lesson plan. The Modelica Playground expects this complete lesson plan to be bundled as a single JSON file that conforms to the Siren specification (Kevin Swiber 2017). But knowledge of Siren or the expected structure of that bundle are not required for content creators. Instead, they can use the `lessonplan` tool (Michael M. Tiller 2022a) to create such a file. Once created, the file *does not* need to be published on the `modelica.university` domain. Instead, it can be hosted *anywhere* and simply referenced via the `toc` query string parameter.

6.3 Examples

For reference, the following are examples of using the lesson plan functionality to create content. Hopefully, over time, users will start to create more such content.

- Lesson Plan Sample: The previously mentioned `lessonplan` tool used to bundle lessons includes, in its repository, an admittedly simple sample lesson plan. The bundled version of this lesson plan is hosted at `https://raw.githubusercontent.com/mtiller/lessonplan/master/sample.json`.
- Tour of Modelica: A more complete lesson plan is one that is bundled with the Modelica Playground. It can be found by clicking on the "Gallery of Lessons" in the upper right corner of the application. The goal of this lesson is to walk users through some of the basic functionality of Modelica.
- Content Creation Tutorial: This lesson plan is also available in the "Gallery of Lessons". Instead of teaching users about Modelica, this lesson is dedicated to teaching users about the Modelica Playground itself. It presents several examples that demonstrate the features discussed in this paper regarding post processing reports with the hope that, armed with this knowledge and combined with the documentation associated with the `lessonplan` bundler, users will create additional Modelica Playground content.

7 Data Management

As mentioned previously, one of the goals of this project was to avoid having to add cookie consent forms. This can be avoided so long as we avoid GDPR related concerns. Since this site is free and does not generate revenue in any way, we have no interest in tracking individual users. Doing so adds many more complications for absolutely zero benefit.

This has implications for how data associated with the application is managed and it is worth spending at least some time discussing this.

7.1 Analytics

It is quite common for web applications to include Javascript code that contacts some third party server to record the activity of visitors. This by itself is not a GDPR concern. It only becomes a GDPR concern when personally identifying information (PII) is recorded.

While we are interested in how many people utilize the site and what they utilize it for, we have no interest in being able to associate that activity with identifiable individuals. For this reason, we wanted to leverage a tracker that *did not* record such information. Fortunately, such analytics tools exist. The first on we tried was from `https://plausible.io`. This worked quite well and we would recommend it as an alternative to Google and other add targeting motivated trackers. Ultimately, we ended up using Cloudflare for our analytics. Like `plausible`, Cloudflare avoids recording PII data. It also has the benefit of being part of the Cloudflare platform which also includes many features related to content distribution, DDoS attack prevention and a whole host of other features.

It is important to note that analytics tools that do not track individual users do not benefit from the revenue associated with tracking users. As such, you should expect to pay for such tools since they do not pay for themselves by selling information about you to third parties.

7.2 History

The most recent addition to the Modelica Playground application is the introduction of user history. Every time a user runs a simulation, a record is made of that simulation. When you return to the Modelica Playground, you can access all your previous models, report templates and results.

Now it might seem like this must be a GDPR concern. But, in fact, it is quite easy to implement such functionality without violating the GDPR guidelines. The reason for this is that the information is not stored server side. Instead, the information is stored *directly in the users browser*. All modern browsers provide something called the IndexedDb API (World Wide Web Consortium 2021). This API allows applications to store data in a relational database directly on the machine that the browser was run from. Because the information *never leaves the users computer*, it doesn't violate the terms of the GDPR.

8 Future Work

Before wrapping up, it is worth some time to discuss potential future work to improve the Modelica Playground even further.

8.1 Improved Link Sharing

As already mentioned, the Modelica Playground allows users to capture their current work in the form of a special link. Such links can then be shared with other users via email, text message, Slack, *etc.*. But these links can be a bit problematic because they can be quite long. While this isn't generally an issue for the browsers (most browsers can tolerate very long URLs), it can be a problem for these various applications used to communicate the links because some applications impose their own limits on URL length. For this reason, users may find a "link shortener" useful.

Of course, there are many existing link shortening services and users are welcome to use those. The Modelica Playground links should work with any such service. But it is slightly inconvenient to visit a third party web site in order to create such a link (unless, of course, you have a browser extension installed that helps with that). An integrated link shortener could help with this.

The complication here is with respect to privacy. This would result in storing more user information. Furthermore, this information would have to be stored server side (unlike our current information which is all stored locally in the browser). Nevertheless, such a system could be made pretty easily GDPR compliant by simply being careful to only store the content but no information about the content creator. Most likely, the link shortener would simply store an association between a content hash and (only) the content.

Note, the same cannot be said for most existing link shortening services. By registering a link with them, you are implicitly opting in to allowing tracking of users who visit the shortened link.

8.2 Support MSL

Another welcome addition would be the ability to reference the Modelica Standard Library (MSL) from within the code written in the Modelica Playground. Although actually loading the MSL in the browser is probably well beyond the scope of practical improvements, it could be loaded server side prior to running the code. This would slow down simulations because loading the MSL takes a non-trivial amount of time. But it is quite possible that references to the MSL could be identified client side and the server could be told *a priori* whether or not loading the MSL was necessary.

Allowing references to the MSL would then allow the Modelica Playground to easily describe more complex models leveraging components available from the Modelica Standard Library. For the foreseeable future, such models would only be represented in their pure text form (*i.e.,* no diagram rendering). But even that should be reasonably intuitive for users.

8.3 Gist support

At the moment, all user work is stored in the browser. But another option for storage would be to store content in a Github Gist. In this case, the content would be available beyond the user's browser. While Gists can be marked "secret", they are still accessible to anybody who has access to the Gist id. So while this is possible, it wouldn't be implemented unless there was significant interest.

8.4 Simulation Caching

The current HTTP API receives the model content as part of the payload. The server could easily cache the simulation results of previous simulations of that particular model. For models presented in lessons (where the model is frequently run, unmodified), such a cache could improve the simulation time as perceived by the user (by avoiding the simulation altogether).

9 Conclusion

Despite being over 20 years old, Modelica remains a compelling technology. It is at least as relevant and useful now as it ever was. The goal of the Modelica Playground is to keep it relevant by making it as accessible or more accessible than alternatives.

By leveraging a variety of open source tools, the Modelica Playground provides a platform not only for exploring the Modelica Language online but also for creating compelling content showcasing Modelica along side interactive 2D and 3D visualizations.

Acknowledgements

This project would have been impossible without the availability of open source tools like the ones mentioned in this paper. This application is truly built on the shoulders of giants.

I'd also like to thank my daughter, Alisha Tiller. Her freshman year at Purdue has helped rekindle my passion for math and physics and I created this tool in large part so that students like her would have a Modelica based alternative to tools like VPython.

References

Apache Software Foundation (2022). *ECharts*. Version 5.3.2. URL: https://echarts.apache.org/en/index.html (visited on 2022-03-31).

Bruce Sherwood (2022). *VPython*. Version 7. URL: https://vpython.org/ (visited on 2022-04-24).

Brunsfeld, Max (2022). *Tree-sitter*. URL: https://tree-sitter.github.io/tree-sitter/ (visited on 2022-03-03).

Empscripten Contributors (2021). *Emscripten*. Version 3.1.9. URL: https://emscripten.org/ (visited on 2021-11-22).

Facebook (2022). *React*. Version 18.1.0. URL: https://reactjs.org/ (visited on 2022-04-26).

Google (2022). *Go Playground*. Version 1.18. URL: https://go.dev/play/ (visited on 2022-04-15).

HTML Canvas 2D Context (2011). URL: https://dev.w3.org/html5/2dcontext-LC/ (visited on 2011-05-24).

John MacFarlane (2021). *Commonmark*. Version 0.30. URL: https://commonmark.org/ (visited on 2021-06-19).

KaTeX (2022). Version 0.5.13. URL: https://katex.org/ (visited on 2022-04-13).

Kevin Swiber (2017). *Siren*. Version 0.6.2. URL: https://github.com/kevinswiber/siren (visited on 2017-04-27).

MDX Community (2022). *MDX Playground*. Version 2.1.1. URL: https://mdxjs.com/playground/ (visited on 2022-03-31).

Michael M. Tiller (2022a). *Lessonplan*. URL: https://github.com/mtiller/lessonplan (visited on 2022-04-30).

Michael M. Tiller (2022b). *Modelica-tree-sitter*. URL: https://github.com/mtiller/modelica-tree-sitter (visited on 2022-04-30).

Microsoft (2022a). *Monaco*. URL: https://microsoft.github.io/monaco-editor/ (visited on 2022-04-24).

Microsoft (2022b). *Monarch*. Version 0.33.0. URL: https://microsoft.github.io/monaco-editor/index.html (visited on 2022-02-03).

Microsoft (2022c). *TypeScript Playground*. Version 4.6.4. URL: https://www.typescriptlang.org/play (visited on 2022-04-28).

Microsoft (2022d). *Visual Studio Code*. URL: https://code.visualstudio.com/ (visited on 2022-04-24).

Mozilla Software Foundation (2020). *Nunjucks*. Version 3.2.2. URL: https://mozilla.github.io/nunjucks/ (visited on 2020-07-20).

Open Modelica Consortium (2022). *Open Modelica Compiler*. Version 1.19.0-dev.beta1. URL: https://openmodelica.org/ (visited on 2022-04-20).

Pallets (2022). *Jinja*. Version 3.1.x. URL: https://jinja.palletsprojects.com/en/3.1.x/ (visited on 2022-04-28).

Rehype (2022). Version 12.0.1. URL: https://github.com/rehypejs/rehype (visited on 2022-01-29).

Remark (2021). Version 14.0.2. URL: https://remark.js.org/ (visited on 2021-11-18).

remark-math (2022). Version 5.1.1. URL: https://github.com/remarkjs/remark-math (visited on 2022-04-24).

reype-sanitize (2021). Version 5.0.1. URL: https://github.com/rehypejs/rehype-sanitize (visited on 2021-12-08).

Scalable Vector Graphics (2018). URL: https://www.w3.org/TR/SVG2/ (visited on 2018-10-04).

Tidefelt, H. and O. Tronarp (2020). *Modelica Change Proposal MCP-0033 Annotations for Predefined Plots*. Tech. rep. Modelica Association.

Vega: A Visualization Grammar (2022). Version 5.22.1. URL: https://vega.github.io/vega/ (visited on 2022-03-25).

WebAssembly Community Group (2022). *WebAssembly Specification*. Version 2.0. URL: https://webassembly.github.io/spec/core/ (visited on 2022-04-27).

WebGL (2022). URL: https://www.khronos.org/webgl/ (visited on 2022-04-24).

World Wide Web Consortium (2021). Version 3.0. URL: https://www.w3.org/TR/IndexedDB/ (visited on 2021-10-06).

X3D (2022). URL: https://www.web3d.org/x3d/what-x3d (visited on 2022-04-24).

Towards an Open Platform for Democratized Model-Based Design and Engineering of Cyber-Physical Systems

Mohamad Omar Nachawati[1] Gianmaria Bullegas[1] Andrey Vasilyev[1] Joe Gregory[1] Adrian Pop[2]
Maged Elaasar[3] Adeel Asghar[4]

[1]Perpetual Labs Ltd., UK, {omar, gian, andrey}@perpetuallabs.io
[2]Linköping University, Sweden, adrian.pop@liu.se
[3]NASA Jet Propulsion Laboratory, USA, maged.e.elaasar@jpl.nasa.gov
[4]Open Source Modelica Consortium, Sweden, adeel.asghar@liu.se

Abstract

This paper reports on the development of GitWorks, an open platform for democratizated Model-Based Design of cyber-physical systems (CPS). The GitWorks platform is currently under development by Perpetual Labs Ltd in collaboration with the Open Source Modelica Consortium (OSMC)[1] and the OpenCAESAR project[2]. In this paper, we present our vision for the platform, its system architecture and a prototype implementation. We also present a case study that demonstrates the use of the proposed platform for enabling the seamless integration of Modelica models into a broader range of systems engineering processes for complex product development. In the long-term, the platform also aims to enable the integration of Modelica tools with advanced systems engineering processes that rely on other domain specific languages (e.g. SysML v2, BPMN, etc.).

Keywords: MBD, MBSE, Modeling, Simulation, Interoperability, Cyber-Physical Systems, Semantic Twin, Real-time Collaboration

1 Introduction

The Modelica language (Elmqvist, Mattsson, and Otter 1998; Fritzson and Engelson 1998) has a growing user community that produces a large and constantly-increasing code base of models. However, there is a lack of tools to address a number of advanced model-management use cases, such as semantic search, analysis, cross-referencing, checking, component selection automation, for a large body of models (Johansson, Pop, and Fritzson 2005). Despite recent developments (Sirin et al. 2015; Isasi, Noguerón, and Wijnands 2015; Hussain et al. 2022), tool support for the integration of Modelica models into advanced Model-Based Systems Engineering (MBSE) practices remains limited (Larsen et al. 2016). This hinders the reuse of models within the Modelica community, and particularly in an industrial context, can greatly limit the potential for adoption of Modelica tools within integrated Model-Based Design (MBD) and product development processes.

There are multiple engineering processes that precede modeling and simulation within a complex product development lifecycle. The information generated by these processes defines the structure, configuration, and input parameter data used by the executable system models. For example:

1. The definition of operational scenarios and associated system requirements. These define the critical behaviors that the system must achieve, the circumstances under which they must be achieved, and other non-functional properties of the product.

2. The definition of the high-level architecture of the system. This includes the system's hierarchical division into different subsystems, their components, parameter values, and interconnections. Alternative design solutions can be evaluated against the criteria defined in (1) via simulation.

Many tools and formalisms can be used in these phases to capture the system information as part of an MBSE framework, such as UML, SysML, AADL, FMDesign, BDPM, and OPM (J. Ma et al. 2022; Basnet et al. 2022). Integration of system simulation and analysis with such MBSE models is difficult to achieve, particularly during the early phases of the system lifecycle, because domain-specific models often lack a common notation (Madni and Sievers 2018). Collecting, aggregating and exchanging information at the system level is complex and often error-prone, which hampers system-wide visibility in a multi-disciplinary concurrent design setting (McDermott et al. 2020). This limits the ability to analyze system-level requirements (such as performance and dependability) in the early design phases, causing the postponement of design decisions to later phases. This, in turn, reduces possible opportunities to study alternative solutions and validation of fitness for purpose. It also increases costs, in terms of time and skill, of design refinements if irreversible consecutive design decisions are made in the early stages of the development (Stirgwolt, Mazzuchi, and Sarkani 2022).

[1]https://openmodelica.org/home/consortium
[2]https://www.opencaesar.io/

The limitations of current MBSE frameworks and their poor integration with system simulation environments, such as Modelica-based tools, contribute to an increased barrier-to-entry for the adoption of MBD. These issues are especially acute for Small and Medium Enterprises (SMEs) that typically do not have the resources to link Commercial-Off-The-Shelf (COTS) tools into integrated tool chains and lack the in-house expertise to develop custom models from scratch.

GitWorks aims to democratize access to MBD for SMEs, independent developers and academia. GitWorks has been designed as a turn-key solution for model management and integration provided with the convenience of a Software as a Service (SaaS) product. It includes the GitWorks Commons which provides a searchable repository of models, tools and services with a try-before-you-buy business model. It also includes a Web application for integrated data and knowledge management, as well as a web-based collaborative Modelica editor.

This paper describes the early design and implementation of GitWorks. Specifically, the contributions of this paper are three-fold: First, we propose a system architecture for the platform to support model-based design and engineering of cyber-physical systems. Second, we develop a prototype implementation of the GitWorks platform that is focused on enabling the seamless integration of Modelica models into a broader range of MBSE activities. Third, we conduct a preliminary case study to demonstrate the use of the proposed platform for the federated design and engineering of an aircraft passenger air conditioner (PACK) system.

The rest of this paper is organized as follows. Section 2 provides an overview of our vision for the GitWorks platform, describing its design goals, conceptual system architecture, and user interfaces. Section 3 describes the prototype implementation of GitWorks and tooling for enabling the use of Modelica in the larger MBSE process. Section 4 presents a preliminary case study to demonstrate the use of the GitWorks for the federated design and engineering of a PACK system. Finally, Section 5 concludes the paper and provides some brief remarks on directions for future work.

2 GitWorks Platform Overview

This section provides an overview of the GitWorks platform, describing its design principles and conceptual system architecture. GitWorks aims to overcome several of the limitations of current systems engineering practices (Elaasar et al. 2019) by introducing three key concepts and their related functionalities:

DEMOps: DevOps for Digital Engineering and Manufacturing. Poor configuration management (CM) practices exacerbate trust issues in current MBSE practices. DEMOps introduces the notion of a Git-like history of changes made across inter-related model fragments. Enables *traceability* of information provenance and design decisions. This refers to the ability to trace from an authority to its design decisions and constraints, and from the latter to their rationales. Without this capability, a system description becomes a disorganized collection of information artifacts. Enables *repeatability*. This refers to the ability to encapsulate the analysis of the system description, including its dependencies, such that it becomes repeatable. This is important to maintain confidence in the analysis over time and use it to assert desirable properties. Enables *durability*. This refers to the ability to version control the information that describes or analyzes a system in such a way that versions become immutable. Without this, it is impossible to perform audits and repeat analyses. Enables *efficiency*. This refers to the ability to automate processes using CI/CD practices that would otherwise be manual and tedious (Elaasar et al. 2019). Without this, such processes become expensive and error-prone. In GitWorks, these functionalities are fulfilled by the Projects environment described in Section 2.1.

Semantic Twin. System information is captured using a precise language that is rooted in mathematics and formal logic. System descriptions are specified using common vocabularies consisting of concepts and their properties and relationships all expressed in a formal language. This enables *digital continuity*, which refers to the interoperability of system information contained in different information artifacts that are produced by different parties during different phases of the product lifecycle. It also enables the augmentation or, in some cases, the replacement of typically human-led processes (such as reporting, model transformation, and validation & verification of system information) through the use of powerful *automations* such as logical reasoning, machine learning, data mining, etc. In GitWorks, these functionalities are fulfilled by the OML language and the versioned triple store and associated reasoner (see Section 2.1).

Digital Prototype. In the context of DEMOps, *Digital Prototype* refers to the usage of digital environments to facilitate the co-simulation of engineering models, connection with HardWare-In-the-Loop (HWIL) frameworks and real-time system data to support *Virtual system Integration*, Validation and Verification from the early stages of the product lifecycle. In GitWorks, these functionalities are fulfilled by the Modelica Studio environment (see Section 3). In its current version, Modelica Studio only supports editing and simulation of Modelica-based models. In the future, we aim to introduce advanced functionalities and analyses, such as co-simulation of FMUs, HWIL, model surrogatization, optimization, uncertainty quantification, etc. The Digital Prototype must be supported by a scalable, cloud-based computational infrastructure to enable the more computationally-demanding workflows, and must also be integrated with the CI/CD pipeline to enable automation. In GitWorks, this is achieved via the integration of a standard CI/CD pipeline with a scalable HPC environment (see Section 2.1).

It should be noted that the concepts of the Digital Proto-

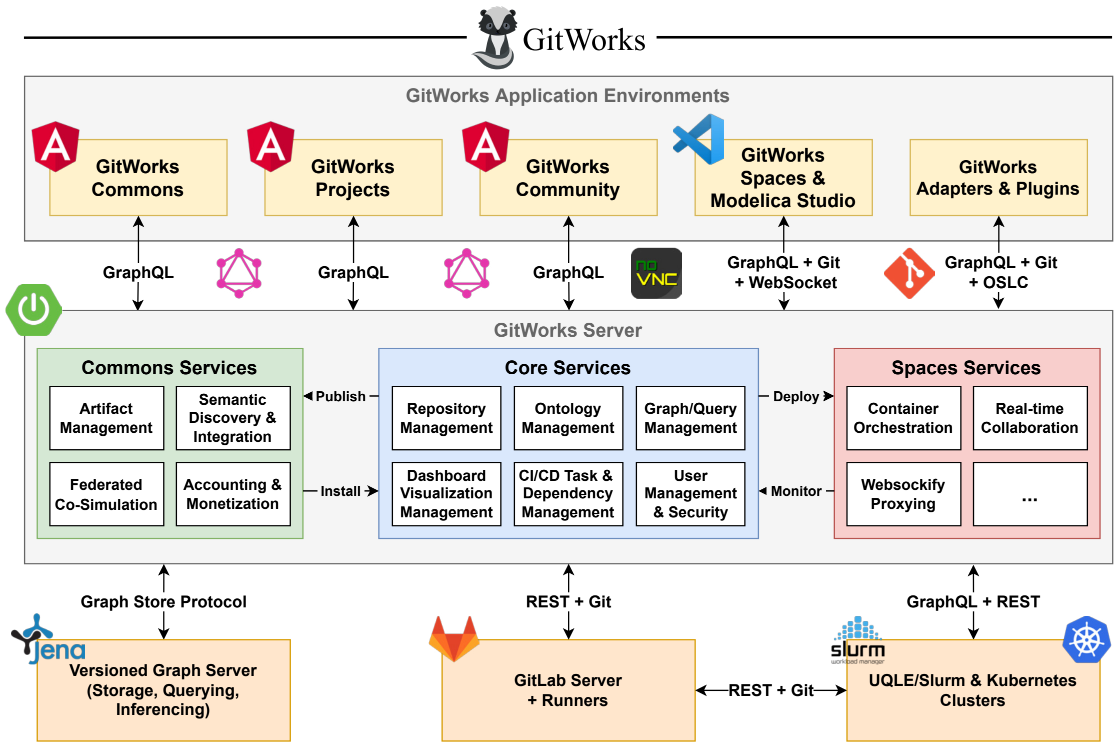


Figure 1. Conceptual software architecture of the GitWorks Platform[3]

type and the Semantic Twin are tightly linked and that the integration of the two enables the realization of new and powerful workflows. The Digital Prototype acts as a backbone to organize and facilitate access to system-related information scattered through the different engineering artifacts. An example application of this idea is provided in the PACK case study in Section 4. The system architecture information contained in a SysML model is used to automatically generate the high-level structure of the Modelica model. Also, queries against the GitWorks Commons, enabled by the OML language representation of Modelica, can be used to find suitable, port-compatible components to complete the model.

2.1 Conceptual System Architecture

The high-level software architecture of the GitWorks platform is shown in Figure 1. From the perspective its users, GitWorks provides multiple application environments for interacting with the platform, to include the *GitWorks Commons* for the publishing and reuse of digital engineering artifacts, the *GitWorks Projects* environment for Semantic Twin-powered DevOps for digital engineering, analysis, reporting and management of the digital thread. In addition to these environments, through Git, OSLC and REST-based APIs, GitWorks is also designed to integrate with third-party adapters and plugins for authoring and reporting.

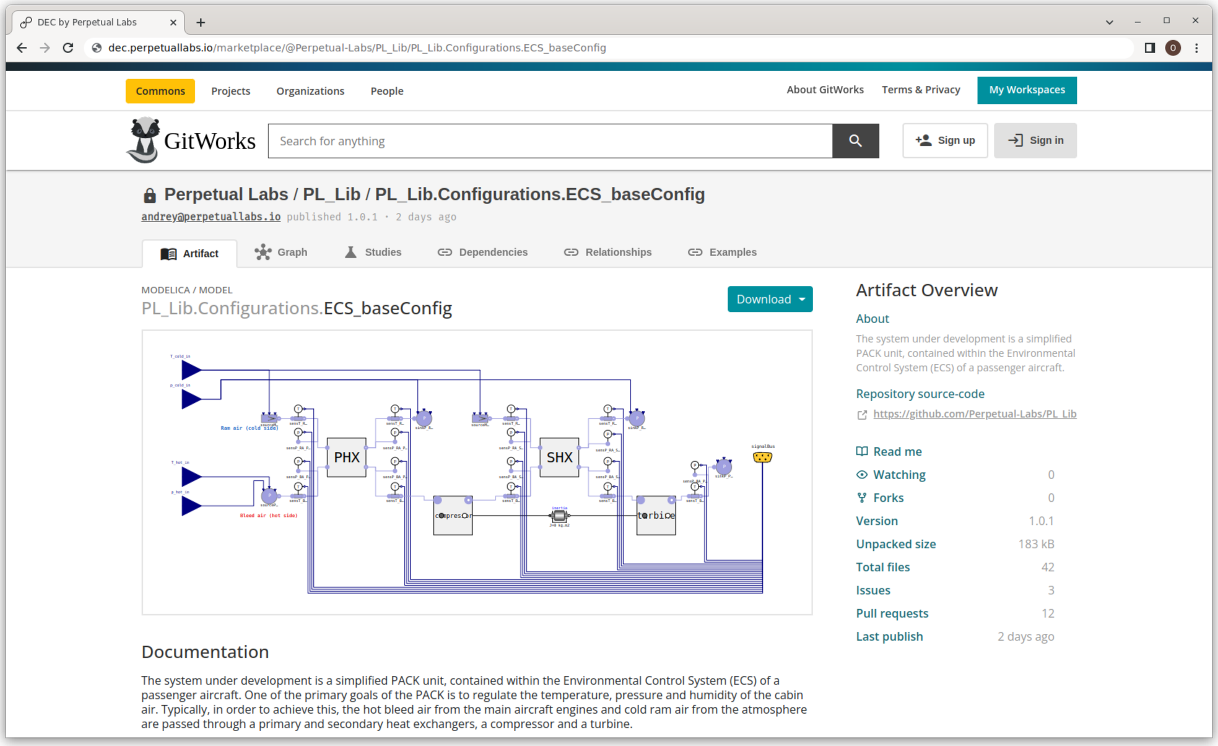
Supporting these application environments, *GitWorks Server* is designed as a middleware of essential services, as well as a nexus for accessing data across organizations and third-party tools and systems. The GitWorks Server is implemented as a Spring Boot application and depends on a GitLab server to provide Git repository hosting and CI/CD capabilities. GitLab[4] is an open-source DevOps platform based on the popular Git version control system. GitLab provides a REST API for programmatic access and manipulation of resources, such as repositories, artifacts and users. This API is used to implement much of the Git-centric capabilities provided by GitWorks through GitLab4J[5].

Unlike traditional version control systems, such as CVS, where changes are managed at the file level, Git manages changes at the repository level so that for any particular commit one can recover the precise state of entire repository at that point. To facilitate the management of artifacts developed and owned by different stakeholders, repositories can be organized into groups that capture the hierarchical relationships between systems and their components or the relationships among the different entities participating in a supply chain.

Leveraging Git's repository level change management mechanism, from the artifacts contained in a GitWorks project repository, GitWorks constructs a versioned Semantic Twin that captures the interdependencies and trace-

[3]All trademarks, logos and brand names are the property of their respective owners.

[4]`https://www.gitlab.com`
[5]`https://github.com/gitlab4j/gitlab4j-api`



Perpetual Labs / PL_Lib / PL_Lib.Configurations.ECS_baseConfig

andrey@perpetuallabs.io published 1.0.1 · 2 days ago

MODELICA / MODEL

PL_Lib.Configurations.ECS_baseConfig

Artifact Overview

About

The system under development is a simplified PACK unit, contained within the Environmental Control System (ECS) of a passenger aircraft.

Repository source-code

https://github.com/Perpetual-Labs/PL_Lib

Read me

Watching	0
Forks	0
Version	1.0.1
Unpacked size	183 kB
Total files	42
Issues	3
Pull requests	12
Last publish	2 days ago

Documentation

The system under development is a simplified PACK unit, contained within the Environmental Control System (ECS) of a passenger aircraft. One of the primary goals of the PACK is to regulate the temperature, pressure and humidity of the cabin air. Typically, in order to achieve this, the hot bleed air from the main aircraft engines and cold ram air from the atmosphere are passed through a primary and secondary heat exchangers, a compressor and a turbine.

Figure 2. The GitWorks Commons UI for Modelica and other digital engineering artifacts

ability links that relate heterogeneous artifacts through the product lifecycle. The Semantic Twin forms a federated knowledge graph that is stored as RDF triples in a custom versioned triple store, which is developed by Perpetual Labs and is based on Jena (Carroll et al. 2004) TDB2. By creating trace links between heterogeneous elements, such as requirements, simulation models, simulation results, and test results, GitWorks provides traceability of design changes and branches at the system level and throughout the product lifecycle, while also allowing each stakeholder the necessary flexibility in managing their own datasets and internal development cycles.

GitWorks provides multiple adapters for automatically enriching the Semantic Twin from a project repository. While the focus of this paper is on the Modelica adapter (see Section 3.3), Perpetual Labs is actively working on adapters for other representations, including SysML and CAD. These adapters read artifact files from the Git repository, for example a Modelica library, and extract a semantic representation in the Ontology Modeling Language (OML).

OML serves as the core ontological language for the GitWorks platform and was originally developed by the Jet Propulsion Laboratory as part of the CAESAR project (Wagner et al. 2020). One of the goals of CAESAR has been to provide a set of OML vocabularies that capture some of the common concepts and relations used in systems engineering (Bayer et al. 2021). OML provides a foundation with well-defined semantics that can be used to model different types of engineering artifacts in a semantically consistent and interoperable fashion. OML extends OWL 2 DL (Web Ontology Language 2 - Description Logic) in such a way that retains the benefits of OWL 2 DL while addressing some of its limitations (Wagner et al. 2020).

GitWorks Commons. The GitWorks Commons enables the seamless reuse of design and engineering artifacts across tool, domain and organization boundaries. Within the GitWorks platform, an artifact is considered as the basic unit of reuse, where its coarsity depends on the tool and domain vocabulary. As shown in Figure 2, a single Modelica file, for example, may contain multiple models, each of which would be considered an artifact in the Commons. When the artifacts from a project repository are ready for release, these artifacts along with their dependencies are bundled together as package and then uploaded to the Commons as a versioned artifact. The GitWorks leverages GitLab Package Registries to support multiple package managers, including Maven and npm. The Commons also supports semantic discovery and integration of artifacts through a SPARQL endpoint that can be used to query the metadata extracted from artifacts via specific adapters (see Section 3.3).

The GitWorks Commons also provides vendors with multiple publication and deployment possibilities. Vendors can choose whether to allow users to download sources and/or binaries, or only provide cloud-based access. For example, a Modelica model can be published as: (1) source code (i.e. a model library), (2) compiled FMU (Model-Exchange or Co-Simulation), and/or (3) as a REST service that can be used in a federated co-simulation. We are investigating different monetization strategies for the GitWorks similar to those proposed for the Digital Manufacturing Commons (DMC) (Beckmann et al. 2016), including (1) payment for each download or execution, (2) payment based on computational usage, and (3) freemium software as a service. To support federated collaboration for enhanced data security and intellectual property protection, we are also actively working on the integration of co-simulation engines, specifically Maestro2 by the INTO-CPS project(Larsen et al. 2016).

GitWorks Projects. The Projects environment is a Semantic Twin-powered Web application for integrated data and knowledge management, and exploration and visualization of the digital thread across multiple disciplines, organizations and product lifecycle stages. It enables the exploration, querying, and modification of OML-based knowledge base using a Web-based GUI, similar to WebProtégé (Tudorache, Vendetti, and Noy 2008) for OWL2, and provides customizable OML vocabularies for different cyber-physical system lifecycle activities, such as requirements analysis, system modeling, verification and maintenance. At its core, a project corresponds to a Git repository of OML vocabularies and descriptions, which can be cloned and edited using an OML IDE, such as Rosetta[6] or Luxor[7].

The GitWorks platform leverages GitLab to provide DevOps capabilities for project repositories, to include Git-based version control, issue management, and CI/CD. To enable Semantic Twin-powered authoring and reporting, every project repository on GitWorks is backed by both a Git repository and a corresponding RDF triple store. The versioned RDF triple store serves as a cache to accelerate semantic queries against the repository and can be reconstructed directly from the files in the Git repository. A project can import artifacts from the GitWorks Commons as dependencies, forming a federated knowledge graph that enables all the stakeholders of a complex engineering system to make specific system information and data available to other project participants independently of the specific tools that they are using (i.e. a Semantic Twin).

An HPC CI environment based on GitLab Runner and the Slurm Workload Manager[8] is under active development, and enables computational expensive analyses such as simulation-based requirements verification, uncertainty quantification and optimization to be seamlessly integrated into the DEMOps pipeline. We have already tested different analysis toolkits using our HPC CI environment, to include UncertainPy (Tennøe, Halnes, and Einevoll 2018) and Dakota (Adams et al. 2020). We are working to provide seamless support for surrogate-assisted methods to accelerate computationally expensive analyses using methods such as pre-trained surrogate models for accelerated simulation, as done by JuliaSim (Rackauckas et al. 2021), and dynamically generated surrogates, as done by GreyOpt (Nachawati and Brodsky 2021), for enhanced optimization.

GitWorks Community. Finally, the Community environment enables users and organizations on the platform to connect with one another in a kind of social network for Digital Engineering. Each user is provided with a profile page that contains a public bio with an activity stream and links to associated published artifacts, project workspaces, and organizations.

3 Modelica Tooling for the GitWorks

This section describes the prototype implementation of the GitWorks tooling for enabling the use of Modelica in the larger MBSE process. Specifically, we report on the progress of our development of: (1) Modelica Studio, a Semantic Twin-powered Modelica text and diagram editor for VSCode for Web, (2) OMFrontend.js[9], a reusable and open-source AGPLv3-licensed library for the parsing and analysis of Modelica source code, which serves as the foundation of Modelica Studio, and (3) the ModelicaOML adapter, also based on OMFrontend.js, for automatically enriching the Semantic Twin from Modelica artifact repositories.

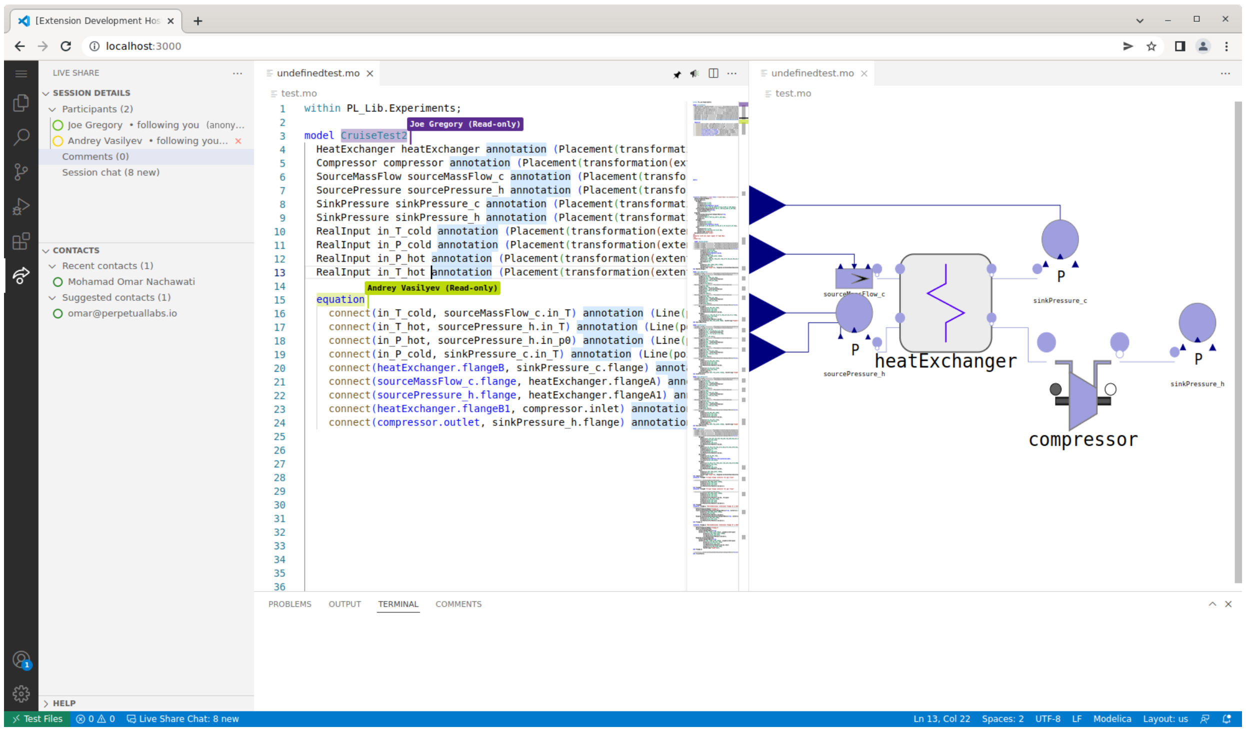
3.1 Modelica Studio

We have developed Modelica Studio as a VSCode for Web extension that serves as a Semantic Twin-powered authoring environment for Modelica. Modelica Studio, shown in Figure 3, is designed to support three levels of collaboration: (1) federated, in-the-large collaboration enabled by the GitWorks Commons, (2) Git-based collaboration, using branches and pull requests, and (3) real-time collaboration, using the VSCode LiveShare extension[10].

