Proceedings of the 7th International Modelica Conference

September 20-22, 2009
Grand Hotel di Como
Como, Italy

Francesco Casella (editor)

organized by
The Modelica Association and Politecnico di Milano

Proceedings of the 7^{th} Modelica Conference

Grand Hotel di Como, Italy, September 20-22, 2009

Editor:
Prof. Francesco Casella

Published by:
The Modelica Association

ISBN: 978-91-7393-513-5
Linköping Electronic Conference Proceedings
ISSN: 1650-3740
DOI: 10.3384/ecp0943

Preface

The first International Modelica Conference took place in October 2000 in Lund, Sweden. Since then, Modelica has increasingly become the preferred language for physical modelling of complex systems, as this 7th edition of the Conference demonstrates. The number of presented papers, 83 regular and 22 posters, has significantly increased compared to the previous Conference, as well as the number of Modelica-enabled tools being presented, which has reached the record number of nine. The keynote talks about the ITEA2 Modelica-related European projects also prove how Modelica is entering mainstream industry usage in system design. The Conference is a key opportunity for all players in the Modelica community (language designers, tool developers, library designers, and end users) to meet and share their ideas and experience, thus promoting further development of the language and of its applications.

The 7th International Modelica Conference was organized by Politecnico di Milano, Italy, and by the Modelica Association. I would like to thank the local organizing committee for the excellent job and the technical programme committee and the programme board for offering their time and expertise in reviewing the papers.

Cremona, September 25, 2009

Francesco Casella

Organizing Committees

Programme Chair

- Prof. Francesco Casella, Politecnico di Milano, Italy

Programme Board

- Prof. Martin Otter, DLR, Oberpfaffenhofen, Germany
- Prof. Peter Fritzson, Linköping University, Sweden
- Dr. Hilding Elmqvist, Dynasim AB, Lund, Sweden
- Dr. Michael Tiller, Emmeskay Inc., Plymouth, Michigan, USA

Programme Committee

- Dr. Johan Åkesson, Lund University, Lund, Sweden
- Dr. Peter Aronsson, MathCore Engineering AB, Linköping, Sweden
- Prof. Karl-Erik Årzén, Lund University, Lund, Sweden
- Prof. Bernhard Bachmann, Univ. Applied Sciences Bielefeld, Bielefeld, Germany
- Dr. John Batteh, Emmeskay Inc., Plymouth, Michigan USA.
- Dr. Ingrid Bausch-Gall, Bausch-Gall GmbH, Munich, Germany
- Daniel Bouskela, EDF R&D, Paris, France
- Prof. Felix Breitenecker, TU Wien, Vienna, Austria
- Prof. François E. Cellier, ETH Zürich, Zürich, Switzerland
- Mike Dempsey, Claytex Services Ltd, UK
- Dr. Hosam Fathy, University of Michigan, USA
- Prof. Gianni Ferretti, Politecnico di Milano, Italy
- Dr. Rüdiger Franke, ABB AG, Mannheinm, Germany
- Dr. Rui Gao, Dassault Systèmes Japan, Nagoya, Japan
- Anton Haumer, Technical consultant, St. Andrae-Woerdern, Austria
- Dr. Kay Juslin, VTT, Espoo, Finland
- Dr. Christian Kral, Arsenal Research, Austria
- Prof. Alberto Leva, Politecnico di Milano, Italy
- Kilian Link, Siemens AG, Erlangen, Germany
- Prof. Wolfgang Marquardt, RWTH Aachen, Germany
- Dr. Jakob Mauss, QTronic GmbH, Berlin, Germany
- Prof. Chris Paredis, Georgia Institute of Technology, Atlanta, Georgia, USA
- Dr. Adrian Pop, Linköping University, Sweden
- Prof. Gerhard Schmitz, Technical University Hamburg-Harburg, Germany
- Peter Schneider, Fraunhofer IIS/EAS, Dresden, Germany
- Dr. Edward D. Tate, General Motors, Michigan, U.S.A.
- Dr. Wilhelm Tegethoff, TLK-Thermo GmbH and TU Braunschweig, Germany
- Dr. Hubertus Tummescheit, Modelon AB, Lund, Sweden
- Dr. Andreas Uhlig, ITI GmbH, Dresden, Germany
- Prof. Alfonso Urquía, UNED, Spain
- Prof. Hans Vangheluwe, McGill University, Montreal, Quebec, Canada

- Dr. HansJürg Wiesmann, formerly ABB Corporate Research, Baden, Switzerland

Local Organizing Committee

- Francesco Casella, Politecnico di Milano
- Martina Maggio, Politecnico di Milano
- Sara Gruppi, Politecnico di Milano, Polo di Cremona
- Maristella Pellini, Politecnico di Milano, Polo di Cremona
- Centro di Cultura Scientifica Alessandro Volta, Como

Contents

Session MoAT1:

Session MoAT2:

Session MoAT3:

Session MoAT4:

Session MoBT1:

Session MoBT2:

Session MoBT3:

Session MoBT4:

Session MoCT1:

Session MoCT2:

Session TuET2:

Session TuET3:

Session TuET4:

Index of Authors

Thermal Modeling of Automotive Lithium Ion Cells using the Finite Elements Method in Modelica

Imke Krüger   Martin Sievers   Gerhard Schmitz
Institute of Thermofluid Dynamics, Applied Thermodynamics
Hamburg University of Technology, 21071 Hamburg, Germany
{imke.krueger, martin.sievers, schmitz}@tuhh.de

Abstract

In this paper, a finite element model for the heat transfer inside an automotive lithium ion cell with Modelica is developed. Convective cooling with several coolants is examined. With the help of the cell model, the effectiveness of several coolants are investigated. As coolants air, 25 and 38 % (v/v) aqueous propylene glycol, and a silicone oil are used.

Keywords: lithium ion cell; Modelica; OpenModelica Compiler; heat transfer; finite element method

1 Introduction

Lithium ion cells are a promising technology for the use in hybrid vehicles due to their high energy and power densities. However, thermal management is necessary to prevent premature aging, and for safety reasons. The modeling and simulation of the temperature distribution in a lithium ion cell is therefore of high interest for the design of a cooling system. Heat sink for the cells is the automotive refrigeration cycle. since the refrigeration cycle will be modeled in Modelica it is obvious to model the battery pack with Modelica as well. In this way, one can avoid time-consuming coupling with other software packages.

Due to the anisotropic heat conduction and the distributed heat generation inside the cell, a simple lumped capacity model is not sufficient to assure that the maximum temperature inside the cell is kept beyond a certain level. To determine the temperature distribution in a cylindrical Li-ion cell, a two-dimensional finite element model is developed and presented in this paper [1]. The thermal models are simulated with Dymola and the OpenModelica Compiler in combination with the GUI simforge.

The paper is structured as follows. The following section describes the structures of the Lithium ion cell and the cooling system. In Section 3, the idea of the finite element method is presented and its Modelica implementation illustrated. In Section 4, the results for several coolants are compared. Conclusions are drawn in the final section.

2 Lithium Ion Cells

Li-ion cells are characterized by the type of electrolyte that is used, the geometry of the cell, the type of the electrodes and the material of the electrodes and other parts. In this project the temperature of a commonly used cell type is simulated.

Fig. 1 shows the architecture of a spirally wound cylindrical cell.

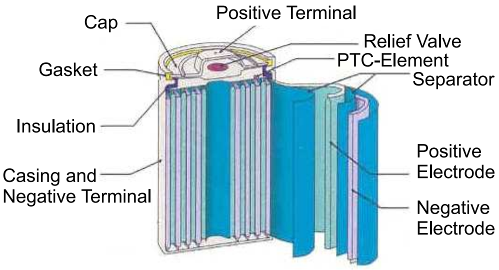


Figure 1: Architecture of a spirally wound cell [2]

A winding consisting of a negative electrode with a negative conductor, a separator layer, a positive electrode with a positive conductor and a further separator layer is rolled up and put in a metal casing. This casing is then filled up with the electrolyte. The investigated cell has a $LiCoO_2$ positive electrode with an aluminum conductor, a graphite negative electrode with a copper conductor and PE/PP separators. The casing is made of steel. The cylindrical cell uses the standard electrolyte $LiPF_6$. Many commercially available cells are

DOI: 10.3384/ecp09430072

Material	Thermal Conductivity / $\frac{\mathrm{W}}{\mathrm{mK}}$
Steel	15
$LiPF_6$	0.6
Aluminum	240
PE / PP	0.22
Graphite	1.04
Copper	395
$LiCoO_2$	1.58

Table 1: Thermal conductivity of the cell components [3, 4, 5, 6, 7]

based on this $LiCoO_2$-graphite system. Thermal conductivities of the cell materials are listed in Table 1. In this model 31 windings are used, the foil thicknesses from 20 to 140 µm are used. This results in a capacity around 7.5 A h.

To construct an HEV battery pack, many Li-ion cells are connected serial and parallel within the pack. An essential part of the battery pack is its thermal and electrical management. Current and voltage of every single Li-ion cell in a battery pack must be controlled because overcharging can lead to thermal runaway. As heat is generated continuously within the cell during charging and unloading, an appropriate cooling system is necessary.

Cell performance generally improves with increasing temperature, as diffusive processes dominate the reaction. On the other hand ageing processes become faster as well with increasing temperatures. At room temperature the reduction of cell life can reach up to 50 % for a temperature increase of 10 deg C. A thermal management system ensures a long life time and good cell performance. At temperatures above 70 °C the cell is at risk of thermal runaway. The separator melts and direct contact between the electrodes results in a highly exothermic reaction. The optimum balance between long life time and cell performance is achieved when the mean temperature lies between 25 and 30 °C [2]. The maximum temperature within the cell should exceed 40 °C in exceptional cases only [2]. Temperature differences within the cell lead to mechanical stress. One measure of this stress is the maximum temperature difference, which should not exceed 5 deg C.

These limits have to be fullfilled for a variety of boundary conditions because the same battery system has to operate efficiently in different climates. Three different cooling systems are possible in a hybrid vehicle:

- air cooling: the air is provided by the air conditioning system,
- secondary loop cooling: a coolant is circulated in an additional loop,
- evaporative cooling: the battery heat exchanger is connected to the refrigeration unit of the air conditioning.

In this paper, the first two approaches are investigated. Forced convection in the longitudinal direction is directly applied to the cell shell and natural convection to the caps. The configuration is shown in Fig. 2.

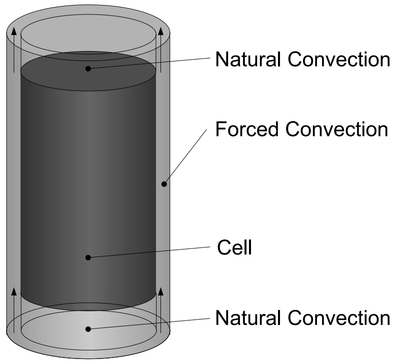


Figure 2: Configuration of the heat transfer at the cell surface

3 Finite Element Method

3.1 Mathematical Aspects

The finite element method is widely used for mechanical problems, but can also be successfully applied to heat transfer problems. This applies especially to problems, for whom an analytical solution is difficult to find due to complex geometries or heat generation inside the object of interest.

The temperature distribution inside the cell is independent of the circumferencial direction if longitudinal flow of the coolant is assumed. The Fourier differential equation of heat transfer in cylindrical coordinates:

$$\lambda \left[\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial \vartheta}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 \vartheta}{\partial \varphi^2} + \frac{\partial^2 \vartheta}{\partial z^2} \right] + \dot{e} = \rho \; c_p \; \frac{\partial \vartheta}{\partial t} \tag{1}$$

is reduced to two dimensions. With the assumption of constant radial und axial heat conductivity, it can be further simplified to

$$\frac{1}{r}\left[\lambda_r \frac{\partial}{\partial r}\left(r\frac{\partial \vartheta}{\partial r}\right)\right]+\lambda_z\frac{\partial^2\vartheta}{\partial z^2}+\dot{e}=0 \qquad (2)$$

with the following boundary conditions

$$\vartheta = \vartheta_b \qquad (3)$$

and

$$\lambda_r\frac{\partial \vartheta}{\partial r}l+\lambda_n\frac{\partial \vartheta}{\partial z}n+\alpha(\vartheta-\vartheta_{inf})+\dot{q}=0. \qquad (4)$$

Equation (3) defines temperatures at the boundaries (Dirichlet boundary conditions). A heat flux can be prescribed by (4), it considers a heat flux due to heat conduction or convection as well as a given heat flux $\dot{q}$ (Neumann boundary condition).

To model a certain cylinder, the area is subdivided into elements. In this approach, triangular elements with linear basis functions are used[8].

For each element, a stiffness matrix **S** and the load vector $\vec{b}$ are calculated. The stiffness matrix **S** defines the heat conduction inside the triangle:

$$\mathbf{S}=\int_V \mathbf{B^T}\,\mathbf{D}\,\mathbf{B}\,dV+\int_O \alpha\,\mathbf{N^T}\,\mathbf{N}\,dO \qquad (5)$$

with

$$\mathrm{B}\;=\begin{bmatrix}\frac{\partial\vartheta}{\partial r}\\ \\ \frac{\partial\vartheta}{\partial z}\end{bmatrix}=\begin{bmatrix}\frac{\partial N_1}{\partial r}\,\frac{\partial N_2}{\partial r}\,\frac{\partial N_3}{\partial r}\\ \\ \frac{\partial N_1}{\partial z}\,\frac{\partial N_2}{\partial z}\,\frac{\partial N_3}{\partial z}\end{bmatrix}=\frac{1}{2\,A}\begin{bmatrix}b_1 & b_2 & b_3\\ c_1 & c_2 & c_3\end{bmatrix} \qquad (6)$$

the heat conductivity matrix

$$\mathbf{D}=\begin{bmatrix}\lambda_r & 0\\ 0 & \lambda_z\end{bmatrix} \qquad (7)$$

and the basis functions matrix **N**

$$\mathbf{N}=[N_1\;N_2\;N_3]\,. \qquad (8)$$

The load vector $\vec{b}$ consists of inner heat sources and the boundary conditions

$$\vec{b}=\int_V \dot{e}\,\mathbf{N^T}dV+\int_O \dot{q}\,\mathbf{N^T}dO+\int_O \alpha\vartheta_a\mathbf{N^T}dO \qquad (9)$$

Applying the Galerkin method (a form of weighted residual method) to Eq. (2) one yields the following equation

$$\int_V \mathbf{N}\left[\frac{1}{r}\lambda_r\frac{\partial}{\partial r}(r\frac{\partial\vartheta}{\partial r})+\lambda_z\frac{\partial^2\vartheta}{\partial z^2}+\dot{e}\right]dV=0 \qquad (10)$$

which can also be written as the finite element equation with $\vec{u}^T=[\vartheta_1\vartheta_2\vartheta_3]$

$$\mathbf{S}\,\vec{u}=\vec{b}. \qquad (11)$$

The global stiffness matrix and the global load vector are formed by adding the corresponding elements of neighboring triangles. The global finite element equation can then be set up and solved.

3.2 Implementation in Modelica

Due to the axisymmetry of the heat transfer inside the cell, the temperature does not depend on the angle φ. Only longitudional flow of the coolant can be examined with this approach, since for transvers flow the heat transfer coefficient depends on φ.

The three nodes of the triangular element are represented by connectors with the temperature as a single variable.

```
connector node
Modelica.siunits.Temp_K T;
end node;
```

The triangle model consists of the local stiffness matrix **S**, the local load vector $\vec{b}$, the local temperature vector $\vec{u}$, three nodes, their geometric parameters in cylindrical coordinates and everything necessary to define the heat transfer. Several parameters have the attribute fixed set to false as their value will be set in the aggregated cell model (see next subsection). The convective heat transfer coefficients depend on the medium temperature at the element and are calculated in the cell model, thus they are declared as input variables. The element size can be varied to be able to account for the terminals and the casing of the cell.