While several attempts have been made towards the development of a Web-based Modelica editor, significant limitations preclude their use as a collaborative Modelica development environment for the GitWorks:

Modelon Impact (Elmqvist, Malmheden, and Andreasson 2019) is a closed-source, cloud-based platform that provides a Modelica diagram and code editor that runs in a Web browser. While it runs in the browser, the implementation appears to require an independent server-side session for each editor instance, where the Optimica Compiler Toolkit (OCT) is used to construct and maintain a semantic model that mirrors what is opened in the editor. This approach simplifies the logic on the client-side, however, the critical dependency on continuous server-side processing for each editor instance can quickly add up

[6] https://github.com/opencaesar/oml-rosetta/
[7] https://github.com/opencaesar/oml-luxor
[8] https://slurm.schedmd.com/documentation.html
[9] https://github.com/OpenModelica/OMFrontend.js
[10] https://visualstudio.microsoft.com/services/live-share/



```
within PL_Lib.Experiments;

model CruiseTest2
  HeatExchanger heatExchanger annotation (Placement(transformat
  Compressor compressor annotation (Placement(transformation(ex
  SourceMassFlow sourceMassFlow_c annotation (Placement(transfo
  SourcePressure sourcePressure_h annotation (Placement(transfo
  SinkPressure sinkPressure_c annotation (Placement(transformat
  SinkPressure sinkPressure_h annotation (Placement(transformat
  RealInput in_T_cold annotation (Placement(transformation(exte
  RealInput in_P_cold annotation (Placement(transformation(exte
  RealInput in_P_hot annotation (Placement(transformation(exten
  RealInput in_T_hot annotation (Placement(transformation(exten

  equation
    connect(in_T_cold, sourceMassFlow_c.in_T) annotation (Line(
    connect(in_T_hot, sourcePressure_h.in_T) annotation (Line(p
    connect(in_P_hot, sourcePressure_h.in_p0) annotation (Line(
    connect(in_P_cold, sinkPressure_c.in_T) annotation (Line(po
    connect(heatExchanger.flangeB, sinkPressure_c.flange) annot
    connect(sourceMassFlow_c.flange, heatExchanger.flangeA) ann
    connect(sourcePressure_h.flange, heatExchanger.flangeA1) an
    connect(heatExchanger.flangeB1, compressor.inlet) annotatio
    connect(compressor.outlet, sinkPressure_h.flange) annotatio
```

Figure 3. Modelica Studio, a Semantic Twin-powered Modelica editor extension for VSCode for Web

to extensive computational resource requirements, especially for open DevOps and collaborative platforms, such as GitHub. Although the Modelon community provides an open-source JavaScript client[11], it is tightly-coupled to the Modelon Impact platform.

WebMWorks (Wan et al. 2013) is also a closed-source, cloud-based platform that provides a Web-based Modelica diagram and code editor. WebMWorks follows a similar implementation approach to Modelon Impact, but uses the OpenModelica Compiler (OMC) (Fritzson, Pop, et al. 2020) instead of OCT to maintain the server-side, semantic model that mirrors what is opened in the editor. TongYuan, the developers of WebMWorks, do not appear to currently offer access to WebMWorks.

WebGME-DSS (Kecskes et al. 2019) is an open-source, cloud-based modeling environment that provides a Web-based Modelica diagram-only editor. The implementation is based on GME, where a translator is used to convert the interface of a Modelica model into an instance of a GME-based meta-model representing a subset of the Modelica language. Although WebGME-DSS is open-source (MIT-licensed), WebGME-DSS only supports a small subset of the Modelica language and does not appear to be actively maintained.

OMWeb (Torabzadeh-Tari et al. 2011) is an open-source platform for editing and simulating Modelica models in a Web browser. Although OMWeb provides the ability to edit Modelica code and visualize simulation results, it does not provide a Web-based Modelica diagram editor. Furthermore, OMWeb appears to be in maintenance mode.

The diagram editor of Modelica Studio provides a user experience similar to that of OMEdit (Fritzson, Pop, et al. 2020), supporting the composition of new Modelica models by dragging and dropping Modelica model components onto the canvas. Changes made in the diagram editor are immediately propagated to the text editor, and vice versa. Noteably, the component palette in Modelica Studio is also designed to integrate with the GitWorks Commons to provide seamless dependency management. Also, unlike the other previously mentioned Web-based Modelica editors, Modelica Studio is largely serverless and the rendering and editing of Modelica models is done on the client without requiring a heavy-weight remote process to maintain a corresponding semantic representation of the contents of the editor. This design decision was made to significantly improve the scalability of the platform and to help realize the goal of making the GitWorks an open platform for digital engineering.

As shown in Figure 4, Modelica Studio largely depends on the OMFrontend.js (see Section 3.2) for the implementation of the Modelica Language Server to provide text and diagram editing support for VSCode for Web. Modelica Studio is designed to integrate with the OMWebService[12] REST API for the simulation of Modelica models in the browser. OMWebService is developed largely as a

[11]`https://github.com/modelon-community/impact-client-js`

[12]`https://github.com/OpenModelica/OMWebService`

wrapper around the OMC and OMSimulator for simulating Modelica models, Functional Mock-up Units (FMU), and System Structure and Parameterization (SSP) models. In response to a request to run a simulation on a model, OMFrontend.js sends the flattened simulation model to OMWebService. The results of the simulation are then returned as a CSV file, which can then be plotted in Modelica Studio or a third-party application. OMWebService is developed in Python and uses the OMPython interface to communicate with the OMC.

3.2 OMFrontend.js

To facilitate the development of Modelica Studio and other Modelica language tools, we have developed the OMFrontend.js library that provides an API for analyzing and manipulating Modelica text documents in both Node.js and Web browser environments. This library handles Modelica parsing, instantiation, flattening, expression evaluation as well as diagram and SVG icon rendering. It simplifies the implementation of language service features in Modelica Studio (see Section 3.1), such as diagnostics, hovers, links, completion, folding, and formatting. We also use OMFrontend.js to implement the ModelicaOML adapter (see Section 3.3) for enriching the Semantic Twin automatically from Modelica source code.
OMFrontend.js provides a context object that manages the collection of opened documents and Modelica libraries and serves as the mechanism for handling references to Modelica classes defined in different files. The context object seamlessly resolves Modelica files stored on the local file system, virtual Web browser file system, and via HTTP, depending on whether it is running in a Node.js or Web browser environment.

While the OpenModelica compiler uses a parser that is generated from an ANTLR3 grammar, to support browser-based editing we found the need to develop a new parser for OMFrontend.js. The new Modelica parser is built using the tree-sitter[13] parser generator. The tree-sitter-modelica[14] project contains the Modelica grammar for the tree-sitter parser generator. OMFrontend.js then constructs an abstract syntax tree from the tree-sitter concrete syntax tree. This abstract syntax tree is akin to the class tree described in the Modelica Language Specification (MLS). Unlike the tree-sitter concrete syntax tree which is incrementally reparsed, the abstract syntax tree needs to be reconstructed every time the underlying text changes. To reduce latency, this is done in a lazy fashion inspired by the red/green trees[15].

3.3 ModelicaOML Adapter

The purpose of the ModelicaOML adapter is to handle the conversion of Modelica source code and OML conforming to the ModelicaOML vocabulary presented in Figure 5. This enables the automatic enrichment of the Semantic Twin directly from the Modelica artifact repositories based on the representation provided by OMFrontend.js.

The ModelicaOML vocabulary[16] defines concepts such as `Class`, `Block`, `Model`, `Package`, `Function`, `Record`, `Type` to model the Modelica class restrictions and `Component` to model the components. The component and class prefixes are also modeled as scalar enumerations: `Prefix` and `ClassPrefix`. There are also relations that bind these concepts together such as `hasType`, `hasPrefix`, `hasClassPrefix`, `contains`, `extendsClass`, etc.

4 PACK Case Study

This case study aims to demonstrate how the GitWorks platform enables collaborative and federated design and development throughout the systems engineering process.

The system under development is a simplified PACK unit, which is itself a subsystem of the Environmental Control System of a passenger aircraft. One of the primary goals of the PACK is to regulate the temperature, pressure and humidity of the cabin air (Jennions et al. 2020).

4.1 Federated Development of the PACK System

The PACK project comprises seven tasks with a focus on the systems engineering process: Define project; Define system requirements; Define system architecture; Define subsystem behavior; Perform analysis; Verify system requirements; Generate Reports. These tasks are to be performed by specialists with various roles: Project Manager, System Architect, Lead Systems Engineer and a team of Design Engineers.

The tasks are performed using either third-party or GitWorks-hosted tools (e.g. Modelica Studio), and each task yields one or more artifacts with a particular filetype (e.g. a SysML or a Modelica model). The GitWorks platform supports this use case in the manner summarized in Figure 6. This diagram describes how OML adapters are employed to consolidate the knowledge contained in different modeling artifacts by creating a unified, RDF-based Semantic Twin.

In this case, the Project Manager creates an OML Git repository (corresponding to the project) using the GitWorks Workbench environment. The knowledge regarding the participating organizations, teams and people is captured by importing the relevant OML vocabularies and populating an OML description file. In the Workbench environment, the Project Manager defines the relevant project tasks and assigns responsibility to members of the project. A small section of the resulting OML description is presented in Listing 1, in which five roles have been defined and tasks assigned.

[13] `https://tree-sitter.github.io/tree-sitter/`
[14] `https://github.com/OpenModelica/tree-sitter-modelica`
[15] `https://ericlippert.com/2012/06/08/red-green-trees/`
[16] `https://github.com/OpenModelica/ModelicaOML`

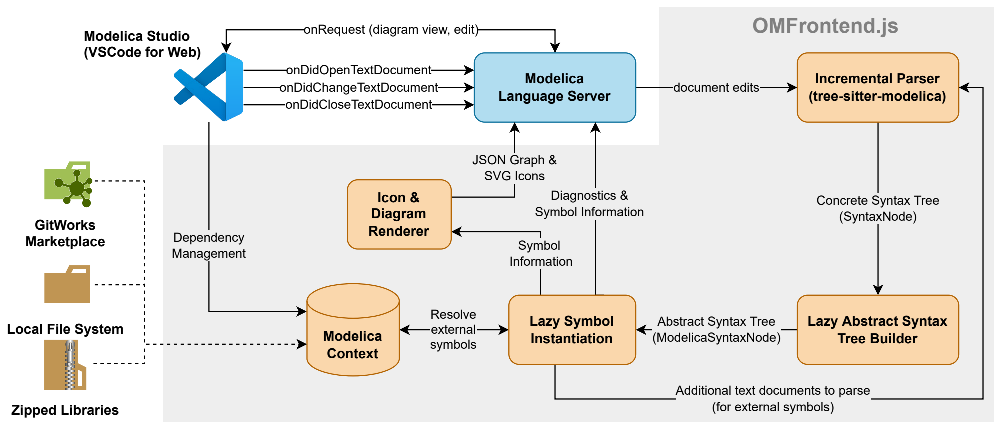

OMFrontend.js
Modelica Studio (VSCode for Web)
onRequest (diagram view, edit)
onDidOpenTextDocument
onDidChangeTextDocument
onDidCloseTextDocument
Modelica Language Server
document edits
Incremental Parser (tree-sitter-modelica)
JSON Graph & SVG Icons
Diagnostics & Symbol Information
Concrete Syntax Tree (SyntaxNode)
Icon & Diagram Renderer
Symbol Information
Dependency Management
GitWorks Marketplace
Local File System
Zipped Libraries
Modelica Context
Resolve external symbols
Lazy Symbol Instantiation
Abstract Syntax Tree (ModelicaSyntaxNode)
Lazy Abstract Syntax Tree Builder
Additional text documents to parse (for external symbols)

Figure 4. Modelica Studio and OMFrontend.js flow diagram

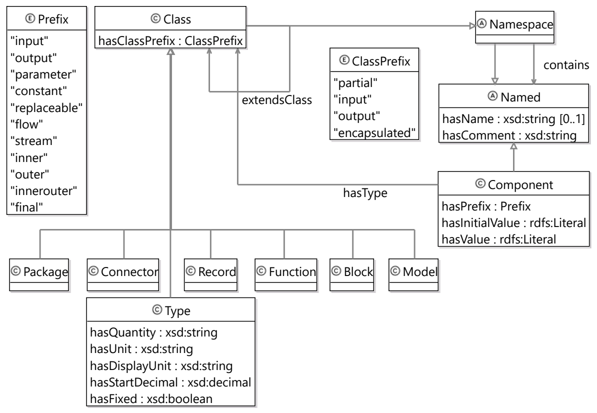


Figure 5. The Modelica Ontology (ModelicaOML Vocabulary)

Listing 1. OML representation of project role definitions

```
ci ProjectManager : project:Role [
    project:hasAssignment DefineProject]
ci SystemEngineer : project:Role [
  project:hasAssignment DefinePACKRequirements]
ci SystemArchitect : project:Role [
  project:hasAssignment DefinePACKArchitecture]
ci DesignEngineer1 : project:Role [
    project:hasAssignment
        DefineSubsystemBehavior]
ci DesignEngineer2 : project:Role [
    project:hasAssignment PerformAnalysis
    project:hasAssignment VerifyPACKRequirements
    project:hasAssignment GenerateReports]
```

Once the Project Manager creates the first commit, the CI/CD pipeline runs the build scripts which interprets the OML code into RDF triples and ingests them into the triple store, effectively integrating the information into the Semantic Twin. The Project Manager can then save the resulting project structure as a template and publish it as an OML artifact in the GitWorks Commons for future reuse and sharing. Once the project repository has been created, the other participants can review the tasks that have been assigned to them, and begin contributing to the project.

The lead Systems Engineer defines the overall systems requirements using a SysML tool. The three requirements for this system are defined as follows:

1. The mass of the PACK shall be no greater than 100kg.
2. The PACK shall produce Conditioned Air with a temperature to ±1°K of 293 °K.
3. The PACK shall produce Conditioned Air with a pressure to ±3kPa of 101 kPa.

The Systems Architect then defines the PACK architecture, also using a SysML tool. For the purposes of this case study, a simplified architecture for the PACK, comprising only the primary heat exchanger and the compressor, is developed. Figure 7 represents this composition of the PACK and the air flows into, out of, and within the system. It is also assumed that the three system requirements apply to the 'Cruise' scenario, during which the temperature and pressure of the input air flows are assumed to be fixed and known.

By specifying the SysML repositories as dependencies within the overall project, the SysML artifacts created by the actors (in this case, requirements and architecture) are translated into OML and populate the Semantic Twin in accordance with the corresponding OML vocabularies. This translation is performed by the dedicated SysMLOML adapter within the GitWorks platform. An example of the resulting OML description (translated from SysML) is presented in Listing 2. In this listing, only a portion of the PACK architecture definition is presented - the flows between the interfaces are also captured by the OML description but are not shown.

Listing 2. OML representation of PACK architecture

```
ci PACK : mission:Component [
    base:contains HeatExchanger
    base:contains Compressor
    PL_Mech:hasMass PACKMass]
ci HeatExchanger : mission:Component [
    mission:presents p1
    mission:presents p2
```

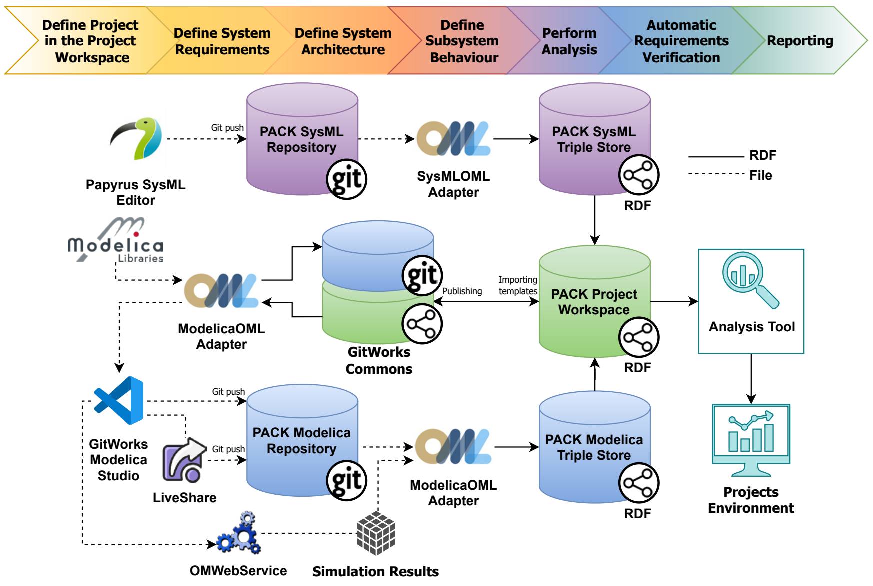


Figure 6. Data flow within the GitWorks platform for the PACK case study.

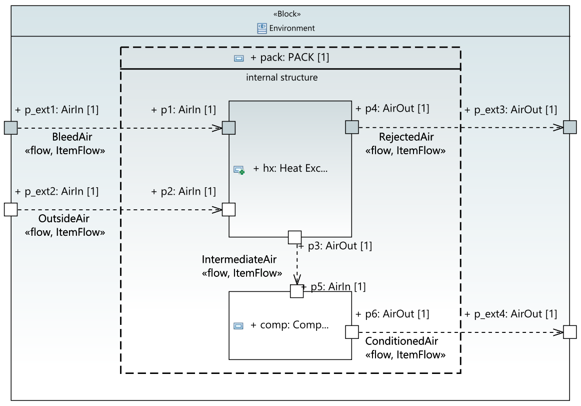


Figure 7. SysML representation of the PACK architecture (hx: Heat Exchanger; comp: Compressor)

```
    mission:presents p3
    mission:presents p4
    PL_Mech:hasMass HXMass]
ci Compressor : mission:Component [
    mission:presents p5
    mission:presents p6
    PL_Mech:hasMass CompMass]
ci PACKMass : PL_Mech:ComponentMass
ci HXMass : PL_Mech:ComponentMass
ci CompMass : PL_Mech:ComponentMass
```

This simple example illustrates how the Semantic Twin can be constructed either directly via the GitWorks Workbench environment (e.g. definition of roles and tasks), or via translation from another artifact (e.g. translation of requirements and architecture from SysML to OML). The Semantic Twin can then be used to efficiently query information about the project and the system (including time-travel queries to investigate evolution of system design) or to automatically generate new modeling artifacts as demonstrated in the next section.

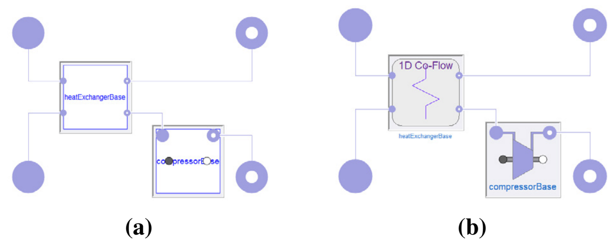


Figure 8. Stages of PACK Modelica model (a) Partial model, (b) Completed model

4.2 Semantic Twin-Powered Authoring

Direct integration of the centralized RDF databases into GitWorks authoring tools allows the users to translate the architectural and requirements knowledge originally expressed as SysML artifacts into corresponding Modelica partial models through the ModelicaOML adapter on the fly. As such, Figure 8a depicts a partial model generated from the OML representation of the PACK architecture.

The generated partial model can be used as a starting point for the Design Engineers to define the behavior of the PACK's components: the heat exchanger and the compressor. Depending on the desired model fidelity, there are multiple ways of implementing the behavior. For example, the total heat transfer rate of the heat exchanger can be obtained from:

$$Q = E_e C_a (T_{bleed} - T_{ram}) \tag{1}$$

where E_e is the effectiveness of the heat exchanger, $C_{a_{min}}$ is the heat capacity of the air stream and T_{bleed} and T_{ram} are the temperatures of bleed and ram air respectively (Poudel 2019). The pressure drop in the heat exchanger can be calculated via the following equation:

$$\Delta P = \frac{f L_{tube} \rho v^2}{2D_h} \tag{2}$$

where f is the friction factor, L_{tube} is the total length of the heat exchanger, ρ is the air density as a function of temperature, v is the mean stream velocity and D_h is the hydraulic diameter (Poudel 2019). Traditionally, Equation 1 and Equation 2 can be implemented manually by engineers in Modelica language. This can be a time-consuming, error-prone and costly process. Alternatively, the engineers can sift through many third-party Modelica repositories available online in the hope of finding the component model they need (Hussain et al. 2022). Existing Modelica tools such as OpenModelica and Dymola (Dempsey 2006) can assist with this task by performing a basic keyword search. However, this comes with a significant limitation of only searching within classes currently loaded into the workspace. Furthermore, such a search is incapable of analyzing the inherent structural and semantic knowledge embedded in the models.

In order to alleviate such modeling bottlenecks and boost model exchange, the GitWorks Commons offers users a convenient interface to query an RDF database of published models and libraries to find required components and blocks. This approach is based on a physics-based simulation ontology currently being developed in OML, and employs the SPARQL query language to offer a set of advanced search techniques such as aggregation, extensible value testing, subqueries, and negation (Hussain et al. 2022; Kollia, Glimm, and Horrocks 2011). For example, a SPARQL query can be designed to find another Modelica component with two compatible connector ports. The connector compatibility is established by ensuring that the connector variables have the same name, prefixes and type. Listing 3 shows an excerpt of such a query which outputs all compatible Modelica models containing two connector ports carrying the thermofluidic variables `m_flow`, `p`, `h_outflow`, `Xi_outflow`, `C_outflow`. We are searching for a model that has two connectors that contain these variables and the names, types and prefixes of the variables match. For example, Modelica.Fluid (Casella, Otter, et al. 2006) and ThermoPower (Casella and Leva 2005) libraries use distinct but compatible connectors which can only be identified manually or through a SPARQL query. As a result, the users are presented with compatible components from all libraries that use the same connector definition. The result of running the query in Listing 3 is given in Listing 4, and shows the three compressor models from the `PL_Lib` library that are matching. One can note that most Modelica tools support `choicesAllMatching` annotation which can help populate the dialogs with a list of matching models - this feature is limited to loaded libraries only and extending it would require tool changes.

Having the Modelica models expressed as individuals using an OML-based vocabulary and searching these using SPARQL queries against the GitWorks Commons populated with all the available libraries on the GitWorks platform is a paradigm-changing feature.

The GitWorks Commons displays a list of models and libraries obtained as a result of the search query and enables the user to inspect the documentation and the code. As highlighted in (Hussain et al. 2022), it is rare that a component model can be reused without any modifications. Therefore, if a suitable component is found, the engineer can import the selected model into the Modelica Studio workspace and invite the rest of the team to use the real-time collaborative functionality of the VSCode Liveshare extension to modify, complete and check the full model definition synchronously. The resulting model definition is shown in Figure 8b. Optionally, component models can also be seamlessly published in the GitWorks Commons with dependencies tracked through Modelica's `uses` annotation.

Development of Modelica models, therefore, is greatly aided by the proposed Semantic Twin technology through automatic generation of model architecture and facilitating model reuse and exchange within the community.

4.3 Simulation-based Requirements Verification and Reporting

Automating the requirement verification allows engineers to accelerate the iterative systems engineering process. In order to enable this capability, the requirements formally captured by the Semantic Twin in Section 4.1 can be expressed alongside a behavioral model developed in Modelica.

Several Modelica libraries for simulation-based requirements verification exist, such as Modelica_Requirements, ReqSysPro discussed in (Bouskela et al. 2022), and vVDR outlined in (Mengist, Buffoni, and Pop 2021). In this case study, the requirements and scenario are captured in a vVDR-style verification scenario model through the ModelicaOML adapter. The resulting model is shown in Figure 9. The model contains the design block containing the PACK system defined in the previous section. It receives the inputs defined in the scenario block and outputs the calculated values of system mass and conditioned air temperature and pressure. These values are then used as inputs in the three requirements blocks to calculate the verification status of the corresponding requirements.

The simulations are performed using the OMWebService (defined in Section 3.1), and the results can be committed to the relevant Git repository at the same time to preserve the traceability of the results. At the same time, the ModelicaOML adapter is invoked to pass the verification status of requirements to the Semantic Twin. Table 1 shows the verification status of each of the requirements defined in Section 4.1.

As a result, all the systems engineering knowledge

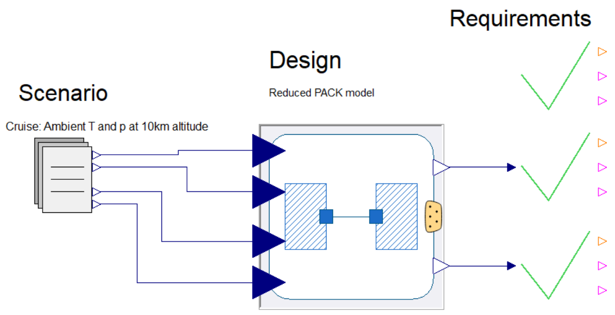


Figure 9. PACK system design model as a part of a verification scenario model.

Table 1. Requirements verification status

	Target	Calculated	Status
Req. 1	100 kg	97 kg	Verified
Req. 2	293 ± 1 K	293.15 K	Verified
Req. 3	101 ± 3 kPA	101.4 kPa	Verified

generated throughout the project is unified and stored in the RDF-based Semantic Twin. This enables automated and intelligent reporting of necessary decision-making information via querying and reasoning across the whole dataset. For example, the Lead Systems Engineer may wish to take a closer look into the project management and requirement verification aspects of the project by constructing a SPARQL query corresponding to the following natural language expression:

- Return all system-level requirements (with their proposed verification test cases) that have not been verified and the person responsible for performing the verification test case.

As another example, the Project Manager may wish to find out more about the model development and trace its metadata information:

- Return the artifact that defines a heat exchanger component which 'presents' a particular interface and the original author of that artifact.

Such queries can be expressed using SPARQL within the Workbench environment, and the outputs can be displayed as tables or graphs. This allows the users to gain insight that would normally be difficult to attain through other means.

The PACK case study presented in this section has demonstrated how the GitWorks platform can be used to effectively integrate Modelica modeling and simulation environments into the systems engineering process to achieve simulation-based system verification from the early stages of the product lifecycle. Artifacts can be automatically generated from other model types (e.g. SysML to Modelica) via the OML adapters. Modelica models can be further defined by searching the GitWorks Commons for relevant and compatible components. Modelica simulation results can be automatically translated into RDF to automatically verify requirements. The resulting integrated Semantic Twin can then be queried. In this way, Modelica models become an invaluable link in the systems engineering chain by providing added value across multiple domains, all while maximizing automation and reducing the effort required.

5 Conclusions and Future Work

We have presented our vision for the GitWorks platform to enable the democratized model-based design and engineering of cyber-physical systems. We have proposed a system architecture for GitWorks and have developed a prototype implementation focused around enabling the use of Modelica in the larger MBSE process. We have conducted a preliminary case study that has demonstrated the use of GitWorks for the federated design and engineering of a passenger air conditioner system.

Plans for future work include further development of OML vocabularies and OML adaptors to increase the number of different modeling paradigms and COTS tools supported by the platform, increase the maturity of the user interface for the web applications, and demonstrate the application to other use cases including satellite systems and composite structures design and fabrication.

Acknowledgments

The work presented here is supported by the Engineering and Physical Sciences Research Council (UK) under the InnovateUK project 97404 and partially supported by the HUBCAP Innovation Action funded by the European Commission's Horizon 2020 Programme under Grant Agreement 872698.

References

Adams, Brian et al. (2020). *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.13 User's Manual.* Tech. rep. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Basnet, Sunil et al. (2022). "A decision-making framework for selecting an MBSE language–A case study to ship pilotage". In: *Expert Systems with Applications*, p. 116451.

Bayer, Todd et al. (2021). "Europa Clipper: MBSE Proving Ground". In: *2021 IEEE Aerospace Conference*. IEEE.

Beckmann, B et al. (2016). "Developing the digital manufacturing commons: a national initiative for US manufacturing innovation". In: *Procedia Manufacturing* 5, pp. 182–194.

Bouskela, Daniel et al. (2022-03). "Formal requirements modeling for cyber-physical systems engineering: an integrated solution based on FORM-L and Modelica". en. In: *Requirements Engineering* 27.1, pp. 1–30. ISSN: 0947-3602, 1432-010X. DOI: 10.1007/s00766-021-00359-z. URL: https://link.springer.com/10.1007/s00766-021-00359-z (visited on 2022-05-22).

Carroll, Jeremy J. et al. (2004). “Jena: Implementing the Semantic Web Recommendations”. In: *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*. WWW Alt. ’04. New York, NY, USA: Association for Computing Machinery, pp. 74–83. ISBN: 1581139128. DOI: 10.1145/1013367.1013381. URL: https://doi.org/10.1145/1013367.1013381.

Casella, Francesco and Alberto Leva (2005). “Object-oriented modelling & simulation of power plants with modelica”. In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, pp. 7597–7602.

Casella, Francesco, Martin Otter, 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.

Dempsey, Mike (2006). “Dymola for multi-engineering modelling and simulation”. In: *2006 IEEE Vehicle Power and Propulsion Conference*. IEEE, pp. 1–6.

Elaasar, Maged et al. (2019). “The case for integrated model centric engineering”. In: *Proceedings of the 10th model-based enterprise summit (MBE 2019). National Institute of Standards and Technology, Gaithersburg, MD*, pp. 9–16.

Elmqvist, Hilding, Martin Malmheden, and Johan Andreasson (2019). “A Web Architecture for Modeling and Simulation”. In: *Proceedings of the 2nd Japanese Modelica Conference, Tokyo, Japan, May 17-18, 2018*. 148. Linköping University Electronic Press, pp. 255–260.

Elmqvist, Hilding, Sven Erik Mattsson, and Martin Otter (1998). “Modelica: The new object-oriented modeling language”. In: *12th European Simulation Multiconference, Manchester, UK*. Vol. 5.

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.

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

Hussain, Mohammad et al. (2022). “Approaches for Simulation Model Reuse in Systems Design—A Review”. In: *SAE Technical Paper* 2022-01-0355. DOI: 10.4271/2022-01-0355.

Isasi, Yago, Ramón Noguerón, and Quirien Wijnands (2015). “Simulation Model Reference Library: A new tool to promote simulation models reusability”. In: *Workshop on Simulation for European Space Programmes (SESP)*. Vol. 24, p. 26.

Jennions, Ian et al. (2020). “Simulation of an aircraft environmental control system”. In: *Applied Thermal Engineering* 172, p. 114925.

Johansson, Olof, Adrian Pop, and Peter Fritzson (2005). “ModelicaDB - A Tool for Searching, Analysing, Crossreferencing and Checking of Modelica Libraries”. In: *Proceedings fo the 4th International Modelica Conference, March 7-8, Hamburg University of Technology, Hamburg-Harburg, Germany, Volume 2 :* Linköping University, The Institute of Technology, pp. 445–454. URL: http://www.modelica.org/events/Conference2005.

Kecskes, Tamas et al. (2019). “Modelica on the Web”. In: *Proceedings of The American Modelica Conference 2018, October 9-10, Somberg Conference Center, Cambridge MA, USA*. 154. Linköping University Electronic Press, pp. 220–226.

Kollia, Ilianna, Birte Glimm, and Ian Horrocks (2011). “SPARQL query answering over OWL ontologies”. In: *Extended Semantic Web Conference*. Springer, pp. 382–396.

Larsen, Peter Gorm et al. (2016). “Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project”. In: *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*. IEEE, pp. 1–6.

Ma, Junda et al. (2022-03). “Systematic Literature Review of MBSE Tool-Chains”. en. In: *Applied Sciences* 12.7, p. 3431. ISSN: 2076-3417. DOI: 10.3390/app12073431. URL: https://www.mdpi.com/2076-3417/12/7/3431 (visited on 2022-05-23).

Madni, Azad and Michael Sievers (2018). “Model-based systems engineering: Motivation, current status, and research opportunities”. In: *Systems Engineering* 21.3, pp. 172–190.

McDermott, Thomas et al. (2020). “Benchmarking the Benefits and Current Maturity of Model-Based Systems Engineering across the Enterprise: Results of the MBSE Maturity Survey”. In: *Technical Report SERC-2020-SR-001*. Systems Engineering Research Center.

Mengist, Alachew, Lena Buffoni, and Adrian Pop (2021). “An Integrated Framework for Traceability and Impact Analysis in Requirements Verification of Cyber–Physical Systems”. In: *Electronics* 10.8, p. 983.

Nachawati, Mohamad Omar and Alexander Brodsky (2021). “Mixed-Integer Constrained Grey-Box Optimization based on Dynamic Surrogate Models and Approximated Interval Analysis”. In: *Proceedings of the 10th International Conference on Operations Research and Enterprise Systems, ICORES 2021, Online Streaming, February 4-6, 2021*. SCITEPRESS, pp. 99–112.

Poudel, Sabin (2019). *Modelling of a Generic Aircraft Environmental Control System in Modelica*.

Rackauckas, Chris et al. (2021). “Composing Modeling and Simulation with Machine Learning in Julia”. In: *Modelica Conferences*, pp. 97–107.

Sirin, Göknur et al. (2015). “A model identity card to support simulation model development process in a collaborative multidisciplinary design environment”. In: *IEEE Systems Journal* 9.4, pp. 1151–1162.

Stirgwolt, Benjamin W, Thomas A Mazzuchi, and Shahram Sarkani (2022). “A model-based systems engineering approach for developing modular system architectures”. In: *Journal of Engineering Design* 33.2, pp. 95–119.

Tennøe, Simen, Geir Halnes, and Gaute T Einevoll (2018). “Uncertainpy: a Python toolbox for uncertainty quantification and sensitivity analysis in computational neuroscience”. In: *Frontiers in neuroinformatics*, p. 49.

Torabzadeh-Tari, Mohsen et al. (2011). “Omweb–virtual web-based remote laboratory for modelica in engineering courses”. In: *Proceedings 8th Modelica Conference, Dresden, Germany*. Vol. 3. Citeseer.

Tudorache, Tania, Jennifer Vendetti, and Natalya Fridman Noy (2008). “Web-Protege: A Lightweight OWL Ontology Editor for the Web.” In: *OWLED*. Vol. 432, p. 2009.

Wagner, David et al. (2020). “CAESAR Model-Based Approach to Harness Design”. In: *2020 IEEE Aerospace Conference*. IEEE, pp. 1–13.

Wan, Li et al. (2013). “A modelica-based modeling, simulation and knowledge sharing web platform”. In: *20th ISPE International Conference on Concurrent Engineering*. IOS Press, pp. 517–525.

A Example SPARQL Query

Listing 3. SPARQL query to find a compatible compressor model

```
PREFIX m:   <http://openmodelica.org/
    openmodelica/modelica#>
PREFIX msl:   <http://examples/
    ModelicaStandardLibrary#>

SELECT DISTINCT ?m ?comp1 ?comp2

WHERE {
  ?m a m:Model .
  ?m m:contains ?comp1 .
  ?m m:contains ?comp2 .

  ?comp1 a m:Component ;
           m:hasType ?con1 .
  ?comp2 a m:Component ;
           m:hasType ?con2 .
  ?con1 a m:Connector . # FlangeA
  ?con1 m:contains [
    a m:Component ;
    m:hasName "m_flow" ;
    m:hasType msl:ThermoPower.Gas.Flange.Medium.
        MassFlowRate
  ] .
  ?con1 m:contains [
    a m:Component ;
    m:hasName "p" ;
    m:hasType msl:ThermoPower.Gas.Flange.Medium.
        AbsolutePressure
  ] .
  ?con1 m:contains [
    a m:Component ;
    m:hasName "h_outflow" ;
    m:hasPrefix "stream"
    m:hasType msl:ThermoPower.Gas.Flange.Medium.
        SpecificEnthalpy
  ] .
  ?con1 m:contains [
    a m:Component ;
    m:hasName "Xi_outflow" ;
    m:hasPrefix "stream"
    m:hasType msl:ThermoPower.Gas.Flange.Medium.
        MassFraction
  ] .
  ?con1 m:contains [
    a m:Component ;
    m:hasName "C_outflow" ;
    m:hasPrefix "stream"
    m:hasType msl:ThermoPower.Gas.Flange.Medium.
        ExtraProperty
  ] .
  ?con2 a m:Connector . # FlangeB
    ...
}
```

Listing 4. The result of running the SPARQL query above

```
{ "head": {
    "vars": [ "m" , "comp1" , "comp2" ] } ,
  "results": {
    "bindings": [
      {
        "m": { "type": "uri" , "value": "http://
            examples/PL_Lib#PL_Lib.Interfaces.
            CompressorBase"},
        "comp1": { "type": "uri" , "value": "
            http://examples/PL_Lib#PL_Lib.
            Interfaces.CompressorBase.inlet"},
        "comp2": { "type": "uri" , "value": "
            http://examples/PL_Lib#PL_Lib.
            Interfaces.CompressorBase.outlet"}},
      {
        "m": { "type": "uri" , "value": "http://
            examples/PL_Lib#PL_Lib.Components.
            Compressor_noMaps"},
        "comp1": { "type": "uri" , "value": "
            http://examples/PL_Lib#PL_Lib.
            Components.Compressor_noMaps.inlet"
            },
        "comp2": { "type": "uri" , "value": "
            http://examples/PL_Lib#PL_Lib.
            Components.Compressor_noMaps.outlet"
            }},
      {
        "m": { "type": "uri" , "value": "http://
            examples/PL_Lib#PL_Lib.Components.
            Compressor_noMaps_mass"},
        "comp1": { "type": "uri" , "value": "
            http://examples/PL_Lib#PL_Lib.
            Components.Compressor_noMaps_mass.
            inlet"},
        "comp2": { "type": "uri" , "value": "
            http://examples/PL_Lib#PL_Lib.
            Components.Compressor_noMaps_mass.
            outlet"}},
    ]
  }
}
```

Enhancing SSP creation using sspgen

Lars Ivar Hatledal[1]   Eirik Fagerhaug[1]

[1]Department of ICT and Natural Sciences, Norwegian University of Science and Technology, Norway
{laht}@ntnu.no

Abstract

The System Structure and Parameterization (SSP) standard is a tool independent standard to define complete systems consisting of one or more components, including its parameterization, that can be transferred between simulation tools. Thus the SSP standard is a natural extension to the Functional Mock-up Interface (FMI) standard, allowing systems of components, rather than just individual components, to be simulated in a growing number of supported tools. This paper introduces sspgen, a textual Domain Specific Language (DSL) for generating SSP archives. The aim of the DSL is to greatly simplify the creation of SSP compatible simulation systems. sspgen is written in the Kotlin programming language, which provide syntax highlighting and static code analysis in selected tools, full access to Java compatible libraries, and more importantly a scripting context so that sspgen definitions can be easily shared and executed on demand. As the DSL is based on a generic programming language, it enables complex expressions to be evaluated for the purpose of e.g., pre-simulation and initialization of variables. The DSL also performs validation and through integration with the Open Simulation Standard - Interface Specification (OSP-IS) even allows more complex connections to be formed than the single scalar connections that the SSP standard defines, while still retaining compliance. Furthermore, the DSL handles automatic packaging of its referenced content into a ready-to-use SSP archive. As a whole, the introduced package makes it easier to create, modify and share SSP systems.

Keywords: Co-simulation, Domain Specific Language, Functional Mock-up Interface, System Structure and Parameterization

1 Introduction

The System Structure and Parameterization (SSP) standard (Köhler et al. 2016) released in 2019, is a tool-independent format for the description, packaging and exchange of system structures and their parameterization. The SSP is comprised of a set of XML-based formats to describe a network of component models with their signal flow and parametrization, as well as a ZIP-based packaging format for efficient distribution of entire systems, including any referenced models and other resources. The SSP contributes to maximizing re-usability of models and parameters across tools and use cases. An

Table 1. Tools supporting SSP import and simulation (v1.0).

Name	Vendor	License
SYNECT Model Management	dSPACE	commercial
OMSimulator	OpenModelica	free
Model.CONECT	AVL	commercial
FMI Bench	PMFS	commercial
easySSP	eXXelent solutions	free + commercial
Simcenter System Architect	Siemens	commercial
Simcenter Studio	Siemens	commercial
Dymola	Dassault Systèmes	commercial
libcosim	OSP	free
Vico	NTNU	free
Ecos	NTNU	free

SSP (file extension .ssp) is a zip archive containing one or more XML documents, at least one named *SystemStructure.ssd*, declaring the structure of the simulation. The archive also contains any referenced components, like Functional Mock-up Units (FMUs) and other resources. Thanks to the SSP, and provided that the SSP does not contain non-optional implementation specific annotations, a simulation system can be defined once and later simulated in multiple tools. Currently, the SSP is supported by a number of free and commercial tools. See Table 1 for an overview of tools that support SSP import and subsequent execution. In (Lars I Hatledal et al. 2021), the authors made use of SSP to describe a simulation system that were simulated in a number of compatible open-source importers. More specifically Vico, OMSimulator, libcosim, FMPy, and FMI Go!. The latter two required a slight modification to the SSP description file as they did not support the final publicly released version of the standard (1.0). Thus, they are not present in the provided table of supported tools. In this example, the SSP proved its usefulness by allowing the same definition of a system to be tested and benchmarked in several tools.

As previously mentioned, the SSP is a collection of XML file(s) declaring the content, connections and parameterization of a simulation as well as any resources, like models, required to run the simulation. Given a set of components, e.g., FMUs, the simulation structure can be formalized in an XML file and zipped together with any resources required. This can be done using nothing else than an text editor and built-in OS capabilities for zipping a folder. However, this approach is tedious, time-consuming and error prone. The resulting SSP archive may include formal and/or logical errors that will not surface until it is loaded into a simulation tool. Depending on the tool used, the source of any errors reported may be non-obvious. Furthermore, XML is a static text-format, which means that parameters must be provided exactly. I.e., a number like $PI/2$ must be pre-computed and manually typed as e.g., 1.57079632679, which is tedious and may lead to accuracy issues depending on the number of decimal points included. Another issue arises with systems that contain similar components, which expects similar or identical connectors and parameterization options. Declaring such a system in XML leads to excessive copy-pasting and performing changes is error prone as the same logical variable needs to be kept track of during modification throughout the document. Another concern is bit-rot, which occurs when some file is left unused and unmaintained, possibly due to poor understanding of the content. Manually generated SSP archives are prime subjects of bit-rot as maintaining them requires substantial knowledge, which typically dwindles over time and might disappear once the archive is transferred to some other recipient.

A domain-specific language (DSL) provide a notation tailored toward some application-domain, and is based on the relevant concepts and features of that domain (Van Deursen and Klint 2002). This paper introduces sspgen, a DSL that aims to ease the creation of SSP archives by providing an accessible and easy to use language construct that is more powerful than manually editing XML or using graphical tools. The solution benefits from existing tooling and provides integration with other standards and tools in order to enhance SSP development.

The rest of the paper is organized as follows; first some related work is given, followed by a presentation of the sspgen software. Finally, a conclusion and notes on future works is given.

2 Related works

Manually creating SSP configurations using basic and readily-available tools, as previously mentioned, is not the only alternative available today.

OMSimulator (Ochel et al. 2019) is a co-simulation framework that allow both import and export of SSP archives. Moreover, it features a simplified Python interface to the underlying C/C++ library for accessibility that can be used to import, define and export SSP configurations. *easySSP* is a graphical, web-based, tool for generating SSP archives. Like graphical tools in general, it favours easy-of-use and accessibility over flexibility. However, large simulations quickly becomes cluttered and editing in a graphical tool has some of the same challenges as editing XML directly.

Additionally, tools using alternative system formats than SSP exists. *Daccosim NG* (Évora Gómez et al. 2019) is a co-simulation framework that features a desktop graphical users interface (GUI) for establishing a simulation graph. The graph may be exported in a custom format only supported by Daccosim, or as an FMU that itself contains other FMUs. The latter allows the system to be loaded into any FMI based simulation tool. While accessible, this solution adds a dependency to an additional solver and the generated FMU is inflexible as it does not allow modifications without re-running the original pipeline. *kopl* is a graphical tool for generating systems compatible with the Open Simulation Platform - System Structure (OSP-SS), which is a format similar to the SSP. By supporting the OSP-IS, connection points are fewer, thus making the system as a whole easier to reason with. The downside is that the format produced is currently only compatible with *libcosim* and eventual tools built on-top of it.

Unlike the graphical tools mentioned, sspgen offers a executable, text-based solution that is more flexible, performs validation, is less verbose than XML, allows arbitrary expressions to be computed as input to the document, and as a novelty, combines the OSP-IS standard with SSP. The DSL defines much of the same concepts and structure as is found in the standard, making the learning curve less steep for users already familiar with the SSP standard. As the solution is text-based, modifications can be easily shared and version-controlled. The solution is further elaborated in the following section.

3 sspgen

This section introduces sspgen, a Kotlin DSL for generating SSP archives that is publicly available as a Maven artifact. Kotlin is a statically typed programming language that runs on the Java Virtual Machine (JVM) that is interoperable and comparable with the more known Java language, but offers additional features and a offer a less verbose syntax. Any and all libraries available for Java are usable by Kotlin and visa versa. Today, Kotlin is mostly known as the main programming language for Android applications, however it is also used as a replacement for JavaScript in web-applications and Java for desktop applications. Thanks to its intuitive type-system and smart use of closures, Kotlin is a very suitable and powerful language for building an embedded DSL. That is, a DSL that is implemented within a host language. While embedded DSLs in general are less flexible than external DSLs, which use an independent interpreter or compiler, embedded DSLs typically benefit from existing tooling. Kotlin for instance, supports a scripting context that allow Kotlin code to be executed without the need for a build-system. While executing a script, any third party dependencies are automatically resolved and the code is compiled on-the-fly. The stand-alone Kotlin compiler that makes this possible is bundled with the IntelliJ integrated development environment (IDE), but it can also be downloaded directly from the official Kotlin repository on GitHub. Using IntelliJ, however, is encouraged as it adds auto-completion, static-code-analysis, syntax highlighting and enables the script to be executed through a GUI as opposed to the command line.

Listing 1. Kotlin script skeleton demonstrating basic usage of sspgen.

```
@file:DependsOn("info.laht.sspgen:dsl:0.5.2")

import no.ntnu.ihb.sspgen.dsl.*

ssp("...") {
 resources {
  file("path/to/FMU.fmu")
 }
 ssd("...") {
  system("...") {
   elements {
    component("FMU", "resources/FMU.fmu") {
     connectors {
      real("output", output) {
       unit("m/s")
      }
      real("input", input)
      integer("counter", output)
     }
    }
   }
   connections {}
  }
  defaultExperiment(startTime = 1.0)
 }
}.build()
```

As mentioned, sspgen is powered by Kotlin, and makes use of closures in such a way that it acts like a *DSL*. Thus offering a DSL context within a generic programming language. While the package is compatible with Java, it can only be intuitively used in the context of Kotlin. As shown in Listing 1, the idea is that users should create a generic Kotlin script, which then adds sspgen as a dependency. The script context allows the code to be easily modified, executed, shared and version-controlled. The actual SSP archive required for simulation is generated on demand by running the script, hopefully reducing the likelihood of bit-rot as maintenance becomes easier. As sspgen runs in a scripting context, generic expressions can be evaluated, which is immensely powerful. For one, loops can be used to declare multiple similar connectors as shown in Listing 2. Furthermore, scripts can make use of any third party library compatible with Java in order to compute e.g., parameterization options. Component references are included either a through a file handle, an URL or as the path to a PythonFMU script. Using the URL option, the script definitions can be shared as a single executable file and easily version-controlled. Another benefit from using URLs is that the referenced components can be updated automatically, as re-running the script can be configured to download the latest version. If this behaviour is not desired, one could naturally point the URL to a fixed version. E.g., if the artifact version-controlled using Git, one could point the URL to a fixed tag rather than an evolving branch.

Listing 2. Using loops to declare similarly named connectors.

```
connectors {
 for (i in 0..10) {
  real("transform[i].position.x", input)
  real("transform[i].position.y", input)
  real("transform[i].position.z", input)
 }
}
```

Listing 3. Declaring connections using sspgen.

```
// SSP type connections
connections {
    "wheel.p1.f" to "chassis.p.f"
    "chassis.p.e" to "wheel.p1.e"
    "ground.p.f" to "wheel.p.f"
    "wheel.p.e" to "ground.p.e"
}

// OSP-IS type connections
ospConnections {
    "chassis.linear mechanical port" to
        "wheel.chassis port"
    "wheel.ground port" to
        "ground.linear mechanical port"
}
```

Listing 4. Declaring annotations using sspgen.

```
val stepSize = 1.0/100
...
defaultExperiment {
  annotations {
    annotation("org.openmodelica") {
      """
        <oms:SimulationInformation resultFile=
            "results.mat"/>
      """
    }
    annotation("com.opensimulationplatform") {
      """
        <osp:Algorithm>
          <osp:FixedStepAlgorithm baseStepSize=
            "$stepSize"/>
        </osp:Algorithm>
      """
    }
  }
}

namespaces {
  namespace("oms",
    "http://openmodelica.org/oms")
  namespace("osp",
    "http://opensimulationplatform.com/SSP/
        OSPAnnotations")
}
```

3.1 Connections

The *connections* closure in Listing 3 shows how regular connections between components are made. By default, outputs are declared on the left hand side of the infix function *to*. It is also possible to swap the ordering for all connections by specifying a boolean flag. Declaring optional linear transformations on the signals formed are achieved by invoking an instance method on the object returned by the individual connection objects.

3.2 Annotations

Several SSP importers, like OMSimulator and libcosim, require tool-specific annotations to be present in the imported XML. This requires users that want to support multiple tools, and use some tool for creating the SSP, to edit the generated XML manually. sspgen allows annotations to be added as plain-text, allowing multiple tools to be supported without further editing. Listing 4 shows how annotations are declared using sspgen.

3.3 Validation

sspgen performs several types of validation of the declared content. Firstly, it checks that the connectors refers to actual variables and that the declared type matches. Secondly it checks that the connections are valid and that a connector has been declared for a given variable. FMI4j (Lars Ivar Hatledal, Zhang, et al. 2018), a JVM library for importing FMUs, is used to validate FMU components based on their *modelDescription.xml*. sspgen is also able to recognise proxy-fmu [1] components (a solution for remote execution of FMUs developed under the umbrella of the Open Simulation Platform), so that their *modelDescription.xml* files can be validated in the same way as regular FMUs. Furthermore, sspgen can perform additional checks as part of the integration with OSP-IS and/or FMI-VDM-Model as explained in more detail below. Currently, FMI version 1.0 and 2.0 for co-simulation is supported. When working with non-compliant models or unsupported FMI versions, it is possible to turn of validation.

3.4 Integration with OSP-IS

The OSP interface specification (OSP-IS) is an addition to the FMI 2.0 standard for co-simulation which provides a method for adding semantic meaning to model interface variables. OSP-IS aims to simplify the model connection process, and enables validation of semantically correct simulations (Open Simulation Platform 2020b). In short, an XML document adhering to the OSP-IS, which declares more complex input and output variable relationships, can be used by tools that support it to form more complex and semantically correct connections between models. Currently, the only tool that natively supports the OSP-IS is libcosim (Open Simulation Platform 2020a) from the OSP foundation. sspgen allows OSP-IS connections to be formed within the DSL, which are later transpiled to single scalar connections that the SSP supports, while retaining the static type checking during the build process. Thus, sspgen enables the OSP-IS to be used by any SSP compatible tool. For example the *ospConnections* shown in Listing 3 are transpiled to the SSP compatible XML as shown in Listing 5

[1] `https://github.com/open-simulation-platform/proxy-fmu`

Listing 5. OSP-IS connections transpiled to SSP.

```
<ssd:Connections>
   <ssd:Connection startElement="wheel"
       startConnector="p1.f" endElement=
       "chassis" endConnector="p.f"/>
   <ssd:Connection startElement="chassis"
       startConnector="p.e" endElement=
       "wheel" endConnector="p1.e"/>
   <ssd:Connection startElement="ground"
       startConnector="p.f" endElement=
       "wheel" endConnector="p.f"/>
   <ssd:Connection startElement="wheel"
       startConnector="p.e" endElement=
       "ground" endConnector="p.e"/>
</ssd:Connections>
```

3.5 Integration with FMI-VDM-Model

sspgen optionally integrates with the FMI-VDM-Model (Battle et al. 2019) tool created by the INTO-CPS project. The FMI-VDM-Model is a formal model of the FMI standard using the VDM Specification Language. The integration allows optional static analysis of the included FMUs for informative purposes. To use, simply provide the path to the FMI-VDM-Model executable when invoking the sspgen functions *validate* or *build*.

3.6 Integration with PythonFMU

PythonFMU (Lars Ivar Hatledal, Collonval, and Zhang 2020) is a Python framework for developing FMUs using the Python programming language. sspgen allows components written using PythonFMU to be declared in its source form. sspgen then calls the PythonFMU packaging tool, which must be pre-installed on the system, during the build process. This makes it easier to prototype SSP systems by shortening the development loop. This command is featured in the DSL, but thanks to the underlying scripting context, it is possible for users to write generic code that produced components from other sources on demand.

4 Conclusion and future work

This paper presents sspgen, a high-level DSL aimed at easing and enhancing the creation of SSP archives. More specifically, the DSL enables evaluation of complex expressions, is less verbose than XML and introduces concepts that makes authoring easier, such as the ability to copy data between components. Furthermore, the DSL is written in Kotlin, a statically typed language that offers auto-completion and syntax highlighting. Moreover, Kotlin features a standard library that further expand the already well-established standard library provided by Java. Additionally, a vast eco-system of third party libraries are readily-available. In the context of sspgen, such libraries can be used to e.g., compute parameterization options for components prior to export. More importantly, sspgen performs validation of its content so that the user can address potential issues before actually loading the SSP into a simulation tool. Furthermore, the DSL supports the OSP-IS standard, allowing more complex connections to be formed, which can be further validated for semantic correctness. All while retaining compliance with the SSP standard. The tool is not feature complete according to the standard, but covers the necessary features in order to be effective by the current users and has seen wide usage internally by researchers and master students at NTNU campus Aalesund for the purpose of modelling maritime systems. The tool is largely stable and future work includes maintenance, documentation and responding to user requests. Additionally, support for FMI 3.0 and further versions of the SSP standard will be considered. sspgen is open-source and available from `https://github.com/Ecos-platform/sspgen` under a permissive license.

Acknowledgements

This work was supported in part by the Project "SFI Offshore Mechatronics", under Grant 237896 from Research Council of Norway.

References

Battle, Nick et al. (2019). "Towards a Static Check of FMUs in VDM-SL". In: *International Symposium on Formal Methods*. Springer, pp. 272–288.

Évora Gómez, José et al. (2019). "Daccosim NG: co-simulation made simpler and faster". In: *Linköping electronic conference proceedings*.

Hatledal, Lars I et al. (2021). "Vico: An entity-component-system based co-simulation framework". In: *Simulation Modelling Practice and Theory* 108, p. 102243.

Hatledal, Lars Ivar, Frédéric Collonval, and Houxiang Zhang (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.

Hatledal, Lars Ivar, Houxiang Zhang, et al. (2018). "Fmi4j: A software package for working with functional mock-up units on the java virtual machine". In: *The 59th Conference on Simulation and Modelling (SIMS 59)*. Linköping University Electronic Press.

Köhler, Jochen et al. (2016). "Modelica-association-project "system structure and parameterization"–early insights". In: *The First Japanese Modelica Conferences, May 23-24, Tokyo, Japan*. 124. Linköping University Electronic Press, pp. 35–42.

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.

Open Simulation Platform (2020a). *libcosim*. (Date accessed 11-May-2022). URL: https://github.com/open-simulation-platform/libcosim.

Open Simulation Platform (2020b). *OSP-IS*. (Date accessed 11-May-2022). URL: https://opensimulationplatform.com/specification/.

Van Deursen, Arie and Paul Klint (2002). "Domain-specific language design requires feature descriptions". In: *Journal of computing and information technology* 10.1, pp. 1–17.

Development of a Modelica Model for the Texas A&M Smart and Connected Homes Testbed

Thomas Firsich[1] Zhiyao Yang[1] Zheng O'Neill[1]

[1]Department of Mechanical Engineering, Texas A&M University, United States, {thomasfirsich17, z.yang, zoneill}@tamu.edu

Abstract

The Texas A&M Smart and Connected Homes Testbed was developed to enable testing and fair comparison of different smart home technologies and grid-interactive capabilities. Having an accurate building model allows researchers to design and optimize smart home systems before implementing them into the experimental testbed. The Modelica Buildings library gives researchers the flexibility to prototype buildings and energy systems to apply to their research projects. This study develops a Modelica model using the Buildings library for the building envelope of the testbed homes using data from the home manufacturer. The A/C system model is developed from the device's rated performance data. Each model goes through independent testing before implementation in the complete building model. In the future, real data from the lab homes will be used to tune these models to ensure accurate performance before a final model is made for testing.

Keywords: Buildings library, testbed modeling, model tuning

1 Introduction

As Modelica grows in popularity among the research community, libraries are built and maintained by labs to further develop the software. Researchers at the Lawrence Berkeley National Laboratory (LBNL) have created an open-source library, the Modelica Buildings library, which enables researchers in the building science community to design and operate buildings and district energy and control systems (Wetter at al. 2014). This library contains dynamic simulation models for HVAC systems, energy storage, multi-zone airflow, and so much more. The contributions to this library come from many researchers among different institutions that aid in developing these tools for building research.

Buildings research is a growing field as the electrification of the grid and modern technology advancements enable the use of new smart home technologies. Whether it is smart thermostats, high-efficiency equipment, or even advanced building controls that are being researched, they all require a dependable model of the physical building to develop these complex fields. With the help of Modelica and the Modelica Buildings library, a flexible building model can be constructed while saving time in simulation-based projects.

The Buildings Energy and HVAC&R research group at Texas A&M University has a smart home testbed that will allow for the testing of smart home technologies, smart grid applications, and other residential building research topics. This paper will briefly go over the testbed before walking through the modeling using Modelica and the Modelica Buildings library. The paper will finish with individual model inspection. The future work of this project is to tune and verify the complete Modelica model performance using data taken at the testbed during operation.

2 Texas A&M Smart and Connected Homes Testbed

The Texas A&M Smart and Connected Homes Testbed (TAM-SCHT) is located at the RELLIS campus in Bryan, Texas, United States. This testbed is used by the Building Energy and HVAC&R research group at Texas A&M University for smart home technology and smart grid applications research (Firsich et al. 2022). It consists of two identical mobile homes of 1,200 ft^2 (111 m^2), each having 3 bedrooms, 2 bathrooms and an open living room and kitchen layout. A picture of the testbed can be seen in Figure 1.

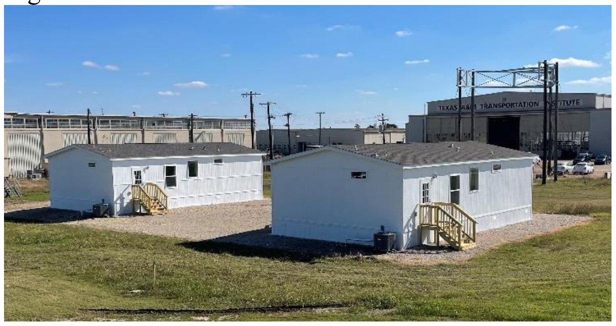


Figure 1. The Texas A&M Smart and Connected Homes Testbed (TAM-SCHT)

This testbed is equipped with a wide-range of sensors to collect building envelope measurements, indoor air-quality, HVAC operation, as well as detailed power consumption. The homes contain removable walls and replaceable windows to provide potential reconfigurable floorplan and advanced building envelope testing. High efficiency HVAC equipment can be installed for testing and the supply ducts can be switched from overhead vents to floor vents. The testbed is also capable of smart grid research with the addition of a PV panel solar farm and on-site battery storage. The PV solar farm location, system configuration, and budget has been established already. Smart appliances will also allow for demand control strategies with the ability to program the operation and scheduling of these loads. The next section will go through the Modelica modeling process for TAM-SCHT.

3 Modelica Models

The Modelica Buildings library is used to construct the model of TAM-SCHT. A complete model of the testbed includes a building model made up of three separate thermal zones: a crawlspace, an attic space, and the living floor, as well as an HVAC model. This building model will be used to test new technologies or control strategies before experimental testing within the testbed. A complete Modelica model, with only a cooling HVAC system implemented, can be seen in Figure 1. The following sections will go over each element of the Modelica model.

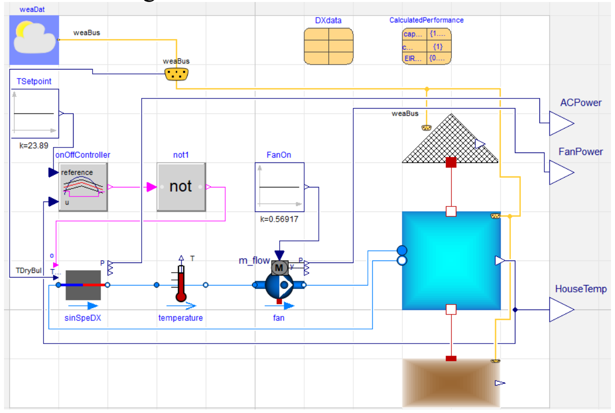


Figure 1. The complete Modelica model of TAM-SCHT, cooling only.

3.1 Building Envelope

The physical building of TAM-SCHT has been constructed using the detailed thermal zone models in the Buildings library. The thermal zones are assumed to be completely mixed air and have been configured with construction details from the building manufacturer. A mixed air thermal zone model has been created for the crawlspace foundation, the actual living zone of the building, and the attic space. These three thermal zones are then connected to form the entire building.

Each thermal zone model is constructed with material data from the building manufacturer to get accurate heat transfer into the building. Starting with the crawlspace model in Figure 2, the testbed is built on top of 6 inches (15.24 cm) of caliche white rock but will be modeled as cement for the time being. Each wall of the crawlspace is 0.375 inch (0.92 cm) of plywood at 28 inches (0.71 m) tall. The ceiling of the crawlspace is the floor of the living area zone and is defined as a surface boundary. The heat transfer for this layer is calculated in the living zone model and input to the crawlspace through the heat transfer port along with the temperature. Heat transfer from the ground into the slab layer of the thermal zone is modeled with a fixed temperature thermal conductance. This data uses the average ground temperature in College Station, Texas with a thermal conductance value found from (GreenCast, Syngenta). Because there is nothing inside the crawlspace, there is zero heat input into the internal gains of the thermal zone.

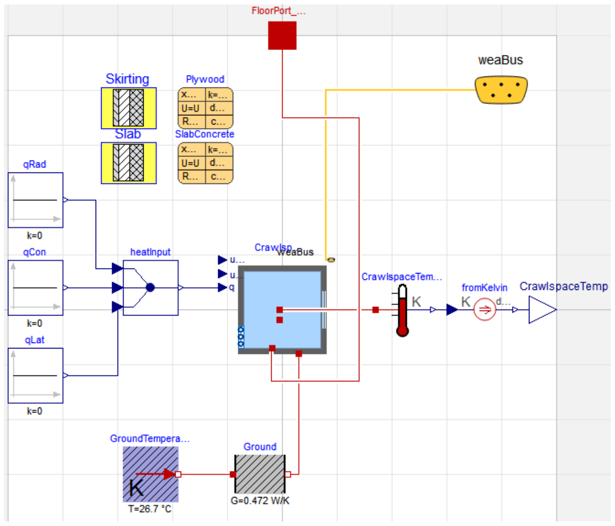


Figure 2. Crawlspace thermal zone model.

The building envelope becomes more involved for the part of the house designed for occupants. Exterior finished layers, weather proofing sheets, insulation, and drywall layers can be found in a typical building wall, from the outside inward. These layers ensure that occupant comfort is maintained throughout the year. For the thermal zone of the living space, the wall construction begins on the outside with a synthetic stucco, then a sheathing layer, next fiberglass insulation between the structural wood framing, and finally 0.5-inch (1.27 cm) gypsum board interior, with a ceiling height of 8.5 ft (2.59 m). Thermal properties of each layer are required and some come from the IECC code compliant house for climate zone 2A (Building Energy Codes Program). The ceiling is made up of cellulose insulation and the same gypsum board for the interior of the zone. The flooring has fiberglass insulation which is beneath the linoleum flooring. Because the floor and ceiling are shared constructions with the crawlspace and attic thermal zones, they are both defined as construction boundaries. The heat conduction through the constructions is modeled in this thermal zone model and then connected to the other thermal zones for the convection, infrared, and solar radiation exchange with the room using the surface ports. The thermal properties of the windows come from the building manufacturer and are defined based on their window area for each exterior wall construction. There is 4 ft^2 (0.37 m^2) of window area on the north wall, 60 ft^2 (5.6 m^2) on the east side, 15 ft^2 (1.4 m^2) on the south side, and 31 ft^2 (2.9 m^2) on the west side of the thermal zone construction. The windows of the testbed currently do not have shading or blinds.

Another crucial element to the thermal zone model of the living space is the internal gains. The lab homes were designed for research to be conducted with simulated internal gains that a typical single-family house would experience. The method used to calculate these internal gains of the house come from the Building America Housing Simulation Protocols (Wilson, E. et al. 2014) and the accompanying spreadsheet tool developed by the National Renewable Energy Laboratory and other Building America partners. With this, the internal gains that we would expect to see in a house of this size are calculated with respect to occupants, appliances, and a typical schedule for all. A weekend hourly profile of these loads can be seen in Table 1.

Table 1. Weekday hourly profile of calculated internal loads.

	Living Room		*Bedroom (x3)*	
Hour	*Sensible (Wh)*	*Latent (Wh)*	*Sensible (Wh)*	*Latent (Wh)*
1	386	20.2	195	6.2
2	362	17.2	184	5.9
3	345	11.5	179	5.7
4	339	11.1	179	5.7

These results can be input into the mixed air thermal zone model as the heat input into the zone, split into radiant, convective, and latent heat transfer. The living space thermal zone also contains supply and return fluid ports for the HVAC to be implemented later. The complete living space thermal zone can be found in Figure 3.

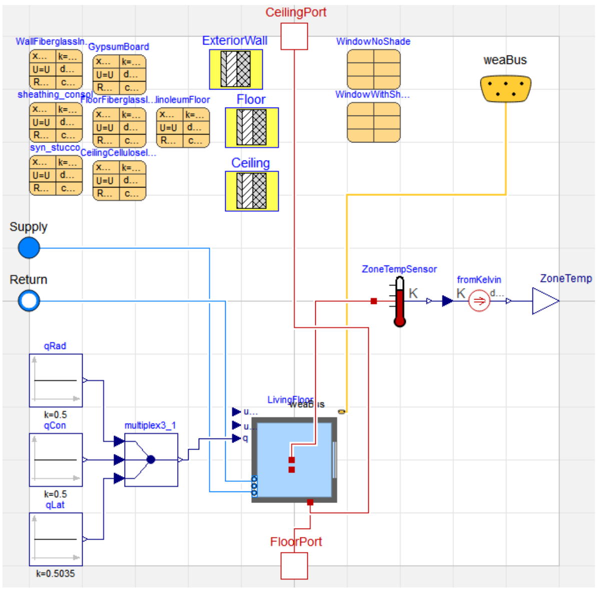


Figure 3. Living space thermal zone model.

The thermal zone model of the attic varies from the other models due to the geometry of the double-pitched roof. Just like the crawlspace model, the floor of the attic is defined as a surface boundary and connected to the heat port of the living space thermal zone. The walls have the same construction but without the fiberglass insulation and gypsum board layers. The double-pitched roof is made up of 0.438-inch (1.11 cm) OSB roof decking with asphalt shingles. The thermal properties of these layers are found in (property ref.). The angle of the roof is 11° (0.192 rad) and the tilt of the roof construction needs to be specified following the *Buildings.Types.Tilt* method. In Figure 4, we can see the orientation specifications that is used in the Buildings library. A ceiling has a tilt of 0 radians due to the solar irradiation being on the other side of the surface that faces the sky, viewing as an occupant in the room. A floor has a tilt of π radians and a wall has a tilt of π/2 radians with respect to the occupants in the building. To get the 11° angle for both roof surfaces, the left roof, letter A in the figure, will have a tilt of 0.192 radians and azimuth facing west, and the right roof, letter B, will also have an angle of 0.192 radians but with an azimuth facing east, matching the building orientation.

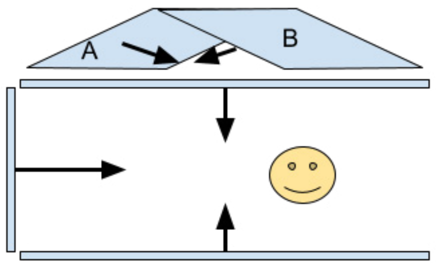


Figure 4. Modelica tilt orientations relative to occupants.

The attic space has no internal gains or heat input into the thermal zone model, just as the crawlspace. Putting all these elements together, the attic thermal zone can be seen in Figure 5.

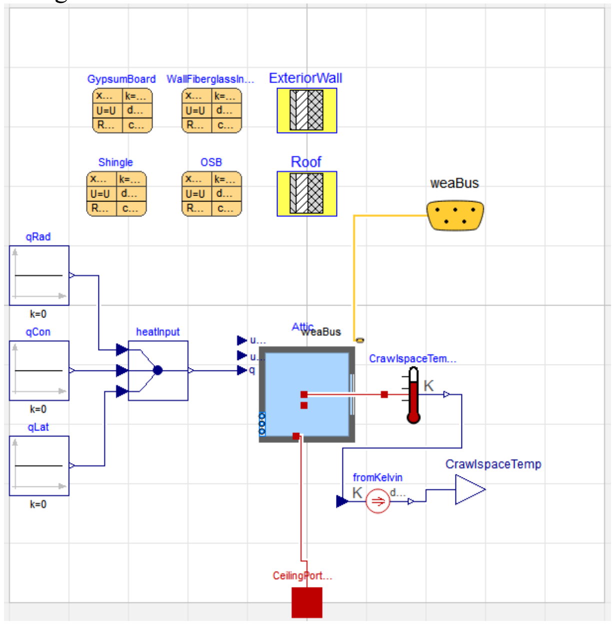


Figure 5. Attic thermal zone model.

3.2 HVAC System

TAM-SCHT is equipped with a split system air conditioning package, serving a single zone, single thermostat building. The outdoor unit is a 2.5-ton air conditioner and the indoor fan coil unit is rated at 3 tons and has a 10-kW electric heating package. The houses come with a thermostat that operates using dual setpoint control.

To model the air conditioning system, a single speed direct expansion, DX, cooling coil, model is used from the Buildings library. This model uses Equation 1 and Equation 2 to calculate the cooling capacity and energy input ration (EIR) of the cooling coil modeled for the building (Wetter et al. 2014).

$$\dot{Q}(\theta_{e,in}, \theta_{c,in}, ff) = cap_{\theta}(\theta_{e,in}, \theta_{c,in}) cap_{FF}(ff) \dot{Q}_{nom} \quad (1)$$

$$EIR(\theta_{e,in}, \theta_{c,in}, ff) = EIR_{\theta}(\theta_{e,in}, \theta_{c,in}) EIR_{FF}(ff) / COP_{nom} \quad (2)$$

$$ff = \dot{m}/\dot{m}_{nom} \quad (3)$$

where $\theta_{e,in}$ is the evaporator inlet wet bulb temperature and $\theta_{c,in}$ is the condenser inlet temperature. The normalized mass flowrate, or flow fraction, is defined in Equation 3, where $\dot{m}$ is the mass flow rate at the evaporator and $\dot{m}_{nom}$ is the nominal mass flow rate. $cap_{\theta}(\theta_{e,in}, \theta_{c,in})$ is cooling capacity modifier as a function of inlet temperatures, Equation 4, and $cap_{FF}(ff)$ is cooling capacity modifier as a function of normalized mass flowrate at the evaporator, Equation 5. Similar to the cooling capacity, $EIR_{\theta}(\theta_{e,in}, \theta_{c,in})$ is EIR modifier as a function of inlet temperatures, Equation 6, and $EIR_{FF}(ff)$ is EIR modifier of normalized mass flowrate at the evaporator, Equation 7.

$$cap_{\theta}(\theta_{e,in}, \theta_{c,in}) = a_1 + a_2\theta_{e,in} + a_3\theta_{e,in}^2 + a_4\theta_{c,in} + a_5\theta_{c,in}^2 + a_6\theta_{e,in}\theta_{c,in} \quad (4)$$

$$cap_{FF}(ff) = b_1 + b_2 ff + b_3 ff^2 + \cdots \quad (5)$$

$$EIR_{\theta}(\theta_{e,in}, \theta_{c,in}) = c_1 + c_2\theta_{e,in} + c_3\theta_{e,in}^2 + c_4\theta_{c,in} + c_5\theta_{c,in}^2 + c_6\theta_{e,in}\theta_{c,in} \quad (6)$$

$$EIR_{FF}(ff) = d_1 + d_2 ff + d_3 ff^2 + \cdots \quad (7)$$

To get the coefficients for the modifier functions that are required for the DX coil performance, the rated performance data is fit using ordinary least squares linear regression functions in the scikit-learn Python package. The Air-Conditioning, Heating, and Refrigeration Institute (AHRI) rated performance data of the HVAC system is found using the model numbers of the equipment and (International Comfort Products). The modifier factors that are functions of inlet temperatures are fit to biquadratic polynomials and the modifier factors that are functions of flow fraction are equal to 1 because the air handler that contains the evaporator has a constant speed fan. A sample of the performance data at rated conditions can be found in Table 2 at one condenser inlet temperature. With the performance curves calculated for the split system AC at TAM-SCHT, the Buildings library DX coil is used in the building model as the HVAC system, shown in Figure 6.

Table 2. Sample rated performance data from AHRI at 1000 CFM.

	Condenser Dry Bulb: 85°F		
Evaporator Wet Bulb (°F)	*Total Cooling (Btu)*	*Sensible Cooling (Btu)*	*Power (kW)*
57	27,536	27,536	2.22
62	27,575	27,575	2.22
67	29,906	23,497	2.21
72	33,025	18,681	2.21
76	35,519	14,827	2.21

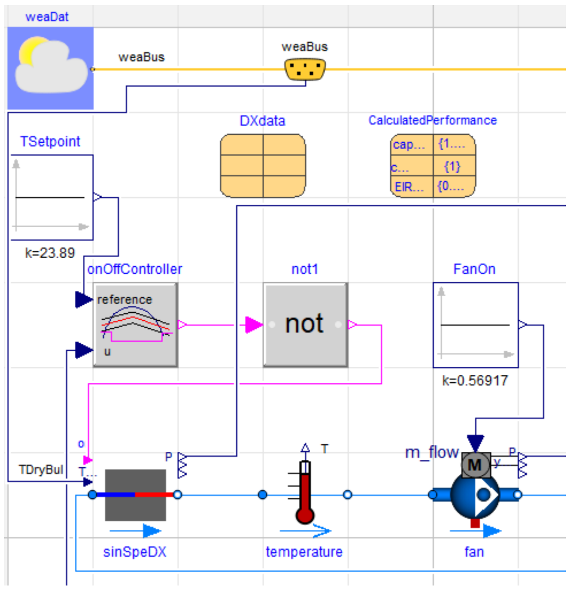


Figure 6. HVAC model for the cooling.

The thermostat control for the model is representative of the thermostat in the testbed. It uses dual setpoint control with a cooling setpoint of 75°F (23.9°C) and a heating setpoint of 69°F (20.6°C). An on/off controller is implemented with a temperature setpoint of 75F with a dead band of 2°F (1.11°C) to mimic the cooling operation in the homes. The thermal zone temperature output from the living floor model is also input to the controller to operate the DX coil. The fan that moves air into the living thermal zone model is a constant mass flow rate fan that operates at the single speed nominal condition.

4 Modelica Model Verification

This section will cover the integration of the thermal zones and HVAC system into one complete model. Initial tests were run to verify the performance of each separate model. The following sections will present the initial results from testing. The period of testing is seven days from August 4th to August 11th of 2022 in which on-site weather conditions are used.

4.1 Building Envelope Verification

The thermal zone models are tested individually to show their performance before implementing into the larger model. An example of the model verification can be seen in Figure 7. Weather data that has been recorded from an on-site weather station and solar radiation data is pulled from (Solcast) to create a TMY3 weather file for Bryan, Texas for the seven days tested. A fan moving the outdoor air into the zone will show the effects on the thermal zone temperature after an intial temperature of 75°F (23.9°C). This can also be seen in Figure 8. We expect this performance from a recently built house with good insulation, even with the outdoor air influencing the zone temperature.

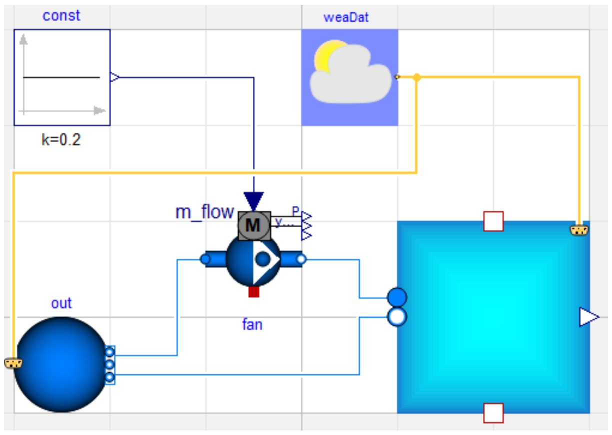


Figure 7. Thermal zone verification model with a fan.