The following code listing shows, how two triangles are connected (see Fig. 3). First, the connectors have to be connected. Then the global stiffness matrix and the load and temperature vectors are constructed. The finite element equation is formulated in the global model only.

```
model 2Elements
siunits.Temp_K u[4,1];
siunits.ThermalConductivity S[4,4];
siunits.Heat b[4,1];
  triangle A(...);
  triangle B(...) ;
equation
connect(A.node2, B.node3);
connect(B.node2, A.node3) ;
S={{A.S[1,1], A.S[1,2],  A.S[1,3],0},
  {A.S[2,1], A.S[2,2]+B.S[3,3],
   A.S[2,3]+B.S[2,3], B.S[3,1]},
```

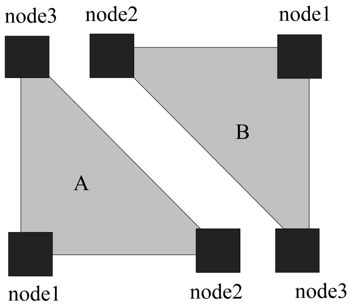


Figure 3: Example 2Elements

```
  {A.S[1,3], A.S[2,3]+B.S[2,3],
  A.S[3,3]+B.S[2,2], B.S[1,2]},
{0,B.S[1,3],B.S[1,2], B.S[1,1]}};
u={{A.u[1,1]},{A.u[2,1]},
{A.u[3,1]}, {B.u[1,1]}};
b={{A.b[1,1]}, {A.b[2,1]+B.b[3,1]},
  {A.b[3,1]+B.b[2,1]},{B.b[1,1]}};
 S*u=b;
end 2Elements;
```

The construction of the global stiffness matrix becomes more and more complicated with a rising number of connected elements. However, once the equations for large stiffness matrices are formulated by means of loops, they can be reused for every other axisymmetric problem if triangular elements with linear basis functions are used.

3.3 Cell model

The model for the lithium-ion cell is made up of two arrays of triangles: A and B. Their directions and coordinates are chosen such that e.g. triangle A[1,1] and B[1,1] are connected to form a rectangle as shown above. All triangles together form the whole cell.

The attempt to take every single layer in the winding into account by at least one element in radial direction would lead to a high number of equations and is therefore dismissed. Instead, the axial and radial heat conductivities are calculated for each layer of sheets as the thermal resistances (axial, radial) vary with the radius. Fig. 4 shows the layers of the winding and the corresponding two triangular elements representing one winding.

For the whole cell a grid with 35 x 10 elements with different element sizes is generated. Using a model with more elements leads to an unreasonable increase in simulation time. The cell model can easily be adopted to simulate different cell dimensions. Besides this, the cell materials can be varied. The lid and the casing of the cell are also taken into account.

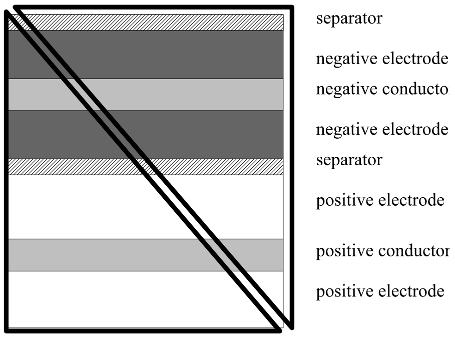


Figure 4: Layers in one winding

3.4 Coolant model

As the OpenModelica Compiler does not yet support the Modelica.Media library, simple linear medium models are used to calculate the medium properties. For air, the relative deviation up to 60 °C is less than 5 % for the relevant properties. Pressure drop of the fluid is neglected.

Fig. 5 illustrates the heat transfer from the cell wall to the fluid. The heat flow from the cell element i to the coolant is described by the following equation

$$\dot{Q}_{w,i} = \alpha \cdot A \cdot \Delta\vartheta_i \tag{12}$$

with the mean temperature difference

$$\Delta\vartheta_i = \frac{\vartheta_{W,i} + \vartheta_{W,i+1}}{2} - \frac{\vartheta_{C,i} + \vartheta_{C,i+1}}{2} \tag{13}$$

between the wall temperature ϑ_W and the coolant temperature ϑ_C .

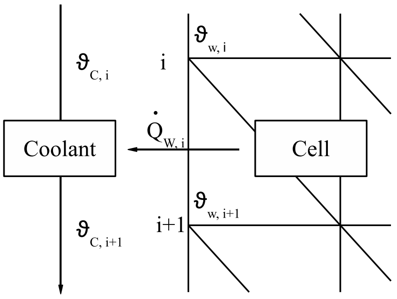


Figure 5: Heat transfer from cell to fluid

The convective heat transfer coefficient α is calculated from a Nusselt number relationship for longitudional flow along a cylinder [3]. As the coolant flows along the cell, the fluid is heated up:

$$\vartheta_{C,\,i+1} = \vartheta_{C,\,i} + \frac{\dot{Q}_{W,\,i}}{\dot{m} c_p}. \qquad (14)$$

The temperature difference between cell wall and fluid decreases, less heat can be transferred to the fluid.

At the caps at the top and the bottom of the cell, only natural convection is taken into account (Fig. 2).

The calculated heat flow out of the cell is added to the load vector $\vec{b}$.

4 Comparison of Coolants

The coolant type, flow velocity, temperature and flow direction were varied for different simulations. The objective was to find a configuration that fulfills the benchmarks for maximum and mean temperature, together with the maximum temperature difference, as specified in Section 2. The coolants investigated were air, 25 and 38 % (v/v) aqueous propylene glycol, and a silicone oil. Further details of the analyses may be found in [1].

In the simulations, it is assumed that the cell is constantly charged or discharged. The amount of heat generated inside the cell is approximated by a relation given by Gibbard [9]. For moderate loading rates, the heat generation is about 200kW/m. For very fast charging/discharging, the heat generation is about 300kW/m.

4.1 Air Cooling

The first set of simulations was performed for air cooling. Fig. 6 shows the result for an air flow velocity of 5 $\frac{\text{m}}{\text{s}}$, an inlet temperature of 20 °C and a heat generation of 200kW/m. The coolant flows upwards along the right-hand side of the diagram; the left-hand side represents the center line of the cell. The maximum temperature exceeds the thermal runaway limit, and the maximum temperature difference greatly exceeds the 5 deg C limit (see Table 2). The heat transfer coefficient decreases in the flow direction, so that the hottest cell zone is close to the center line in the upper half of the cell, and heat is predominantly removed through the cap at the coolant inlet. Since the forced convective heat transfer is considerably greater than the natural convective heat transfer at the caps, most of the heat is conducted radially to the coolant within the caps. The isotherms are almost parallel within the electrolyte-filled cavity along the center line of the cell, as no heat is generated there. Some of the heat from the cell is transported to the caps within this cavity.

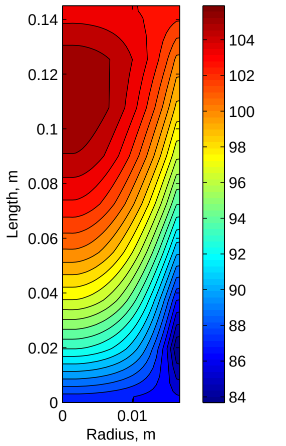


Figure 6: Temperature field for air cooling with a flow velocity of 5 $\frac{\text{m}}{\text{s}}$, an inlet temperature of 20 °C and 200kW/m^3 heat generation

Chen et al. [5] state that radiative heat transfer should be taken into account when modeling surface heat transfer. This applies to cooling with air, whose optical thickness is low. When liquid coolants are used, radiative heat transfer is neglected because of the higher optical thickness. A simple model which assumes grey Lambert bodies for a single cell was used to demonstrate the influence of radiative heat transfer for cells located in the outermost regions of the battery pack.

For the same boundary conditions as in Fig. 6, the maximum temperature is reduced from 105.9 °C to 82.7 °C, and the maximum temperature difference is reduced from 22.2 deg C to 17.8 deg C, see Fig. 7 and Table 2.

Table 2: Characteristic temperatures of the different simulations. A: air, inlet flow velocity 5m/s, inlet temperature 10 °C, 200kW/m^3 heat generation

	ϑ_{max} °C	ϑ_{avg} °C	$\Delta\vartheta_{max}$ deg C
Fig. 6	105.9	97.6	22.2
Fig. 7	82.7	75.9	17.8
A	95.6	87.3	22.2
Fig. 8	31.5	25.9	11.1
Fig. 9 left	42.0	33.8	16.6
Fig. 9 right	41.6	33.3	16.3

Another way of lowering the cell temperature is to reduce the inlet temperature. A simulation was con-

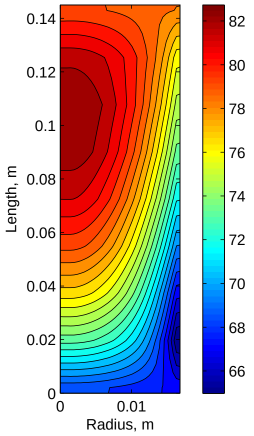


Figure 7: Temperature field for cooling with air at a flow velocity of 5 $\frac{\mathrm{m}}{\mathrm{s}}$, an inlet temperature of 20 °C and 200kW/m^3 heat generation, taking radiative heat transfer into account

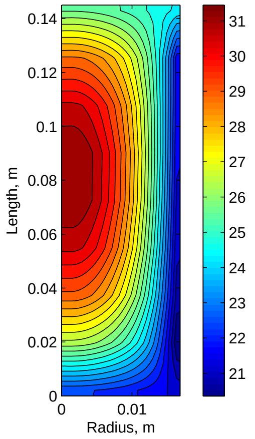


Figure 8: Temperature field for cooling with 38 %(v/v) aqueous propylene glycol at a flow velocity of 1 $\frac{\mathrm{m}}{\mathrm{s}}$, an inlet temperature of 20 °C and 200kW/m^3 heat generation

ducted for an air flow velocity of 5 $\frac{\mathrm{m}}{\mathrm{s}}$, an inlet temperature of 10 °C and a heat generation of 200kW/m^3 (see Table 2, case A). Comparison with Fig. 6 shows that the temperature reduction within the cell is approximately equal to the inlet temperature reduction. The higher energy demand of the refrigeration cycle in order to cool the fluid below ambient temperature should be noted.

No air cooling configuration has been found that keeps the temperature within the permissible temperature range. Air cooling is therefore not recommended for the stationary case.

4.2 Cooling with 38 %(v/v) Aqueous Propylene Glycol

Aqueous propylene glycol is used as a coolant in internal combustion engines. The thermal capacity of glycol is higher than for air, so the flow velocity is reduced. The result of a simulation for an inlet flow velocity of 1 $\frac{\mathrm{m}}{\mathrm{s}}$, an inlet temperature of 20 °C and a heat generation of 200kW/m^3 is shown in Fig. 8.

The high heat transfer coefficients lead to a considerable improvement in thermal performance. The mean temperature of 25.9 °C and maximum temperature of 31.5 °C are both within limits; the maximum temperature difference of 11.1 deg C exceeds the limit. Temperature benchmarks are nearly satisfied, and temperature management is possible.

The heat generation is increased to 300kW/m^3, which corresponds approximately to a charging rate that allows 20 complete chargings and dischargings per hour. Fig. 9 shows the effect of varying the inlet flow velocity for an inlet temperature of 25 °C and 300kW/m^3 heat generation.

An increase in flow velocity from 1 $\frac{\mathrm{m}}{\mathrm{s}}$ to 1.5 $\frac{\mathrm{m}}{\mathrm{s}}$ reduces the temperature within the cell only slightly, by about 0.5 °C. This is because the wall temperature is already close to the coolant temperature. Increasing the coolant flow velocity above 1 $\frac{\mathrm{m}}{\mathrm{s}}$ is thus of minor advantage for temperature control.

A change in inlet temperature has a considerable and therefore useful influence upon cell temperature. Reducing the inlet temperature to 15 °C results in a maximum cell temperature of 32.2 °C and a mean temperature of 24.0 °C, temperatures which are very close to the benchmark temperature. The maximum temperature difference of 16.6 deg C still exceeds the limit (see Table 3, case B).

To reduce the maximum temperature difference within the cell even further, a change in cell geometry must be considered. This may involve cell material, external diameter and length, and also the dimensions of internal components or the thickness of the caps. Changes in cell design have been evaluated in [1].

To ensure safe operation, the maximum temperature within the cell should never exceed the critical temperature of 60 °C. simulations have shown that, when cooling with 38 %(v/v) aqueous propylene glycol at a flow velocity of 1 $\frac{\mathrm{m}}{\mathrm{s}}$ and with a heat generation of 300kW/m^3, the maximum inlet temperature is 43 °C.

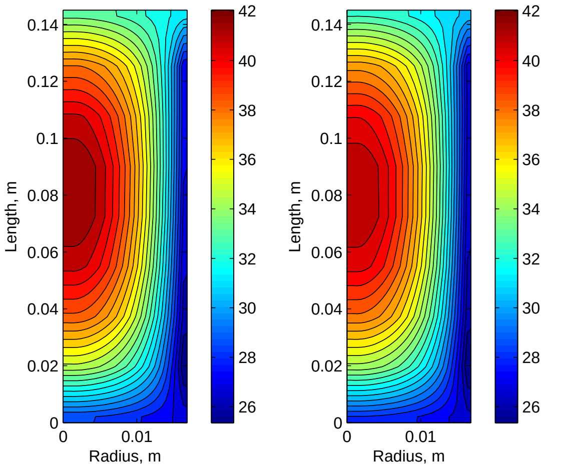


Figure 9: Comparison of temperature fields for cooling with 38 %(v/v) aqueous propylene glycol at a flow velocity of 1 $\frac{m}{s}$ (left) and 1.5 $\frac{m}{s}$ (right), an inlet temperature of 25 °C and 300kW/m^3 heat generation

4.3 Comparison with other Coolants

To show the effect of different coolants, simulations with 25 %(v/v) aqueous propylene glycol and Syltherm 800 were performed. Fig. 10 shows a comparison between all three coolants for an inlet flow velocity of 1 $\frac{m}{s}$, an inlet temperature of 25 °C and a heat generation of 300kW/m^3. The propylene glycol concentration is seen to have only a minor influence on the temperature, whilst the use of Syltherm 800 silicone oil results in much higher temperatures.

Table 3: Characteristic temperatures of the different simulations. B: 38 %(v/v) aqueous propylene glycol, inlet flow velocity 1 $\frac{m}{s}$, inlet temperature 15 °C, 3300kW/m^3 heat generation

	ϑ_{max}	ϑ_{avg}	$\Delta\vartheta_{max}$
	°C	°C	deg C
B	32.2	24.0	16.6
Fig. 10 left	36.4	30.9	11.1
Fig. 10 center	36.1	30.6	11.0
Fig. 10 right	40.7	34.9	13.4

5 Conclusion

The stationary thermal model simulates the two-dimensional temperature distribution within the cell for natural and forced convective as well as for radiative heat transfer. It can be easily adapted to other cell geometries, and can therefore be used to simulate other cell designs.