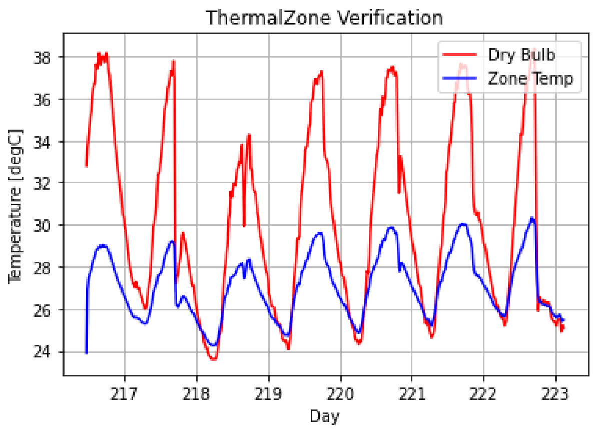


Figure 8. Temperature profile of the outdoor dry bulb (red) and the indoor thermal zone (blue).

The attic and the crawlspace undergo the same verification simulations as the thermal zone but without the fan connection because the two zones do not have air flow through them. The crawlspace perfrmance over the week can be seen in Figure 9 and the attic performance can be seen in Figure 10. These results are what is expected as the crawlspace temperature influence comes from the solar irradiation and the ground heat transfer. For the attic, solar irradiation plays a big role in the temperature influence of the zone. If the horizontal global radiation is plotted alongside the outdoor dry bulb and zone air temperatures, the zone temperature effects follow the radiation during the week simulation period, resulting from the local weather.

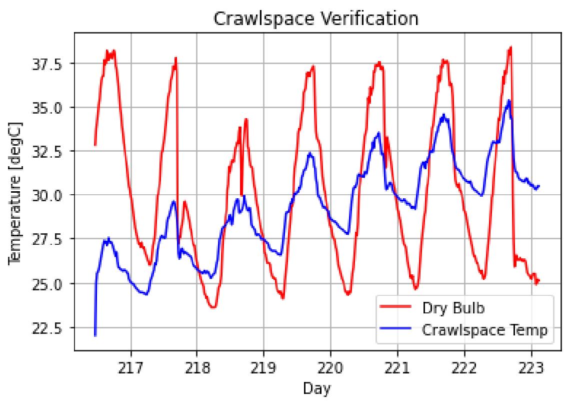


Figure 9. Crawlspace temperature verification results with outdoor dry bulb (red) and zone air (blue).

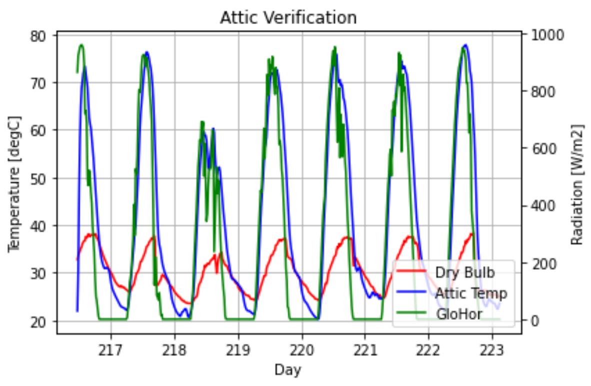


Figure 10. Attic temperature verification with outdoor dry bulb (red), zone air (blue), and the horizontal global radiation (green).

After the building thermal zone models, the HVAC cooling system is next to check.

4.2 DX Coil Verification

For the DX coil verification, the AHRI rated conditions performance data is used to check the accuracy of the model. These rated conditions come from the same data as table 2. The condenser inlet temperature, the outdoor dry bulb temperature for a split system AC unit, and the evaporator inlet temperature are supplied to the model. A quick simulation is run and the cooling capacities and power consumption is compared to the tabulated ratings data.

The model that is used to verify the cooling capacity and power consumption of the DX coil can be seen in Figure 11. This model comes from the air cooled DX coil example called single speed. Modifications have been made to this example. A constant condenser inlet dry bulb temperature is wired to the DX coil model along with a constant on signal for operation. A wether file is used to supply the wet bulb temperature to the moist air media that is supplied as the evaporator inlet air, also a rated condition from AHRI performance data. A constant mass flow rate fan is used to pull the moist air media through the DX coil at the nominal condition. Relative humidity, wet bulb, and dry bulb temperatures are measured for the inlet air to ensure that the rated condition being run is satisfied.

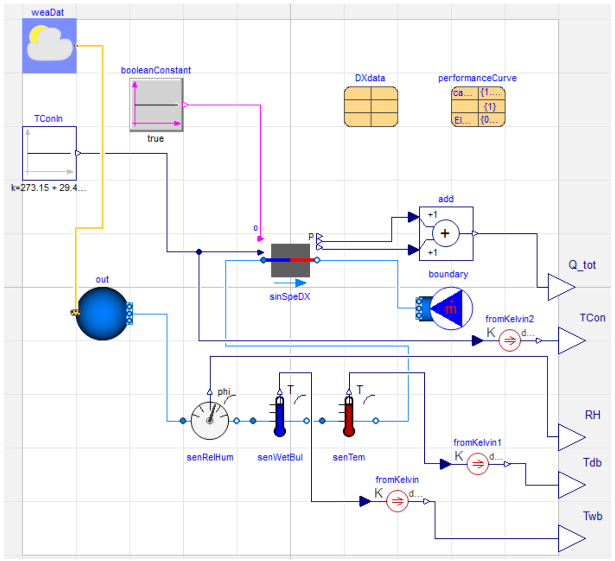


Figure 11. DX Coil verification model, derived from the single speed example.

Each rated condition was run to compare the total and sensible cooling capacity, as well as the power consumption from the DX coil. The results from the rated performance and the simulated perforamnce can be seen in tables 3-5. The DX coil model performs accurately at the highest 4 temperatures at the evaporator inlet. The lowest 2 temperature conditions perform significantly worse at every condenser inlet condition. Improvements to the model will be discussed in the future work section of this paper that follows.

Table 3. DX coil performance at a condenser inlet temperature of 29.4°C, dry bulb.

	Total Capacity (W)	
Twb (°C)	*AHRI Data*	*Simulation*
13.7	8,070.0	11,568.7
16.5	8,081.4	11,568.7
19.3	8,764.6	8,709.1
22.2	9,678.7	9,552.0
24.4	10,409.6	10,459.6

Table 4. DX coil performance at a condenser inlet temperature of 29.4°C, dry bulb.

	Sensible Capacity (W)	
Twb (°C)	*AHRI Data*	*Simulation*
13.7	8,070.0	11,568.7
16.5	8,081.4	11,568.7
19.3	6,8886.3	6,704.5

22.2	5,474.9	5,269.2
24.4	4,345.4	4,169.1

Table 5. DX coil performance at a condenser inlet temperature of 29.4°C, dry bulb.

	Total Power (W)	
Twb (°C)	*AHRI Data*	*Simulation*
13.7	2,220	2,030.3
16.5	2,220	2,030.31
19.3	2,210	2,245.8
22.2	2,210	2,237.3
24.4	2,210	2,175

The next step is to compile the models together.

5 Integrated Envelope and HVAC Model

With all three thermal zone models and the HVAC cooling model completed and verified, they can now be connected together to simulate the operation of TAMSCHT. Because the surface boundaries were defined in the attic and crawlspace model, a heat port is used to connect with the boundary construction that calculates the heat transfer. Figure 1 shows the complete model and each thermal zone is connected to each other. The HVAC cooling system implementation in the complete model is discussed in section 3.2.

The indoor temperature of the living space is plotted in Figure 12 along with the outdoor dry bulb temperature and the setpoint for the thermal zone. The Modelica simulation of the one week period shows that even with the high temperatures, the room setpoint was met. The testbed does not haev any problems reaching this setpoint as it is a newly built home. However, t he zone temperature control can be seen as faulty as the temperature drops to the lower bound of the setpoint deadband. This will be discussed in the future work.

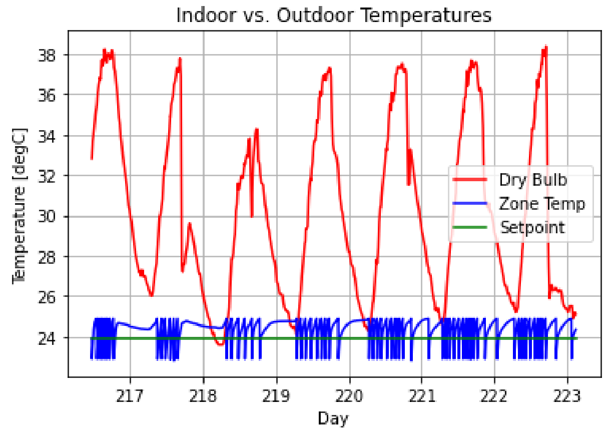


Figure 12. Temperature comparison of outdoor dry bulb (red), zone air (blue), and the setpoint (green).

For validating the Modelica model of TAM-SCHT, we look to compare the simulated AC power consumption against measured power consumption. Detailed power metering in the testbed contains data sampled every minute on the power consumption from each end use. Figure 13 shows the comparison of the simulated power consumption against the measured data. The simulated results managed to capture the trend of the AC power consumption, while further model calibration is necessary to corrrect the minor over-estimation.

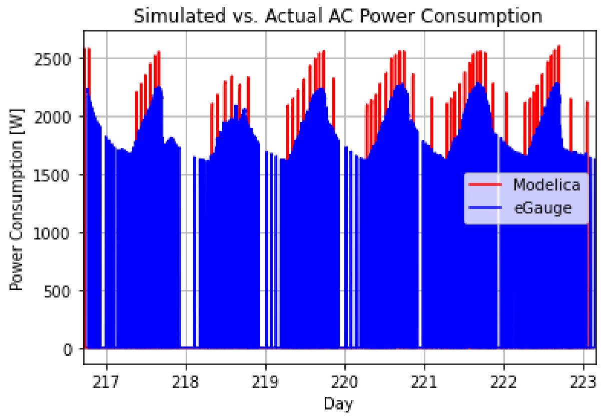


Figure 13. Simulated (red) versus actual (blue) power consumption of TAM-SCHT AC unit.

6 Future Work

Based on the initial simulation results of this integrated model using on-site measure weather data illustrated in te previous section, tuning and model calibration is necessary to improve the model accuracy. On the envelope side, the envelope of the living space thermal zone contains a few materials that are common of code compliant construction in IECC climate zone 2A. Due to limited information from the manufacturer, these materials were substituted in for the layers with descriptions too vague to find thermal properties for, like the “engineered wood panels“ on the exterior of the houses. For the DX coil model, the accuracy at lower evaporator inlet temperature needs to be improved. The DX coil modifier functions need to be calibrated as shown by the over-estimated power consumption from Figure 13. The actual data allows us to tune these model inputs to match the actual on-site measurement.

Further improvement and calibration of this Modelica model is being carried out with the increasing stream of on-site measurement data following the progress of sensor deployment. The model for electric furnace will also be implemented for the testing during the heating season.

7 Conclusion

An accurate building model is crucial for simulation based research. Data from the building manufacturer, equipment distributor, and knowledge of modeling techniques in Modelica are necessary to have a model for testing. In this study, a building model for the Texas A&M Smart and Connected Homes Testbed including its envelope and cooling HVAC system has been developed. Models for both the building envelope and the HVAC cooling system were based on modules from the Modelica Buildings library and set according to the manufacturer specifications. These two models were combined into the testbed model. Initial simulations were carried out separately on the envelope and the HVAC model as well as on the integrated testbed model with local weather data to verify the expected behavior of these models. The simulation results showed reasonable behavior out of the models while further calibration is needed for improved accuracy. This will be carried out with more data being collected on-site on the building envelope thermostat control, and HVAC performance as well as detailed power metering. With the planned calibration, this Modelica model will provide a digital twin to the actual testbed, laying the foundation to future smart homes research at TAM-SCHT.

References

Building Energy Codes Program, U.S. DOE. “Prototype Building Models”. URL: https://www.energycodes.gov/prototype-building-models

Firsich, Thomas, Zhiyao Yang, Fan Feng, and Zheng O’Neill (2022). “Texas A&M Smart and Connected Homes Testbed (TAM-SCHT): An Evaluation and Demonstration Platform for Smart & Grid-interactive Technologies”. Accepted: *2022 ASHRAE Annual Meeting*. Toronto, ON, Canada. June 25-29, 2022.

GreenCast, Syngenta. “Soil Temperature Maps”. URL: https://www.greencastonline.com/tools/soil-temperature

International Comfort Products. AHRI Split System Ratings. URL: https://www.icpeqp.com/AHRIratings/ratings.aspx

“Property Tables and Charts (SI Units)”. 2010

Solcast (2019). Global solar irradiance data and PV system power output data. URL: https://solcast.com/

Wetter, Michael, Wangda Zuo, Thierry S. Nouidui, and Xiufeng Pang. Modelica Buildings library. Journal of Building Performance Simulation, 7(4), 253-270, 2014. DOI: 10.1080/19401493.2013.765506

Wilson, E., C. Engebrecht Metzger, S. Horowitz, and R. Hendron (2014). “2014 Building America House Simulation Protocols”. Tech. rep. National Renewable Energy Laboratory. Report number: NREL/TP-5500-60988

Performance Enhancements for Zero-Flow Simulation of Vapor Compression Cycles

Hongtao Qiao* Christopher R. Laughman

Mitsubishi Electric Research Laboratories
Cambridge, MA, USA
{qiao,laughman}@merl.com

Abstract

Models that correctly describe the dynamic behavior of vapor compression cycle at low or zero refrigerant mass flow rates are valuable because they can be used to handle low load, on/off cycling and inactive component conditions. However, low- or zero-flow simulation imposes significant computational challenges because of high frequency oscillations in mass flow. We explore techniques that may be used for improving robustness and performance of low- or zero-flow simulation. Comparisons are conducted to demonstrate the efficacy of the proposed techniques. It is shown that these techniques can result in simulations that are more robust and significantly faster than real-time.

Keywords: Modelica, zero flow, heat exchanger modeling, dynamic simulation, vapor compression cycle

1 Introduction

Numerical simulations are widely employed in the modern day HVAC&R (Heating, Ventilation, Air-Conditioning and Refrigeration) industries to assist in the design and optimization of advanced products in response to the increasing pressure of cost reduction and high-energy efficiency standards. With the aid of simulation tools, design engineers can evaluate a conceptual product design on computers instead of building real systems and conducting expensive tests in the laboratory, thereby shortening the time required for design cycles.

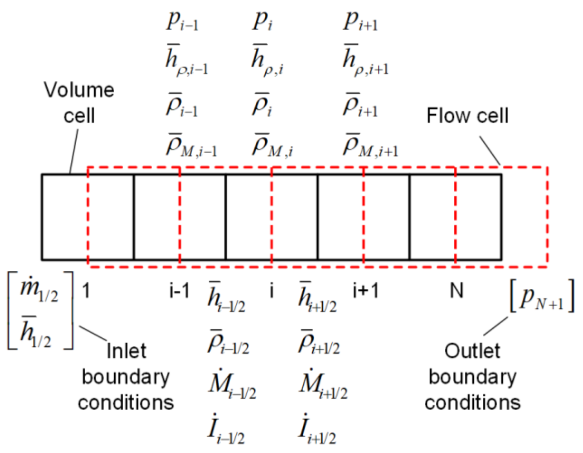
In general, vapor compression system simulations can fall into one of two categories: steady-state or transient. The evaluation of the steady-state, full-load performance of vapor compression systems is often used to determine the system capacity and size. However, vapor compression systems rarely operate under steady-state conditions, and dynamic models are more adequate for a realistic representation of the system response. These models are typically used for two types of studies: (1) examining small-scale changes in the refrigerant-side behavior, such as flow instabilities, and (2) examining larger system-level changes, such as system behavior during start-up, shut-down, or defrosting. While many models can typically reproduce small-scale behavior quite accurately, the increased complexity and nonlinearity associated with large-scale transients often results in predictions that have much lower accuracy.

Low and zero-flow phenomena are often encountered in the operation of vapor compression systems with large transients, including on/off cycling of air-conditioning systems, operating mode switch of variable refrigerant flow systems, and reverse-cycle defrosting of heat pump systems as examples. Simulation of system dynamics under such conditions presents numerical challenges, such as problems with robustness and an attendant increase in simulation time due to direction switching flows. In recognition of these problems, Dermont et al. (2016) proposed an approach to improve zero-flow simulation based on a systematic analysis of heat transfer coefficients. Although this approach was shown to increase simulation robustness under (near) zero-flow conditions, the presented use cases ran much slower than real time. In his sole-authored paper, Zimmer (2020) suggested that regularization schemes were required to improve model robustness against zero mass flow without giving further details. Li (2020) discussed the computational improvement from table-based refrigerant property calculation models with Analytical Jacobians. However, the implementation of such methods is a long-term task and requires significant effort. We are thus motivated to explore effective numerical techniques to improve the performance of zero-flow simulations, especially focusing on robustness and improvements in the simulation speed, with a goal of achieving faster than real-time dynamic simulation.

The remainder of the paper is organized as follows. In Section 2, we present a regularized pressure drop model that has significant benefits for these on/off transient simulations. In Section 3, we discuss the advantages and disadvantages of static and dynamic heat transfer coefficient models. In Section 4, we describe the single pressure heat exchanger model and its potential to speed up the zero-flow simulation. Conclusions from this work are then summarized in Section 5.

2 Pressure Drop Model

The finite volume method is often used to discretize the governing equations that describe the dynamics of refrigerant 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. 1.



p_{i-1} $\bar{h}_{\rho,i-1}$ $\bar{\rho}_{i-1}$ $\bar{\rho}_{M,i-1}$
p_i $\bar{h}_{\rho,i}$ $\bar{\rho}_i$ $\bar{\rho}_{M,i}$
p_{i+1} $\bar{h}_{\rho,i+1}$ $\bar{\rho}_{i+1}$ $\bar{\rho}_{M,i+1}$
Volume cell
Flow cell
$\begin{bmatrix}\dot{m}_{1/2}\\ \bar{h}_{1/2}\end{bmatrix}$
1
i-1
$\bar{h}_{i-1/2}$ $\bar{\rho}_{i-1/2}$ $\dot{M}_{i-1/2}$ $\dot{I}_{i-1/2}$
i
$\bar{h}_{i+1/2}$ $\bar{\rho}_{i+1/2}$ $\dot{M}_{i+1/2}$ $\dot{I}_{i+1/2}$
i+1
N
$[p_{N+1}]$
Inlet boundary conditions
Outlet boundary conditions

Figure 1. Staggered grid scheme

Since the inertia term, dynamic pressure wave and gravity effect in the momentum equation are usually of minor importance in these applications, they are often neglected in heat transfer analyses to reduce the modeling complexity (Brasz and Koenig, 1983). As a result, the discretized governing equations for 1-D flow are often given as

$$\frac{dM_i}{dt} = \dot{m}_{i-1/2} - \dot{m}_{i+1/2} \tag{1}$$

$$\frac{dU_i}{dt} = \dot{m}_{i-1/2}\bar{h}_{i-1/2} - \dot{m}_{i+1/2}\bar{h}_{i+1/2} + P\Delta z q_i'' \tag{2}$$

$$p_i - p_{i+1} = \Delta p_{f,i} \tag{3}$$

Pressure and specific enthalpy are often selected as state variables to avoid non-linear algebraic equations when they are the independent properties in the media models. The equations of mass and energy can then be rewritten using the time-derivative of pressure and specific enthalpy based on the chain rule.

It is evident from Eq. (3) that the pressure difference between adjacent volumes is solely determined by the frictional pressure loss. The frictional pressure drop is often a complex nonlinear function of Reynolds number and thus mass flow rate, and is often impossible to invert analytically. Since the pressures in each control volume are known at each time step, numerical iterations may be required to solve mass flow rate based on pressure difference, resulting in slower simulations.

Unlike the steady-state models that are often used for system design and require high prediction accuracy, dynamic models are widely used over much wider operating ranges and thus require high robustness and high efficiency, which is sometimes achieved at the expense of accuracy or fidelity to published frictional pressure loss correlations (Idelchik, 1986). Therefore, simplified models are often used to reduce modeling complexity and improve simulation speed. Among those, the following model approximating the frictional pressure loss (Laughman and Qiao, 2018) $\Delta p = f(\dot{m})$ is expressed as

$$\Delta p = K\Delta p_0 \left(\dot{m}/\dot{m}_0\right)^b \tag{4}$$

where Δp_0 and $\dot{m}_0$ are the parameters in the nominal condition, and K and b (often greater than 1) are curve-fitting constants. This relation is not only less nonlinear than the original correlation-based relations, but it is also easily invertible and can allow the pressure loss to be calculated as a function of the mass flow rate, or vice versa. As such, the resulting system simulations have much faster performance, since the nonlinear dependence on the variety of input variables is removed from the relation and the integrator can take much larger steps. One can easily invert Eq. (4) and obtain the derivative of mass flow rate with respect to pressure drop

$$\frac{d\dot{m}}{d(\Delta p)} = \frac{\dot{m}_0^b}{bK\Delta p_0}\frac{1}{\dot{m}^{b-1}} \tag{5}$$

Eq. (5) suggests that $\dot{m} = f^{-1}(\Delta p)$is not Lipschitz continuous and becomes increasingly sensitive to pressure drop as it approaches zero and eventually the derivative becomes infinite when mass flow is zero. Meanwhile, we can examine the time derivative of the mass flow

$$\frac{d\dot{m}}{dt} = \frac{d\dot{m}}{d(\Delta p)}\frac{d(\Delta p)}{dt} = \frac{\dot{m}_0^b}{bK\Delta p_0}\frac{d(\Delta p)}{dt}\dot{m}^{1-b} \tag{6}$$

Eq. (6) shows that the time derivative of the mass flow rate is very large when the mass flow rate is small, indicating that Eq. (6) is a stiff ODE. One can make a simple analogy between Eq. (6) and the ODE $y' = -ay\ (a > 0)$. To obtain a stable solution with the Explicit Euler method, the time step must satisfy $\Delta t < 2/a$. When a is very large, the time step becomes very small, which is exactly the case for Eq. (6) since $a = -\frac{\dot{m}_0^b}{(bK\Delta p_0)}\frac{d(\Delta p)}{dt}\frac{1}{\dot{m}^b}$ is infinite when the mass flow is zero.

The point with infinite derivative is often called singularity point. In consequence of this behavior, the simulation often stalls for off-cycle conditions of vapor compression systems in which mass flow rate is extremely low. A conventional remedy for this behavior is to replace the singular part with locally non-singular substitute resulting in a finite derivative at the point; this process is often referred to as regularization. According to Dermont et al. (2016), a regularized pressure drop correlation is necessary for a complex thermo-fluid model to compute under zero flow conditions. The built-in implementation of power function that employs such regularization for terms in Eq. (4) can be found in `Modelica.Fluid.Utilities.regPow`.

```
function regPow
  extends Modelica.Icons.Function;
  input Real x;
  input Real a;
  input Real delta=0.01;
  output Real y;
algorithm
  y := x*(x*x+delta*delta)^((a-1)/2);
end regPow;
```

This `regPow` function approximates $sign(x)|x|^a$ and is regularized with finite derivatives around the singular point. In this function, the parameter `delta` is used to specify the regularization range. When `abs(x) << delta`, the regularization results in a linear approximation for the original function with the Lipschitz constant to be unity. Although the built-in implementation successfully eliminates the singularity point, it can be further improved. Consequently, one more parameter can be added to `regPow` so that different types of regularization can be formulated.

```
function regPowGen
  extends Modelica.Icons.Function;
  input Real x;
  input Real a;
  input Real delta=0.01;
  input Real b=1;
  output Real y;
algorithm
  y := x^b*(x*x+delta*delta)^((a-b)/2);
end regPowGen;
```

With different values for the parameter `b`, different approximations can be obtained for the original function. When `b = a`, the `regPowGen` function is equivalent to the original power function without regularization. With `b = 1`, the function is the same as the built-in function `regPow`. Fig. 2 shows different regularization schemes for the pressure loss relation in the neighborhood around the singularity point. Without regularization, the derivative at the origin is infinite and the mass flow rate is extremely sensitive to the change in pressure loss around the singularity point, which can cause simulation of low- or zero-flow conditions to crash. The simulation performance does improve with the built-in approach, but is still far from satisfactory because small pressure differences can still result in large variations in mass flow. With `b = 3`, a cubic approximation is used around the singularity point and the Lipschitz constant is much smaller. As a result, the mass flow rate becomes far less sensitive to pressure differences so that the solver can take much larger step sizes.

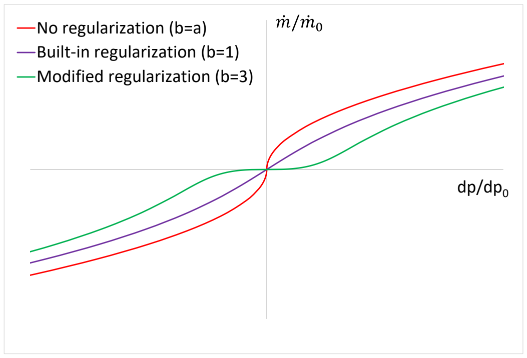


Figure 2. Different types of regularization for pressure loss relation

To evaluate the efficacy of different types of regularization, off-cycle transients of a room air-conditioning system, illustrated in Fig. 3, were simulated. The system ran steadily before it was shut down at 500 sec. The off-cycle period then lasted for 4500 sec and the simulation ended at 5000 sec. As shown in Fig. 4, adequate regularization can make a substantial improvement to the simulation performance. With the built-in `regPow` function for the simplified pressure loss relation, it took more than 23000 sec of CPU time to finish a 5000 sec simulation. However, with modified regularization scheme (`b=3`), it only took around 1700 sec to finish the simulation and CPU time was reduced by more than 10 times. Reducing the sensitivity of the mass flow rate to the pressure difference around the singularity point can thus be a key for faster simulation of low or zero flow conditions. No changes in the component models were required with the proposed regularization scheme. Please note that the proposed cubic approximation should be directly applied to the function that calculates the mass flow rate given the pressure drop, i.e., $\dot{m} = f^{-1}(\Delta p)$. This regularization scheme exhibits no robustness issues in our models since the inverse function $\dot{m} = f^{-1}(\Delta p)$ can be easily computed. If the inverse function $\dot{m} = f^{-1}(\Delta p)$ cannot be obtained, one needs to be cautious when applying the proposed scheme due to the possibly resulting numerical issues.

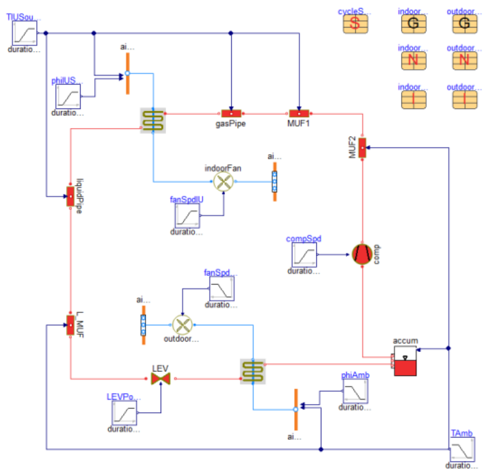


Figure 3. Modelica model of a room air-conditioning system

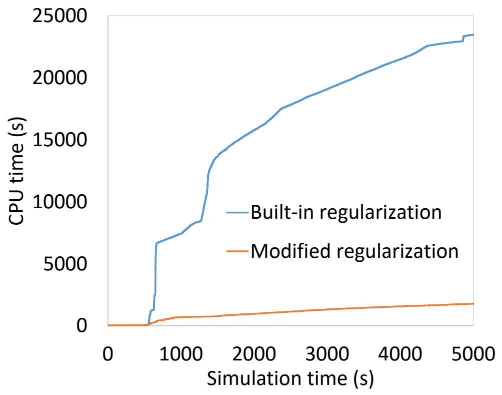


Figure 4. CPU time vs. simulation time with different types of regularization

3 Heat Transfer Coefficient Model

The description of local heat transfer coefficients (HTCs) in the simulation models of thermofluid systems can be particularly challenging, as the correlations are usually formulated with accuracy as the primary concern, and with little regard for computational considerations. Consequently, they can be difficult to incorporate into system-level models of thermofluid systems as they may be extremely nonlinear. Meanwhile, these correlations are usually defined only for specific flow conditions or refrigerant phases, so that there will inevitably be significant discontinuities between regions of the validity for specific correlations. Dynamic simulation presents additional difficulties as the unknown refrigerant mass flow rates, pressures, and specific enthalpies preclude the use of any initial information about the phase of the refrigerant (condensation, evaporation, liquid, or vapor) or the flow regime (laminar or turbulent), so the correlations must be defined in a manner which encompasses a wide range of flow conditions.

One alternative approach that has been successfully used to mitigate the nonlinearities of detailed heat transfer coefficient correlations has been the creation of simplified models that capture the general trends of those detailed correlations without implementing their complexity. These simplified correlations can be justified via the improved numerical performance of the simulation models, which may not even function with some of the complex correlations found in the literature, as well as the fact that the overall heat transfer coefficient for many refrigerant-to-air heat exchangers is dominated by the air-side heat transfer coefficient, rather than the refrigerant-side heat transfer coefficient.

A wide variety of forms can be used for these relations, depending on the required parametric dependence or level of fidelity to the behavior of the original correlations. For example, we used a simplified heat transfer relation for each phase according to

$$\alpha = K\alpha_0\left(\dot{m}/\dot{m}_0\right)^b \tag{7}$$

The constants α_0 for the liquid, two-phase, and vapor flow regions were calculated by coarsely approximating the behavior of the full correlations over their regions of validity, and a trigonometric interpolation method was used to smoothly transition between phases (Richter, 2008).

Laughman and Qiao (2018) proposed the incorporation of dynamics into the closure models to decouple the heat transfer coefficient from the other state variables. This makes the closure variables into state variables of the system, and will decouple the value of the closure variable in the fluid computations with the value of the closure variable calculated from the other state variables. In the case of the heat transfer coefficient, this may be calculated by

$$\frac{d\alpha}{dt} = \frac{1}{\tau}\left(\hat{\alpha} - \alpha\right) \tag{8}$$

where $\hat{\alpha}$ represents the algebraic heat transfer coefficient which can be calculated using either detailed or simplified relations, and α represents the filtered version of the heat transfer coefficient. The parameter τ should be tuned to be substantially faster than other time constants of the system in order to ensure that it will not change the system response.

This dynamic heat transfer coefficient model has proved to be effective at eliminating the spurious oscillations caused by the high gain of $\partial\alpha/\partial x$ in the transition region from vapor phase to two-phase and increase model robustness. However, for the case of off-cycle simulation, the filtered heat transfer model can potentially slow down the simulation because it increases the number of state variables in the system. As demonstrated in Fig. 5, it took around 3700 sec CPU time to finish the same off-cycle simulation of the air-conditioning system described in Fig. 3 with filtered heat transfer coefficient model, which was 2 times longer than the CPU time of the simulation with static heat transfer coefficient model. It was evident that the filtered model slowed down the simulation remarkably for the first 500 sec after the system was shut down. During this period of time, the refrigerant mass flow rates declined dramatically, resulting in a rapid change in heat transfer coefficients. Setting 'log norm true' during the running simulation when using the Modelica compiler Dymola 2020x (Dymola, 2020) can determine that some of heat transfer coefficient states were causing the integrator to be slow. In summary, the filtered heat transfer coefficient model can help improve model robustness and eliminate the high-frequency numerical oscillations, but not necessarily speed up the off-cycle simulation. It is recommended that modelers try both static and filtered approaches to the heat transfer coefficients to choose a more appropriate approach on a case-by-case basis.

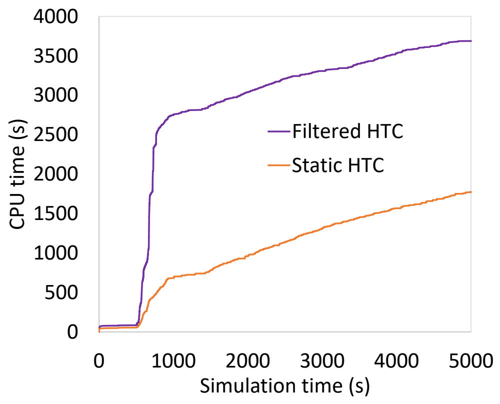


Figure 5. CPU time vs. simulation time with different HTC models

4 Single Pressure Heat Exchanger Model

Heat exchangers usually require particular attention in the modeling of vapor compression cycles because they are the main components where exchange of mass, energy and momentum take place. Accurate mathematical and physical representations for heat transfer and fluid flow phenomena in heat exchangers are always crucial for the overall cycle simulation. In general, three modeling paradigms are often used for heat exchanger simulations. In order of increasing complexity and sophistication, they are the lumped parameter method, the moving boundary method and the finite volume method, respectively.

Lumped parameter models simplify the description of the characteristics of an inherently spatially distributed physical system with mean properties that are assumed homogeneous throughout the heat exchanger by averaging out the spatial variations. With this approach, only the overall mass and energy balances (2 equations) are considered and the thermal behavior of heat exchangers is modeled as a single control volume. Since this approach disregards the spatial variation in properties and the distinct differences of the heat transfer mechanisms between single-phase and two-phase, these models inevitably result in the most inaccurate predictions amongst these three modeling approaches.

Recently, Qiao and Laughman (2022) developed a new low-order heat exchanger model based on the lumped parameter approach. Unlike conventional lumped parameter models, this new model assumes a distribution of refrigerant enthalpy so that the spatial variations of refrigerant properties such as density and specific enthalpy can be taken into account. The overall mass and energy balances to describe the refrigerant dynamics are given as

$$V\left(\frac{\partial\bar{\rho}}{\partial p}\frac{dp}{dt}+\frac{\partial\bar{\rho}}{\partial\bar{h}}\frac{d\bar{h}}{dt}\right)=\dot{m}_{in}-\dot{m}_{out} \tag{9}$$

$$\begin{aligned}&\bar{\rho}V\frac{d\bar{h}}{dt}-V\frac{dp}{dt}=\\&\dot{m}_{in}h_{in}-\dot{m}_{out}h_{out}-\left(\dot{m}_{in}-\dot{m}_{out}\right)\bar{h}+\sum_{j=1}^{3}q_{r,j}\end{aligned} \tag{10}$$

It is assumed that refrigerant enthalpy varies exponentially in the heat exchanger. Therefore, the local refrigerant quality is determined by

$$x=\frac{x_{in}-x_{out}}{1-\exp(-1)}\exp(-\zeta)+\frac{x_{out}-x_{in}\exp(-1)}{1-\exp(-1)} \tag{11}$$

where ζ is the fraction of the heat exchanger covered by the portion from the inlet to the location of interest. Fractions of vapor, two-phase and liquid zone can be readily computed with this enthalpy distribution profile. The mean specific enthalpy of refrigerant in the heat exchanger can also be determined by

$$\begin{aligned}\bar{h}&=\frac{\bar{\rho}_{vap}\bar{h}_{vap}\zeta_{vap}}{\bar{\rho}}+\frac{\left[\bar{\gamma}\rho_g h_g+\left(1-\bar{\gamma}\right)\rho_f h_f\right]\zeta_{tp}}{\bar{\rho}}\\&+\frac{\bar{\rho}_{liq}\bar{h}_{liq}\zeta_{liq}}{\bar{\rho}}\end{aligned} \tag{12}$$

where $\bar{\gamma}$ is the mean void fraction of the two-phase flow. Since $\bar{h}$ is a state variable and is known at each time step, Eq. (12) can be used to iterate h_{out} so that the entire system is closed. The accuracy of the new models can be significantly improved with the addition of refrigerant enthalpy profile as compared to moving boundary models, but with minimum additional computational cost. A full description of this modeling approach, which is beyond the scope of the present work, can be found in the referenced paper.

In comparison, the moving boundary method is characterized by dividing the heat exchanger into different control volumes, each of which exactly encompasses a particular fluid phase (vapor, two-phase or liquid) and is separated by a moving boundary where the phase transition occurs. In contrast to the distributed parameter models, the number of control volumes in moving boundary models may vary because fluid phases can disappear or appear under large disturbances. These models may consist of at most three control volumes and at least one at a time (6 equations at most). The objective of such models is to capture the thermal behavior inside these control volumes and time-varying position of phase boundaries. Moving boundary models generally result in much faster simulations compared to distributed parameter models due to their small size while more accurately capturing the time-varying dynamics of these systems, but these models are inherently fragile due to their variable model structures. For instance, moving boundary models cannot manage zero or reverse flows because these models are designed with the strict assumption that refrigerant flow enters the heat exchanger from one end, and leaves from the other. These models are either are either over- or underdetermined if these assumptions are violated (Qiao et al., 2016).

Finite volume heat exchanger models are particularly useful for describing spatially dependent phenomena and detailed component performance, such as the effect of local heat transfer and pressure drops or the branching and joining of refrigerant pipes as a result of particular circuiting configurations. As discussed earlier, finite volume models are comprised of an alternative sequence of volume cells and flow cells. The resultant modular nature allows great flexibility in system configurations, and different component models can be seamlessly linked together (Qiao et al., 2015). However, one of the disadvantages of this modeling approach is that it creates many dynamic pressure states. For a model with N control volumes, it has 2N dynamic states, i.e., N pressure states and N specific enthalpy states, resulting in N mass flow rates that need to be computed based on pressure differences. Therefore, 3N equations are needed to solve the model. Under off-cycle conditions, these N mass flow rates will all decline rapidly and each will enter the region where the mass flow rate is highly sensitive to pressure difference. This will inevitably increase the likelihood that the integrator will substantially reduce the step size during the solving process of the model. Based on this reasoning, it is anticipated that the off-cycle simulation can be greatly accelerated if the dependence of mass flow upon pressure difference can be removed. We thus propose a heat exchanger model with a single pressure state and the governing equations are given as

$$A\Delta z\left(\frac{\partial\bar{\rho}_i}{\partial p}\frac{dp}{dt}+\frac{\partial\bar{\rho}_i}{\partial\bar{h}_{\rho,i}}\frac{d\bar{h}_{\rho,i}}{dt}\right)=\dot{m}_{i-1/2}-\dot{m}_{i+1/2} \qquad (13)$$

$$A\Delta z\left(\bar{\rho}_i\frac{d\bar{h}_{\rho,i}}{dt}-\frac{dp}{dt}\right)=$$
$$\dot{m}_{i-1/2}\left(\bar{h}_{i-1/2}-\bar{h}_{\rho,i}\right)\bar{h}_{i-1/2}-\dot{m}_{i+1/2}\left(\bar{h}_{i+1/2}-\bar{h}_{\rho,i}\right)+P\Delta z q_i'' \qquad (14)$$

In this new modelling approach, the volume cells within the component model share the same pressure. The number of dynamic states is N+1, i.e., one pressure and N specific enthalpies. Mass flow rates between volume cells will be computed through the coupling between the equations of mass and energy (Qiao and Laughman, 2018). The momentum equation is therefore not needed, so that the whole model consists of only 2N equations. It is worthwhile to point out that pressure drop between components is still taken into account in the system model, though the pressure drop is lumped together and calculated at the inlet or the outlet of the component model depending upon the model structure. As a result, the number of pressure states is significantly reduced, while the number of flow models calculating mass flow based on pressure differences is also decreased. These changes can substantially speed up the off-cycle simulation.

With the modified regularization scheme for pressure loss relation, static heat transfer coefficient model, and single pressure heat exchanger model, the same off-cycle simulation finished with around 200s of CPU time, which was 9 times faster than the conventional finite volume models, as shown in Fig. 6. The speedup improvement achieved using all of the techniques discussed in this work was substantial, given that the off-cycle simulation without any of these enhancements was more than 100 times slower. The discrepancies arising from the approximation of lumped pressure drop were minimal, as the system pressures equalize quickly under off-cycle conditions, and pressure drops between volume cells are negligible. Fig. 7 illustrates the suction and discharge pressure transients as well as compressor mass flow during off-cycle. The compressor mass flow instantly dropped to zero after system was shut down, and suction and discharge pressures came to an equilibrium shortly afterwards, which somewhat justified the key assumption of the proposed single pressure heat exchanger modeling approach.

Fig. 8 illustrates a vapor compression system with two evaporators, which was modified based on the results of the single-evaporator system described in Fig. 3. To further demonstrate the efficacy of the proposed enhancement techniques for zero-flow simulation, we present another case study, in which the system described in Fig. 8 was operated normally for the first 500 sec with two active evaporator branches, after which the first evaporator branch was turned off (the fan was off and the valve was closed) and the system continued running before being completely shut off at 3000 sec. The changes in the actuators and the CPU time as a function of simulation time were given in Fig. 9. This simulation finished smoothly and only took 600 sec of CPU time, indicating the effectiveness of the proposed techniques.

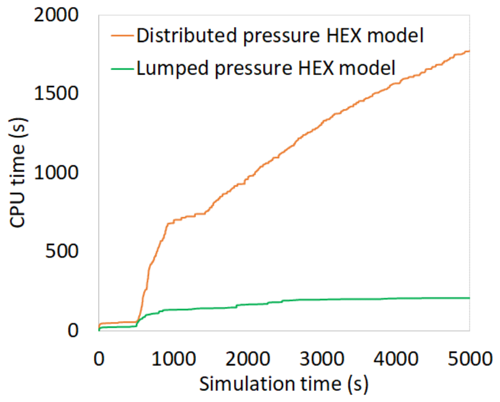


Figure 6. CPU time vs. simulation time with different heat exchanger models

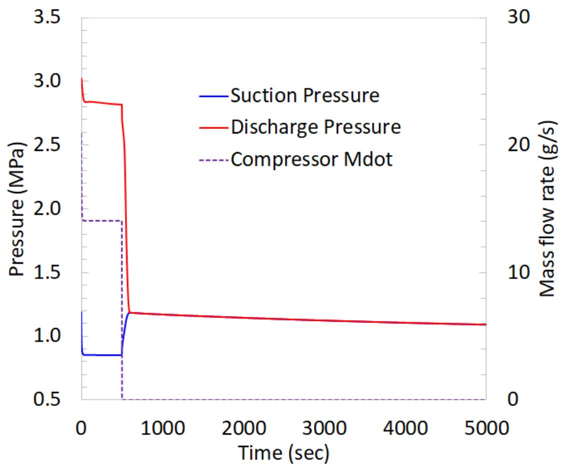


Figure 7. Pressure and compressor flow transients under off-cycle operation

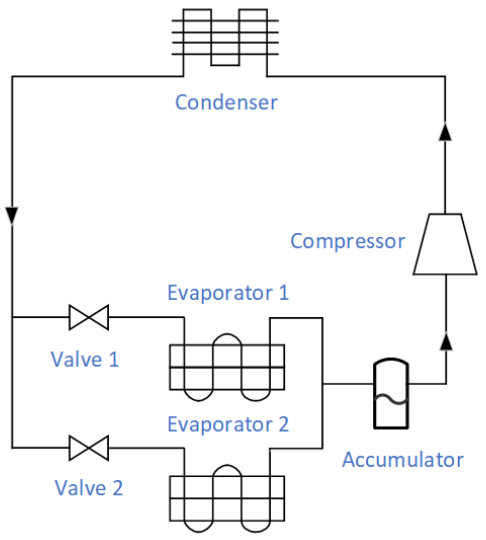


Figure 8. A vapor compression system with two evaporators

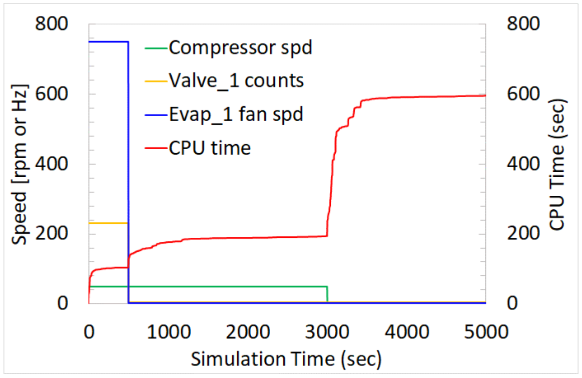


Figure 9. Actuator changes and CPU time vs. simulation time for the system in Fig. 8

5 Conclusions

This paper explored a set of techniques to improve the robustness and speed for zero-flow simulation of vapor compression cycles. It was found that reducing the sensitivity of mass flow to pressure differences was an important key to accelerating the zero-flow simulation. This can be achieved by regularizing the pressure loss relation with cubic approximation in the neighborhood around the singularity point. We also recommend using a static heat transfer model because it reduces the number of dynamic states if no spurious oscillations appear in the simulation. Lumping refrigerant pressure drops at the inlet or outlet of heat exchangers or pipes also demonstrated value in further speeding up the zero-flow simulation. These techniques proved to be efficient to handle refrigerant dynamics in on/off cycling and inactive component conditions.

References

Christopher Laughman and Hongtao Qiao. On closure relations for dynamic vapor compression cycle models. American Modelica Conference, 2018.

Christopher Richter. Proposal of new object-oriented equation-based model libraries for thermodynamic systems. Ph.D. thesis, Technische Universität Braunschweig, Institut für Thermodynamik, 2008.

Dassault Systemes, AB. Dymola 2020x, 2020.

Dirk Zimmer. Robust object-oriented formulation of directed thermofluid stream networks. *Mathematical and Computer Modeling of Dynamical Systems*, Vol. 26, No. 3, pp. 204–233, 2020.

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 and Christopher Laughman. A Low-Order Model for Nonlinear Dynamics of Heat Exchangers. 18th International Refrigeration and Air Conditioning Conference at Purdue. 2022.

Hongtao Qiao, Christopher Laughman, Vikrant Aute and Reinhard Radermacher. An advanced switching moving boundary heat exchanger model with pressure drop. *International Journal of Refrigeration*, No. 65, pp. 154-171, 2016.

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.

Idelchik, I.E. Handbook of hydraulic resistance. Washington. 1986.

Joost J. 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.

Lixiang Li, Jesse Gohl, John Batteh, Christopher Greiner, Kai Wang. Fast simulations of air conditioning systems using spline-based table look-up method (SBTL) with analytical Jacobians. American Modelica Conference, 2020.

Pieter Dermont, Dir Limperich, Johan Windahl, Katrin Prolss and Cartsen Kubler. Advances of zero flow simulation of air conditioning systems using Modelica. Proceedings of the 1st Japanese Modelica Conference, Tokyo, Japan. 2016.

Tradeoffs Between Indoor Air Quality and Sustainability for Indoor Virus Mitigation Strategies in Office Buildings

Cary A. Faulkner[1] John E. Castellini Jr.[1] Yingli Lou[2] Wangda Zuo[3,4] David M. Lorenzetti[5]
Michael D. Sohn[5]

[1]Department of Mechanical Engineering, University of Colorado Boulder, USA
[2]Department of Architectural Engineering, University of Colorado Boulder, USA
[3]Department of Architectural Engineering, Pennsylvania State University, USA
[4]National Renewable Energy Laboratory, USA
[5]Energy Analysis and Environmental Impacts Division, Lawrence Berkeley National Laboratory, USA

Abstract

The COVID-19 pandemic has motivated building operators to improve indoor air quality (IAQ) through long-term sustainable solutions. This paper develops a modeling capability using the Modelica *Buildings* library to evaluate three indoor virus mitigation strategies: use of MERV 10 or MERV 13 filtration and supply of 100% outdoor air into a building with MERV 10 filtration. New evaluation metrics are created to consider the impact of improving IAQ on financial and environmental costs. The mitigation strategies are studied for medium office buildings in three locations in the United States with differing climates and electricity sources. The results show that use of 100% outdoor air can significantly improve IAQ with limited increases in costs in the milder climate, but leads to very high costs in the hot and humid and very cold climates. MERV 13 filtration can improve IAQ relative to MERV 10 filtration with small increases in costs in all locations.

Keywords: Indoor air quality, costs, COVID-19, sustainability.

1 Introduction

Improving indoor air quality (IAQ) during the COVID-19 pandemic while limiting environmental impact in the age of rapid climate change is challenging. Operation of building heating, ventilation, and air-conditioning (HVAC) systems can mitigate indoor virus concentration and reduce risk of infection (Pease, Wang, et al. 2021; Shen et al. 2021; Vlachokostas et al. 2022; Pease, Salsbury, et al. 2022), but can also increase HVAC energy consumption (Faulkner et al. 2022; Cortiços and Duarte 2021). For example, higher outdoor air ventilation rates can increase heating/cooling energy usage, or use of efficient, high pressure drop filters can increase fan energy consumption. Improving IAQ while minimizing increases in energy costs and emissions is a challenge dependent on several factors, including mitigation strategy, climate, and electricity sources.

Previous literature studied tradeoffs between IAQ and HVAC energy consumption. Santos and Leal (Santos and Leal 2012) examined the impact of increased ventilation rate on energy consumption in European cities. They found the increase in energy is dependent on climate and building type. Aviv et al. (Aviv et al. 2021) proposed a novel HVAC strategy to couple radiant systems and natural ventilation to increase outdoor air ventilation while minimizing energy consumption. Schibuola and Tambani (Schibuola and Tambani 2021) found high mechanical ventilation rates with efficient air handling units could reduce the risk of infection of COVID-19 and improve energy efficiency in Italian secondary schools. Ben-David and Waring (Ben-David and Waring 2018) studied the effects of increased filtration and ventilation on indoor exposure to $PM_{2.5}$ and ozone. They found that improving filtration tended to have a greater impact on the cost function incorporating energy and exposure costs.

Despite significant recent progress in the literature, there is potential for further analysis. First, studies often assume steady-state scenarios and neglect the dynamics of the HVAC system. For example, constant ventilation rates and outdoor air fractions may be assumed, when these values are dynamic in practice and affect both IAQ and energy consumption. Additionally, researchers often consider energy and costs to quantify sustainability, but do not always include greenhouse gas emissions. This is especially important as new policies incentivize reducing building emissions.

To address this research gap, we propose a study to analyze the tradeoffs between IAQ and costs, including costs associated with filters, HVAC energy consumption, and CO_2 emissions. Newly available dynamic CO_2 emission data is used to quantify emissions in different locations based on electricity sources. Three mitigation strategies are studied in three locations with distinct climates and electricity sources. The mitigation strategies include different levels of filtration, such as MERV 10 and MERV 13 filtration, as well as supply of 100% outdoor air into a building with MERV 10 filtration. We simulate the scenarios using detailed system modeling of a prototype medium office building initially sized for MERV 10 filtration based on the Modelica *Buildings* library (Wetter, Zuo, et al. 2014; Wetter, Bonvini, et al. 2015). New component models for HVAC filtration and viral transmission are developed to support the analyses in this study.

The remainder of this paper is organized as follows. We describe the Modelica modeling to support the analyses in this study in Section 2. Next, methods to evaluate and compare the mitigation strategies are detailed in Section 3. The scope of analysis for this study including the three mitigation strategies and three locations is described in Section 4. The results in terms of IAQ and costs are presented in Section 5. Finally, conclusions are drawn in Section 6.

2 Modelica Modeling

The modeling based on the Modelica *Buildings* library is detailed in this section. First, we describe the developed component models to support the analyses in this study. We then detail the system modeling of the studied medium office building.

2.1 Component Modeling

We describe the new component models for HVAC filters and viral transmission in this section.

2.1.1 HVAC Filter Model

We first develop an HVAC filter model to simulate filtration of viral particles. The model includes: removal of viral particles based on a defined efficiency and static pressure drop due to the resistance the filter imposes on the airflow.

The removal of virus can be described as:

$$c_{out} = (1 - \eta_{filter}) c_{in}, \tag{1}$$

where c_{out} is the virus concentration exiting the filter, η_{filter} is the filter removal efficiency in terms of percentage of virus removed, and c_{in} is the virus concentration entering the filter. The filter efficiency can be between 0-100%, where $\eta_{filter} = 100\%$ describes a filter that completely removes all virus in the airflow.

Next, the static pressure drop caused by the resistance of the filter is:

$$\Delta p_{filter} = k_{filter} \dot{m}_{filter}^2, \tag{2}$$

where Δp_{filter} is the static pressure drop caused by the filter, $\dot{m}_{filter}$ is the mass flow rate of air though the filter, and k_{filter} is:

$$k_{filter} = \frac{\Delta p_{nom}}{\dot{m}_{nom}^2}, \tag{3}$$

where Δp_{nom} is the nominal pressure drop at the nominal mass flow rate, $\dot{m}_{nom}$. These two values are inputs to the filter model. The quadratic relation between static pressure drop and mass flow rate can be approximated using the Bernoulli equation and captures the general trend from experimental data (ASHRAE 2017). It should be noted that the filter pressure drop increases over time as the filter collects particles (Xia and Chen 2021), but the nominal pressure drop was assumed to be constant in this study for simplicity.

2.1.2 Viral Transmission Modeling

We use the building level concentration of COVID-19 virus to represent IAQ in the majority of this study. Sick people generate viral particles directly into each well-mixed zone at a constant generation rate. The balance of concentration in a zone can be described as:

$$\dot{c}_{zone} = (1/m_{air,zone}) \Sigma(\dot{m}c)_{zone} + \dot{c}_{gen,zone} - \dot{c}_{decay,zone}, \tag{4}$$

where $\dot{c}_{zone}$ is the rate of change of virus concentration in the zone with respect to time, $m_{air,zone}$ is the mass of air in the zone, $\Sigma(\dot{m}c)_{zone}$ is the net sum of the virus concentration flowrates into/out of the zone, $\dot{c}_{gen,zone}$ is the virus concentration generation rate within the zone, and $\dot{c}_{decay,zone}$ is the rate of viral decay in the zone, which is modeled based on a first order method:

$$\dot{c}_{decay,zone} = k_{decay} c_{zone}, \tag{5}$$

where k_{decay} is a defined constant rate of viral decay, and c_{zone} is the virus concentration in the zone.

The presence of one sick person in each zone within the building is simulated from 9:00 AM - 5:00 PM, Monday through Friday throughout the year. This allows for the evaluation of the mitigation strategies during different conditions, such as weather, throughout the year. The virus generation rate is dependent on many factors, such as the activity level of the sick person. We select a typical virus generation rate of 25 $quanta/hr$ (Buonanno, Stabile, and Morawska 2020; Buonanno, Morawska, and Stabile 2020) and a viral decay rate of 0.48 hr^{-1} (Pease, Wang, et al. 2021) based on data from the literature.

2.2 System Modeling

We provide an overview of the studied medium office building system and modeling of this system in this section.

2.2.1 Studied Building System

The building system in this work is based on the DOE commercial reference medium office building (Department of Energy n.d.), with a focus on the bottom floor. The schematic for this system is shown in Figure 1. The floor consists of five zones, including a core zone and four perimeter zones, with a total floor area of 1,664 m^2. A central air handling unit with heating and cooling coils services this floor, with VAV terminal boxes containing reheat coils for each zone. An outdoor air economizer is used to supply the minimum outdoor airflow based on ASHRAE standards (ASHRAE 2019) as well as provide free cooling. Natural gas is used to provide heating, while electricity is used to provide cooling and power the

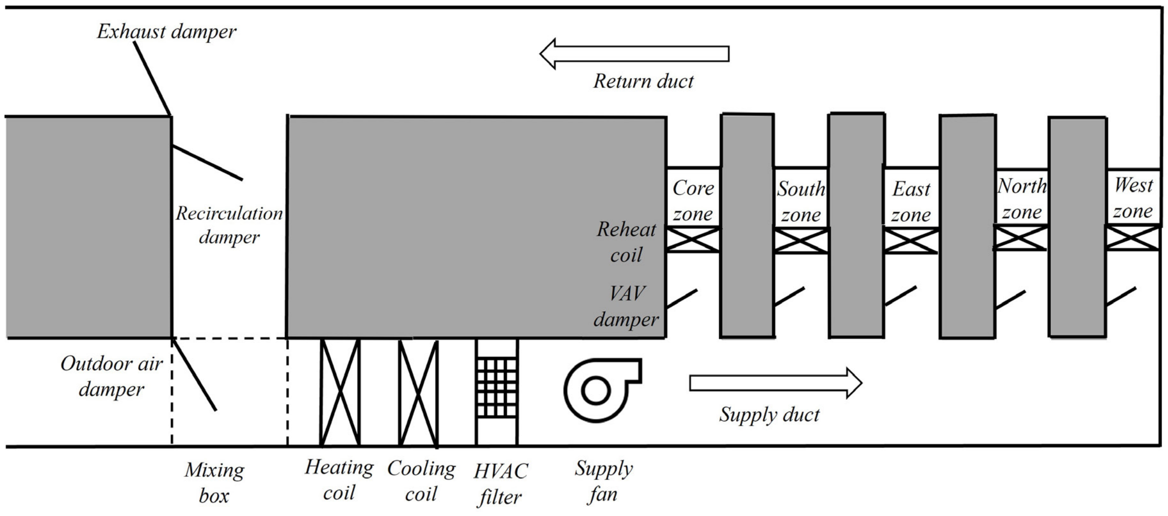


Figure 1. Schematic of VAV system servicing the bottom floor of the five zone medium office building.

fan. The HVAC system is controlled based on the VAV 2A2-21232 sequence from the Sequences of Operation for Common HVAC Systems described in (Wetter, Hu, et al. 2018).

2.2.2 Modeling of Studied System

The new component models are added to a medium office building system model, which is based along a prototype provided in the Modelica *Buildings* library (Lawrence Berkeley National Laboratory 2013), to create the final modeling capability. The developed medium office building system model for this study is shown in Figure 2. The HVAC system is sized for each climate using EnergyPlus^{TM} and the fan is assumed to be sized for MERV 10 filtration. We use typical meteorological year data for each location (EnergyPlus n.d.). The entire system model contains the following key subsystems: (1) building envelope and room airflow model, including the generation and decay of virus in the zones; (2) HVAC system model which includes the central air handling unit, as well as VAV terminal boxes and return duct; (3) control system, which includes PI controllers for the heating and cooling coils, outdoor air economizer, and supply fan; and (4) the weather conditions, including dry bulb temperature, wind speed, and radiative exchange.

For the system located in Tampa in this study, the model is adapted to supply air through the building at all times, including unoccupied hours to avoid development of mold due to the high humidity. The outdoor air damper is closed during unoccupied hours and only recirculated air is supplied to the building (including for the 100% outdoor air case). For cooling scenarios, the supply air temperature setpoint is reset from 12 °C to 27 °C and the zone temperature setpoints are reset from 24 °C to 30 °C in unoccupied hours. For heating scenarios, the zone temperature setpoints are reset from 20 °C to 12 °C in unoccupied hours. This allows for the system to run and prevent buildup of mold, while limiting the increase in energy during the unoccupied hours.

3 Methods to Compare Mitigation Strategies

This section describes the methods to compare the mitigation strategies in terms of IAQ and costs. This includes calculating the predicted number of infections based on virus concentration, as well as determining total costs based on costs associated with filters, HVAC energy consumption, and CO_2 emissions.

3.1 Predicted Number of Infections Calculation

To quantify the impact of the virus concentrations, risk of infection is calculated using the Wells-Riley approach (E. Riley, Murphy, and R. Riley 1978), which determines this risk based on the amount of virus inhaled by an occupant. Risk of infection is calculated as:

$$R(t) = 1 - \exp(-IR \int_{t_0}^{t} c(t)\,dt), \tag{6}$$

where $R(t)$ is risk of infection in terms of percentage, IR is the volumetric inhalation rate of air for an occupant, and $\int_{t_0}^{t} c(t)\,dt$ is the integral of virus concentration in the room with respect to time since initial time t_0. The predicted number of infections, R_0, can be calculated based on the risk, R. The predicted number of infections over time, $R_0(t)$ is calculated accounting for the variable occupancy in the zones for this study. This is done by calculating $R_0(t)$ for a given time interval when the occupancy is constant and adding the predicted number of infections calculated from the previous time interval. This can be described as:

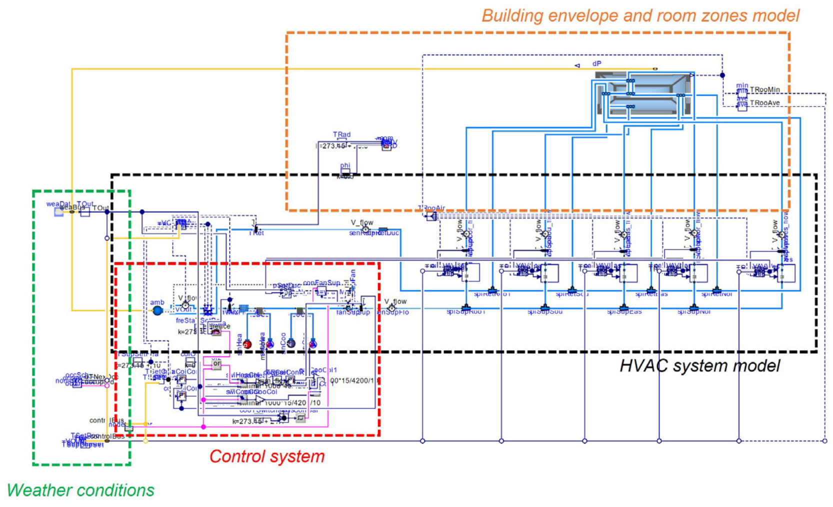


Figure 2. Modelica model of the medium office building system.

$$R_{0,T}(t) = S_T[1 - \exp(-IR\int_{t_0}^{t} c(t)\,dt)] + R_{0,T-1}(t_0), \quad (7)$$

where $R_{0,T}(t)$ is the predicted number of infections in the zone for time interval T, S_T is the number of susceptible occupants in the zone during T, t_0 is the time at the beginning of interval T, and $R_{0,T-1}(t_0)$ is the predicted number of infections from the previous time interval, $T-1$, ending at time t_0. Susceptible occupants is determined as $S = N - 1$, where N is the number of occupants. This way S does not account for the sick person, since they cannot infect themselves.

3.2 Financial Cost Calculation

The annual financial costs for the different mitigation strategies are calculated based on the following equation:

$$J_{total} = J_{filter} + J_{elec} + J_{gas} + J_{CO_2}, \quad (8)$$

where J_{total} is the total annual costs, J_{filter} are the costs associated with filtration, J_{elec} are the electricity costs to run the HVAC system, J_{gas} are the costs for natural gas heating, and J_{CO_2} are costs associated with CO_2 emissions. The costs associated with filtration include purchase costs and labor costs for replacing the filters throughout the year based on their expected life. The electricity costs to run the HVAC system come from fan and cooling power, while natural gas costs are calculated based on the heat supplied in the HVAC system from natural gas. Finally, we use a cost of \$12 (USD) per ton of CO_2 emissions based on average prices in the U.S. described by the Regional Greenhouse Gas Initiative and California Cap-and-Trade Program (The World Bank 2021) to determine costs associated with CO_2 emissions. It should be noted J_{filter}, J_{elec}, and J_{gas} are charged in current practice, but J_{CO_2} has not been implemented in the building sector in the United States yet.

3.3 CO_2 Emissions Calculation

The annual CO_2 emissions for the mitigation strategies are determined based on emissions associated with natural gas heating and electricity consumed by the HVAC system, using the method adopted in (Lou, Yang, et al. 2021; Lou, Ye, et al. 2022). The emission factor for natural gas heating is constant and independent of location. However, the emission factor for electricity is dynamic and depends on the electricity sources of the location. Different locations use various portions of renewable, nuclear, or fossil fuel energy. The electricity sources vary based on the time of day as well as season, for example depending on the availability of solar or wind energy. The emission factor data comes from the Cambium project lead by the National Renewable Energy Laboratory (Gagnon et al. 2020).

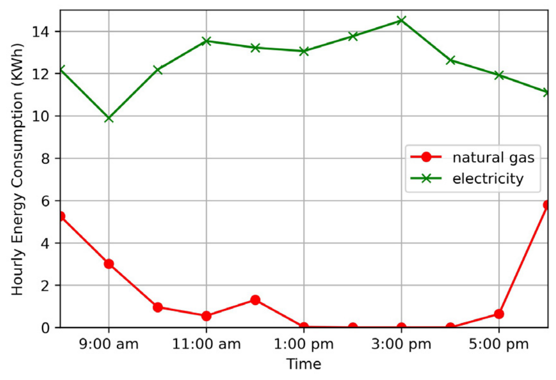
Figure 3 shows an example of how CO_2 emissions are calculated for a sample day based on the natural gas and electricity usage. Figure 3a shows the energy consumption for one heating day in San Diego. We see the natural gas usage varies based on the heating demand and the electricity consumption changes based on the fan power. The emission factor of electricity in Figure 3b varies during the day based on the availability of renewable energy, while the emission factor of natural gas remains constant. Finally, Figure 3c shows the hourly CO_2 emissions are the product of the hourly energy usage and emission factor.

3.4 Analysis of Combined Metrics

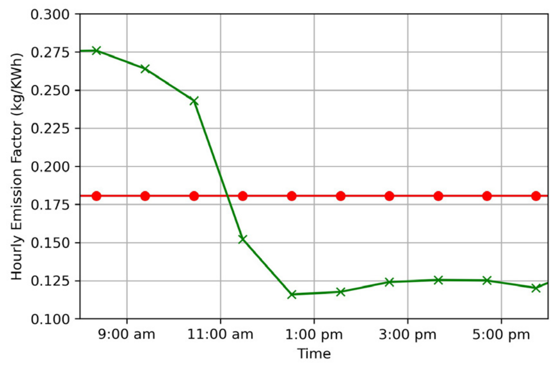
To evaluate the performance of the two strategies relative to MERV 10 filtration, we define metrics to consider the improvement in IAQ relative to an increase in costs. These are relative metrics, since they are calculated for the strategies relative to MERV 10 filtration. First, we calculate the percent increase in costs relative to MERV 10 filtration. This is described as:

$$\Delta J_i = J_i / J_{M10} - 1, \quad (9)$$

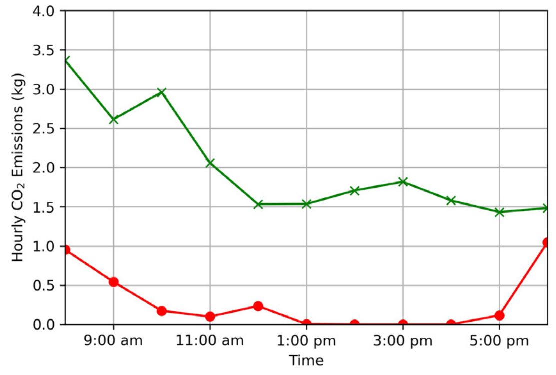
where ΔJ_i is the percent increase in costs associated with a strategy i relative to MERV 10 filtration, J_i is the costs for strategy i, and J_{M10} is the costs for MERV 10 filtration in that location.



(a) Hourly energy consumption.



(b) Hourly emission factors.



(c) Hourly CO_2 emissions.

Figure 3. Calculation of CO_2 emissions based on electricity and natural gas usage for Jan 1, 2020 in San Diego.

The percent improvement in IAQ relative to the percent increase in costs can then be calculated as:

$$\Delta IAQ / \Delta J_i = (1 - IAQ_i / IAQ_{M10}) / \Delta J_i, \quad (10)$$

where $\Delta IAQ / \Delta J_i$ is the marginal improvement in IAQ per increase in costs for a strategy i relative to MERV 10 filtration, IAQ_i is the IAQ metric for a strategy i, and IAQ_{M10} is the IAQ metric for the MERV 10 strategy.

4 Scope

We describe the scope of our analysis in this section, including the selected mitigation strategies and summary of the chosen geographic locations.

4.1 Mitigation Strategies

Three mitigation strategies are chosen for this study, including use of MERV 10 and MERV 13 filtration, or supply of 100% outdoor air into the building. MERV 10 filtration may be used in existing buildings, while improved MERV 13 filtration has been recommended for use during the COVID-19 pandemic by ASHRAE (ASHRAE Epidemic Task Force 2021). The 100% outdoor air strategy also uses MERV 10 filtration, since filtration is needed for outdoor contaminants as well. This study assumes the viral particles have diameters between 1-3 μm, and a constant, typical removal efficiency is chosen based on filter data for particles of this size. Table 1 shows the settings for the HVAC filters used in the simulations. The filtration efficiencies come from ASHRAE technical resources (ASHRAE 2017) and the pressure drop values come from data for MERV 10 (Dwyer n.d.[a]) and MERV 13 (Dwyer n.d.[b]). It should be noted the pressure drop across the filter can increase over time as the filter accumulates particles (Xia and Chen 2021) and the pressure drop can vary for filters with the same rating, depending on the depth or type of filter (Ben-David and Waring 2018). For simplicity, a constant nominal pressure drop for each filter is chosen based on the average of the typical initial and final pressure drops.

Table 1. HVAC filter simulation settings.

Filter	*Nominal Pressure Drop (Pa)*	*Filtration Efficiency*
MERV 10	143	50%
MERV 13	162	85%

The costs of the HVAC filters, which are obtained from (Azimi and Stephens 2013), are shown in Table 2. The total annual costs are determined by the purchase and labor costs throughout the year based on the expected life of the filters.

4.2 Studied Locations

International Falls, MN, San Diego, CA, and Tampa, FL are the locations studied in this paper. A summary of the climates, electricity sources, energy prices, and average electricity emission factor is shown in Table 3. The electricity (U.S. Energy Information Administration 2021b)

Table 2. HVAC filter costs.

Filter	*Purchase Cost (USD)*	*Replacement Labor Costs (USD)*	*Expected Life*	*Total Annual Costs*
MERV 10	$7	$17	4 months	$72
MERV 13	$11	$17	4 months	$84

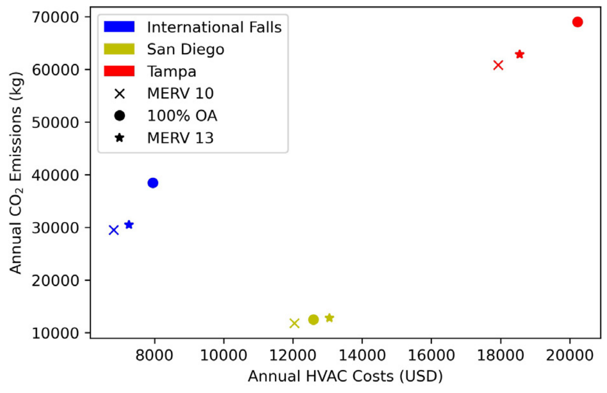
and natural gas (U.S. Energy Information Administration 2021a) prices for each location are also included. The natural gas price is based on the total price paid by end-users per thousand cubic feet of natural gas, and is inclusive of all taxes and other fees. Compared to International Falls and Tampa, San Diego has a lower average electricity emission factor. San Diego is able to utilize significant renewable energy, such as solar power, and limit its fossil fuel usage. International Falls and Tampa instead rely more on fossil fuels such as coal and natural gas for electricity generation.

5 Results and Discussion

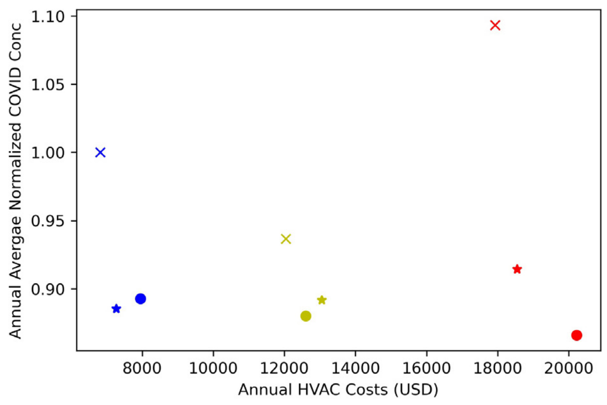
We first show an overview of the results for the three mitigation strategies in the three locations in terms of IAQ, HVAC costs, and CO_2 emissions. We then discuss the tradeoffs between IAQ and totals costs.

5.1 Overview of Results

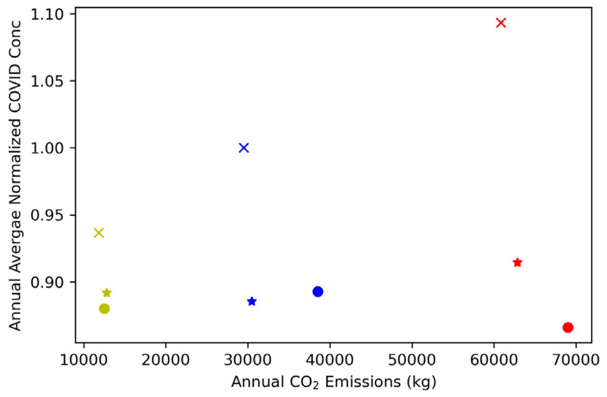
The annual results for IAQ, HVAC costs, and CO_2 emissions are shown in Figure 4. The virus concentrations are normalized by the annual average virus concentration for the MERV 10 case in International Falls. The HVAC costs in this section include costs associated with filters and HVAC energy consumption, while the total costs including those associated with CO_2 emissions are used in the next section. The results show dependencies on mitigation strategy, climate, and electricity sources. The trends for emissions and costs can be seen in Figure 4a. San Diego has lower costs and emissions compared to Tampa, due to less HVAC energy consumption in the milder climate. The breakdown of HVAC energy consumption for the three mitigation strategies in the three locations is shown in Figure 5. International Falls has lower costs but more emissions compared to San Diego. This is because natural gas heating is the dominant energy consumption in the very cold climate of International Falls, which has much lower costs compared to electricity. Due to its climate, San Diego uses very little heating and most of the HVAC energy consumption comes from electricity to provide cooling and power the fan. The lower emissions in San Diego compared to International Falls can be explained by the lower HVAC energy consumption, as well as the lower average electricity generation emission factor.



(a) Annual CO_2 emissions vs HVAC costs.



(b) Average virus concentration vs annual HVAC costs.

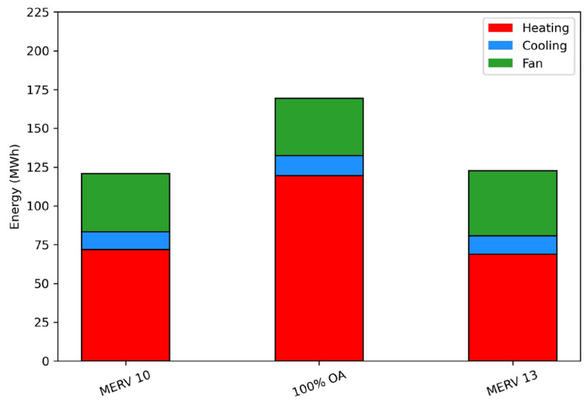


(c) Average virus concentration vs annual CO_2 emissions.

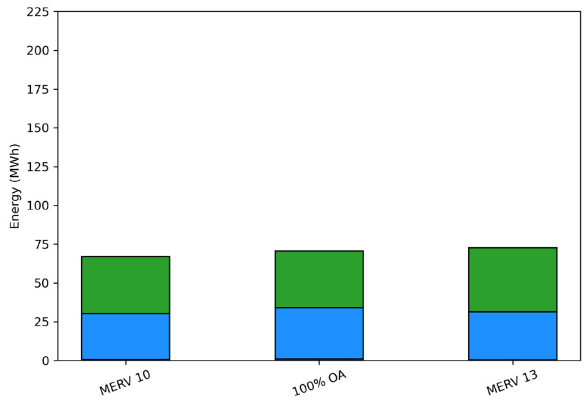
Figure 4. Results for average virus concentration, annual HVAC costs, and annual CO_2 emissions for the three mitigation strategies in the three locations.

Table 3. Summary of selected locations.