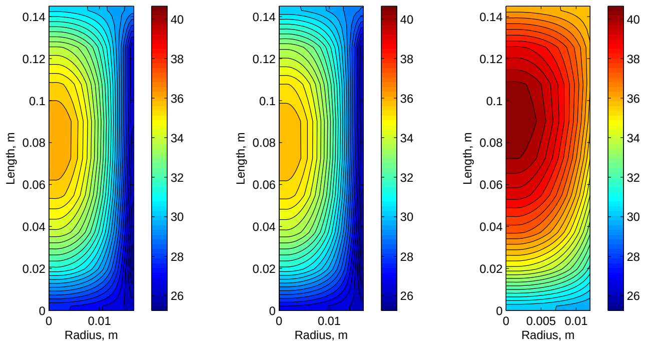


Figure 10: Comparison of temperature fields for cooling with 38 (left), 25 %(v/v) aqueous propylene glycol (center) and Syltherm 800 (right) at inlet flow velocity of 1 $\frac{m}{s}$, an inlet temperature of 25 °C and 300kW/m^3 heat generation

simulations show that air cooling is insufficient to keep temperatures within the required limits. Meanwhile, liquid coolants keep the maximum and mean temperature within the desired range for a heat generation of 200kW/m^3, and thus ensure safe cell operation. The cell cannot be operated safely at a heat generation of 300kW/m^3.

The simulation of heat transfer with the finite element method has been succesfully implemented in Modelica. Various cylindrical bodies can be modeled. The quality of the model is only limited by the maximum number of equations that can be handled by the tool. The construction of the global stiffness matrix is quite complex, but can be reused for other cylindrical models.

The modeling attempt was also tested with the OpenModelica Compiler. Arrays with large dimensions could not be succesfully simulated, the model for the whole cell can therefor not be translated. For small models consisting of four triangles, the simulation was possible. Only little modifications of the models were necessary.

The introduction of a capacitance matrix into each element would enlarge the range of application to dynamic problems as well.

Smaller numbers of equations can be reached by using different element types for the inside (only heat conduction) and the outside of the cell (additional heat transfer by convection and radiation).

References

[1] Sievers, M.: Modellierung der Temperaturverteilung in Lithium-Ionen-Batterien mit der Finite-Elemente-Methode. Bachelor thesis, Hamburg University of Technology, Department of Mechanical Engineering, 2008.

[2] Jossen, A., Weydanz, A.: Moderne Akkumulatoren richtig einsetzen. Reichert Verlag, Untermeitingen, 2006.

[3] Verein Deutscher Ingenieure, VDI-Gesellschaft Verfahrenstechnik und Chemieingenieurwesen (GVC): VDI-Wärmeatlas. 10. Auflage, Springer-Verlag, Berlin, 2006.

[4] Julien, C., Stoynov, Z.: Materials for lithium-ion batteries: Proceedings of the NATO Advanced Study Institute on Materials for Lithium-Ion Batteries, Design and Optimization. Sozopol, Bulgaria, September 21 - October 1, 1999, Kluwer Academic Publishers, Dordrecht, 2000.

[5] Chen, S.-C., Wang, Y.-Y., Wan, C.-C.: Thermal Analysis of Spirally Wound Lithium Batteries. Journal of the Electrochemical Society, 153 (2006) 4, pp. A637-A648.

[6] Al-Hallaj, S., Maleka, H., Selman, J. S.: Thermal modeling and design considerations of lithium-ion batteries. Journal of Power Sources, 83 (1999) 1-2, pp. 1-8.

[7] Wu, M.-S., Liu, K.H., Wang, Y.-Y., Wan, C.-C.: Heat dissipation design for lithium ion batteries. Journal of Power Sources, 109 (2002) 1, pp. 160-166.

[8] Lewis, R. W., Nithiarasu, P., Seetharamu, K. N.: Fundamentals of the Finite Element Method for Heat and Fluid Flow, John Wiley, Chichester, 2004.

[9] Gibbard, H. F.: Thermal properties of battery systems, Journal of the Electrochemical Society, 125 (1978), pp. 353-358.

Object-Oriented Decomposition of Tire Characteristics based on semi-empirical Models

Markus Andres
Vorarlberg Univ. of Appl. Sc.
Austria
Markus.Andres@students.fhv.at

Dirk Zimmer
ETH Zürich
Switzerland
DZimmer@Inf.ETHZ.CH

François E. Cellier
ETH Zürich
Switzerland
FCellier@Inf.ETHZ.CH

Abstract

This article introduces a new and freely available Modelica library, called *Wheels and Tires*, for modeling wheels and tires. The contained models are intended to be used in vehicle simulations, where computational performance is a major concern. Semi-empirical single contact point models are well suited for this kind of applications and are therefore applied in the presented library.

The *Wheels and Tires* library provides a tool to quickly build custom tire models, and allows a convenient customization of existing models. This is achieved by a modular and expandable design system utilizing well established models. In addition, a set of ready-made models is provided to allow a quick insight in the used modeling structure and to enable a direct application in vehicle models. The final version of the library will be published as a free library via the Modelica website as well as the website of F. E. Cellier.

Keywords: object-oriented tire modeling; object-oriented tyre modelling; semi-empirical tire model; tire decomposition

1 Introduction

During the last decades a fairly large number of tire models of varying levels of complexity suiting basically differing fields of application have been developed. These range from simple non-slipping tires to very complex FEA (finite element analysis) models for performance prediction [6].

The library developed is intended to be used in simulations that cover entire vehicles, therefore computational effort is an important issue. Hence, the selection of the appropriate level of detail for the used models is essential for the overall simulation performance. Semi-empirical single contact point models provide a very good trade-off between accuracy and computational effort. Such models are based on physical considerations, like those emerging from multibody dynamics. These physical aspects get enhanced with empirical formulas representing measurement results that cover e.g. friction and slip characteristics. Two of these semi-empirical models are commonly accepted and widely used. These are TMeasy by G. Rill [11] and the magic-formula model by H. B. Pacejka [10]. However, both are often implemented in a flat and mainly unstructured fashion, which makes them difficult to understand and maintain. Customizing these models for particular situations or expanding them in order to cover new aspects of tires can be cumbersome and is often error-prone.

A paper by D. Zimmer and M. Otter [14] builds on the previously mentioned models and demonstrates how models of varying levels of complexity can be integrated within the object-oriented framework of Modelica. However, the object orientation in these models limits itself primarily to their external interfaces. The models themselves continue to be mostly flat. For instance the most complex tire model created, defines approximately 200 equations [14] and is a good example showing the difficulties that arise from the common flat structure.

Another example for a quite flat structure can be found in the freely available but outdated Vehicle Dynamics library [2]. There, a wheel-base model gets extended with friction models of [11] and [10], but not much further effort was spent regarding object-orientation.

In [1], a tire model is modularized in hub, belt and road elements. A further enhancement is made in [5] by redesigning the model's structure as well as enabling uneven road surfaces and losing contact to the ground due to enhanced vertical dynamics. This modularization is well defined, but still the different aspects of friction are summarized in the *Tyre-Road* class

DOI: 10.3384/ecp09430029

and can not be customized easily. Moreover the libraries presented in [14], [1] and [5] are not freely available.

The newly developed library takes the object-orientation even further than in [5]. Therefore the focus of this research effort concerns itself less with modeling new tire properties, but more with an improved organization of existing knowledge. This will enable future modelers to conveniently customize the models to their own purposes.

2 Basic Considerations

2.1 A Closer Look at Tires

In motion dynamics of vehicles the forces exerted by the tire-road contact are of major importance. This section is intended to provide basic knowledge about tires buildup. For a more detailed information the reader is referred to [8], [9], [10] or [11].

The modern tire is a complex construction resulting from clashing requirements. They basically have to carry the vertical load, transmit forces to accelerate (and slow down) the vehicle and generate cornering forces to guide the vehicle through curves securely. This has to be fulfilled under a large variety of environmental conditions with a long life time ensured. The rolling resistance has to be as small as possible, with damping and acoustic properties suiting modern demands. As one can imagine there is no optimal solution to this problem resulting in a large variety of different tires for varying demands.

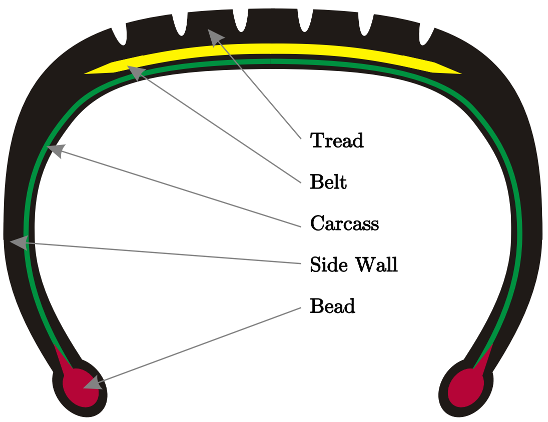


Figure 1: Basic structure of a tire.

A quite basic design example for a tire is depicted in Figure 1. For today's passenger cars steel-belt tires are used exclusively, which differ in construction only marginally, when treated from such a basic point of view. On the inside of the tire a coating (not depicted in Figure 1) inherits the function of the tube, preventing the over-pressurized air to leak to the outside. The *Bead* is usually built of steel wire with synthetic rubber components, ensuring a tight fit of the tire on the rim, allowing a reliable operation under difficult conditions e.g. when driving over a curbstone. The rubber elements building the sidewall strongly affect the vertical dynamics of the tire and are important when it comes to handling precision and stability. The *carcass* is the element absorbing the tension from the inflation pressure. Therefore, it has to be protected from damage, which is ensured by the *side wall*. The *tread* is responsible for the force generation by establishing a reliable contact to road and is therefore a very central element of the tire. Its composition is a major factor when it comes to the frictional properties of the tire. The *tread* is reinforced by the steel *belt* that enhances mileage and reduces the rolling resistance. Overall a mixture of more than 20 rubber composites form the tire, which makes them quite difficult to describe as well as enabling the tire engineer to adjust the tire properties to different needs.

2.2 Definition of Coordinate Systems

Figures 2 to 5 are intended to present basic assumptions made. Figure 2 shows the two coordinate systems used to describe the orientation of a wheel. The contact point's coordinate system is described by e_{Long}, e_{Lat} and e_N, whereas e_{Long}, e_{Axis} and e_{Plane}[1] form the rim's system. Figure 4 shows the standardized

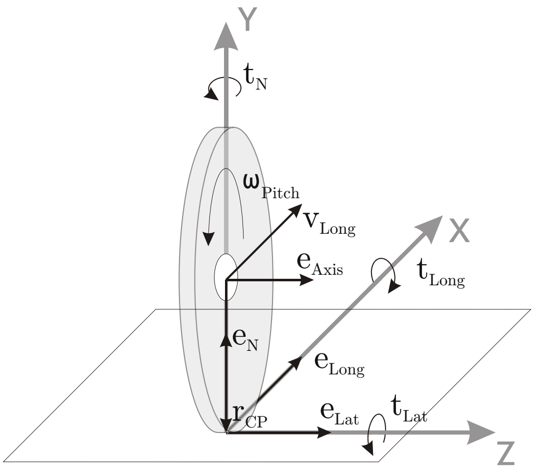


Figure 2: The unitary vectors of the tire without a lean angle and the vector r_{CP} pointing from the rim's center to the contact point.

[1] Pointing in the opposite direction of e_N in Figure 2 and shown in Figure 3 and 4.

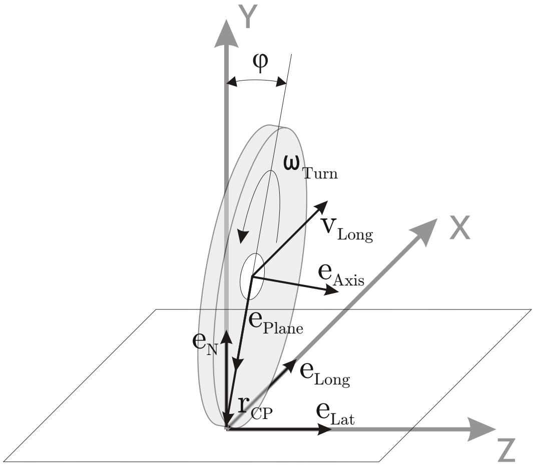


Figure 3: The unitary vectors of the tire with a lean angle φ of 10°.

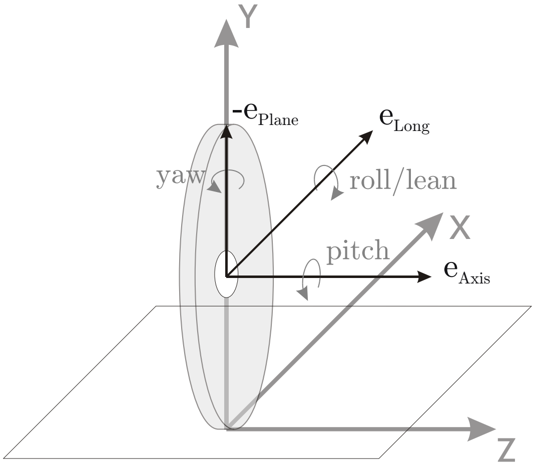


Figure 4: Standardized names of the angles and angular velocities.

names of the angels and their corresponding angular velocities.

An enlarged view of the contact point is depicted in Figure 5. It shows important properties like the translational velocity of the contact point in longitudinal (v_{Long}) and lateral (v_{Lat}) directions, the overall velocity v and the slip angle α. The sliding velocity in longitudinal $v_{SlipLong}$ direction is calculated by $(\omega \times r_{CP}) \cdot e_{Long} + v_{Long}$.

3 Object-oriented Tire Modeling

Section 3.1 is intended to demonstrate the considerations that lead to the actual structure of the tire model. Afterwards Section 3.2 explains the resulting structure from the modeler's point of view. Sections 3.3 to 3.10 shallowly introduce the classes forming the tire.

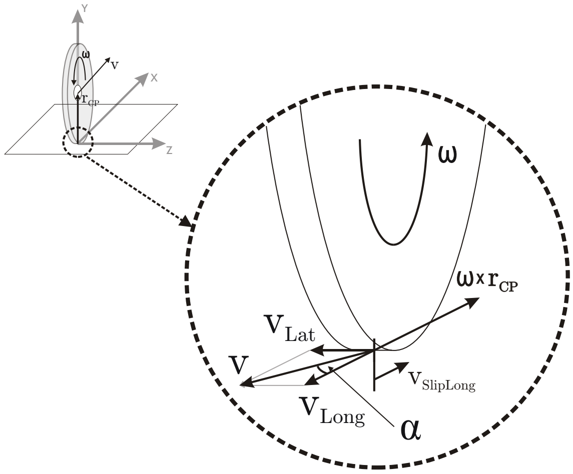


Figure 5: An enlarged view of the contact point with sliding velocities.

3.1 Decomposition into Objects

Thinking about wheels, the first division of the model is quite obvious, as there are two physical components: the rim and the tire. The rim does not need to be split up into further objects, as its properties can be modeled in a quite simple fashion in a semi-empirical tire model. This and the main properties of tires regarding modeling are shown in Figure 6.

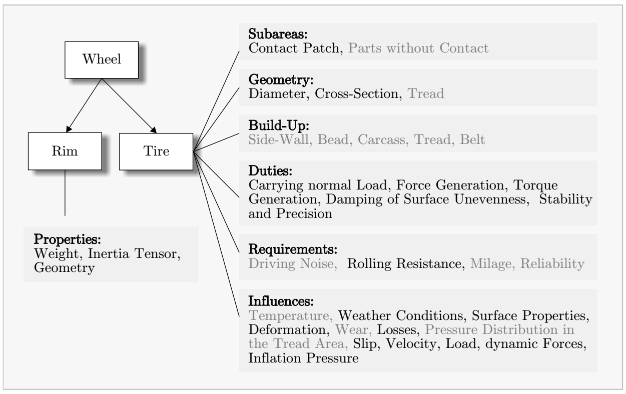

Wheel
Rim
Tire
Properties:
Weight, Inertia Tensor, Geometry
Subareas:
Contact Patch, Parts without Contact
Geometry:
Diameter, Cross-Section, Tread
Build-Up:
Side-Wall, Bead, Carcass, Tread, Belt
Duties:
Carrying normal Load, Force Generation, Torque Generation, Damping of Surface Unevenness, Stability and Precision
Requirements:
Driving Noise, Rolling Resistance, Milage, Reliability
Influences:
Temperature, Weather Conditions, Surface Properties, Deformation, Wear, Losses, Pressure Distribution in the Tread Area, Slip, Velocity, Load, dynamic Forces, Inflation Pressure

Figure 6: Composition of wheels and properties of tires. Properties depicted in gray are neglected in the presented tire model.

The tire does require a much closer look concerning its properties. Modeling every single of the properties shown in Figure 6 by a class of its own is an infeasible task, suggesting a certain grouping of properties. Due to the semi-empirical single contact point model one constraint is fixed. There has to be a model describing the contact point. It is named the *Contact Physics* model.

One of the most challenging tasks when modeling a tire, is to calculate the forces the tire excites in different driving situations. There are several different models trying to describe the relations resulting in the forces that act on the contact point. Hence a decision was made to create a class that gathers the different effects that are responsible for the generation of forces and torques acting on the contact point. Due to its origin it is called the *Friction* class, resulting in Figure 7.

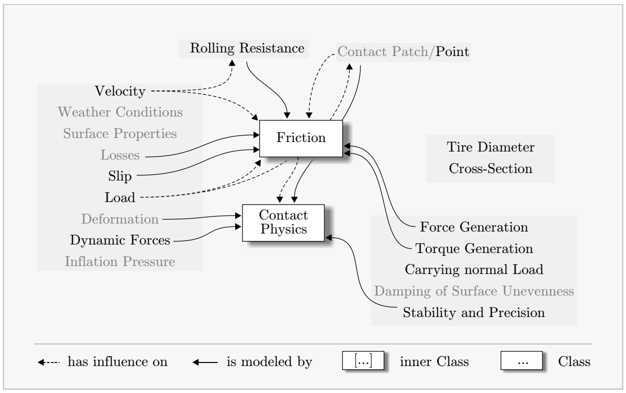


Figure 7: Properties from Figure 6 shown in relation to the corresponding *Tire Component* classes which are closely related with the semi-empirical contact point model. Properties depicted in gray are considerably simplified in the model.

Still, absolutely basic properties of the tire are not yet part of the model as shown in Figure 7. The probably most obvious are the *Tire Diameter* and the *Cross-Section*. These properties basically define the geometry of the tire, which shall be changeable conveniently in the final tire model, allowing basically different geometric properties of the tire. Therefore a *Geometry* class is introduced, defining the positional relation between the tire hub and the contact point and some other properties.

The next very basic duty the tire has to fulfill, is the *Carrying of normal Load* resulting in a normal force. Due to the changing requirements to these aspects, the effects are described in a class named *Vertical Dynamics*.

Until now, all possible tire models would be totally rigid. To enable a certain deformation of the tire, the *Belt Dynamics* class is introduced. It allows a flexibility of the still rigid tire, defined by the *Geometry* class, related to the tire's hub.

The wheel is now modularized into six classes including the *Rim* class which is not depicted in Figure 8, as it was shown in Figure 6. Section 3.2 explains the implemented version of these classes from the modeler's point of view. It also explains why the final model of the tire contains seven classes, introducing a *Center to Contact Point* class.

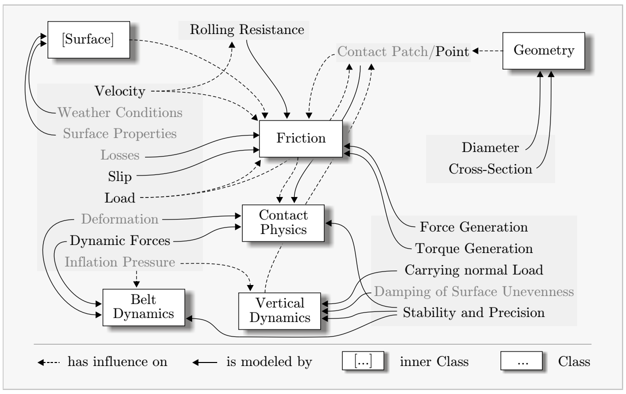


Figure 8: Final decomposition of the tire.

Still there is one class found in Figure 8 that has not been introduced until now. This is justified as it is an *inner* model and does not form a part of the actual tire model. It realizes uneven surfaces and allows a position-depending friction coefficient. This model is described in Section 5.

3.2 The Tire Model's Structure

The tire is split up into seven objects shown in Figure 9. It consists of objects modeling

- *Vertical Dynamics*,
- *Friction*,
- *Geometry*,
- *Contact Physics*,
- the translation from *Center to Contact Point*,
- *Belt Dynamics* and
- the *Rim*.

The single classes are described in the following sections. Here the relations between the classes shall be illustrated.

The connection to the superordinate vehicle model is established by the three-dimensional frame named *Tire Hub* depicted in Figure 9. The *Tire Hub*, a standard element of the *MultiBondLib* [13], is rigidly connected to the model of the *Rim*. The rim's frame connected to the *Tire Hub* therefore models the center-point of the rim. The "output" of the *Rim* again models the center-point of the *Rim*, and is connected to the *Belt Dynamics* model. In this model the relative movement between the rigid belt and the rim can be described. The "output" of the *Belt Dynamics* is again

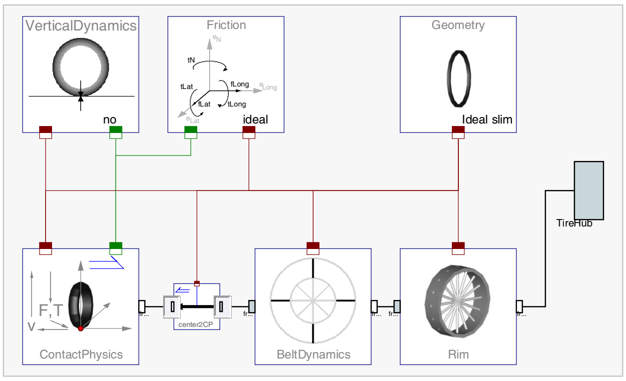


Figure 9: Model of the ideal tire showing the seven used objects and the communication structure on the top level. The *Tire Bus* is colored red whereas the *Contact Point Connector* is green.

positioned at the center-point of the belt. Therefore an element is needed to realize the translation from this point to the contact point. As this cannot be modeled by a standard element, the *Center To Contact Point* model has been created. It connects the *Contact Physics* model that applies forces and torques to the contact point. This lower part of the model represents the mechanically connected part of the model. The upper three classes are not directly connected by a mechanical connector, although the *Contact Point Connector* implies kind of a mechanical connection. The *Vertical Dynamics* class determines if and how the tire is able to lift from the ground and how it responds to normal load. The *Friction* class determines the longitudinal and lateral forces as well as the torques that act on the contact point. Finally, the *Geometry* class determines the unitary vectors shown in Figure 2 and the position of the contact point depending on the actual geometry, position and orientation of the tire.

The visualization is implemented in the corresponding object, e.g. the rim is visualized in the *Rim* object and the tire is visualized in the *Geometry* class. This makes it possible to recognize basic changes to the models in the animation directly.

All of the featured classes that model a certain type of effect are extended from the corresponding base class, ensuring that the necessary output is calculated. This guarantees also that all models stay exchangeable not depending on the implementation of the class. In this way, the user can add new classes or adapt existing ones being sure that the other elements stay unchanged and remain exchangeable.

3.3 Rim Class

The *Rim Class* is a rather simple model, as the rim can be modeled ideally just consisting of a body that has a mass and an inertia tensor.

3.4 Friction

For the sake of object oriented modeling, the different modeled effects are put into subclasses (see Figure 10 as an example) that are of varying complexity. The communication between the classes is established using *inner/outer* statements. Therefore no connections between the sub-models are visible in Figure 10.

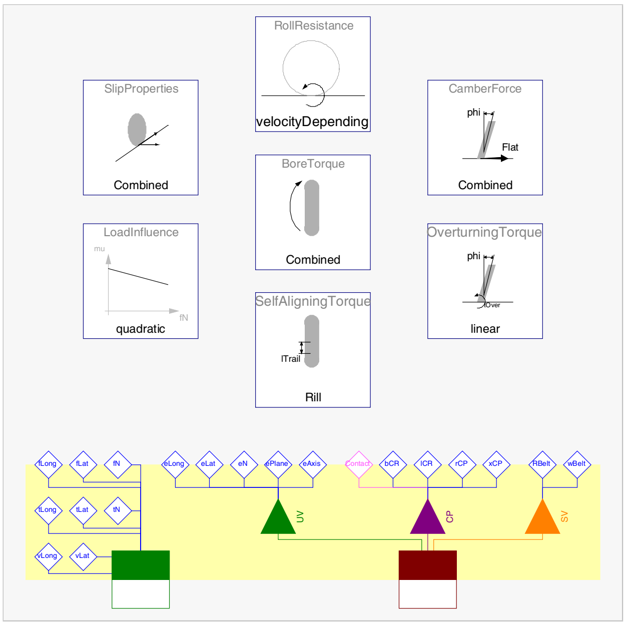


Figure 10: Model of the advanced friction.

To use a certain combination of frictional effects in a tire model, a class gathering these effects has to be created as shown in Figure 10. This class can then e.g. replace the *Ideal Friction* depicted in Figure 9. In the library three different combinations of *Friction* classes are composed, forming an *Ideal Friction*, one simple *Dry Friction With Rolling Resistance* but without e.g. *Bore Torque* or *Camber Force* and one *Advanced Friction* that is shown in Figure 10. New frictional models can be created easily, which is the reason why only three frictional classes were included in the library.

3.5 Geometry

All geometric classes have two main tasks to fulfill. The first is to determine the contact point properties including the vector *rCP* that points from the center of the rim to the contact point and the penetration depth.

Secondly the unit vectors shown in Figure 2 get computed by the *Geometry* class. To enable that functionality it utilizes the *Surface*'s functions *get_eN* and *get_elevation* establishing the connection from the tire to the surface.

Three different *Geometry* classes are provided for ideally *Slim*, *Circular* tires with cross-section being modeled as a semi-circle, and a "*Belt*" profile with side walls and a sector of a circle modeling the tread area.

3.6 Contact Physics

The *Contact Physics* model applies forces and torques on the contact point, as well as measuring its velocities. All these physical quantities are connected to the models determining frictional and vertical dynamic properties via the *Contact Point Connector*. Measuring and setting of speeds and forces has to be done in an a-casual way as ideal models set velocities rather then apply forces.

The second property determined by the *Contact Physics* class is the dynamic behavior of the contact point. It can introduce flexibility of the contact point in longitudinal and lateral direction.

3.7 Center to Contact Point

The *Center To Contact Point* class is a model without a nice physical interpretation. It is kind of a *Fixed Translation* element known from the *Modelica Standard Library*. The model is built using multi-bond graphs and therefore derived from the *Fixed Translation* in the *MultiBondLib* [13]. It describes the connection between the contact point and the belt's center point.

3.8 Vertical Dynamics

The models in the *Vertical Dynamics* package determine the behavior of the tire normal to the surface. The normal force is related to the penetration depth computed in the *Geometry* class. The *Vertical Dynamics* models can either set the penetration depth to a certain value, or use it as a base for the calculation of the normal force f_N.

There are basically three different approaches to model vertical dynamics in the library. One is to not allow any elevation from the ground or penetration into it introducing a holonomic constraint. The second utilizes an *ElevationGap* model that compensates forces of a attached 1D mechanical model when the tire lifts from the ground. It is a derivation from the *ElastoGap* model found in the *BondLib* [7]. It makes the vertical dynamics easily changeable by a modification of the 1D mechanical system. The third is a derivation of the *ElastoGap* model of the *Modelica Standard Library 3.0*. It overcomes the sticking effect of the *ElevationGap* but is harder to adapt to different dynamics.

3.9 Belt Dynamics

The models described in this section allow the tire to have a certain flexibility. This is realized by connecting an ideal (virtual) belt to the ideal rim in a flexible fashion. All models except the *Rigid Belt* model add a considerable amount of complexity to the simulation. Different models are provided to realize the dynamic behavior. One allows translation of the belt in longitudinal and lateral direction, another models a rotational degree of freedom around the axis of rotation, both defining the dynamics by 1D mechanical elements. The last model is the most complex with the dynamics defined by four ideally stiff translational elements for rim and belt respectively, connected by 3D spring damper systems.

3.10 Communication Structure

The tire's objects compute a bunch of different variables, some of which are used in different objects as well. The *Tire Bus*, with a connector as shown in Figure 11, gathers the variables used in most of the objects. Additionally a division into records of similar variables ensures a better overview and allows a more convenient graphical connection. For each of these sets of variables separate inputs and outputs have been created. This makes it possible to show, in which objects variables are computed or just used (see Figure 11). The second communication structure is the connector that contains the velocities as well as the forces and torques that act on the contact point. It

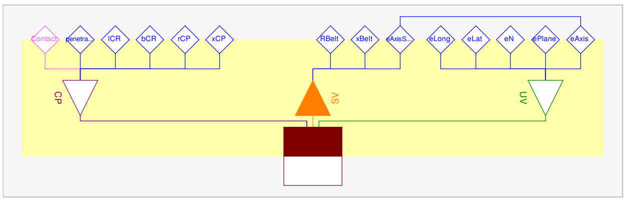


Figure 11: Separation of the *Tire Bus* connector (lower quadratic element) within the *Geometry Class* in an input for *Sensor Values* (SV) and outputs for *Contact Properties* (CP) and *Unit Vectors* (UV). The rhombuses define protected variables used in the model.

is named *Contact Point Connector*. This connector is implemented in an a-causal manner due to the requirements of this connection. It can be found connecting *Contact Physics*, *Vertical Dynamics* and *Friction*. Both bus connectors are illustrated in Figure 9.

4 Provided Tire Models

All tires are basically built up like the ideal tire in Figure 9. In the library eleven ready-made tires are provided for a quick application and a better understanding of the model structure. Four models of slim tires include three rigid versions and one with a dynamically behaving contact point. Two tires with semicircular cross section are provided, one rigid and the other having rotational dynamics. The so-called belted tires feature three rigid models differing in frictional behavior. The other two belted tires behave dynamically.

5 Environment

The *Environment* in the current version of the library is limited to even or uneven but slowly changing surfaces, which is a limitation arising from the single contact point model. Basically the method to be implemented has to be able to compute elevation (the *y* coordinate shown in Figure 12) and the normal vector on the surface. There are many significantly different approaches, whereas the one chosen is rather simple but still conveniently configurable.

5.1 Surface Base

The *Surface Base* (Section 5.1) class ensures compatibility with future enhancements. This partial inner model defines the functions necessary to calculate the behavior of the tire. These include the following functions.

- *get_eN_Base* – returning the normal vector of the surface with the actual *x* and *z* coordinates as inputs.
- *get_elevation_Base* – returns the y coordinate of the surface (*x* and *z* are inputs again).
- *get_mu_Base* – returns the frictional coefficient at the *x* and *z* coordinates of the contact point.

5.2 Surface Class

The implemented version of the *Surface* in the *Wheels and Tires* library is based on the method shown in [4]. It is an interpolation in a unit square based on four y values and the eight corresponding partial derivatives. A sketch of the unit square is shown in Figure 12. To enable the user to define more complex surfaces an arbitrary amount of interpolated elements can be combined to one surface of definable size. The only fact limiting the amount of rectangles is simulation time, especially the time consumed for compilation that rises quickly with growing surfaces.