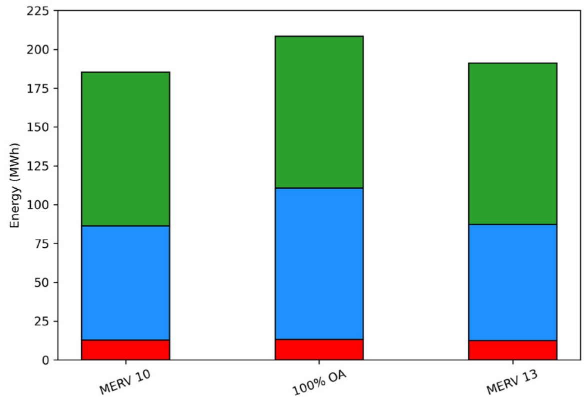
Location	*Climate*	*Electricity Price (cents/kWh)*	*Natural Gas Price (cents/kWh)*	*Avg. Electricity Emission Factor (kg CO_2/MWh)*
International Falls, MN	Very Cold	10.57	2.18	302
San Diego, CA	Warm and Marine	18.00	3.34	196
Tampa, FL	Hot and Humid	10.06	3.93	338



(a) International Falls.

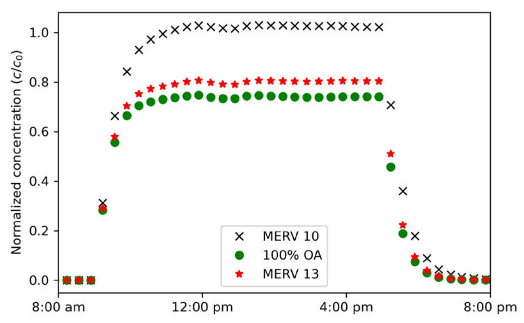


(b) San Diego.

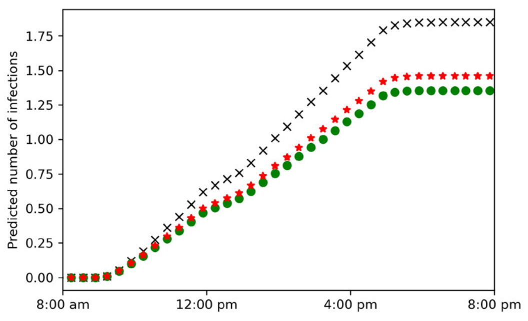


(c) Tampa.

Figure 5. Breakdown of HVAC energy consumption for the three strategies in the three locations.



(a) Virus concentration.



(b) Predicted number of infections.

Figure 6. Virus concentration and predicted number of infections in the Core zone for the three strategies on June 24, 2020 in Tampa.

Figures 4b and 4c similarly show the trends based on climate and electricity sources, as well as the IAQ trends for the different mitigation strategies. The 100% outdoor air strategy provides the best IAQ in Tampa and San Diego, but not in International Falls. This is because the economizer only decreases the outdoor air usage for the 100% outdoor air strategy when it is very cold outside to prevent freezing, which happens more often in the very cold climate of International Falls. The 100% outdoor air strategy leads to significant increases in costs and CO_2 emissions in International Falls and Tampa, but not as significantly in San Diego. This is because significant energy is required either to cool and dehumidify the outdoor air in the hot and humid Tampa climate or to heat the very cold outdoor air in International Falls. In San Diego, however, the weather is milder throughout the year, so the 100%

outdoor air strategy leads to smaller increases in costs and emissions. Figure 5 shows the increase in energy consumption for the 100% outdoor air case in Tampa and International Falls, as well as overall higher energy consumption in these two locations compared to San Diego because of climate.

To compare the impact of the differences in virus concentrations for these strategies, an example of the virus concentration and predicted number of infections in the Core zone for a hot summer day (June 24, 2020) in Tampa is shown in Figure 6. The minimum outdoor airflow is used for the MERV 10 and MERV 13 cases on this day. Use of MERV 13 filtration reduces the peak virus concentration on this day by 22% compared to MERV 10 filtration, and use of 100% outdoor air reduces the peak virus concentration by 27% compared to MERV 10 filtration. The predicted number of infections is above one for all three strategies in this zone during this day, which means at least one infection is expected to occur. MERV 13 filtration reduces the expected chance of a second infection occurring by 39% compared to MERV 10 filtration, while supply of 100% outdoor air decreases the expected chance of a second infection occurring by 50% compared to MERV 10 filtration.

MERV 10 filtration is the cheapest and lowest emission strategy due to having the lowest energy consumption, but also provides the worst IAQ in all locations. MERV 13 filtration improves the IAQ relative to MERV 10 filtration, but with moderate increases in costs and emissions because of the increase in fan energy consumption. It can be seen that the improvement in IAQ for the other strategies relative to MERV 10 filtration differs between the two locations. Additionally, the costs and emissions for the mitigation strategies also differ for these locations. Analysis of these tradeoffs is performed in the following section.

5.2 Analysis of Tradeoffs

The tradeoffs between IAQ and costs for the mitigation strategies relative to MERV 10 filtration are analyzed for the three locations in this section. Associating a cost with CO_2 emissions allows us to directly compare the marginal improvement in IAQ to both costs and emissions simultaneously, as described in Section 3.4. This is shown for the two strategies relative to MERV 10 in the three locations in Figure 7.

Use of 100% outdoor air outperforms MERV 13 filtration in San Diego. This is because supply of 100% outdoor air is able to provide better IAQ compared to MERV 13 with less of an increase in costs in this location. The milder weather in San Diego allows for limited increases in heating/cooling costs throughout the year, while the increase in fan energy for the MERV 13 case slightly increases the overall costs compared to 100% outdoor air. On the other hand, MERV 13 filtration appears to be the most beneficial strategy in International Falls and Tampa. Unlike San Diego, use of 100% outdoor air significantly increases the costs due to the energy required to heat or cool and dehumidify the outdoor air in these locations. MERV 13 filtration also shows a more significant improvement in IAQ in the two locations relative to San Diego due to the limited amount of outdoor air usage in those climates.

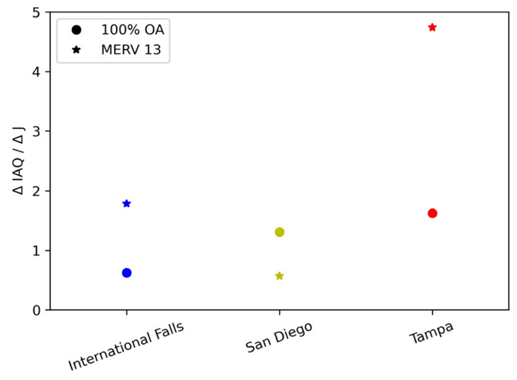
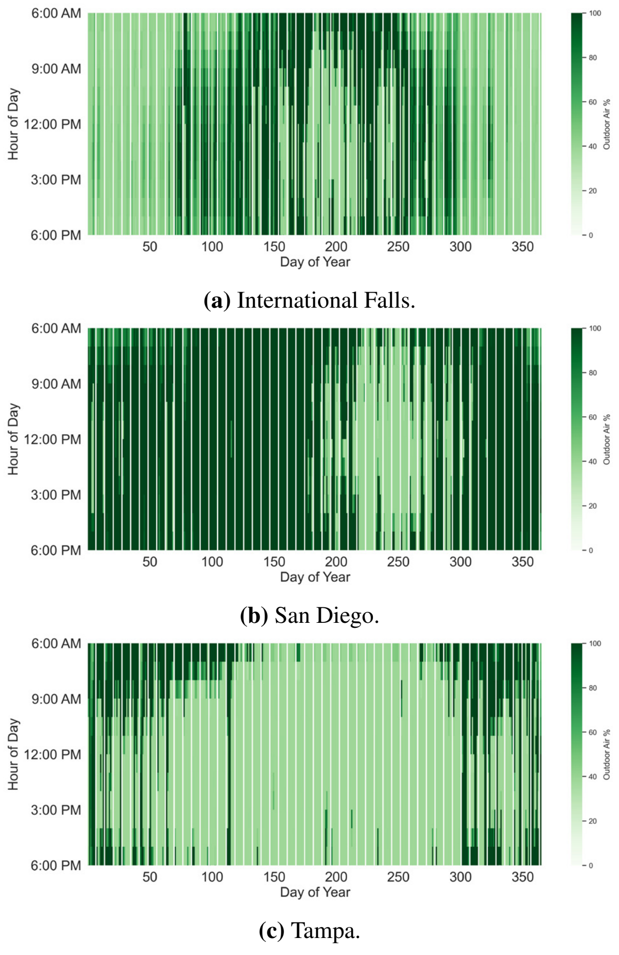


Figure 7. Marginal improvement in IAQ relative to total costs for the three locations.



(a) International Falls.

(b) San Diego.

(c) Tampa.

Figure 8. Dynamic usage of outdoor air using MERV 10 filtration in the three locations.

Figure 8 shows the outdoor air usage for the MERV 10 cases in the three locations. Because of its milder weather, the MERV 10 case in San Diego can use high outdoor airflow rates most of the year, except in the peak of summer in July - September. In International Falls, the MERV 10 case uses less outdoor air during the very cold winter, as well as during the peak of summer around July. In Tampa, much less outdoor air is used for the MERV 10 case throughout the year, with an exception during the cooler winter mornings. Because of these trends for the MERV 10 cases, the additional filtration in the MERV 13 case or outdoor air usage in the 100% outdoor air case significantly improves the IAQ in International Falls and Tampa.

6 Conclusion

The tradeoffs between IAQ and sustainability for three strategies to mitigate indoor virus are compared for three locations in the United States. The mitigation strategies include different levels of filtration, such as MERV 10 or MERV 13 filtration, as well as supply of 100% outdoor air into the building. The locations have differing climates and their electricity profiles are also comprised with varying portions of renewable energies and fossil fuels for generating electricity. The strategies are evaluated using a prototypical medium office building model initially sized for MERV 10 filtration, developed using the Modelica *Buildings* library.

The results show the tradeoffs between IAQ and costs for the different strategies have a strong dependency on climate and electricity sources. MERV 10 filtration is always the cheapest option, since this strategy tends to use the least energy, but also provides the worst IAQ. Use of 100% outdoor air provides the best IAQ in San Diego and Tampa, and significantly increases costs in the hot and humid climate of Tampa and very cold climate of International Falls. Use of 100% outdoor air can be a good option in the relatively milder climate of San Diego, where the increase in costs and emissions is limited. MERV 13 filtration can improve IAQ with limited increases in costs in all locations due to its high virus filtration efficiency and relatively smaller increases in energy consumption. This strategy outperforms use of 100% outdoor air in International Falls and Tampa, since it avoids the significant increase in cooling/dehumidification or heating of the outdoor air.

Future studies can be conducted based on the work in this paper. The models we used in this study can be applied to other contaminant scenarios, for example $PM_{2.5}$ which can infiltrate the building from outdoor air. They can also be used to evaluate advanced control strategies to improve IAQ, such as occupant-based strategies. Finally, this study focuses on applying mitigation strategies to an existing building, since redesigning an HVAC system is costly. However, the models can be used to evaluate HVAC system designs for new buildings, for example to study a system designed for high-efficiency filters.

Acknowledgements

This research was supported in part by the U.S. Defense Threat Reduction Agency and performed under U.S. Department of Energy Contract No. DE-AC02-05CH11231. 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. This research was supported by the National Science Foundation under Awards No. CBET-2217410.

References

ASHRAE (2017). “Standard 52.2. Method of Testing General Ventilation Air-Cleaning Devices for Removal Efficiency by Particle Size”. In: *American Society of Heating, Refrigerating and Air-Conditioning Engineers*.

ASHRAE (2019). “Standard 62.1 Ventilation for Acceptable Indoor Air Quality”. In: *American Society of Heating, Refrigerating and Air-Conditioning Engineers*.

ASHRAE Epidemic Task Force (2021). “Core Recommendations for Reducing Airborne Infectious Aerosol Exposure”. In: *ASHRAE*.

Aviv, Dorit et al. (2021). “A fresh (air) look at ventilation for COVID-19: Estimating the global energy savings potential of coupling natural ventilation with novel radiant cooling strategies”. In: *Applied Energy* 292, p. 116848.

Azimi, Parham and Brent Stephens (2013). “HVAC filtration for controlling infectious airborne disease transmission in indoor environments: predicting risk reductions and operational costs”. In: *Building and environment* 70, pp. 150–160.

Ben-David, Tom and Michael S Waring (2018). “Interplay of ventilation and filtration: Differential analysis of cost function combining energy use and indoor exposure to $PM_{2.5}$ and ozone”. In: *Building and Environment* 128, pp. 320–335.

Buonanno, Giorgio, Lidia Morawska, and Luca Stabile (2020). “Quantitative assessment of the risk of airborne transmission of SARS-CoV-2 infection: prospective and retrospective applications”. In: *Environment international* 145, p. 106112.

Buonanno, Giorgio, Luca Stabile, and Lidia Morawska (2020). “Estimation of airborne viral emission: Quanta emission rate of SARS-CoV-2 for infection risk assessment”. In: *Environment international* 141, p. 105794.

Cortiços, Nuno D and Carlos C Duarte (2021). “COVID-19: The impact in US high-rise office buildings energy efficiency”. In: *Energy and Buildings* 249, p. 111180.

Department of Energy (n.d.). *Commercial Reference Buildings*. https://www.energy.gov/eere/buildings/commercial-reference-buildings.

Dwyer (n.d.[a]). *MERV 10 Pleated Filters*. https://www.dwyer-inst.com/PDF_files/Priced/DF10_cat.pdf.

Dwyer (n.d.[b]). *MERV 13 Pleated Filters*. https://www.dwyer-inst.com/PDF_files/Priced/DF13_cat.pdf.

EnergyPlus (n.d.). *Weather Data*. https://energyplus.net/weather.

Faulkner, Cary A et al. (2022). “Investigation of HVAC operation strategies for office buildings during COVID-19 pandemic”. In: *Building and Environment* 207, p. 108519.

Gagnon, Pieter et al. (2020). *Cambium data for 2020 Standard Scenarios*. National Renewable Energy Laboratory. https://cambium.nrel.gov/.

Lawrence Berkeley National Laboratory (2013). *Buildings.Examples.VAVReheat*. https://simulationresearch.lbl.gov/modelica/releases/v5.0.0/help/Buildings_Examples_VAVReheat.html#Buildings.Examples.VAVReheat.

Lou, Yingli, Yizhi Yang, et al. (2021). "The effect of building retrofit measures on CO2 emission reduction–A case study with US medium office buildings". In: *Energy and Buildings* 253, p. 111514.

Lou, Yingli, Yunyang Ye, et al. (2022). "Long-term carbon emission reduction potential of building retrofits with dynamically changing electricity emission factors". In: *Building and Environment* 210, p. 108683.

Pease, Leonard F, Timothy I Salsbury, et al. (2022). "Size dependent infectivity of SARS-CoV-2 via respiratory droplets spread through central ventilation systems". In: *International Communications in Heat and Mass Transfer* 132, p. 105748.

Pease, Leonard F, Na Wang, et al. (2021). "Investigation of potential aerosol transmission and infectivity of SARS-CoV-2 through central ventilation systems". In: *Building and Environment* 197, p. 107633.

Riley, EC, G Murphy, and RL Riley (1978). "Airborne spread of measles in a suburban elementary school". In: *American journal of epidemiology* 107.5, pp. 421–432.

Santos, Hugo RR and Vıtor MS Leal (2012). "Energy vs. ventilation rate in buildings: A comprehensive scenario-based assessment in the European context". In: *Energy and Buildings* 54, pp. 111–121.

Schibuola, Luigi and Chiara Tambani (2021). "High energy efficiency ventilation to limit COVID-19 contagion in school environments". In: *Energy and Buildings* 240, p. 110882.

Shen, Jialei et al. (2021). "A systematic approach to estimating the effectiveness of multi-scale IAQ strategies for reducing the risk of airborne infection of SARS-CoV-2". In: *Building and environment* 200, p. 107926.

The World Bank (2021). *Carbon Pricing Dashboard*. https://carbonpricingdashboard.worldbank.org/map_data.

U.S. Energy Information Administration (2021a). *Natural Gas Prices*. https://www.eia.gov/dnav/ng/ng_pri_sum_a_EPG0_PCS_DMcf_a.htm.

U.S. Energy Information Administration (2021b). *State Electricity Profiles*. https://www.eia.gov/electricity/state/.

Vlachokostas, Alex et al. (2022). "Experimental evaluation of respiratory droplet spread to rooms connected by a central ventilation system". In: *Indoor air*.

Wetter, Michael, Marco Bonvini, et al. (2015). "Modelica buildings library 2.0". In: *Proc. of The 14th International Conference of the International Building Performance Simulation Association (Building Simulation 2015), Hyderabad, India*.

Wetter, Michael, Jianjun Hu, et al. (2018). "OpenBuildingControl: Modeling feedback control as a step towards formal design, specification, deployment and verification of building control sequences". In: *Building Performance Modeling Conference and SimBuild*.

Wetter, Michael, Wangda Zuo, et al. (2014). "Modelica buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270.

Xia, Tongling and Chun Chen (2021). "Evolution of pressure drop across electrospun nanofiber filters clogged by solid particles and its influence on indoor particulate air pollution control". In: *Journal of Hazardous Materials* 402, p. 123479.

Guidelines and Use Cases for Power Systems Dynamic Modeling and Model Verification using Modelica and OpenIPSL

Giuseppe Laera[1] Luigi Vanfretti[1] Marcelo de Castro Fernandes[1] Sergio A. Dorado-Rojas[1] Fernando Fachini[1] Chetan Mishra[2] Kevin D. Jones[2] R. Matthew Gardner[2] Hubertus Tummescheit[3] Stéphane Velut[3] Ricardo J. Galarza[4]

[1]ECSE, Rensselaer Polytechnic Institute, Troy (NY), USA,
{laerag,vanfrl,decasm3,dorads,fachif}@rpi.edu
[2]Dominion Energy, Richmond (VA), USA,
{chetan.mishra,kevin.d.jones,matthew.gardner}@dominionenergy.com
[3]Modelon, Glastonbury (CT), USA & Lund, Sweden,
{hubertus.tummescheit,stephane.velut}@modelon.com
[4]PSM Consulting, Inc., Guilderland (NY), USA, rgalarza@psm-consulting.com

Abstract

This paper offers systematic guidelines for modeling power systems components in the phasor time-domain using the Modelica language and their verification. It aims to share the authors' experience in power system modeling with Modelica and the approaches used to meet the high expectations of the power industry w.r.t. to the models' simulation results. While the modeling guidelines are generic, the verification procedure includes the validation against a domain-specific commercial software tool called PSS®E that is the *de facto* tool used for power system transmission planning and analysis. To formalize the proposed approaches, a schematic description of the processes of model implementation and validation is elicited through flowcharts. Challenging use cases are presented to point out some of the major difficulties that can be faced in the modeling steps because of unclear or missing documentation of the models' dynamics in the reference tool. Finally, unique features of the Modelica language that allow for power system modeling and verification unavailable in traditional tools are illustrated.

Keywords: Modelica, OpenIPSL, PSS®E, Dymola, Modelon Impact, SystemModeler, OpenModelica

1 Introduction

This paper aims to formalize the process of power systems dynamic modeling and model verification using the Modelica language. The mail goal is to provide, for the first time, a formal description of the steps necessary for complete re-implementation of power systems components of closed-source commercial software like PSS®E in the Modelica language. A generic methodology for re-implementing existing models from different software tools and domains can be derived from the proposed steps. Key use cases that have been challenging to implement and verify are presented to highlight the value of the proposed approaches for model implementation and validation. Moreover, it is shown how the self-documenting nature of object-oriented equation-based modeling offered by Modelica provides unique advantages for human and computer readable model implementation, as compared to existing modeling tools that do not always offer transparency regarding the dynamic behavior implemented in their tools. In this collection of examples, models for the OpenIPSL library[1] have been considered and validated against PSS®E. Finally, we emphasize the value of the open-access standardized Modelica specification as a key enabler of open-access standards-based interoperability (Gómez et al. 2020), showing how the newly implemented models can be re-utilized in multiple Modelica-standard-compliant tools without a need for re-implementation.

The reminder of this paper is organized in four sections: *Introduction* describing some motivations and contributions, *Guidelines* formalizing in flowcharts the process of models implementation and validation, *Use Cases* illustrating the proposed approaches with examples of implemented models and *Future Work and Conclusions* with some final comments.

1.1 Motivations

Modeling of power systems has always been fundamental for the design, operation and planning of electric networks. To help all the players of the electric power systems sector to perform their studies and analyses, over the last decades several software tools have been developed. The *de facto* tools used by industry include proprietary software like PSS®E, PSCAD, EMTP-RV, PowerFactory, etc., that require the user to become an expert of their functionalities and poses intimate tool-and-domain specific knowledge to be productive. In addition, each tool has its own way of defining the data used to characterize their discretized model (i.e. "data format") mak-

[1]https://github.com/OpenIPSL/OpenIPSL

ing it inflexible when attempting to share models and data between tools (Hongesombut et al. 2005). Another drawback is represented by the inconsistency of dynamic simulation results between different simulation platforms. This is due to the need to re-implement models in each simulation platform, which has tremendous costs. As reported by the Australian Energy Market Commission [2], the order of magnitude costs for model implementation in each individual tool can reach almost \$500,000.00 in the case of existing components with power electronic interfaces, such as wind and solar, thus making it challenging to study the effects of the integration of renewable energy resources. Therefore, the idea to remove ambiguity in power systems modeling was suggested in (Vanfretti et al. 2013) by using the object-oriented equation-based modeling language Modelica (Tiller 2001). Following this innovative idea a power systems library called OpenIPSL has been developed (Baudette et al. 2018). The library continues to be maintained and expanded based on the concepts of regression testing and Continuous Integration (CI) (Rabuzin, Baudette, and Vanfretti 2017), that allow the models to be verified against a traditional commercial software tool like PSS®E giving the user of the power systems community the confidence about its reliability for performing dynamic simulations and studies.

1.2 Previous Works

The proposed process for testing newly implemented models in Modelica and their verification make use of a Single Machine Infinite Bus (SMIB) equivalent system model (Zhang et al. 2015). This model is typically used to test the implementation of new models and designs of control strategies, which has been emphasized in the literature (Chaudhary and Singh 2014; Kumar 2018; Jayapal and Mendiratta 2010; Wang et al. 2015). This small network is useful in describing the key behavior of a power plant within a power system for most practical purposes. It offers a good framework for studying basic power systems stability concepts and the application of different control techniques to understand their effects on the network.

Regarding the development of new open-source software for power systems studies several examples based on different languages have been reported in the literature. During the 1990's and 2000's, MATLAB saw an exponential adoption in academia, resulting in a number of power system simulation software, such as *MATPOWER* (Zimmerman, Murillo-Sánchez, and Thomas 2010), focusing on steady state computations and the Power System Analysis Toolbox that offers an array of analysis types, both steady state or dynamic (F. Milano 2005). With the rise of Python, a new generation of Python-based tools for modeling, analysis and optimization of electric power systems have been developed. Among these, *pandapower* (Thurner et al. 2018) which aids with steady state computations. *GridCal*[3] is another platform for power systems research and simulation based on Python, again, focusing on steady state computations. Meanwhile, another Python-based tool for power system simulation is *ANDES* (Cui, Li, and Tomsovic 2020). It is an open-source Python library for power system modeling, computation, analysis, and control and it uses a hybrid symbolic-numeric framework for numerical analysis. What these software have in common is that they define both their models in discretized form, interlinking a specific numerical solver (e.g. Newton solver for steady state analysis or trapezoidal integration for dynamic simulation) during implementation.

In recent years, the Julia language has been gaining popularity by the modeling and simulation community (Elmqvist, Henningsson, and Otter 2016). Naturally, Julia packages for power system modeling and simulation of power systems have also emerged, with *PowerSimulationsDynamics.jl* for power system dynamics and for power systems operations called *PowerSimulations.jl* (Henriquez-Auba et al. 2021). While these packages do require the user to specify their models in discretized form and target a specific solver, as in the case of the MATLAB and Python-based tools described above, they do require the user to specify models with pre-defined data structures and the resulting models can only be used with the solvers available within the Julia ecosystem (Henriquez-Auba et al. 2021), not to mention the models have not been validated against any other reference software tool.

Regardless, as each of the afromentioned tools define their own approach for model implementation, the means to define parameter data and support only specific solvers, the results obtained will be different.

In contrast, Modelica facilitates the re-use of models among different Modelica-compliant tools by defining interoperable libraries, with the models of each component implemented separately and without the need of a numerical solver. There are several power system analysis libraries based on Modelica, (Winkler 2017) gives an overview of all available open-source libraries for power system dynamic analysis and highlights the advantages and disadvantages of each library. Note that most libraries listed are for positive sequence phasor-based dynamic simulation. In addition, it is worth mentioning *MSEMT*: an advanced Modelica library for power system electromagnetic transient studies (Masoom et al. 2021). This paper enhances and expands the OpenIPSL library, which focuses on power system models in the phasor-domain that have been validated against PSS®E. In that regard, some recent examples of models validation for renewable energy sources and battery energy storage systems are given in (Fachini et al. 2021).

[2]Online: `https://tinyurl.com/aemc-rule2017`

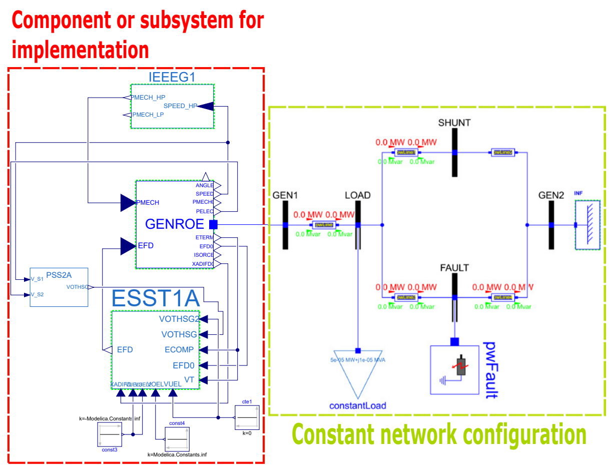
[3]`https://github.com/SanPen/GridCal`

1.3 Contributions

The use of the object-oriented equation-based modeling language, Modelica, allows for unambiguous power systems modeling. The flexibility of the language offers options to model a component, either by using typical block diagram representations or by defining mathematical equations that describe its dynamic behavior. The classical approach to validate a component's dynamics is by creating a small reference power network model, like an SMIB as used herein. As shown in Section 3, using the Modelica language it is also possible to validate each individual sub-system component within a model by using the reference tool's output signals and connecting them to the inputs of the newly implemented model. This simplifies the modeling process as it allows to isolate the specific individual part of the model that is being implemented in Modelica, instead of having to use it in a small reference power network as it is necessary for the traditional software tools to perform simulations.

Explicitly, the contributions of this paper are as follows:

- To formalize the model implementation approach used for component model development in Modelica to meet the requirements of the power industry.
- To formalize the software-to-software model validation process for Modelica model validation against reference domain-specific tools, illustrated with the *de facto* standard in the power industry, PSS®E software.
- To propose an approach to validate sub-system model components without the need to define entire system models in Modelica by replaying the reference tool simulation results into the Modelica model.
- To illustrate the proposed approach with challenging implementation and validation use cases.
- Synthesizing the know-how described above into guidelines which fully elicit the model implementation and validation process using flowcharts.



Component or subsystem for implementation
IEEEG1
GENROE
PSS2A
ESST1A
GEN1
LOAD
SHUNT
FAULT
GEN2
constantLoad
pwFault
Constant network configuration

Figure 1. SMIB template in Modelica.

2 Guidelines

In this section the approach of modeling power systems components in Modelica is described.

2.1 Template Models for Modeling and Validation

A basic example of electric power system network is the SMIB. Its block diagram in Modelica is given in Figure 1. This system is used to model a power plant with its controls connected to the rest of the grid through transmission lines and substations represented by buses. This small network also defines the model to standardize the testing of different device models implemented in Modelica and having PSS®E models as reference.

In Figure 1, in the red block, a complete power plant is modeled with a generator, connected to bus GEN1, and its controls. The grid is represented by the generator connected to bus GEN2. Between GEN1 and GEN2 other power systems components are considered like lines, a load and a ground fault. The bus SHUNT also allows to connect other components to this small network. Depending on the type of power systems component to model, the configuration of the generating unit (red block) connected to bus GEN1 varies whereas the configuration of the rest of the network (green block) remains the same.

In the sequel, components in the red block of Figure 1 will be implemented and validated. Note that for all the tests, the remainder of the power system in the green block of Figure 1 remains unchanged.

2.2 Model Implementation Guide

The modeling implementation process is defined in Figures 2 and 3. While the approach is generic, the process depicted considers PSS®E the reference software tool. In Figure 2 the process of implementation starts with the assignment of a model to implement (a). The identification of available technical information (b) about the model is necessary to the model implementation in Modelica. If the PSS®E manuals do not present sufficient information about the model's dynamics then it is necessary to find additional literature (c) that can help understanding how the models were implemented within the reference tool. Once the documents describing the model have been identified, collected and analyzed by finding the block diagrams and/or the equations of the model (d), it is possible to determine if the building blocks of the model are already present in the OpenIPSL library (e). It might be required to implement missing blocks or functions (f) before building the entire component model with the appropriate initialization of its sub-blocks (g). To assess the validity of the model to implement, a small test network (SMIB) is used in both Modelica and the reference software, and generating the reference results from PSS®E (h). That means the SMIB network with the target model needs to be assembled both in PSS®E (i) and Modelica (l). After that the software-to-software validation can be performed

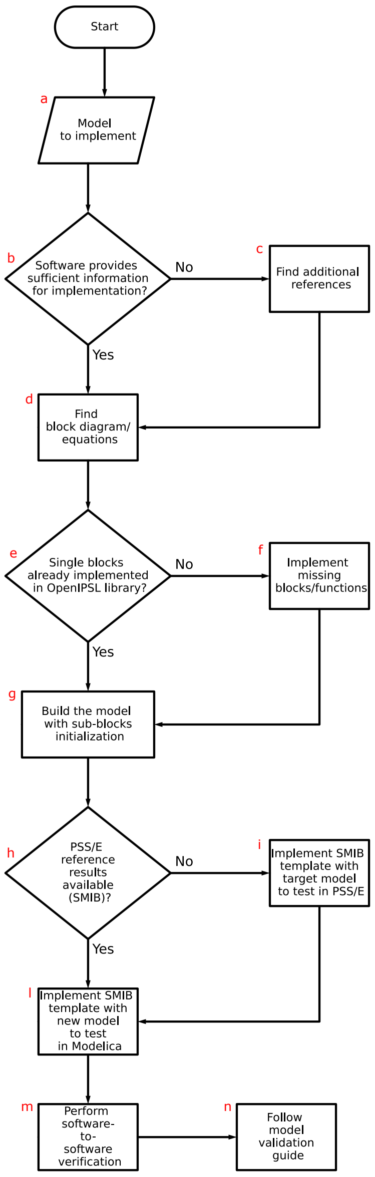

Start
a
Model to implement
b
Software provides sufficient information for implementation?
No
c
Find additional references
Yes
d
Find block diagram/ equations
e
Single blocks already implemented in OpenIPSL library?
No
f
Implement missing blocks/functions
Yes
g
Build the model with sub-blocks initialization
h
PSS/E reference results available (SMIB)?
No
i
Implement SMIB template with target model to test in PSS/E
Yes
l
Implement SMIB template with new model to test in Modelica
m
Perform software-to-software verification
n
Follow model validation guide

Figure 2. Flowchart of the process of implementation of power systems models.

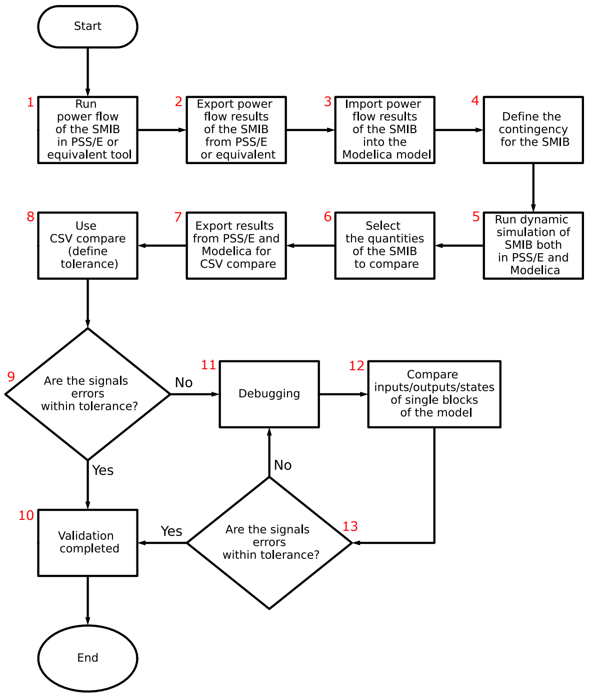

Start
1
Run power flow of the SMIB in PSS/E or equivalent tool
2
Export power flow results of the SMIB from PSS/E or equivalent
3
Import power flow results of the SMIB into the Modelica model
4
Define the contingency for the SMIB
5
Run dynamic simulation of SMIB both in PSS/E and Modelica
6
Select the quantities of the SMIB to compare
7
Export results from PSS/E and Modelica for CSV compare
8
Use CSV compare (define tolerance)
9
Are the signals errors within tolerance?
No
11
Debugging
12
Compare inputs/outputs/states of single blocks of the model
Yes
No
10
Validation completed
Yes
13
Are the signals errors within tolerance?
End

Figure 3. Flowchart of the process of software-to-software validation of power systems models.

(m) following the model validation guide (n) (see Figure 3).

2.3 Model Validation Guide

The process of model validation is defined through the flowchart in Figure 3 and it comes after the process of Figure 2. After assembling the SMIB in PSS®E it is necessary to obtain the steady state computation results of a "power flow" (1), export them (2) and provide them as initial guess values to solve the initialization problem of the corresponding SMIB in the Modelica compliant software tool (3). The OpenIPSL library used in this validation process describes the dynamic behavior of power system components therefore it relies on external tools for the power flow calculations necessary for initializing the models. The import of the power flow results can be made manually or automatically (Dorado-Rojas et al. 2021), the latter increasing numerical accuracy and reducing human errors. Next, the scenario for the dynamic simulation in both tools can be defined (4) and a dynamic simulation of the SMIB in both softwares can be run (5). Once the results are generated the quantities to compare can be chosen (6) and exported in the appropriate format (7) to be used in another tool, for example CSV Compare[4] (8). Tools like CSV Compare allow to quantify the discrepancies between the simulation software tools after defining an acceptable tolerance level (see Figure 4). If the errors between the quantities to compare are within the tolerance band (9) then the validation is complete (10). If the er-

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

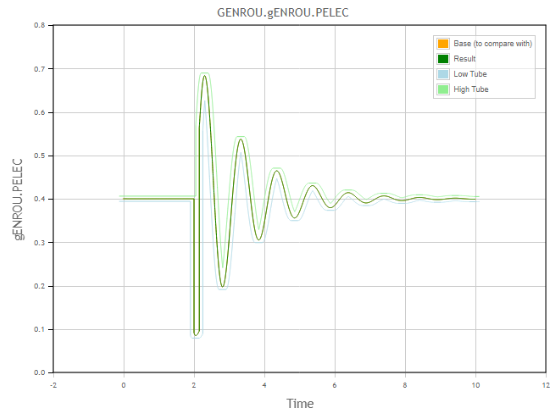


Figure 4. Example of the use of the CSV Compare tool.

rors are bigger than the defined tolerance then more debugging (11) of the model is required. A better insight of the model dynamics can be obtained by comparing subcomponent inputs, outputs and states of the implemented model (an example is given in Section 3) with the analogous signals from the SMIB in PSS®E (12). The iterative process continues until the difference between signals is lower than the tolerance (13), meaning that the validation is completed. This process can be part of a continuous integration and regression methodology, such as the one described in (Baudette et al. 2018).

3 Use Cases

In this section some key use cases of the implementation of power system components are described following the steps in Section 2. The verification illustrated through the plots of some quantities includes a software-to-software validation between PSS®E and different Modelica compliant platforms.

3.1 Power System Stabilizer PSS2A model

3.1.1 Implementation

An example of the difficulties faced when implementing a standard power system model like those of power system stabilizers (PSSs) PSS2A and PSS2B is given in this section. The reference software tool PSS®E comes with several manuals to help understanding the implementation and behavior of the different components present in its libraries. In some cases, like for the aforementioned PSSs models, the documentation is insufficient and not helpful to understand the behavior of one of the blocks of the models. In particular, this block is the ramp tracking filter highlighted in Figures 5 and 6.

An initial implementation of the ramp tracking filter in Modelica revealed to be not accurate compared to PSS®E implementation. This result led to additional investigations about the ramp tracking filter block. Because the PSS®E documentation does not offer more details, then the idea of analysing the continuous time trajectories of the states of this block was used. This idea was derived

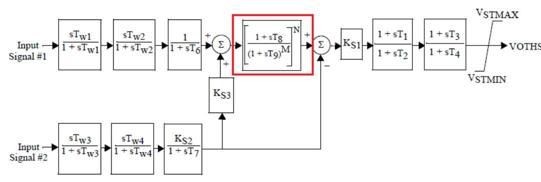


Figure 5. Block diagram of PSS2A from PSS®E manual (Siemens Industry 2013).

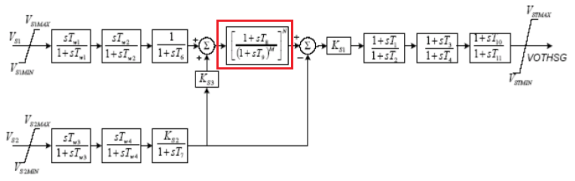


Figure 6. Block diagram of PSS2B from PSS®E manual (Siemens Industry 2013).

considering the available information, that needs to be appropriately selected, included in the simulation results of a SMIB test system in PSS®E with one of the PSSs. So from PSS®E it is possible to analyze the dynamics of the ramp tracking filter block through its input, output and states. A visual illustration of the implemented corrections when the filter model was debugged is given in Figure 7.