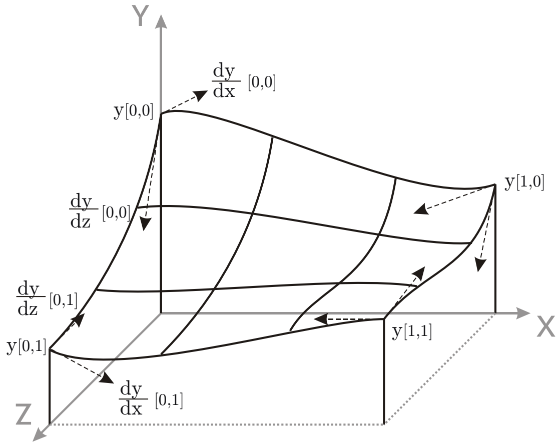


Figure 12: The basis for the interpolation used to find the elevation *y* and normal vector in one square.

6 Simulation Results

The following sections are intended to give an impression of what the simulation results look like. Therefore, results from the *Test Bench* package and from the *Example* package are depicted.

6.1 Test Bench

Figures 13 to 16 show results from different models in the *Test Bench* package. The models are used to test basic functionalities of the tire and the surface classes.

6.2 Examples

Six examples are included in the library demonstrating the application of the models in vehicles. Five of the Examples are two-wheeled (single-track) vehicles

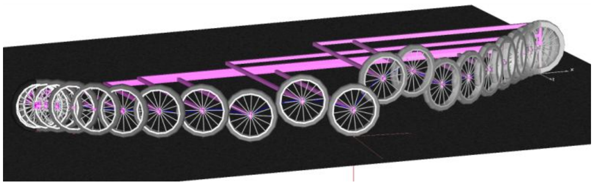


Figure 13: Tire elevating from the ground while driving a curve.

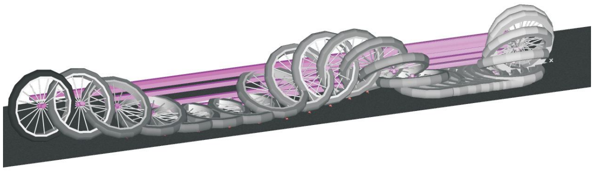


Figure 14: Tire's overturning torque *Test Bench* result.

that are provided by [12] and copied to the *Used Models* in the *Examples* package of the *Wheels and Tires* library. Three of the five examples are unsprung bicycle models composed of rigid elements exclusively. The other two models are sprung motorcycles including front and rear suspension. For both vehicles, models are provided to analyze the uncontrolled stability, e.g. with different tire properties for geometry and friction as shown in Figures 17 and 18.

The other two models are "nice to show" models with the bicycle being accelerated by a strong torque to make the front wheel lift from the ground as shown in Figure 19 and the motorcycle jumping over a gap as depicted in Figure 20.

The last example is a very basic unsprung model of a four-wheeled vehicle with a predefined steering angle profile and driving torques acting on the rear tires. An impulse in the driving torque makes the vehicle-model slide after a certain simulation time, with the model "reacting" by a full breaking, which makes the wheels slide to a standstill of the vehicle.

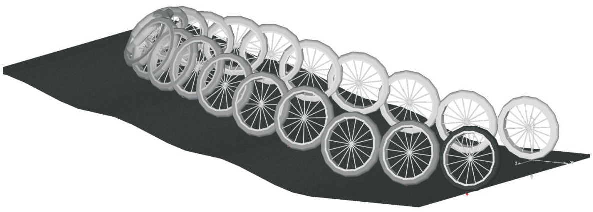


Figure 15: A tire rolling up an uneven surface with an initial lean angle.

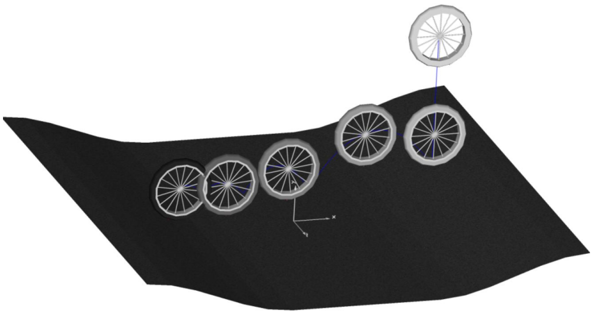


Figure 16: Tire dropping on an uneven surface.

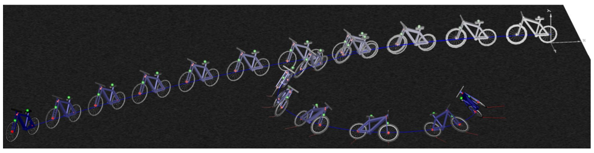


Figure 17: Two similar bicycles with non-slipping tires differing in their geometric properties. The upper bicycle is equipped with ideally slim tires, whereas the lower one driving a narrowing curve has a belted tire with a width of 2cm.

Regarding the simulation speeds of the examples it can be stated, that the bicycle models take about half of the simulated time for the computation of the result. The motorcycle jumping over the gap takes about 250s of calculation time for 16s of actual simulation. The reason for that is the more complex structure of the motorcycle in comparison with the bicycle. The again undamped four-wheeled models compute the results in a little less time than the simulation time. All models used non-dynamic tires either with *slipping* or the *advanced* friction models. The simulations have

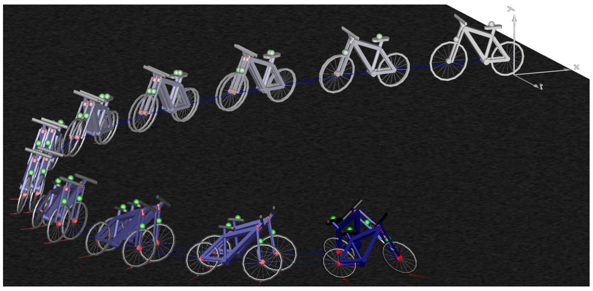


Figure 18: Two similar bicycles with slipping tires differing in their frictional properties. The one driving the inner circle is equipped with the *Advanced Friction* class, whereas the outer bicycle is equipped with the *Dry Fricition with Constant Roll Resistance* class.

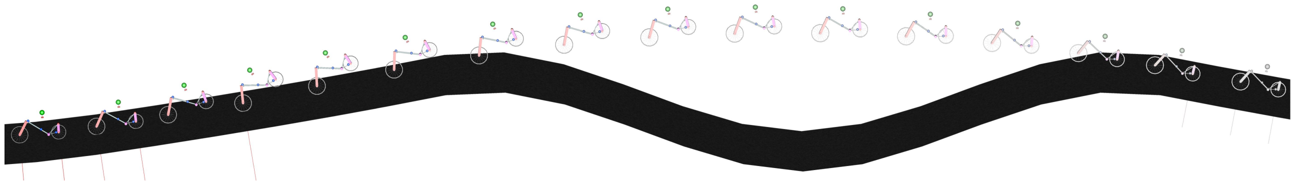


Figure 20: A motorcycle jumping over a gap.

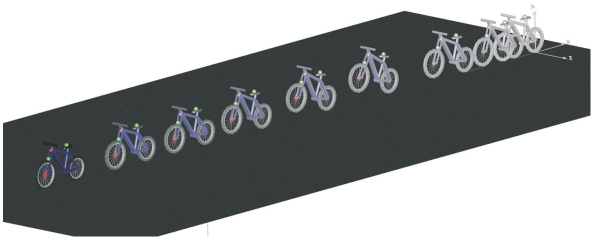


Figure 19: A bicycle with non-ideal tires accelerated by a torque on the rear tire, with the front tire lifting from the ground.

been carried out on a Intel Core2Duo clocked at 2GHz and equipped with 4GB RAM, computing 250 output intervals per second. It has to be mentioned that no big effort was spent regarding the optimization of simulation speed of the overall models. A more suitable combination of state variables would most likely lead to a considerable enhancement in speed.

7 Library Structure

This section introduces the top level packages that are contained in the *Wheels and Tires* library and are depicted in Figure 21.

The *Tire Components* package covers the classes that the *Tires* are built of. The contained sub-packages are described in Sections 3.3 to 3.10. These classes form the ready-made tires contained in the *Tires* package as one example shows for the *Ideal Tire* as depicted in Figure 9.

The *Environment* package contains the *Surface Base* as well as a possible implementation for even and uneven surfaces. It is described in Section 5. The *Test Bench* and the *Example* Package are top-level packages as well. They are provided to test the tire models' functionality and get an insight into the usage of the models. A short description can be found in Sections 6.1 and 6.2 respectively. Finally the *Visualization* package gathers a few models used to enable the animation of all necessary parts of the library.

8 Conclusion

To sum up, the library enables a quick and convenient building of easily customizable tire models. There are some further enhancements possible but the structure of the tire models should persist as it is well expandable, fulfilling the major requirements that were demanded at the beginning of the work. Still the lack of comparison to real applications and measurement data is considered to be a drawback as some minor malfunctions could probably not be identified.

For the use of the models in real-time applications a further optimization of the computational efficiency would be desirable to ensure quick enough simulation. Furthermore a prediction of the influences that different objects have on the computational effort of the overall tire would be of advantage.

A more detailed description of the library can be found in the corresponding master's thesis [3].

References

[1] J. Andreasson and J. Jarlmark. Modularised tyre modelling in modelica. In *Proceedings of the Second International Modelica Conference, Oberpfaffenhofen, Germany*, pages 267–274, 2002.

[2] Johan Andreasson. Vehicledynamics library. In *Proceedings of the Third International Modelica Conference, Linköping, Sweden*, pages 11–18, 2003.

[3] Markus Andres. Object-oriented modeling of wheels and tires in dymola/modelica. Master's thesis, Vorarlberg University of Applied Sciences, 2009.

[4] G. Aumann and K. Spitzmüller. *Computerorientierte Geometrie*. BI-Wiss.-Verl, 1993.

[5] Mats Beckmann and Johan Andreasson. Wheel model library for use in vehicle dynamic studies. In *Proceedings of the third Modelica Conference, Linköping, Sweden*, pages 385–392, 2003.

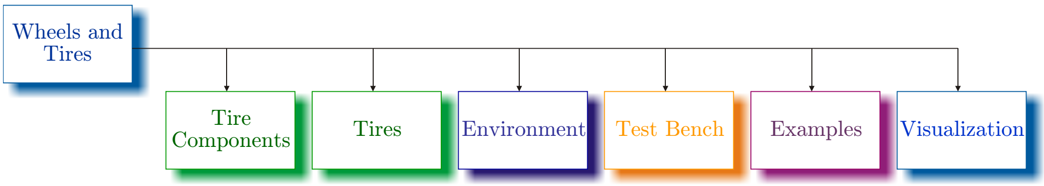


Figure 21: The library's top level packages.

[6] P. Bohara, A. Saha, P. Ghosh, and M. Roopak. Tyre perfomance prediction through fea. Hasetri - Hari Shankar Singhania Elasotmer and Tyre Research Institute, 2008.

[7] F. E. Cellier and À. Netbot. The modelica bond-graph library. In *Proceedings of the 4th International Modelica Conference, Hamburg*, pages 57–65, 2005.

[8] John C. Dixon. *Tires, Suspension and Handling*. Cambridge University Press, 1996. Second Edition.

[9] Günter Leister. *Fahrzeugreifen und Fahrwerkentwicklung*. Vieweg + Teubner, 2009. 1. Auflage.

[10] Hans B. Pacejka. *Tyre and Vehicle Dynamics*. Butterworth-Heinemann, 2006. Second Edition.

[11] Georg Rill. *Simulation von Kraftfahrzeugen*. Vieweg-Verlag, 2007. genehmigter Nachdruck.

[12] Thomas Schmitt. Modeling of a motorcycle in dymola/modelica. Master's thesis, Vorarlberg University of Applied Sciences, 2009.

[13] Dirk Zimmer. A modelica library for multibond graphs and its application in 3d-mechanics. Master's thesis, ETH Zürich, 2006.

[14] Dirk Zimmer and Martin Otter. Real-time models for wheels and tires in an object-oriented modeling framework. Accepted for publication in Vehicle Dynamics, 2009.

A Virtual Motorcycle Rider Based on Automatic Controller Design

Thomas Schmitt
Vorarlberg Univ. of Appl. Sc.
Austria
Thomas.Schmitt@students.fhv.at

Dirk Zimmer
ETH Zürich
Switzerland
DZimmer@Inf.ETHZ.CH

François E. Cellier
ETH Zürich
Switzerland
FCellier@Inf.ETHZ.CH

Abstract

This paper introduces a new and freely available *Modelica* library for the purpose of simulation, analysis and control of bicycles and motorcycles (single-track vehicles). The library is called *MotorcycleLib* and focuses on the modeling of virtual riders based on automatic controller design.

For the single-track vehicles, several models of different complexity have been developed. To validate these models and their driving performance, virtual riders are provided. The main task of a virtual rider is to track either a roll angle profile or a pre-defined trajectory using path-preview information. Both methods are implemented and several test tracks are also included in the library.

Keywords: virtual rider; automatic controller design; state-space controller, bicycle and motorcycle modeling; pole placement

1 Introduction

Among the vehicle models, models of bicycles and motorcycles turn out to be particularly delicate. Whereas a four-wheeled vehicle remains stable on its own, the same does not hold true for a single-track (two-wheeled) vehicle. For this reason, the stabilization of such a vehicle, a control issue, requires special attention.

A key task for a virtual rider is to stabilize the vehicle. To this end, a controller has to generate a suitable steering torque based on the feedback of appropriate state variables of the vehicle (e.g. lean angle and lean rate). One major problem in controlling single-track vehicles is that the coefficients of the controller are strongly velocity dependent. This makes the manual configuration of a controller laborious and error-prone. To overcome this problem, an automatic calculation of the controller's coefficients is desired. This calculation requires an eigenvalue analysis of the corresponding uncontrolled vehicle which is performed in order to determine the self-stabilizing area. The library includes the means for such an analysis and its results can be interpreted by three different modes that qualitatively describe the vehicle's motion [12]. This enables a convenient controller design and hence several control laws that ensure a stable driving behavior are provided. The corresponding output represents a state feedback matrix that can be directly applied to ready-made controllers which are the core of virtual riders. The functionality of this method is illustrated by several examples in the library.

In 2006, F. Donida et al. introduced the first Motorcycle Dynamics Library in Modelica [5] and [4]. The library focuses on the tire/road interaction. Moreover different virtual riders (rigidly attached to the main frame or with an additional degree of freedom (d.o.f.) allowing the rider to lean sideways) capable of tracking a roll angle and a target speed profile are presented. Until now these virtual riders include fixed structure controllers only [4]. This means that virtual rider stabilizes the vehicle only correctly within a small velocity range.

Using the automatic controller design functions provided by the *MotorcycleLib* this major deficiency can be overcome. Furthermore, to validate the motorcycle's performance, the virtual rider is capable of either tracking a roll angle profile (open-loop method) or a pre-defined path (closed-loop method).