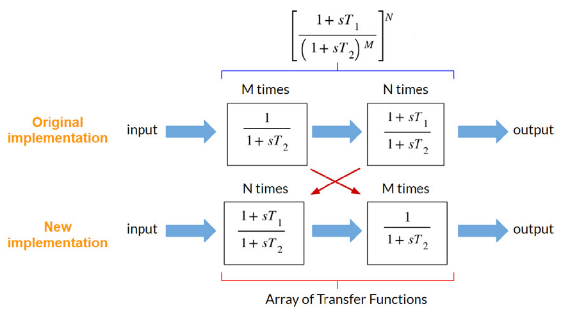


Figure 7. Original and new block diagram representation of the ramp tracking filter for the Modelica implementation.

The Modelica code for the ramp tracking filter model is given in Listing 1. With this kind of implementation several features of the Modelica language have been used. The language is object-oriented and, in this case, it allows for the creation of arrays of transfer functions to build the model. This implementation is transparent and self-documented without leaving any uncertainty about the dynamics of the component as opposed to the *de facto* standard proprietary tools of the power systems domain that are not always well documented, or as in this case, the documentation of the software was erroneous. In addition to that, the level of abstraction and portability with the Modelica language is superior.

```
model RampTrackingFilter "Ramp-tracking filter"
  extends Modelica.Blocks.Interfaces.SISO;
  import Modelica.Blocks.Continuous;
  parameter Real T_1;
  parameter Real T_2;
  parameter Integer M = 5 ">=0, M*N<=8";
  parameter Integer N = 1 ">=0, M*N<=8";
  parameter Real y_start = 0 "Output start value";
  final parameter Boolean bypass = if M == 0 or N == 0 then true
      else false "Boolean parameter" annotation(Evaluate =
      true);
  Continuous.TransferFunction TF1[M](b=fill({1},M), a=fill({T_2
      ,1},M),each y_start=y_start) if bypass == false "
      Conditional component";
  Continuous.TransferFunction TF2[N](b=fill({T_1,1},N), a=fill({
      T_2,1},N), each y_start=y_start) if bypass == false "
      Conditional component";
equation
  if M == 0 or N == 0 then
    u = y;
  elseif M == 1 then
    connect(u, TF2[1].u);
    for i in 1:N-1 loop
      connect(TF2[i].y, TF2[i+1].u);
    end for;
    connect(TF2[N].y, TF1[1].u);
    connect(TF1[1].y, y);
  elseif N == 1 then
    connect(u, TF2[1].u);
    connect(TF2[1].y, TF1[1].u);
    for i in 1:M-2 loop
      connect(TF1[i].y, TF1[i+1].u);
    end for;
    connect(TF1[M-1].y, TF1[M].u);
    connect(TF1[M].y, y);
  elseif M == 1 and N ==1 then
    connect(u, TF2[1].u);
    connect(TF2[1].y, TF1[1].u);
    connect(TF1[1].y, y);
  else
    connect(u, TF2[1].u);
    for i in 1:N-1 loop
      connect(TF2[i].y, TF2[i+1].u);
    end for;
    connect(TF2[N].y, TF1[1].u);
    for i in 1:M-2 loop
      connect(TF1[i].y, TF1[i+1].u);
    end for;
    connect(TF1[M-1].y, TF1[M].u);
    connect(TF1[M].y, y);
  end if;
end RampTrackingFilter;
```

Listing 1. Modelica code for *Ramp Tracking Filter* model

The implementation of the ramp tracking filter has been tested first with the component alone as in the system of Figure 8. A ramp signal has been applied as input and the following parameters: $T_1 = 0.5$, $T_2 = 0.1$, $M = 5$, $N = 1$ for the filter have been used. The choice of the parameters of the ramp tracking filter has been derived from the reference example of Figure 11. By varying those parameters it is possible to obtain the desired ramp tracking behavior (Bérubé and Hajagos 2007; Berube, Hajagos, and Beaulieu 1999).

The results of the test in Figure 8 are given in Figure 9.

3.1.2 Validation

The new implementation of the ramp tracking filter has been introduced in the PSS2A component model. The Modelica model of PSS2A corresponding to the one in Figure 5 is given in Figure 10.

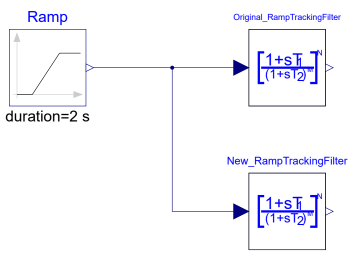


Figure 8. Ramp tracking filter test with the original and final version (see Figure 7).

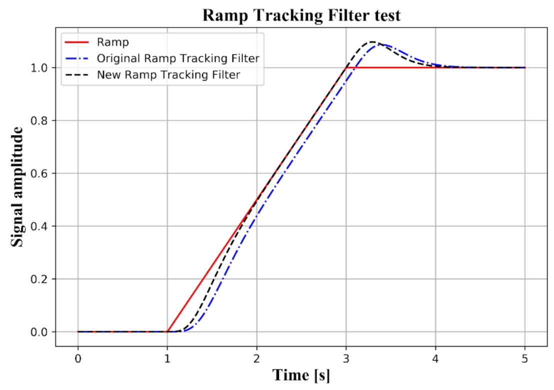


Figure 9. Results of the ramp tracking filter test: reference ramp signal (red), original Modelica implementation of ramp tracking filter (blue) and final new implementation of ramp tracking filter (black) (see Figure 7).

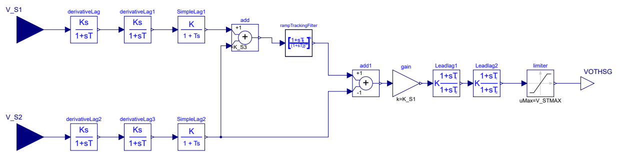


Figure 10. Block diagram of PSS2A in Modelica.

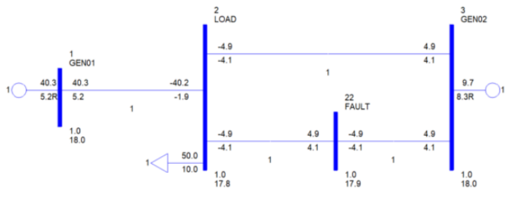


Figure 11. SMIB test system for PSS2A in PSS®E.

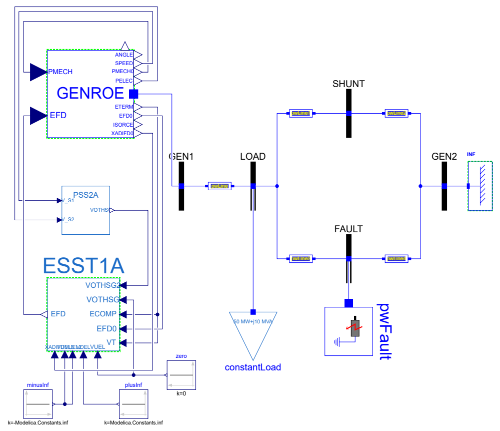


Figure 12. SMIB test system for PSS2A in Modelica.

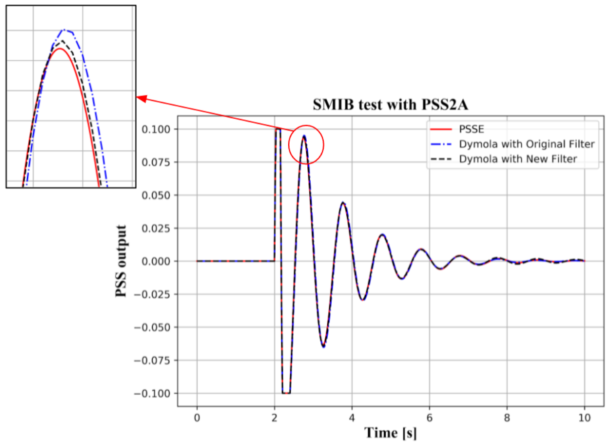


Figure 13. Results comparison between the Modelica SMIB with the original (see Figure 7) and final implementation (see Listing 1) of the ramp tracking filter and the SMIB from PSS®E.

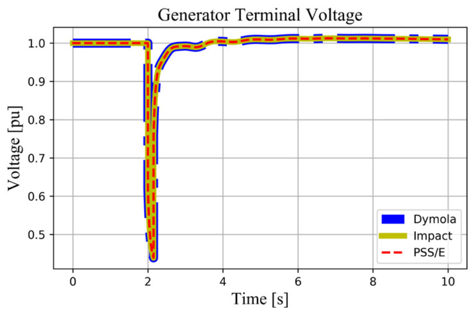


Figure 14. Generator terminal voltage at bus GEN1 of the system in Figure 12.

The component PSS2A has then been tested in a small network like a SMIB. The PSS®E benchmark system is illustrated in Figure 11.

The corresponding SMIB implementation in Modelica is given in Figure 12.

The tested scenario for the SMIB consists of a 3 phase fault to ground applied at bus FAULT at $t = 2s$ for $0.15s$. The results are plotted in Figure 13.

A multi-platform software-to-software validation of PSS2A using the same SMIB system in Figure 12 with the same scenario is given in Figures 14, 15, 16 and 17. The simulations in the different tools have been performed with the same tolerance (1e-06) and the same output interval length ($0.001s$).

Another feature of the Modelica language is the possibility to implement models in the traditional way using block diagrams, like PSS2A, or with coding, like the ramp tracking filter. The graphic layer and the text layer of a model are linked to each other without the need to work on both separately like in the standard software tools, which is illustrated next.

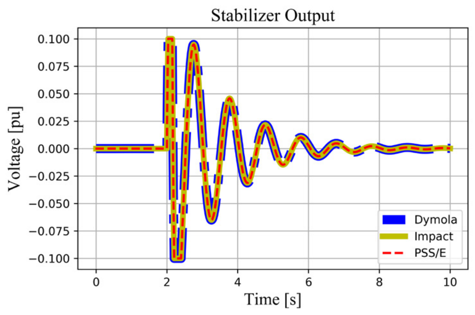


Figure 15. PSS2A output of the system in Figure 12.

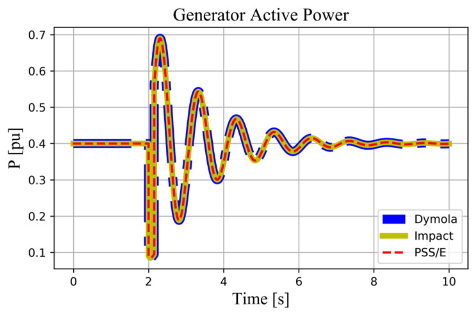


Figure 16. Active Power of the generator at bus GEN1 of the system in Figure 12.

3.2 IEEE 421.5 2005 DC4B Excitation System model

3.2.1 Implementation

Another example of model implementation is the exciter model DC4B. In Figure 18 the block diagram of the component from a PSS®E manual is given.

The PID with non-windup limits included in the model in Figure 18 is represented in Figure 19.

The challenge of the implementation of this component was represented by the integrator block inside the PID block. From (Murad and Federico Milano 2019) it is clear that for the same component, like a PI, there can be different implementations. In our case, to find the right representation of the dynamics of the PID block it has been necessary to check the state of the integrator. The interpretation has been facilitated by observing the output of the PID block together with the state of the integrator of the PID during a dynamic simulation in PSS®E of the SMIB including the DC4B model. The considered scenario is a 3-phase fault applied at bus FAULT at $t = 2s$ for $0.15s$. The corresponding SMIB in Modelica is given in Figure 20. The time trajectories of the PID output and its integrator state from PSS®E for the bus fault scenario are illustrated in Figure 21.

From Figure 21 it is possible to see that when the output of the PID reaches its limits the state of the integrator freezes to avoid windup effects (the so called *conditional integrator*). This observation led to the Modelica imple-

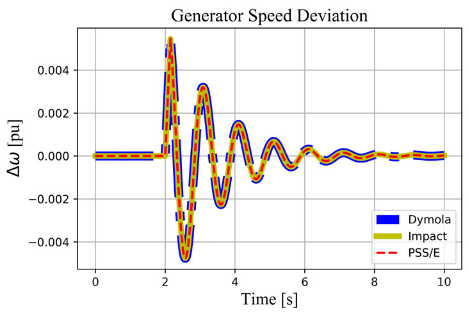


Figure 17. Speed Deviation of the generator at bus GEN1 of the system in Figure 12.

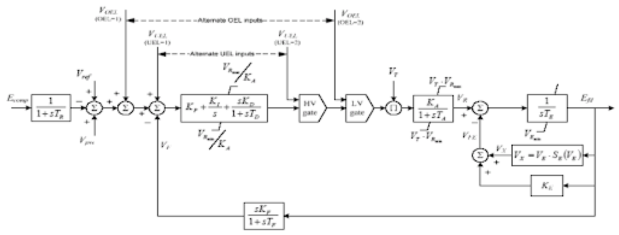


Figure 18. Block diagram of DC4B from PSS®E manual (Siemens Industry 2013).

mentation of the PID with non-windup limits as in Figure 22.

The Modelica text layer of the model in Figure 22 explains the behavior of the integrator of the PID (see Listing 2). To make the code more easily readable, in Listing 2 all the annotations, the lines indicating the blocks and additional parameters for the initialization of the model have been removed.

```
model PID_No_Windup
  import     Modelica.Units.SI;
  parameter SI.PerUnit K_P "Voltage regulator proportional gain
       (pu)";
  parameter SI.TimeAging K_I "Voltage regulator integral gain (
       pu)";
  parameter SI.PerUnit K_D "Voltage regulator derivative gain (
       pu)";
  parameter SI.Time T_D "Voltage regulator derivative channel
       time constant (sec)";
  parameter SI.PerUnit V_RMAX "Maximum regulator output (pu)";
  parameter SI.PerUnit V_RMIN "Minimum regulator output (pu)";
equation
  reset_switch.u2 =
```

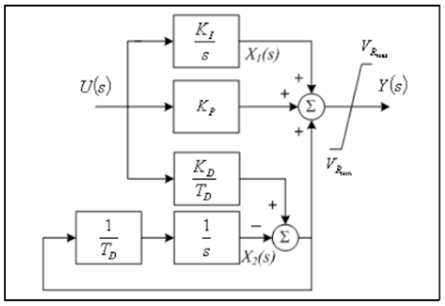


Figure 19. Block diagram of PID with non-windup limits from PSS®E manual (Siemens Industry 2013).

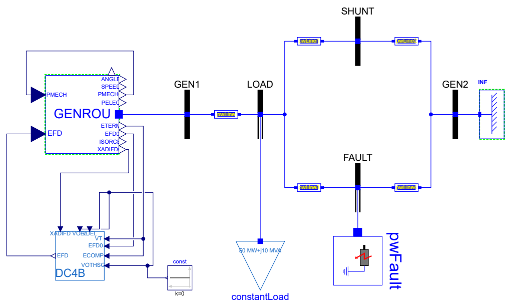


Figure 20. SMIB with DC4B in Modelica.

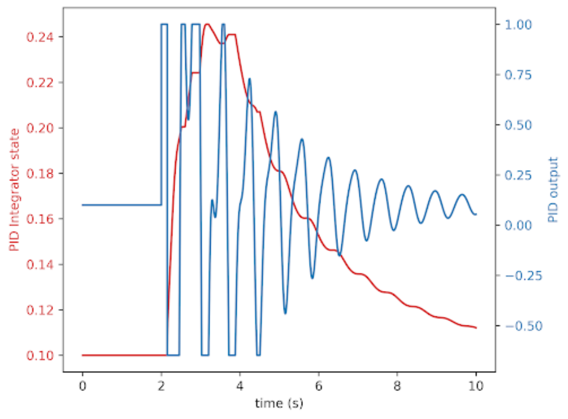


Figure 21. Plot of the output of the PID and the state of its integrator from a bus fault simulation in PSS®E.

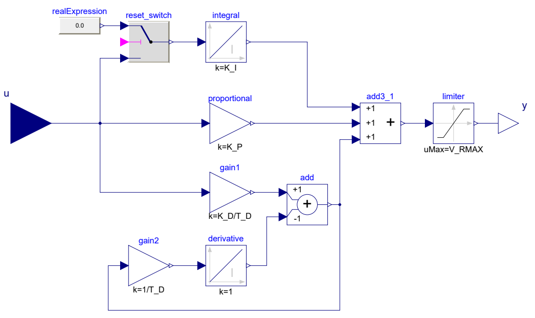


Figure 22. PID with non-windup limits implemented in Modelica.

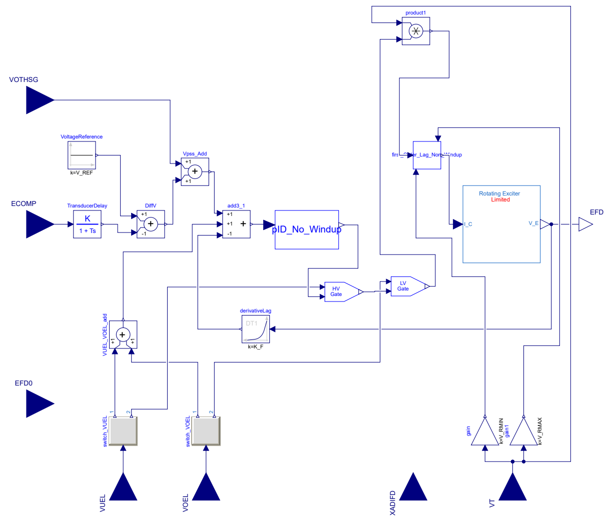


Figure 23. IEEE 421.5 2005 DC4B Excitation System model in Modelica.

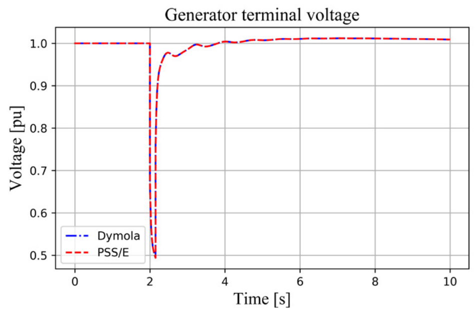


Figure 24. Generator terminal voltage at bus GEN1.

```
    if (abs(V_RMAX - y) <= Modelica.Constants.eps and der(integral
        .y)>0)
      then true
    else if (abs(V_RMIN - y) <= Modelica.Constants.eps and der(
        integral.y)<0)
      then true
    else false;
  end PID_No_Windup;
```

Listing 2. Modelica code for *PID with non-windup limits* model

Then the implementation of the DC4B model in Modelica has been completed as in Figure 23.

3.2.2 Validation

The software-to-software validation of the DC4B model against PSS®E has been performed to check the validity of the adopted modeling approach. For the verification the SMIB in Figure 20 has been considered with a bus fault applied at bus FAULT at $t = 2s$ for $0.15s$. Some results of the validation are given in Figures 24, 25 and 26.

An alternative approach for the validation of the DC4B excitation system, that Modelica flexibility allows for, consists of simulating only the block representing the DC4B component driven by external input signals collected from the reference SMIB network in PSS®E. This means that the Modelica model of DC4B can also be run

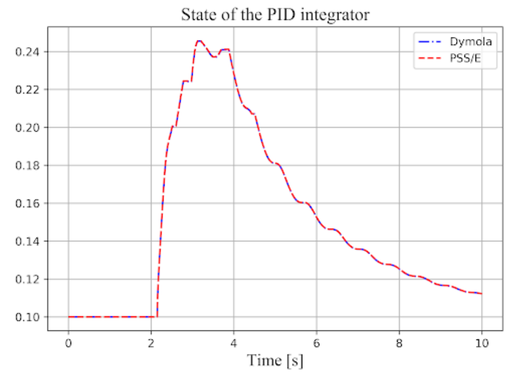


Figure 25. Time trajectory of the state of the integrator of the PID with non-windup limits included in the DC4B model.

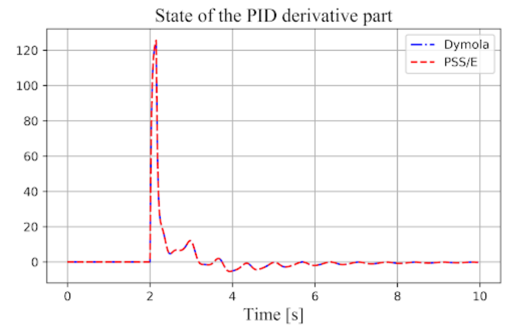


Figure 26. Time trajectory of the state of the derivative part of the PID with non-windup limits included in the DC4B model.

using different input signals coming, for example, from real measurements.

To prove this new approach the PID model in Figure 22 has been modified replacing the integrator blocks with an input so to use the integrator state trajectory as input to the model (see Figure 27).

Then a new test system in Modelica has been created as in Figure 28. Instead of assembling an SMIB to test the new implemented model it is possible to build a test system with only the target model with all the inputs depending on the available signals and a table collecting the external signals driving the model. To clarify the link between the external signals and the model to test, the text layer of the Modelica model in Figure 28 is given in Listing 3 keeping only the *equation* section.

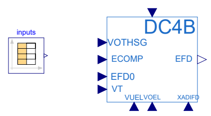


Figure 28. New Modelica test system for DC4B.

```
model DC4B_state_input_test
equation
  dC4B_state_input.VOTHSG = inputs.y[1];
  dC4B_state_input.ECOMP = inputs.y[2];
  dC4B_state_input.EFD0 = inputs.y[3];
```

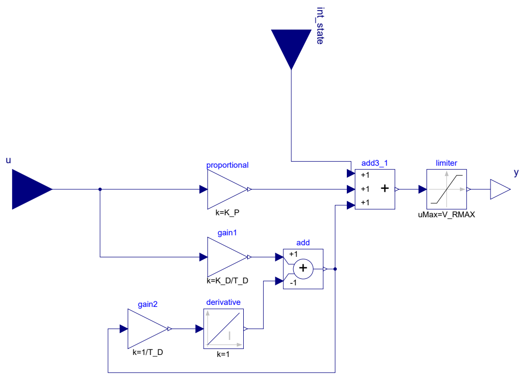


Figure 27. Modified PID with non-windup limits model.

```
    dC4B_state_input.VUEL = inputs.y[4];
    dC4B_state_input.VOEL = inputs.y[5];
    dC4B_state_input.XADIFD = inputs.y[6];
    dC4B_state_input.VT = inputs.y[7];
    dC4B_state_input.int_state = inputs.y[8];
end DC4B_state_input_test;
```

Listing 3. Modelica code for the DC4B test system in Figure 28

Then the validation of the model has been performed comparing the DC4B exciter output (EFD) in a scenario of SMIB system, as in Figure 20, with a bus fault applied at bus FAULT at $t = 2s$ and duration $t = 0.15s$. The results are given in Figure 29.

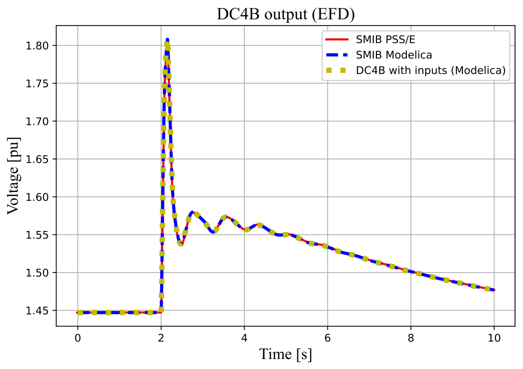


Figure 29. DC4B exciter output (EFD).

In Figure 29 the output of DC4B for the SMIB in Figure 20, the corresponding SMIB in PSS®E and the new test system in Figure 28 has been plotted. The system in Figure 28 takes the required input signals exported from the SMIB in PSS®E simulated with a bus fault as described before.

3.3 Examples of interoperability: portable system modeling with unambiguous and homogeneous results

Finally, to demonstrate an additional value of implementing and validating models with the Modelica language, we highlight the portability of system models using validated components. This can be shown by running the simulation of an example of the OpenIPSL library in different software environments. The example of a SMIB with the exciter EXST1 has been chosen (see Figure 30).

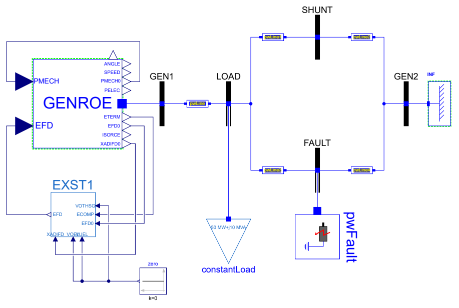


Figure 30. SMIB example for EXST1 from the OpenIPSL library.

Some results of the simulation of the network in Figure 30 are reported in Figures 31 and 32. The plots show that the model gives the same results regarless of the tool being used, which means that tool-specific re-implementation can be avoided.

Another example is illustrated with the network model IEEE14 in Figure 33.

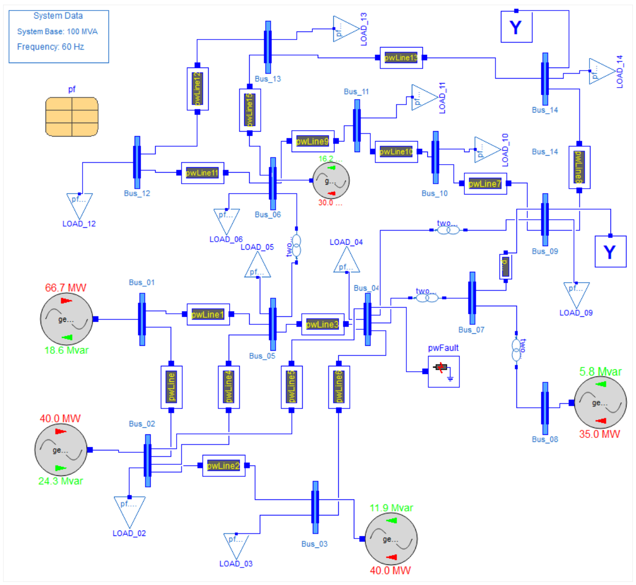


Figure 33. Modelica model of the power systems network IEEE14.

A simulation of the network has been run in the reference tool PSS®E and three different platforms that are Modelica based. They are Dymola, Modelon Impact and SystemModeler. The same scenario has been considered in all three software and it consists of applying a three-phase fault to ground at Bus 4 at time $t = 3s$ for $0.01s$. The fault to ground has an impedance $Z = R + jX = (0.01 + j0.02)pu$. The same simulation settings about the output interval length ($0.01s$) and the tolerance (1e-06) have been used. Results of the

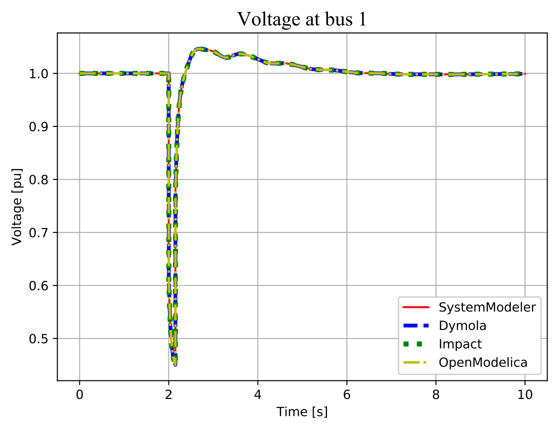


Figure 31. Voltage at bus GEN1 of the system in Figure 30.

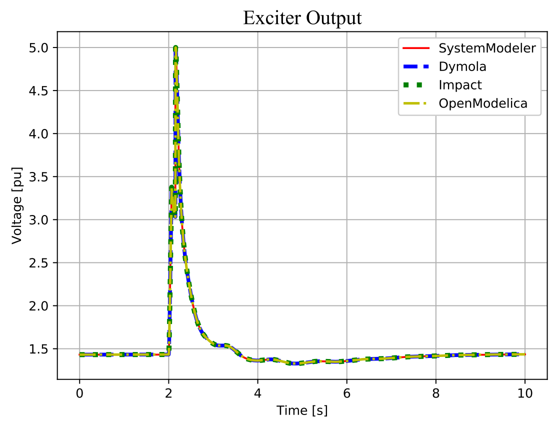


Figure 32. EXST1 output of the system in Figure 30.

simulation are given in Figure 34, where it is possible to see that once the models are validated they can give the same results as the reference tool with the same simulation settings even for larger networks.

4 Conclusions and Future Work

This paper formalizes and illustrates a procedure to implement power system models in Modelica with a software-to-software validation methodology that is an important step for developing and maintaining a Modelica library using the concepts of regression testing and continuous integration. The guidelines illustrated in this paper are indispensable for the initial debugging of new models addressing all possible challenges of the time consuming re-implementation process in a systematic way. The guidelines were illustrated through key use cases that proved to be challenging to implement, this gives an idea of the difficulties faced when developing power system models from a reference software tool. Once the results of an initial validation are satisfactory then the software-to-software

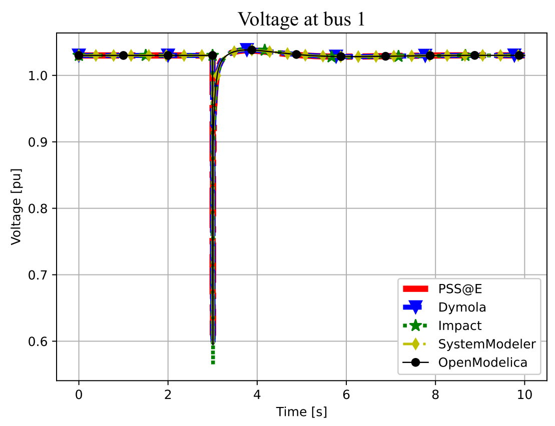


Figure 34. Voltage at bus 1 of the IEEE14 model with a fault to ground at bus 4.

validation process can be automated using scripts to test different scenarios in the different simulation platforms. This work will be presented in a future publication.

Acknowledgements

This material is based upon work supported in whole or in part by Dominion Energy, New York State Energy Research and Development Authority (NYSERDA) and the New York Power Authority (NYPA) under agreement numbers 137940, 37951, and 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. The views expressed herein do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

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.

Berube, GR, LM Hajagos, and Roger Beaulieu (1999). "Practical utility experience with application of power system stabilizers". In: *1999 IEEE Power Engineering Society Summer Meeting. Conference Proceedings (Cat. No. 99CH36364)*. Vol. 1. IEEE, pp. 104–109.

Bérubé, GR and LM Hajagos (2007). "Accelerating-power based power system stabilizers". In: *Year not known*, p. 10.

Chaudhary, Rekha and Arun Kumar Singh (2014). "Transient stability improvement of power system using non-linear controllers". In: *Energy and Power Engineering* 2014.

Cui, Hantao, Fangxing Li, and Kevin Tomsovic (2020). "Hybrid symbolic-numeric framework for power system modeling and analysis". In: *IEEE Transactions on Power Systems* 36.2, pp. 1373–1384.

Dorado-Rojas, Sergio A. et al. (2021-09). "Power Flow Record Structures to Initialize OpenIPSL Phasor Time-Domain Simulations with Python". 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. 147–154. ISBN: 978-91-7929-027-6. DOI: 10.3384/ecp21181147.

Elmqvist, Hilding, Toivo Henningsson, and Martin Otter (2016). "Systems Modeling and Programming in a Unified Environment Based on Julia". In: *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications*. Ed. by Tiziana Margaria and Bernhard Steffen. Cham: Springer International Publishing, pp. 198–217. ISBN: 978-3-319-47169-3.

Fachini, Fernando et al. (2021). "Modeling and Validation of Renewable Energy Sources in the OpenIPSL Modelica Library". In: *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, pp. 1–6. DOI: 10.1109/IECON48115.2021.9589148.

Gómez, Francisco J. et al. (2020). "Software requirements for interoperable and standard-based power system modeling tools". In: *Simulation Modelling Practice and Theory* 103, p. 102095. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2020.102095.

Henriquez-Auba, Rodrigo et al. (2021). "Transient Simulations With a Large Penetration of Converter-Interfaced Generation: Scientific Computing Challenges And Opportunities". In: *IEEE Electrification Magazine* 9.2, pp. 72–82. DOI: 10.1109/MELE.2021.3070939.

Hongesombut, K. et al. (2005). "Object-oriented modeling for advanced power system simulations". In: *2005 IEEE Russia Power Tech*, pp. 1–6. DOI: 10.1109/PTC.2005.4524823.

Jayapal, R and JK Mendiratta (2010). "H_∞ Controller Design for a SMIB-Based PSS Model 1.1." In: *Journal of Theoretical & Applied Information Technology* 11.

Kumar, Ajit (2018). "Damping enhancement for smib power system equipped with partial feedback linearization avr". In: *2018 20th National Power Systems Conference (NPSC)*. IEEE, pp. 1–6.

Masoom, Alireza et al. (2021). "MSEMT: An Advanced Modelica Library for Power System Electromagnetic Transient Studies". In: *IEEE Transactions on Power Delivery*.

Milano, F. (2005). "An open source power system analysis toolbox". In: *IEEE Transactions on Power Systems* 20.3, pp. 1199–1206. DOI: 10.1109/TPWRS.2005.851911.

Murad, Mohammed Ahsan Adib and Federico Milano (2019). "Modeling and simulation of PI-controllers limiters for the dynamic analysis of VSC-based devices". In: *IEEE Transactions on Power Systems* 34.5, pp. 3921–3930.

Rabuzin, Tin, Maxime Baudette, and Luigi Vanfretti (2017). "Implementation of a continuous integration workflow for a power system Modelica library". In: *2017 IEEE Power Energy Society General Meeting*, pp. 1–5. DOI: 10.1109/PESGM.2017.8274618.

Siemens Industry, Inc. (2013-03). *MODEL LIBRARY*. English. Siemens Power Technologies International. 748 pp.

Thurner, Leon et al. (2018). "pandapower—an open-source python tool for convenient modeling, analysis, and optimization of electric power systems". In: *IEEE Transactions on Power Systems* 33.6, pp. 6510–6521.

Tiller, Michael (2001). *Introduction to physical modeling with Modelica*. Springer Science & Business Media.

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.

Wang, Xiaodong et al. (2015). "Nonlinear dynamic analysis of a single-machine infinite-bus power system". In: *Applied Mathematical Modelling* 39.10-11, pp. 2951–2961.

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.

Zhang, Mengjia et al. (2015). "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), October, 7-9, 2015, Linköping University, Sweden*. Linköping University Electronic Press, pp. 105–112.

Zimmerman, Ray Daniel, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas (2010). "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education". In: *IEEE Transactions on power systems* 26.1, pp. 12–19.

Material Production Process Modeling with Automated Modelica Models from IBM Rational Rhapsody

John Batteh[1] Jesse Gohl[2] James Ferri[3] Quang Le[4]
Bill Glandorf[5] Bob Sherman[6] Rudolfs Opmanis[7]

[1,2]Modelon Inc., USA, {john.batteh, jesse.gohl}@modelon.com
[3,4]Virginia Commonwealth University, {jkferri, leq2}@vcu.edu
[5]Procter and Gamble, USA, glandorf.wm@pg.com
[6]Occam Systems Inc., USA, bob@occamsystemsinc.com
[7]Tom Sawyer Software, USA, rudolfs@tomsawyer.com

Abstract

This paper describes a method to author dynamic simulation models in Modelica from a manufacturing architectural model structure in SysML. Modelica models are generated from IBM Rational Rhapsody and simulated using Modelon Impact. Following a brief overview of the overall modeling approach and tool coupling, the Modelica modeling is detailed for computing process throughput and processing time. Following the model overview, a sample application for the production of the pharmaceutical ingredient atropine is demonstrated.

Keywords: SysML, Rhapsody, process modeling, pharmaceuticals, manufacturing

1 Introduction

The increasing complexity of modern manufacturing systems has created a paradigm shift from the traditional document-centric approach in systems engineering to Model Based System Engineering (MBSE). MBSE provides stakeholders access to an authoritative thread of information describing the system design throughout the product lifecycle. Complementary to MBSE, Multidisciplinary Design Analysis and Optimization (MDAO) focuses on creating an analysis model to demonstrate properties or outcome of the intended system by leveraging high performance computing and dynamic simulations. However, there is little effort in bridging the gap between the two approaches. We propose a method to author dynamic simulation models from a structural model. Specifically, we are developing infrastructure to automatically generate simulations in the Modelica language from a manufacturing architectural model structure in SysML (SysML 2022). This digital coupling allows for the unified design vision be developed with analytical rigor throughout the product lifecycle.

This work is part of a U.S. Defense Advanced Research Projects Agency (DARPA) project to build a digital infrastructure that the chemicals and materials industry needs to improve efficiencies. The goal of this project is to make it easier for chemical manufacturers in the U.S. to produce critically needed medicines. The models and workflow described in this paper allow manufacturers to explore production duration, synchronization, requirements, and constraints to improve and optimize chemical synthesis processes. Addressing challenges in the chemical industry to encourage domestic production of these medications will also pave the way for manufacturing a broad range of active pharmaceutical ingredients (APIs) for drugs in the U.S., instead of overseas. These include in-demand APIs related to COVID-19 and several others on the federal government's Strategic National Stockpile list.

This paper focuses on just the modeling and simulation portion of the project. Following a brief overview of the overall approach and tool coupling to generate Modelica models from SysML using IBM Rational Rhapsody (IBM 2022), the Modelica modeling is detailed for computing process throughput and processing time. An overview of the modeling approach and key components is provided. Following the model overview, a sample application for the production of atropine is demonstrated with simulations in Modelon Impact (Modelon 2022).

2 Modeling Process Overview

This section provides a high level overview of the modeling process to generate an executable Modelica model starting from an MBSE system model in IBM Rational Rhapsody. Figure 1 illustrates key components of the modeling process and the various software tools involved. The overall steps in the modeling process are as follows for the drug manufacturing use case:

- Based on research regarding the process to create a particular pharmaceutical ingredient, a SysML model is created in IBM Rational Rhapsody that outlines the manufacturing process steps.