2 Bicycle and Motorcycle Models

The mathematical modeling of single-track vehicles is a challenging task which covers a wide range of models of varying complexity. The library provides several single-track vehicle models of different complexity. The models are composed of multibody el-

DOI: 10.3384/ecp09430030

ements and are based on bond graphs [2] and multibond graphs [17]. Basically two types of models are provided. Some include out-of-plane modes only, while others include both in-plane and out-of-plane modes. Roughly speaking, out-of-plane modes are related to stability and handling of single-track vehicles whereas in-plane modes are dealing with riding comfort. The wheels used in this library are either provided by D. Zimmer's *MultiBondLib* [17] or by M. Andres' *WheelsAndTires* library [1]. The former are ideal, whereas in the latter models non-ideal effects such as slip can be considered. For the bicycle, both 3 and 4 d.o.f. out-of-plane mode models are included in the library. The former are composed of four rigid bodies, namely a front frame, a rear frame including a rigidly attached rider and two wheels, connected via revolute joints. The wheels are infinitesimally thin (knife-edge). The latter introduce an additional d.o.f. that allows the rider's upper body to lean sideways. Both models are based on those introduced by Schwab et al. [12] and [11].

The out-of-plane mode motorcycle model is a 4 d.o.f. model that is based on a model established by V. Cossalter [3]. Basically, V. Cossalter's model is the same as the one introduced by R. S. Sharp in 1971 [13]. This model allows a lateral displacement of the rear frame since the wheels are no longer ideal. Due to the fact that the wheels of D. Zimmer's *MultiBondLib* [17] are ideal, the model is reduced to 3 d.o.f.Later, with reference to the *WheelsAndTires* library [1], it is possible to consider non-ideal effects of wheels and tires and thus simulate the lateral displacement of the wheels caused by tire slip. The animation of a 3 d.o.f. motorcycle is depicted in Figure 1. To incorporate in-plane modes two more complex models are included in the library. The first model was originally developed by C. Koenen during his Ph.D. Thesis [9]. R. S. Sharp and D. J. N. Limebeer introduced the SL2001 model which is based on Koenen's model [15]. They reproduced Koenen's model as accurately as possible and described it by means of multibodies. The model developed in this library is based on the SL2001 motorcycle. The second model is based on an improved more state-of-the-art version of the former developed by R. Sharp, S. Evangelou and D. J. N. Limebeer [14]. A very detailed description of these models can be found in S. Evangelou's Ph.D. Thesis [6]. Such models are composed of a front frame including the front forks and handle bar assembly, a rear frame including the lower rigid body of the rider, a swinging arm including the rear suspensions, the rider's upper body, a front and a rear wheel. Furthermore several additional freedoms due to twist frame flexibility at the steering head, suspensions, non-ideal tire models and aerodynamics are taken into account. The animation of the SL2001 model is depicted in Figure 2. In contrast to the former models each body is created in a fully object-oriented fashion. As with the out-of-plane models these models only include all degrees of freedom in combination with the *WheelsAndTires* library. Without this library several freedoms are inhibited.

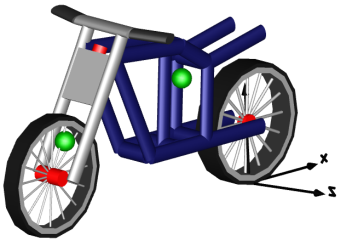


Figure 1: Animation of a 3 d.o.f. motorcycle model

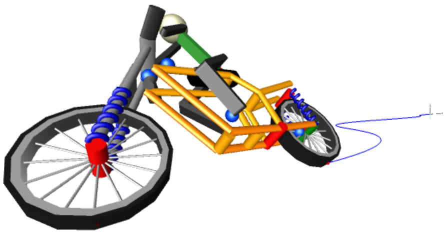


Figure 2: Animation of the SL2001 motorcycle model

It is important to keep in mind that vehicles in combination with ideal wheels include so called holonomic constraints. Such constraints are based on location and in case of single-track vehicles prevent them from sinking into the ground.

3 Eigenvalue Analysis

Due to geometry and gyroscopic forces Klein and Sommerfeld [8] found out that a single-track vehicle is self-stabilizing within a certain velocity range. That is, the vehicle performs a tail motion in the longitudinal direction. Below this range the steering deflections caused by gyroscopic forces are too small in order to generate enough centrifugal force. Thus the amplitude of the tail motion increases and the vehicle falls over. Although, these interactions are damped by the trail[1] it is still impossible to achieve stable behavior. Hence

[1] The trail is the distance between the front wheel contact point and the point of intersection of the steering axis with the ground line (horizontal axis).

the rider has to apply a steering torque to ensure that the vehicle stays upright. Above this range, for high speeds, the gyroscopic forces are almost unnoticeable for the rider. That is, the amplitude of the tail motion is close to zero. More precisely, although the vehicle feels stable, after a certain time, it falls over like a capsizing ship. However, by applying a steering torque it is rather simple to stabilize the vehicle. In most cases it is sufficient that one solely touches the handle bars in order to compensate for the instabilities.

An eigenvalue analysis is performed in order to determine the self-stabilizing range of an uncontrolled bicycle or motorcycle. For this purpose the state variables of the vehicle that are responsible for stability are of interest. These are the steer angle δ, the lean angle ϕ, and their derivatives.

$$x = \begin{pmatrix} \delta \\ \dot{\delta} \\ \phi \\ \dot{\phi} \end{pmatrix}$$

In case of vehicles with an additional d.o.f. allowing the rider's upper body to lean sideways, the state variables γ and $\dot{\gamma}$ are also taken into account, where γ is the lean angle of the rider's upper body relative to the rear frame and $\dot{\gamma}$ the corresponding lean rate. All the other state variables (e.g. lateral- and longitudinal position) of the state vector have no influence on the stability of single-track vehicles. Now, the eigenvalues (one for each state variable) are calculated as a function of the vehicle's forward velocity $\lambda = f(v)$ (e.g. $v = 10ms^{-1}$ to $v = 50ms^{-1}$). Thus, for each specific velocity the model is linearized. The result of such an analysis are three different velocity ranges at which the motion of the vehicle changes qualitatively. Figure 3 depicts a typical result of such an analysis. The first velocity range is below the stable region, the second one is within, and the third one above the stable region. Positive eigenvalues, or more precisely eigenvalues with a positive real part, correspond to unstable behavior whereas eigenvalues with a negative real part correspond to stable behavior. Eigenvalues including an imaginary part emphasize that the system is oscillating whereas eigenvalues without an imaginary part are non-oscillating. A stable region exists, if and only if all real parts of the eigenvalues are negative. In the following, the modes of single-track vehicles are explained with reference to Figure 3.

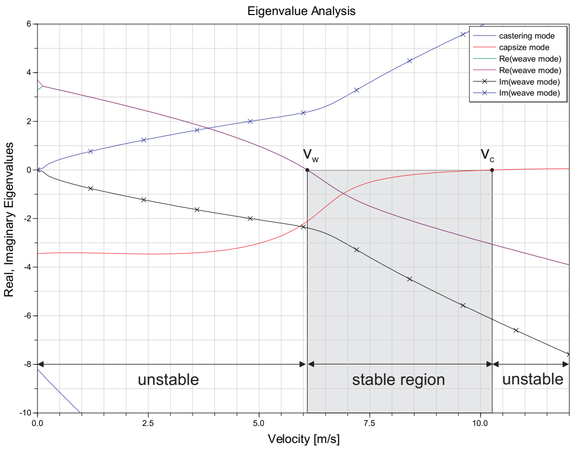


Figure 3: Result of the eigenvalue analysis for a 3 d.o.f. motorcycle model. The stable region is determined by eigenvalues with a negative real part. Here it is from $v_w = 6.1ms^{-1}$ to $v_c = 10.3ms^{-1}$.

3.1 Weave Mode

The weave mode begins at zero velocity. This mode is non-oscillating in the beginning and after a certain velocity passes over into an oscillating motion. The non-oscillating motion at very low speeds states that the bicycle is too slow to perform a tail motion and thus falls over like an uncontrolled inverted pendulum. As soon as it passes a certain value of approximately $v_w = 0.12ms^{-1}$ the real parts of the eigenvalues merge and two conjugate complex eigenvalues appear. Hence, a tail motion in the longitudinal direction emerges. This motion is still unstable but becomes stable as soon as the real parts of the eigenvalues cross zero. This happens at a velocity of about $v_w = 6.1ms^{-1}$. For all velocities greater than v_w this motion is stable.

3.2 Capsize Mode

The capsize mode is a non-oscillating motion that corresponds to a real eigenvalue dominated by the lean. As soon as the bicycle speed passes the upper limit of the stable region of about $v_c = 10.3ms^{-1}$, it falls over like a capsizing ship. However, above the stable region the bicycle is easy to stabilize although the real eigenvalue is positive. In the paper [12] of Sharp et al. this motion is called "mildly unstable" as long as the absolute value of the eigenvalues is smaller than $2s^{-1}$.

3.3 Castering Mode

The castering mode is a non-oscillating mode that corresponds to a real negative eigenvalue dominated by the steer. In this mode the front wheel has the tendency to turn towards the direction of the traveling vehicle.

4 Controller Design

4.1 An Introduction to State-Space Design

In general, the state-space representation of a linear system is given by:

$$\dot{x} = A \cdot x + B \cdot u, \quad x(0) = x_0 \tag{1}$$

$$y = C \cdot x + D \cdot u \tag{2}$$

where x is a $(n \times 1)$ state vector, y is a $(m \times 1)$ output vector, A is referred to as system matrix with a dimension of $(n \times n)$, B is a $(n \times r)$ input matrix, C is a $(m \times n)$ output matrix and D the "feedthrough" matrix with a dimension of $(m \times r)$. Usually D is set to zero, except if the output directly depends on the input. The state vector x at time $t = 0$ includes the initial conditions, sometimes referred to as initial disturbances x_0. The block diagram of a system described in state-space is illustrated in Figure 4. One major advantage of state-space control compared to classic control is that each state of the system can be controlled. In order to control the system, the state vector x is fed back. The state feedback control law for a linear time-invariant system is given by:

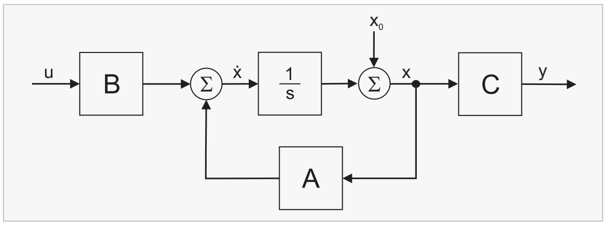


Figure 4: Block diagram of a system described in state-space

$$u(t) = -F \cdot x(t) \tag{3}$$

where F is a constant matrix.

By substituting u of Equation 1 with Equation 3 the state equation results in

$$\dot{x} = A \cdot x - B \cdot F \cdot x = (A - B \cdot F)x \tag{4}$$

The block diagram of the equation above is shown in Figure 5.

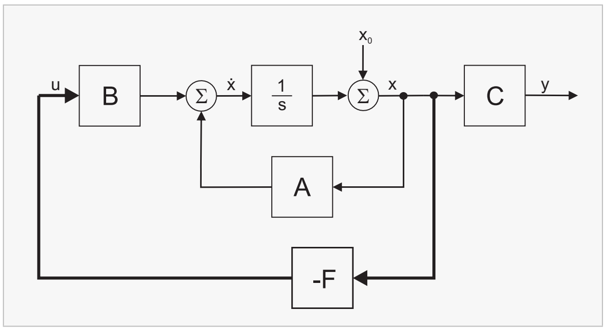


Figure 5: State feedback

The elements of the feedback matrix F have to be chosen in such a way that the *initial disturbances* $x_0(t)$ for $t \longrightarrow \infty$ converge towards zero

$$\lim_{t \to \infty} x(t) = 0 \tag{5}$$

and that the system becomes stable.

The main task of the state feedback control is to find appropriate coefficients for the feedback matrix F in order to achieve the desired dynamical behavior of the system. One method that fulfills all the requirements is the so-called pole placement technique (refer to [7]). Figure 6 illustrates the graphical interpretation.

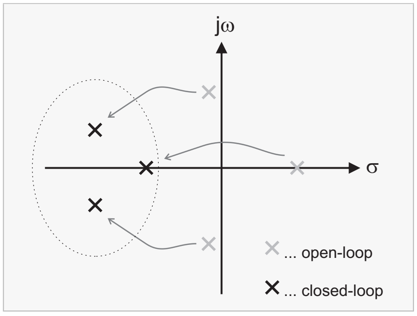


Figure 6: Graphical interpretation of the pole placement technique. Eigenvalues (poles) of the system located in the left-half plane correspond to stable behavior.

4.2 State-Space Controller Design Based on a Preceding Eigenvalue Analysis

The library includes several different stabilizing controllers. Although classic controllers and linear quadratic regulators (LQR) are included in the library, the focus lies in state-space controller design via the pole placement technique. In the simplest case the lean angle and the lean rate of the vehicle are fed back in order to generate an appropriate steering torque. However, since a physical interpretation of these eigenvalues is not possible an alternative approach is introduced in this paper. This approach is based on a preceding eigenvalue analysis. That is, exactly the same state variables, namely the steer angle δ, the lean angle ϕ, and their derivatives, are used to design the controller (see Figure 7). Of course if the upper body of the rider is movable, the states γ and $\dot{\gamma}$ are taken into account as well. Thus, a physical interpretation of the poles is available.

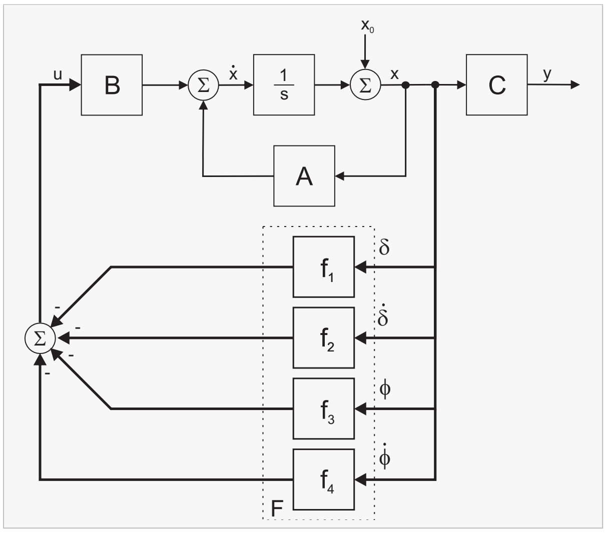


Figure 7: State-space controller based on a preceding eigenvalue analysis

As already mentioned the eigenvalues are a function of the velocity, i.e. the trajectory of each eigenvalue is thus perfectly known. With this knowledge the velocity dependent coefficients of the state feedback matrix can be conveniently calculated. To this end, three different approaches were developed. In the first approach all eigenvalues (poles) of the system are simply shifted towards the left-half plane (see figure 6) by the same value (offset). A typical result for the controlled version of the 3 d.o.f. motorcycle is illustrated in Figure 8.

Two improved control laws have been established. Both are based on solely shifting those poles towards the left-half plane that are unstable. Within the stable region the motorcycle needs no control and thus no offset. Above the stable region (for velocities greater than v_c) the behavior of the bicycle is dominated by the capsize mode. Hence, it is absolutely sufficient to shift just this pole towards the left-half plane and leave all other poles unchanged. Below the stable region, for velocities lower than v_w, the instability of the motorcycle is caused by the weave mode (see Figure 9). To ensure stable behavior the two real parts of the conjugate complex poles have to be shifted towards the left-half plane. Now, a control law for the regions below and above the stable region is set up:

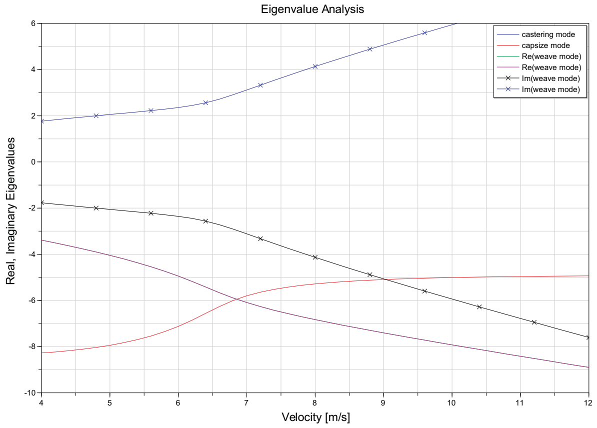


Figure 8: Result of the controller design for a velocity range from $4ms^{-1}$ to $12ms^{-1}$, where the offset is $d = 5$.

$$control\ law \begin{cases} v < v_w : & d = d_w \cdot (v_w - v) \\ v_w < v < v_c : & d = 0 \\ v_c < v : & d = d_c \cdot (v - v_c) \end{cases}$$

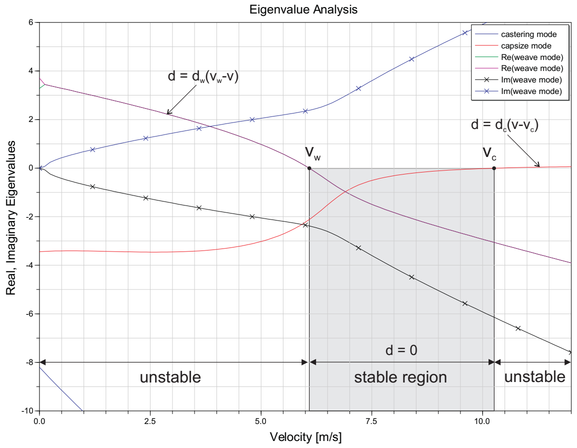


Figure 9: Controller design with reference to a preceding eigenvalue analysis. The stable region is left unchanged - below v_w, the weave mode eigenvalues are modified - above v_c, the capsize mode eigenvalue is modified.

Figure 10 shows the result of the individual controller design. Although the results of the individual controller are rather good, there is still potential for improvements. For velocities equal to v_w or v_c the eigenvalues that are responsible for stability are close or equal to zero. To be more precise, for such velocities the stability of the motorcycle is critical since a real part equal to zero has no damping. Somewhere in

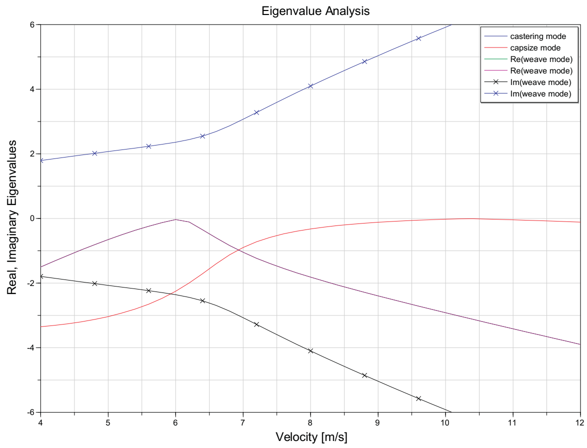


Figure 10: Result of the individual controller design for a velocity range from $4ms^{-1}$ to $12ms^{-1}$, where $v_w = 6.1ms^{-1}$, $v_c = 10.3ms^{-1}$, $d_w = 1.5$ and $d_c = 0.1$.

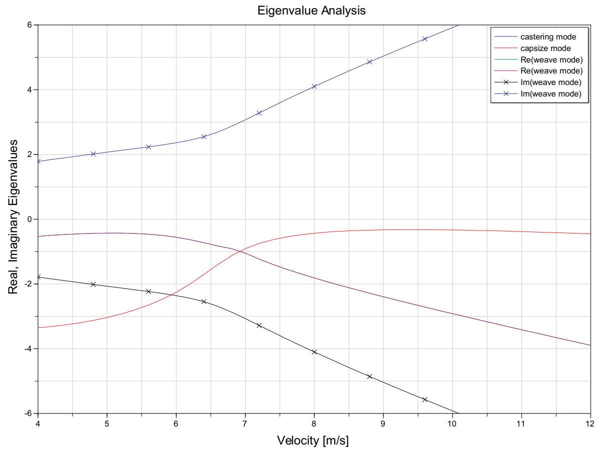


Figure 12: Result of the improved individual controller design for a velocity range from $4ms^{-1}$ to $12ms^{-1}$, where $v_i = 6.9ms^{-1}$, $d_w = 0.75$, $d_c = 0.1$ and $d_0 = 0$

the stable region the weave and the capsize mode have an intersection point v_i. Instead of the previous control law, the improved control law results in:

$$control\ law \begin{cases} v < v_i: & d = d_0 + d_w \cdot (v_i - v) \\ v_i < v: & d = d_0 + d_c \cdot (v - v_i) \end{cases}$$

A graphical interpretation of the control law is illustrated in Figure 11.

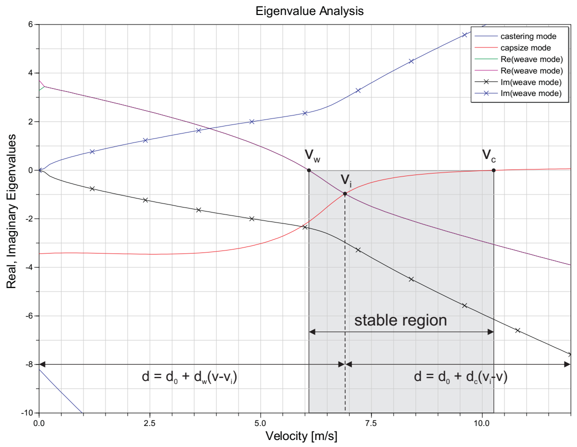


Figure 11: Controller design with reference to a preceding eigenvalue analysis. Below v_i, the weave mode eigenvalues are modified - Above v_i, the capsize mode eigenvalue is modified. In addition, the weave and capsize eigenvalues can be shifted by an offset d_0.

Figure 12 depicts the result of the improved individual controller design.

4.3 Results

The results are several pole placement functions that automatically calculate the controller coefficients, i.e. the elements of the feedback matrix. The corresponding output represents a state feedback matrix that can be directly applied to ready-made controllers. The algorithm of the functions is based on *Ackermann's formula*. Unfortunately, it is just valid for single-input, single-output (SISO) systems. In order to design a multiple-input, multiple-output (MIMO) controller, e.g. for vehicles including rider's capable of leaning sideways, a MATLAB *m-file* based on the *place*-function is provided.

Finally, the coefficients of the state feedback matrix are automatically fed into a ready-made controller which is incorporated into a virtual rider. With respect to the virtual rider the vehicle's performance can now be evaluated.

5 Development of a Virtual Rider

5.1 Roll Angle Tracking

For virtual riders capable of tracking a roll angle profile, several test tracks are provided. So far, no reference input was used, i.e. the set-value of the state variables was zero. Instead of a set-values equal to zero, the roll angle profile (e.g. of a standard 90°-curve) is fed into the virtual rider. The corresponding block diagram is illustrated in Figure 13. Since each vehicle has its own specific profile, some records including such profiles are provided. The corresponding Mod-

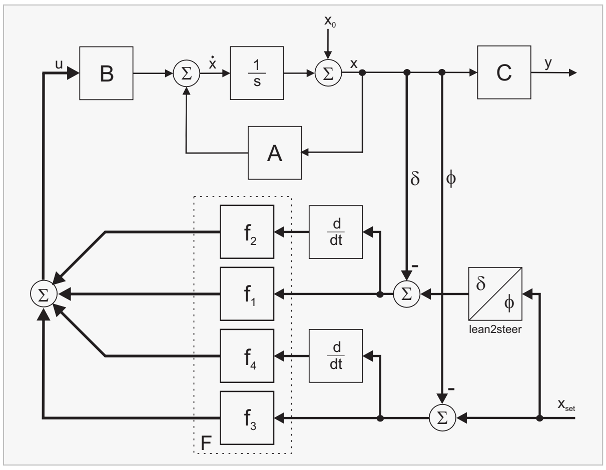


Figure 13: Block diagram of a virtual rider composed of a state-space controller and an additional block in order to calculate the corresponding steer angle. The reference input x_{set} is the desired roll angle profile.

elica model is depicted in Figure 14. The incorporated controller is shown in Figure 15.

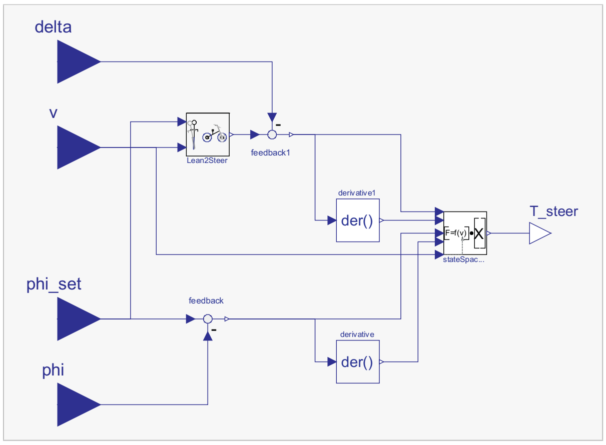


Figure 14: Wrapped model of the virtual rider composed of a state-space controller for a user defined velocity range. The inputs (blue) are lean and steer angle, lean angle set-value and the velocity of the motorcycle, the output T_{steer} (white) is the steering torque

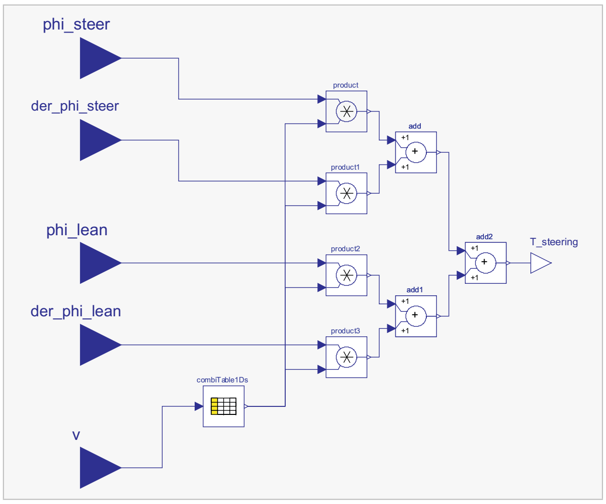


Figure 15: Wrapped model of a state-space controller for a specific velocity range. The table includes the state feedback matrix coefficients which by default are stored in *place.mat*

5.2 Path Tracking

In order to track a pre-defined trajectory using path preview information, a randomly generated path is included in the library. The path generation was done with MATLAB. This path is defined by its lateral profile [16]. To emulate the behavior of a human rider, single-point path preview is performed by the virtual rider. That is, the rider looks a pre-defined distance ahead in order to follow the path. It is worth noting that a similar deviation pattern is actually observed from human riders. In order to track a path, the controllers have to be extended [16] (see Figure 16). In the simplest case the lateral position x_{lat} of the rear frame's center of mass is fed back in order to generate an additional steering torque that keeps the vehicle on the desired path. For the utilized state-space controllers the lateral rate $\dot{x}_{lat}$ is additionally taken into account. The corresponding Modelica model is basically the same as the one depicted in Figure 14. To cover the path tracking capabilities two additional inputs, namely x_{lat} and $\dot{x}_{lat}$ are included. For the state-space path tracking controller the pole placement functions were extended in order to conveniently design such a controller.

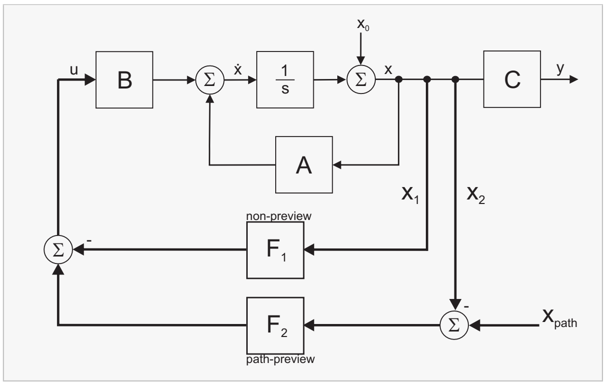


Figure 16: Basic structure of a state-space path preview controller. The state vector x_1 includes the states that are responsible for the stability (non-preview), whereas $x_2 = (x_{lat}\ \dot{x}_{lat})^T$ includes the states required for path tracking.

6 Examples

The first example demonstrates a 3 d.o.f. motorcycle stabilized by a virtual rider. The animation of the uncontrolled vehicle with a velocity of $4ms^{-1}$ is depicted in Figure 17. In order to determine the self-stabilizing range of the motorcycle an eigenvalue analysis is carried out. The results are shown in Figure 3. According to these results it can be seen that the vehicle is truly unstable for a velocity of $4ms^{-1}$. Furthermore, it can be seen that an offset of $d = 2$ is absolutely sufficient to achieve stable behavior. With this information the coefficients of the state feedback matrix are calculated. For this purpose the pole placement function based on the first approach is executed. The corresponding output is stored in the controller of the ready-made virtual rider introduced in Figure 14. The model of the controlled motorcycle is depicted in Figure 18. The animation of the controlled vehicle is shown in Figure 19.

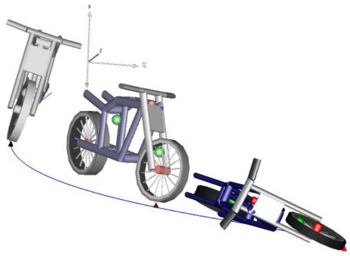


Figure 17: Animation result of the uncontrolled 3 d.o.f. motorcycle. After about 2s the motorcycle falls over like an uncontrolled inverted pendulum

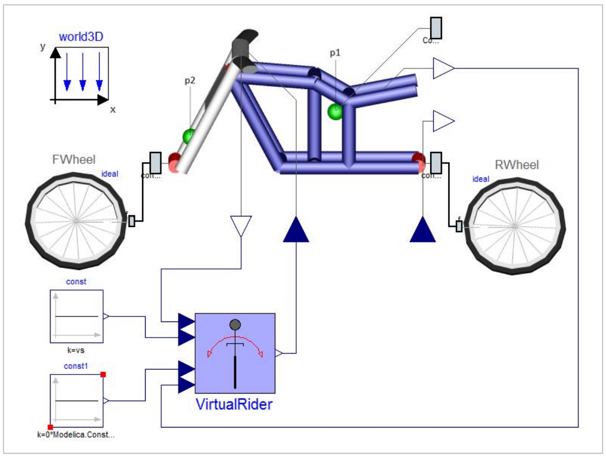


Figure 18: Example: controlled 3 d.o.f. motorcycle. The wrapped model of the virtual rider corresponds to Figure 14.

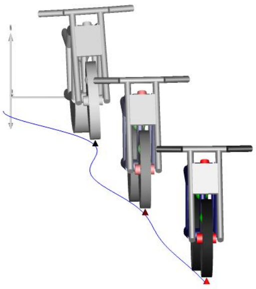


Figure 19: Animation result of the controlled 3 d.o.f. motorcycle

In the second example the motorcycle tracks a roll angle profile. The utilized model is depicted in Figure 18. Instead of a *constant source* block, the model of a 90°-curve is included. The coefficients of the feedback matrix were automatically calculated for an offset $d = 5$. The resulting eigenvalues are equal to those depicted in Figure 8. The animation result is depicted in Figure 20.