- Based on the SysML model and an equipment database for chemical plants, a fully parameterized Modelica model is automatically created using Tom Sawyer Software technology for the manufacturing process based on a Modelica model library for the various process primitives
- The resulting Modelica model is then simulated in Modelon Impact using its simulator API
- Key simulation results, including process throughput and process time, are then returned to Tom Sawyer Software for visualization and also for use in an optimization process regarding the manufacturing process and equipment allocation

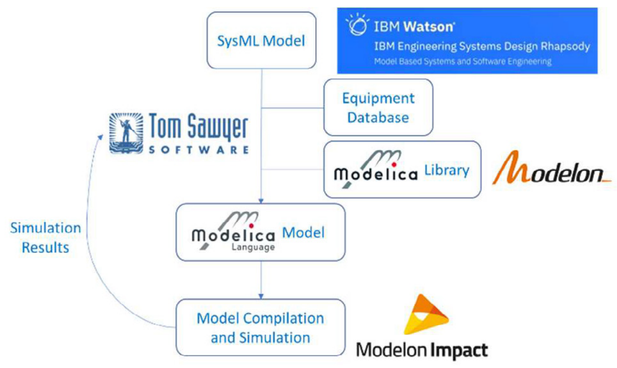


Figure 1. The modeling process overview.

This modeling process will be outlined in detail in separate future publications.

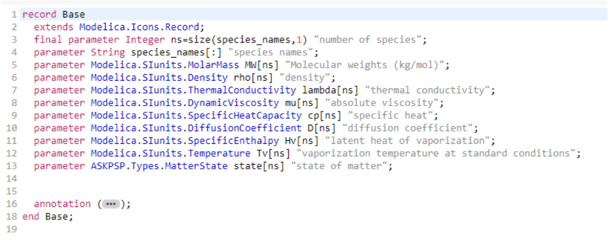
3 Modelica Models

As outlined in Section 2, the intended studies supported by this work are focused on the process layout for analyzing and optimizing the process time and yield for batch based material processing. The degrees of freedom for such an optimization are not only individual steps' boundary conditions (parameters) but also the overall process topology. Since the process steps required for manufacturing a given pharmaceutical ingredient are not wholly documented in public references and since the models are expected to be used in massive optimization loops, the models are intentionally implemented to be computationally fast due to low fidelity implementations and the use of time events.

The distinct stages of the process are simulated as individual components in the Modelica model. These stages compute their status (complete or not) and the batch material characteristics based on the behavior of the specific stage. The material characteristics and status feed successive stages. This data that is shared between components are described in Section 3.2. Details about specific implementations are provided in Section 3.3. Finally, examples of a fully assembled process are given in section 3.4.

3.1 Material Data Record

The properties of individual species are maintained within a data record that can be customized to each process. The records contain an array of the species names which are necessary for the process along with arrays of the species properties. The material data record approach is used in lieu of the more rigorous and detailed medium model approach to provide material information and thermophysical properties due to the lack of rigorous species information at various stages of the manufacturing process.



```
record Base
  extends Modelica.Icons.Record;
  final parameter Integer ns=size(species_names,1) "number of species";
  parameter String species_names[:] "species names";
  parameter Modelica.SIunits.MolarMass MW[ns] "Molecular weights (kg/mol)";
  parameter Modelica.SIunits.Density rho[ns] "density";
  parameter Modelica.SIunits.ThermalConductivity lambda[ns] "thermal conductivity";
  parameter Modelica.SIunits.DynamicViscosity mu[ns] "absolute viscosity";
  parameter Modelica.SIunits.SpecificHeatCapacity cp[ns] "specific heat";
  parameter Modelica.SIunits.DiffusionCoefficient D[ns] "diffusion coefficient";
  parameter Modelica.SIunits.SpecificEnthalpy Hv[ns] "latent heat of vaporization";
  parameter Modelica.SIunits.Temperature Tv[ns] "vaporization temperature at standard conditions";
  parameter ASKPSP.Types.MatterState state[ns] "state of matter";


  annotation (•••);
end Base;

```

Figure 2. The material data record.

As an example, the process simulated in Section 3.4, uses the species listed in Table 1.

Table 1. An example process data record species list.

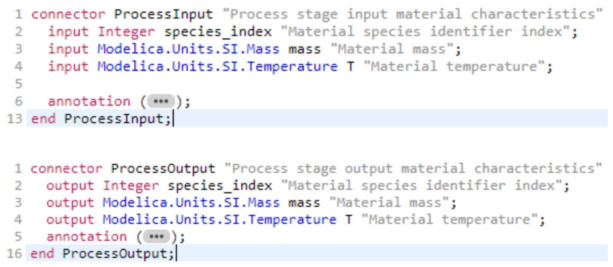
Index	Species
1	DCM
2	Tropine
3	Methanesulfonic acid
4	DCM+Tropine
5	DCM+Methanesulfonic acid
6	DCM+Tropine+Methanesulfonic acid
7	DCM+Tropine methanesulfonate

This record is instantiated as an `inner` component at the top level of any simulation model for access by model components via an `outer` relationship.

3.2 Component Interfaces

Material Interface

The individual "steps" in the production process implies an explicit ordering to the stages. This approach allows the interface between Modelica model components to use a causal interface with distinct inputs and outputs that define the material state.



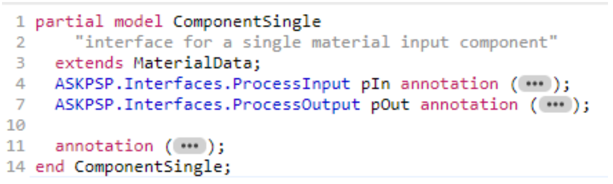
```
connector ProcessInput "Process stage input material characteristics"
  input Integer species_index "Material species identifier index";
  input Modelica.Units.SI.Mass mass "Material mass";
  input Modelica.Units.SI.Temperature T "Material temperature";

  annotation (...);
end ProcessInput;
```

```
connector ProcessOutput "Process stage output material characteristics"
  output Integer species_index "Material species identifier index";
  output Modelica.Units.SI.Mass mass "Material mass";
  output Modelica.Units.SI.Temperature T "Material temperature";
  annotation (...);
end ProcessOutput;
```

Figure 3. A process step's material data input and output.

These connectors represent the material type, quantity, and state that is shared between process stages. Because all of the steps represent processes that operate on batches of material (e.g. heating, stirring), these connectors are instantiated in interface base classes.

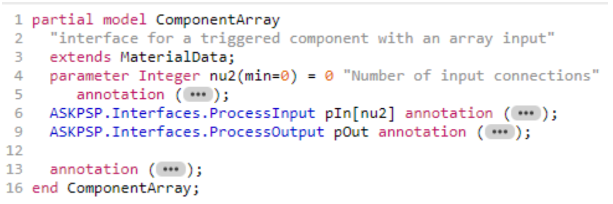
The first of these is `ComponentSingle` that supports components that operate on a single batch of material. In this case a single input and output is needed.



```
partial model ComponentSingle
    "interface for a single material input component"
  extends MaterialData;
  ASKPSP.Interfaces.ProcessInput pIn annotation (...);
  ASKPSP.Interfaces.ProcessOutput pOut annotation (...);

  annotation (...);
end ComponentSingle;
```

Figure 4. The single component base class.

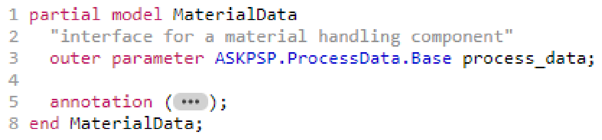
To support processing steps that operate on multiple batches of material (e.g. mixing and reactions), the `ComponentArray` base class is used.



```
partial model ComponentArray
  "interface for a triggered component with an array input"
  extends MaterialData;
  parameter Integer nu2(min=0) = 0 "Number of input connections"
    annotation (...);
  ASKPSP.Interfaces.ProcessInput pIn[nu2] annotation (...);
  ASKPSP.Interfaces.ProcessOutput pOut annotation (...);

  annotation (...);
end ComponentArray;
```

Figure 5. The array component base class.

Notice that the dynamically incremented connector size parameter is named `nu2`. This naming is to avoid conflicts with the connector size parameter for the trigger inputs, discussed in the next section. Components that extend from these base classes are required to include at least three equations to balance the three variables mass, temperature (`T`), and species_index on the output connector `pOut`. The ancestor base class for both of these base classes is the `MaterialData` model that instantiates an `outer` instance of the `process_data` record described in Section 3.1.



```
partial model MaterialData
  "interface for a material handling component"
  outer parameter ASKPSP.ProcessData.Base process_data;

  annotation (...);
end MaterialData;
```

Figure 6. The material data base class.

Logical Interface

In addition to the material data, stages also share their status with adjacent steps. This Boolean status is the local variable `complete` and is an output from all stages that serves as the trigger for successive stages to start their process. Because a stage could depend on previous steps (for example when process steps occur simultaneously), the input `trigger` is a Boolean array. Processing for a stage begins when <u>all</u> trigger inputs are true. Thus, all components include a scalar Boolean output named `complete` and an array Boolean input named `trigger`. These Boolean inputs and outputs are collected in a base class named `TriggeredComponent` that includes the logic to consider all input triggers to start the process step. The diagram and text of this base class appears in the following figure.

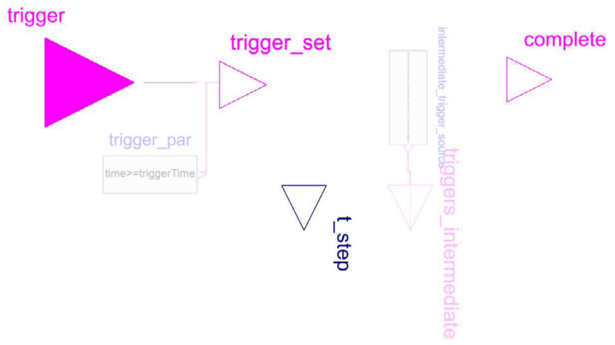
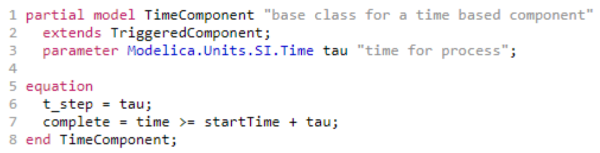


Figure 7. The triggered component base class diagram.

The base class includes the parameter option `paraOption_trigger` which allows the same component to be used either with external trigger connectors or from a parameter for absolute trigger time, `triggerTime`. The `triggered` internal Boolean variable becomes true when all triggers are true. This also starts the internal timer by setting the discrete variable `startTime` to the time instant of the trigger.

Fixed Time

The required duration for some stages can be specified directly. For example, stirring stages are specified as "stir the material for x seconds". These stages extend from the fixed `TimeComponent` base class which includes the process time parameter `tau`. Notice the state is complete when the simulation time exceeds the process time after that stage starts.

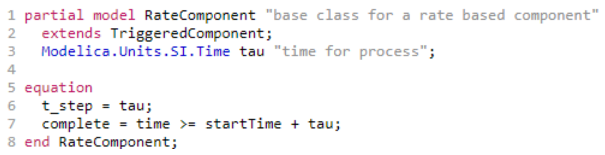


```
partial model TimeComponent "base class for a time based component"
  extends TriggeredComponent;
  parameter Modelica.Units.SI.Time tau "time for process";

equation
  t_step = tau;
  complete = time >= startTime + tau;
end TimeComponent;
```

Figure 8. The fixed time component base class.

Prescribed Rate

Other stages operate at prescribed rates. The processing time required for these stages is computed based on a rate equation using the ratio of the total quantity to be processed and the processing rate. For example, the duration of mass transfer is computed as the total mass to be transferred divided by the rate of transfer. For these stages the `RateComponent` base class is used.



```
partial model RateComponent "base class for a rate based component"
  extends TriggeredComponent;
  Modelica.Units.SI.Time tau "time for process";

equation
  t_step = tau;
  complete = time >= startTime + tau;
end RateComponent;
```

Figure 9. The rate component base class.

3.3 Process Stages

Elemental Stages

The following components represent fundamental stages used to describe an overall material handling process. These elemental components are responsible for computing the time required to complete their stage based on the established equations that describe the process physics. In addition, each stage is responsible for computing the exit state mass, temperature, and species index. Some stages can also produce a change of species (e.g. reactions, mixing, and drying). Due to the limited information available regarding each process step and the species mix, mass conservation is handled based on the incoming mass but also based on a yield input or parameter to allow the output mass to be calculated.

Time and Hold

The `Time` and `Hold` components represent simple timed stages that require a fixed time delay. Both of these components extend from the `TimeComponent` base class and add equations for the output connector's contents. These components behave as pure delays with no additional modification to the material. The same material species leaves that entered, no mass is stored in the component, and no heat is transferred. When the stage is complete, the mass and the temperature at the outlet become the same as the inlet.

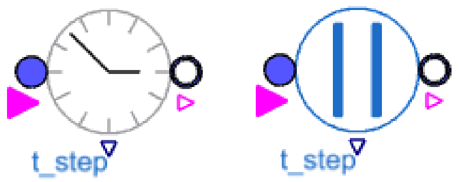


Figure 10. Time and Hold components' icons.

Parameter Rate

The `FixedRate` component describes the time required for handling mass at a fixed rate. It adds a parameter for the fixed mass flow rate and species the outlet connector similar to the `Time` and `Hold` components discussed above. When the stage starts (`triggered` becomes true), the processing time is computed based on the incoming mass divided by the fixed mass flow rate from the parameter.

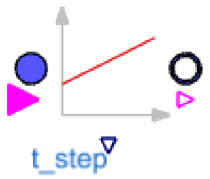


Figure 11. The FixedRate component icon.

Stir

The `Stir` component is another trivial component used to simulate the duration required to stir the material for a fixed amount of time. It specifies the outlet state in the same way as the components above.

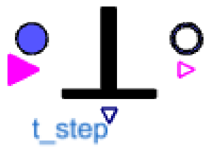


Figure 12. The Stir component icon.

Mix

The `Mix` component extends from the `ComponentArray` base class. It is used to simulate stages where multiple materials are combined over a fixed amount of time. The new, outgoing species is specified by a parameter. The outgoing mass is conserved as the sum of incoming masses and the temperature of the outlet material is the mass-average of the incoming materials (note: not a rigorous conservation of energy due to lack of detailed information about the process species at each stage).

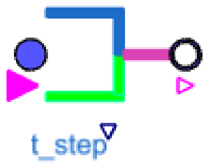


Figure 13. The mix component icon.

Reaction

The `Reaction` component is similar to the `Mix` component because it combines multiple materials to produce a new outgoing species. It also extends from the `ComponentArray` base class in order to support multiple incoming materials. In addition to the `species_index_out` parameter, there are parameters for the reaction time (`tau_reaction`), yield fraction, and reaction temperature change (`dT`). The yield fraction reduces the outgoing mass to account for losses and incomplete reactions. The temperature difference applies an additional offset to the outgoing temperature from the mass averaged inlet temperatures.

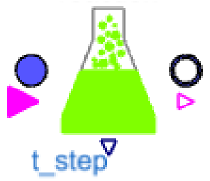


Figure 14. The Reaction component icon.

Volumetric Rate

The `VolumetricRate` component simulates a process that applies a fixed volumetric flow rate to the incoming material. This component also demonstrates the use of the "process_data" record. The density of the incoming material, `rho`, is selected based on the incoming species index. This density is then used together with the incoming mass, to compute the volume of the material that must be transferred by this stage. When the stage starts (`triggered` becomes true), the process time is updated based on the ratio of the material volume, `Vf`, with the fixed volumetric flow rate, `Q`.

Figure 15. The Volumetric rate component icon.

Liquid Transfer

The `LiquidTransfer` component is similar to the `VolumetricRate` component except the volumetric flow rate is computed assuming hydraulic flow through a smooth pipe driven by a constant power pump. The Fanning friction factor (Incropera 2007) relates the pressure drop to the mean fluid velocity and the Reynolds number characteristic of the flow (Munson 2009).

$$Re = \frac{4\rho Q}{\pi \mu D} \tag{1}$$

$$\Delta p = \frac{2\rho f L v^2}{D} \tag{2}$$

$$f = 0.0791 \cdot Re^{-1/4} \tag{3}$$

$$Q = v \cdot A \tag{4}$$

$$P_b = Q \cdot \Delta p \tag{5}$$

For *Re* – Reynolds number, ρ – fluid density, Q – volumetric flow rate, μ – dynamic viscosity, D – pipe diameter, Δp – pressure drop, f – Fanning friction factor, v – mean fluid velocity, A – pipe cross-sectional area, *Pb* – pump power

The unknown variables in these five equations are *Re*, Δp, Q, f, and v, which can be solved from these five equations. The volume of the fluid, *Vf*, is directly computed from the incoming mass and density. With the solution of the volumetric flow rate, *Q*, the duration can be directly computed.

Dissolution

The `Dissolution` component computes the duration required to fully dissolve a specified mass of solute in a stirred, fixed volume vat. The solute can be specified either as a fixed parameter, or it can enter from a conditional connector. The time for dissolution is calculated based on mass transfer analysis for a stirred vessel.

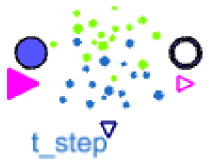


Figure 16. The Dissolution component icon.

Heat Transfer

As with the `Dissolution` component, the `HeatTransfer` component assumes a liquid in a fixed volume, stirred vessel. The formulation starts with Newton's law of cooling and the First Law of Thermodynamics to relate the heat capacity, heat transfer rate, and the material temperature for a constant pressure process (Castellan 1983).

$$\Delta H = Q = C_p \Delta T = C_p (T - T_0) \tag{5}$$

$$\dot{Q} = C_p \dot{T} = hA(T_{ht} - T) \tag{6}$$

T – material temperature, C_p – heat capacity at constant pressure, h – heat transfer coefficient, A – heat transfer area, T_{ht} – temperature of the heat transfer element. From these equations, the solution to the initial value problem for $T(0) = T_0$ is:

$$T(t) = T_{ht} + (T_0 - T_{ht}) e^{-t\frac{hA}{C_p}} \tag{7}$$

The final time, t_f, can be calculated from this equation to heat (or cool) the material to the final temperature, T_f.

$$t_f = -\tau \log\left(\frac{T_{ht} - T_f}{T_{ht} - T_0}\right) \tag{8}$$

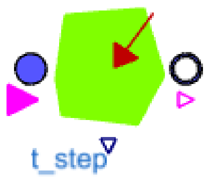


Figure 17. The heat transfer component icon.

Drying Fixed Rate

The `DryingFixedRate` component accounts for two physical phenomena. First is the addition of heat to evaporate a fraction of the incoming material stream. Second is the optional change of species to the outlet stream. This change depends on whether the evaporate or the dessicate (remainder) continues to the outlet or is discarded. This implementation allows the same component to be used for both a drying operation as well as a distillation step. The component assumes heat is added at a fixed rate defined by a parameter `Qmax`. The process time accounts for the duration to add sensible heat to raise the temperature of both the evaporate and dessicate and to add the latent heat of the evaporate. The fraction of the incoming stream that evaporates is specified by a parameter. From this information, the masses of both the evaporate and dessicate are known. From these masses and the heat capacity at constant pressure of each species, the total energy required for the sensible heat necessary to raise the temperature of both species from the incoming temperature to the vaporization temperature of the evaporate species can be directly computed. The total heat required also includes the latent heat of vaporization. This heat is directly computed from the evaporate mass and the evaporate species specific latent heat of vaporization. The duration of this step is then directly computed from the fixed rate of heating, `Qmax`.

This component includes an optional Boolean flag that allows the user to select whether the process is a drying or distillation step. This selection determines which species continues to the outlet stream. An assumption of this component is that the species index of the incoming stream is the evaporate species.

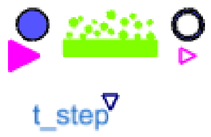


Figure 18. The Drying component icon.

Filter/Wash Base Class

Both the `Filter` and `Wash` component use Darcy's law to model the flow rate of a solvent through a permeable solid. This law relates the volumetric flow rate of a liquid to the hydraulic permeability, cross sectional area, dynamic viscosity, flow length, and pressure difference.

$$\boldsymbol{Q} = \frac{\Delta V}{\Delta t} = \frac{kA}{\mu L}\boldsymbol{\Delta p} \qquad (9)$$

Q – volumetric flow rate, ΔV – volume, Δt – duration, k – hydraulic permeability, A – cross sectional area of the filter path, μ – dynamic viscosity, L – length of the filter path, Δp – pressure difference

For gravity driven filtrations, this equation can be rearranged to use hydraulic conductivity, fluid density, and gravity in place of the permeability and dynamic viscosity.

$$\frac{K}{\rho g} = \frac{k}{\mu} \qquad (10)$$

K – hydraulic conductivity, ρ – density, g – gravitational constant

These equations are implemented in the `FilterWash` base class model to calculate the filtration time.

$$\Delta t = \frac{\Delta V \rho g}{A \Delta p} \cdot \left(\frac{L_f}{K_f} + \frac{L_c}{K_c}\right) \qquad (11)$$

L_f – filter length, K_f – filter conductivity, L_c – cake path length, K_c – cake conductivity

In this case, L_f/K_f is assumed to be a constant and is specified as a parameter. L_c is computed from the solid volume, packing efficiency, and cross sectional area. The outlet mass is the solute mass computed from a fixed yield fraction of the filtration.

The `Filter` component extends from the `FilterWash` base class and assumes the incoming material already contains the solvent and solute. The solvent mass is based on the yield fraction of the filtration and is used to compute the volume. The filtration is also assumed to be isothermal, so the outlet temperature is the same as the inlet.

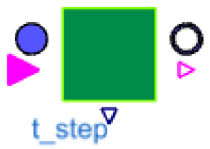


Figure 19. The Filter component icon.

Like the `Filter` component, the `Wash` component also extends from the `FilterWash` base class. In this case a new connector is added for a separate inlet for the solvent. The solvent volume is explicitly computed from the solvent mass and density. The outlet temperature for the `Wash` component is specified by a parameter. An additional condition for completion for this stage is that all the solvent must be delivered. Because prior stages hold their outlet mass at zero until they are complete, the

Wash stage uses the inlet solvent mass greater than epsilon to indicate completion. This implementation prevents the stage from completing until all the solvent mass has been transferred.

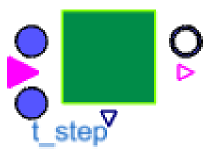


Figure 20. The Wash component icon.

Phase Cut

The `PhaseCut` stage represents a process separation step that splits the material on the input connector into two separate outputs based on a fixed fraction (`usable_fraction`). An additional `yield_fraction` parameter is defined to account for losses that occur during the separation process.

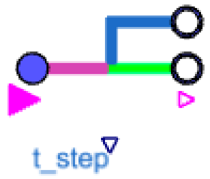
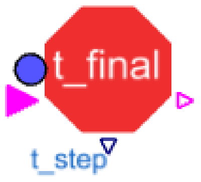
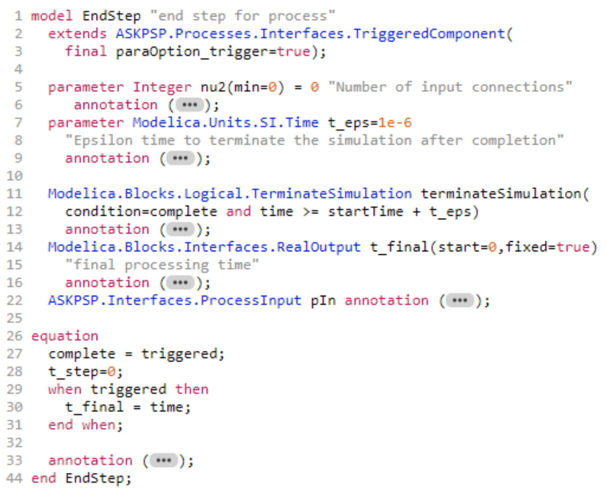


Figure 21. The PhaseCut component icon.

End

The `End` component performs two functions. First it records the overall completion time in the discrete variable `t_final`. This value is updated when the `End` component is triggered. The second function the component performs is to terminate the simulation. The additional condition for termination guarantees the simulation does not terminate before the solver has updated all outputs (e.g. `t_final`). Notice the final product species, mass, and temperature is available on the input connector.



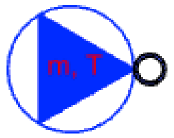


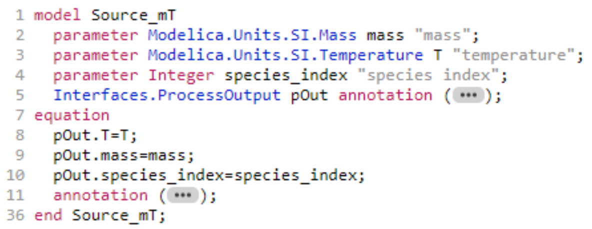
```
model EndStep "end step for process"
  extends ASKPSP.Processes.Interfaces.TriggeredComponent(
    final paraOption_trigger=true);

  parameter Integer nu2(min=0) = 0 "Number of input connections"
     annotation (•••);
  parameter Modelica.Units.SI.Time t_eps=1e-6
    "Epsilon time to terminate the simulation after completion"
    annotation (•••);

  Modelica.Blocks.Logical.TerminateSimulation terminateSimulation(
    condition=complete and time >= startTime + t_eps)
    annotation (•••);
  Modelica.Blocks.Interfaces.RealOutput t_final(start=0,fixed=true)
    "final processing time"
    annotation (•••);
  ASKPSP.Interfaces.ProcessInput pIn annotation (•••);

equation
  complete = triggered;
  t_step=0;
  when triggered then
    t_final = time;
  end when;

  annotation (•••);
end EndStep;
```

Figure 22. The End component icon and implementation.

Source

The `Source` component provides a fixed amount of material (mass), at a fixed temperature, as a specific species. It does not include an output `complete` signal because it is assumed to be a source of material. If the delivery time must be considered at the start of a process, one of the transfer components can be placed immediately after the source.





```
model Source_mT
  parameter Modelica.Units.SI.Mass mass "mass";
  parameter Modelica.Units.SI.Temperature T "temperature";
  parameter Integer species_index "species index";
  Interfaces.ProcessOutput pOut annotation (•••);
equation
  pOut.T=T;
  pOut.mass=mass;
  pOut.species_index=species_index;
  annotation (•••);
end Source_mT;
```

Figure 23. The Source component icon and implementation.

Composite Stages

In many cases during material handling, multiple stages are often associated with each other. For this reason, it is convenient to create composite stages that combine these steps into a single component. This approach to create composite stages reduces the effort required for the automated code generation process.

Heat Transfer and Dissolution

In many cases of dissolution during material processing, the instructions specify simultaneous heating during the dissolution. The elemental `Dissolution` component however assumes an isothermal process (the outlet material maintains the same temperature as the inlet). For this reason, a composite component was created that includes both these steps in parallel. This component starts with a mixing stage that combines the solute with the inlet material. After this stage is complete the heating and dissolution stages can start. The overall stage cannot complete until both stages are done.

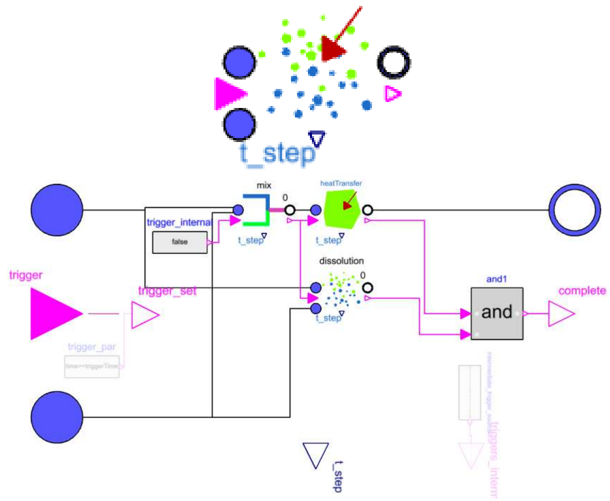


Figure 24. The Heat transfer and Dissolution component.

Add Liquid and Wash

Because washing steps require the addition of a liquid, the `AddLiquidAndWash` convenience component combines both the `Wash` component with the `VolumetricRate` liquid transfer component. Because washing can commence immediately, no mixing component is necessary. The volumetric transfer component only needs to generate its completion signal to complete the composite step.

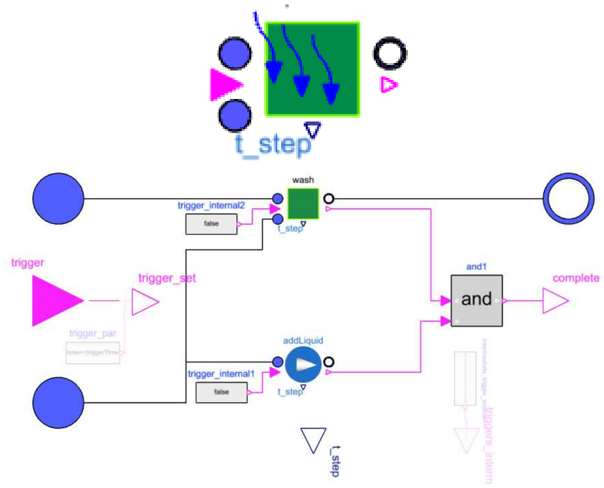


Figure 25. The AddLiquidAndWash component system.

3.4 Complete System Configuration

To demonstrate the usage of these components, the following model was created. This model is used to simulate the synthesis of tropine methanesulfonate, which is a precursor in the synthesis of atropine. The process starts by mixing species dichloromethane (DCM) to tropine as indicated by the species indices, 1 and 2 as listed in Table 1, as shown on the `source_DCM` and `source_Trop` components. The `liquidTransfer` and `solidTransfer` components compute the duration required to transfer the supplied materials to the mixing vessel (`mix`).

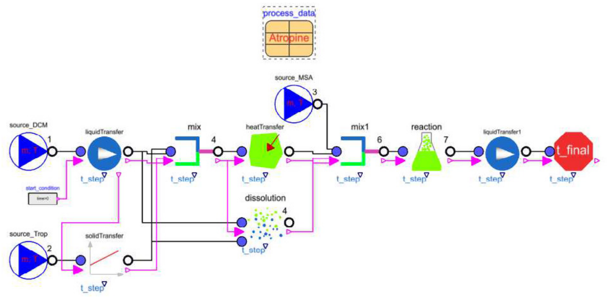


Figure 26. The Tropine methanesulfonate synthesis system.

Notice the `mix` component gets its material from `liquidTransfer` and `solidTransfer` and starts its process when both these steps are complete. Because it is based on the `TimeComponent` base class, it also includes a process time, tau for the additional duration after it starts. Finally the mix component also changes the species index to 4, which is the identifier for the mixture of DCM and Tropine (see Table 1).

After mixing, the mixture is heated and the tropine is dissolved in solution. This process is simulated by the `heatTransfer` and `dissolution` components. Notice the DCM liquid transfer is connected to the main material inlet of the dissolution component. The tropine material outlet connector is connected to the `dissolution` component solute material inlet. This approach allows the dissolution step to be computed for cases when the materials are not already mixed.

Alternatively the dissolution component also includes the parameter option to premix the solvent and solute, in which case the solute connector is disabled and the solute species can be specified via parameter. In this case the materials are already mixed so the dissolution component only needs to indicate when dissolution occurs. This demonstrates the flexibility of the models to be connected in multiple ways to accommodate different processes. This version of the library does not account for thermal effects of dissolution so there is no feedback from the heating step to the dissolution step. In this case they occur in parallel and are connected to the next `mix1` component.

The `mix1` component accounts for the time to complete both the heating, dissolution, and to add the third component, methanesulfonic acid, that is the reagent for the reaction. The outlet is the new species 6, DCM, tropine, and methanesulfonic acid.

The `reaction` component is another fixed time component that accounts for the reaction time based on a specified duration as a parameter. The outlet of this component is the final species 7, DCM and tropine methanesulfonate.

The final step in this sub-process is to transfer the liquid from the reaction vessel. This step is simulated with an additional liquid transfer component, `liquidTransfer1`. When this step is complete, it triggers the end step which records the overall duration and terminates the simulation. The progression of completion the stages can be tracked throughout the process by observing the `complete` local Boolean variable for individual stages as shown in the following figure.

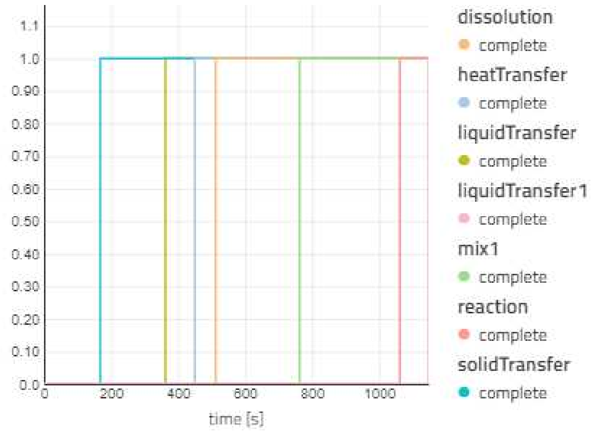
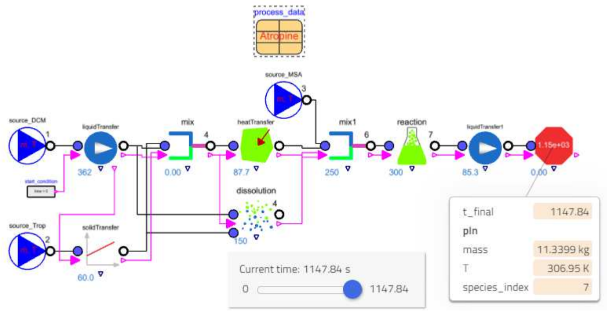


Figure 27. Atropine precursor process, completion progression.

The final results are shown in the following image. The overall process time (`t_final`), the mass, temperature (`T`), and final species index are visualized.

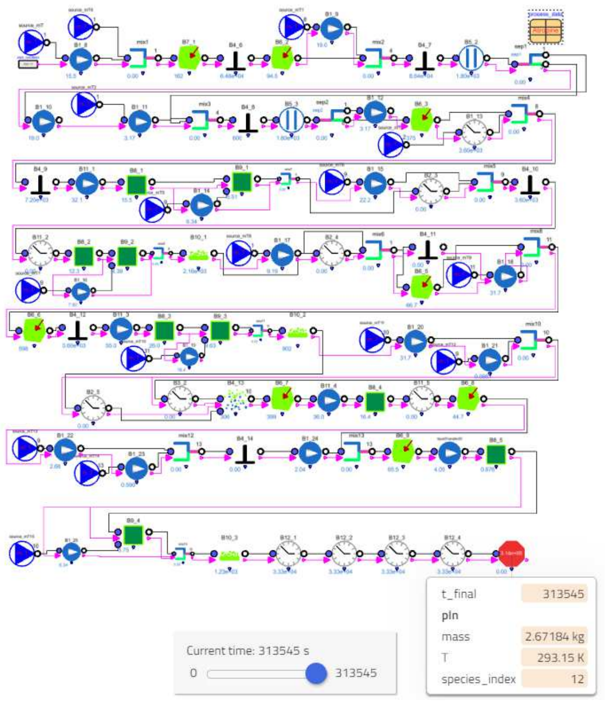


Current time: 1147.84 s

t_final	1147.84
pIn	
mass	11.3399 kg
T	306.95 K
species_index	7

Figure 28. Precursor final results.

A larger process system is shown in the following image. This system simulates the final stages in the synthesis of atropine starting from the precursor created by the previous process. The stages are numbered according to the identifiers used in the original synthesis instructions. Again, the final results are indicated in the diagram and the progression of individual stages completion can be tracked by plotting the time trajectory for the Boolean variable, `complete`, shown in Figure 30.



Current time: 313545 s

t_final	313545
pIn	
mass	2.67184 kg
T	293.15 K
species_index	12

Figure 29. Atropine synthesis system model, and final results.

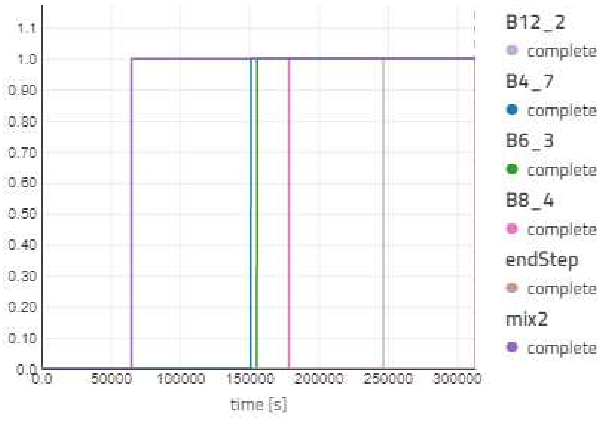


Figure 30. Atropine synthesis, completion progression.

4 Conclusions

This paper focuses on the modeling and simulation capability for the DARPA project to build a digital infrastructure to improve efficiencies in the chemicals and materials industry. Based on the work in the DARPA project to build SysML models that describe the manufacturing process for given pharmaceutical agents, simulation models for process throughput and yield are automatically generated in Modelica using the model library detailed in the paper. The models are simulated in Modelon Impact using its simulator API and key results returned to the software stack for use in visualization and optimization processes. A sample use case for the synthesis of atropine is demonstrated.

This paper focuses on the model library developed to support this digitalization approach and an initial use case. Ultimately this project will support efforts to analyze the chemical supply chain and identify capability gaps in pharmaceutical manufacturing for key drugs.

Acknowledgements

This work has been supported by U.S. Defense Advanced Research Projects Agency (DARPA).

References

Castellan, Gilbert W. (1983). *Physical Chemistry*. 3rd ed. The Benjamin/Cummings Publishing Co., Inc. ISBN: 0-201-10386-9.

IBM (2022). *IBM Engineering Systems Design Rhapsody*. URL: https://www.ibm.com/products/systems-design-rhapsody

Incropera, Frank P., David P. Dewitt, Theodore L. Bergman, and Adrienne S. Lavine (2007). *Fundamentals of Heat and Mass Transfer*. 6th ed. John Wiley & Sons, Inc. ISBN: 978-0-471-45728-2.

Modelon (2022). *Impact*. URL: https://www.modelon.com/modelon-impact/.

Munson, Bruce R., Donald F. Young, Theodore H. Okiishi, and Wade W. Huebsch (2009). *Fundamentals of Fluid Mechanics*. 6th ed. John Wiley & Sons, Inc. ISBN: 978-0470-26284-9.

SysML (2022). *SysML Open Source Project - What is SysML? Who created SysML?* URL: https://sysml.org/

Modelica