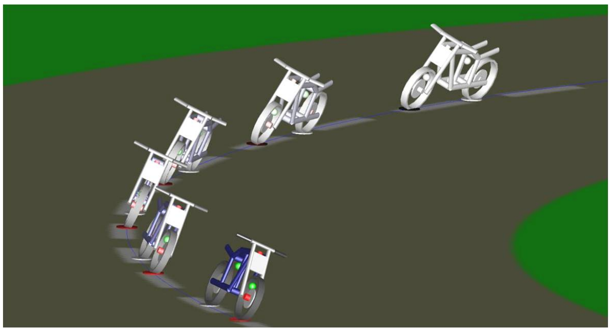


Figure 20: Animation result of a 3 d.o.f. motorcycle tracking a 90°-curve

In the last example the motorcycle tracks a predefined path. The utilized model is depicted in Figure 21.

Again, the coefficients of the feedback matrix are automatically calculated for an offset $d = 5$. Additionally, an offset $d_{lat} = 5$ is needed in order to keep the motorcycle on the desired path. The results are shown in Figure 22.

7 Structure of the Library

The structure of the *MotorcycleLib* is depicted in Figure 23. For each single-track vehicle a separate sub-package is provided. The basic bicycle sub-package is composed of a rigid rider and a movable rider sub-package. Both include the corresponding wrapped model and a function in order to perform an eigenvalue analysis. The motorcycle sub-package also in-

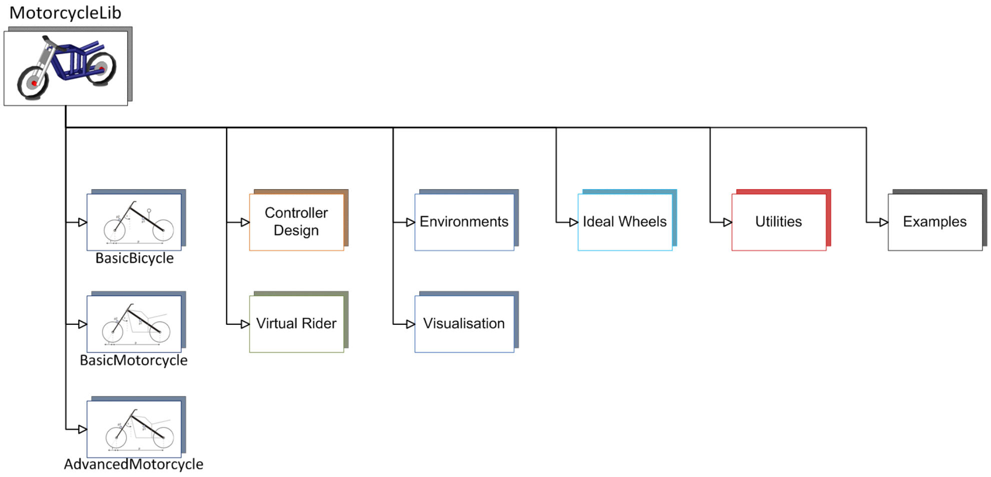


Figure 23: Library structure

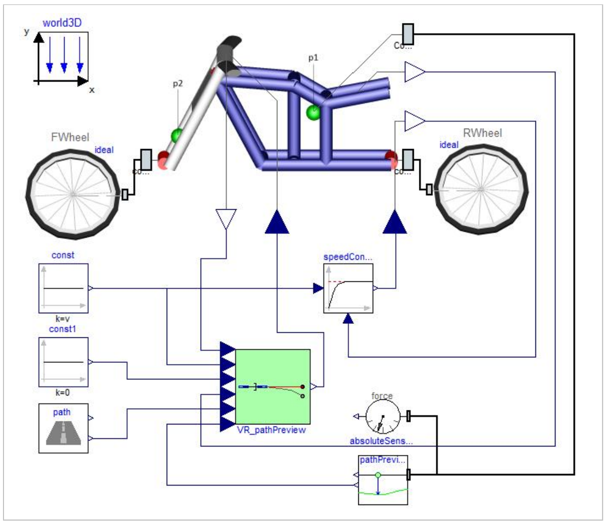


Figure 21: Example: model of a 3 d.o.f. motorcycle tracking a pre-defined path

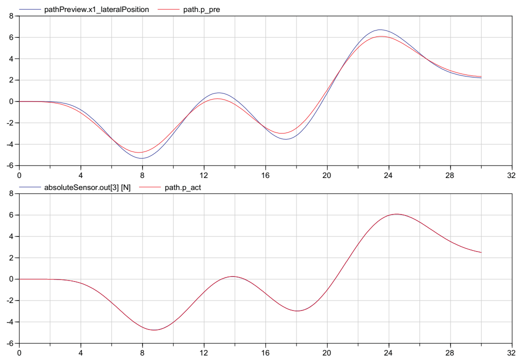


Figure 22: Simulation result of a motorcycle tracking a pre-defined path. Upper plot: The red signal is the path a pre-defined distance ahead, the blue signal is the preview distance of the rider. Lower plot: The red signal is the actual path, the blue signal is the traveled path measured at the rear frame's center of mass

cludes a wrapped model and an eigenvalue analysis function. The structure of the advanced motorcycle models is much more detailed since each part (e.g. front frame) is created in a fully object-oriented fashion. It is composed of a parts, an aerodynamics and a stability analysis sub-package. The parts sub-package includes several different front and swinging arms, a rear frame, the rider's upper body, a torque source (engine), an elasto-gap and a utilities sub-package. In the latter one, models of characteristic spring and damper elements are stored. The aerodynamics sub-package includes a lift force, a drag force and a pitching moment model.

The controller design sub-package contains pole placement functions in order to design appropriate controllers. The virtual rider sub-package includes, among others, a virtual rigid rider and a virtual movable rider sub-package. In both the riders are capable of either tracking a roll angle profile or a pre-defined trajectory. To this end, several different controllers (e.g. classic, state-space and LQR) are incorporated into the virtual riders.

The environments sub-package provides tracks for both roll angle tracking and path tracking. In addition, the models for single-point path tracking are included. The visualization sub-package provides the graphical information for the environments sub-package. In the ideal wheels sub-package the visualization of the rolling objects from D. Zimmer's *MultiBondLib* were modified such that the appearance is similar to real motorcycle wheels. The utilities sub-package provides some additional functions and models which are partly used in the library. The purpose of the examples subpackage is to provide several different examples that demonstrate how to use the library.

8 Conclusion

The library provides an appropriate eigenvalue function for each vehicle. Beside the controller design such an analysis is beneficial for the optimization of the vehicle's geometry. By changing the geometry or the center of mass' locations of a vehicle, the eigenvalues of the system are changing as well. It is thus possible to optimize the design of a vehicle regarding self-stability.

Furthermore, due to the results of the eigenvalue analysis it is now possible to conveniently design a state-space controller valid for a specific velocity range of the vehicle. Thus, for the calculation of the state feedback matrix coefficients, a pole placement function was developed. In order to design an LQR, MATLAB functions are provided.

To test the performance of the vehicles, the virtual riders are capable of tracking both, a roll angle profile and a pre-defined path. Therefore, several test tracks are included in the library.

A very detailed description of this paper can be found in the corresponding master's thesis [10].

References

[1] M. Andres. Object-oriented modeling of wheels and tires. Master's Thesis, 2009.

[2] F. E. Cellier and À. Netbot. The modelica bondgraph library. In *Proceedings of the 4th International Modelica Conference, Hamburg*, pages 57–65, 2005.

[3] V. Cossalter. *Motorcycle Dynamics*. 2006. 2nd edition.

[4] F. Donida, G. Ferreti, S. M. Savaresi, and M. Tanelli. Object-oriented modeling and simulation of a motorcycle. *Mathematical and Computer Modelling of Dynamic Systems*, 14, No. 2:79–100, 2008.

[5] F. Donida, G. Ferreti, S. M. Savaresi, M. Tanelli, and F. Schiavo. Motorcycle dynamics library in modelica. *Proceedings of the Fifth International Modelica Conference*, 5:157–166, 2006.

[6] S. Evangelou. *The control and stability analysis of two-wheeled road vehicles*. PhD thesis, Imperial College London, September, 2003.

[7] Otto Föllinger. *Regelungstechnik, Einführung in die Methoden und ihre Anwendungen*. Hüthig, 2008. 10. durchgesehene Auflage.

[8] F. Klein and A. Sommerfeld. Über die theorie des kreisels. *Quarterly Journal of Pure and Applied Mathematics*, Chapter 9, Section 8:863–884, Leipzig, 1910.

[9] C. Koenen. *The dynamic behaviour of motorcycles when running straight ahead and when cornering*. PhD thesis, Delft University, 1983.

[10] Thomas Schmitt. Modeling of a motorcycle in dymola/modelica. Master's thesis, Vorarlberg University of Applied Sciences, 2009.

[11] A. L. Schwab, J. D. G. Kooijman, and J. P. Meijaard. Some recent developments in bicycle dynamics and control. *Fourth European Conference on Structural Control*, page 8, 2008.

[12] A. L. Schwab, J. P. Meijaard, and J. M. Papadopoulos. Benchmark results on the linearized equations of motion of an uncontrolled bicycle. *KSME International Journal of Mechanical Science and Technology*, pages 292–304, 2005.

[13] R. S. Sharp. The stability and control of motorcycles. *Journal Mechanical Engineering Science*, Volume 13:316–329, 1971.

[14] R. S. Sharp, S. Evangelou, and D. J. N. Limebeer. Advances in the modelling of motorcycle dynamics. *Multibody System Dynamics*, Volume 12:251–283, 2004.

[15] R. S. Sharp and D. J. N. Limebeer. A motorcycle model for stability and control analysis. *Multibody System Dynamics*, Volume 6:123–142, 2001.

[16] R. S. Sharp and V. Valtetsiotis. Optimal preview car steering control. *Vehicle System Dynamics*, Supplement 35:101–117, 2001.

[17] D. Zimmer and F. E. Cellier. Multibond graph library. *Proceedings of the Fifth International Modelica Conference*, pages 559–568, 2006.

Modeling and Optimization with Modelica and Optimica Using the JModelica.org Open Source Platform

Johan Åkesson[a,b] Tove Bergdahl[a] Magnus Gäfvert[a] Hubertus Tummescheit[a]
[a] Modelon AB, Sweden
[b] Department of Automatic Control, Lund University, Sweden

Abstract

This paper reports a new Modelica-based open source project entitled JModelica.org, targeted towards dynamic optimization. The objective of the project is to bridge the gap between the need for high-level description languages and the details of numerical optimization algorithms. JModelica.org is also intended as an extensible platform where algorithm developers, particularly in the academic community, may integrate new and innovative methods. In doing so, researchers gain access to a wealth of industrially relevant optimization problems based on existing Modelica models, while at the same time facilitating industrial use of state of the art algorithms. In this contribution, an overview of the platform is presented and the main features of JModelica.org are highlighted.

Keywords: Modelica; Optimica; Optimization; Model Predictive Control

1 Introduction

Optimization is becoming a standard methodology in many engineering disciplines to improve products and processes. The need for optimization is driven by factors such as increased costs for raw materials and stricter environmental regulations as well as a general need to meet increased competition. As model-based design processes are being used increasingly in industry, the prerequisites for optimization are often fulfilled. However, current tools and languages used to model dynamic systems are not always well suited for integration with state of the art numerical optimization algorithms. As a result, optimization is not used as frequently as it could, or less efficient, but easier to use, algorithms are employed.

More often than not, systems to be optimized are complex and dynamic. Such problems offer several challenges at different levels. Much effort has been devoted to encapsulating expert knowledge in model libraries encoded in domain specific languages such as VHDL-AMS [30] and Modelica [44]. While such model libraries have been primarily intended for simulation, it is desirable to enable also other usages, including optimization. From a user's perspective, it is desirable that the optimization specification is expressed in a high-level language in order to provide a comprehensive description both of the dynamic model to be optimized and of the optimization problem. Another aspect that requires attention is that of enabling flexible use of the wealth of numerical algorithms for dynamic optimization, based on the high-level descriptions specified by the user.

Several common engineering tasks are conveniently cast as optimization problems. This includes parameter estimation problems to obtain models that match plant data, design optimization for improving product performance, and controller parameter tuning. In addition, dynamic optimization is a key to implementing for example model predictive controllers and receding horizon state estimators.

This contribution reports a new Modelica-based open source initiative targeted at dynamic optimization entitled JModelica.org. JModelica.org [36] is a novel open source project with the mission:

> *"To offer a community-based, free, open source, accessible, user and application oriented Modelica environment for optimization and simulation of complex dynamic systems, built on well-recognized technology and supporting major platforms."*

JModelica.org is primarily focused on dynamic optimization of Modelica models. To meet this end, JModelica.org supports Optimica, which is an extension to the Modelica language that offers language constructs for encoding of cost functions, constraints and the optimization interval with fixed or free end points. The platform consists of compilers for translating Model-

DOI: 10.3384/ecp09430057

ica and Optimica models into C and XML code, a C API for evaluation of model equations and Python bindings to enable scripting and custom algorithm development. The software is distributed freely under the GPL license.

The paper is outlined as follows. In Section 2, a review of optimization tools and the Optimica extension are given. Section 3 describes the JModelica.org platform. Previous case studies performed based on JModelica.org, and the opportunities provided by abstract syntax tree access are discussed in Section 4. In Section 5, an example of a model predictive control application is given. The paper ends with a summary and comments on future work in Section 6.

2 Background

It is typical that numerical algorithms for dynamic optimization is written in C or Fortran. Often, the user is required to encode the dynamic model and the optimization specification in the same languages. While C and Fortran enables efficient compilation to executable code, such languages are not well suited for encoding of dynamic models and optimization problems. In particular, it is difficult to write the code in a modular way that enables reuse. This observation was made several decades ago in the context of modeling and simulation and resulted in high-level modeling languages, including ACSL and later Omola, [4], VHDL-AMS [30], and Modelica [44]. See [5] for a comprehensive overview of the evolution of continuous-time simulation languages and tools.

2.1 Optimization Tools

There are several tools for optimization on the market, offering different features. In essence, three different categories of tools can be distinguished, although the functionality is sometimes overlapping. *Model integration tools* addresses the problem of interfacing several design tools into a a single computation environment, where analysis, simulation and optimization can be performed. Examples are ModelCenter, [41], OptiY, [40], modeFRONTIER [21], and iSIGHT, [11]. Typically, such tools are dedicated to design optimization of extremely complex systems which may be composed from subsystems encoded in different tools. Accordingly, model integration tools typically offers interfaces to CAD and finite element software as well as simulation tools for, e.g., mechanical and hydraulic systems. As a result of the heterogeneity and complexity of the target models, models are usually treated as black boxes, i.e. the result of a computation is propagated to the tool, but the structure of a particular model is not explored. Accordingly, heuristic optimization algorithms which do not require derivative information or detailed structural information, are usually employed. In addition, model integration tools often have sophisticated features supporting model approximation and visualization.

Several *Simulation tools* comes with optimization add-ons, e.g., Dymola [14], gPROMS [42] and Jacobian [37]. Such tools typically offer strong support for modeling of physical systems and simulation. The level of support for optimization in this category differs between different tools. Dymola, for example, offers add-ons for parameter identification and design optimization, [18]. gPROMS, on the other hand, also offers support for solution of optimal control problems. Tools in this category are usually limited to a predefined set of optimization algorithms. Integration of new algorithms may be difficult if the tools do not provide the necessary API:s.

In the third category we have *numerical packages* for dynamic optimization, often developed as part of research programs. Examples are ACADO [39], Muscod II [46], and DynoPC [33], which is based on Ipopt [48]. Such packages are typically focused on efficient implementation of an optimization algorithm for a particular class of dynamic systems. Also, detailed information about the model to optimize is generally required in order for such algorithms to work, including accurate derivatives and in some cases also sparsity patterns. Some of the packages in this category are also targeting optimal control and estimation problems in real-time, e.g., non-linear model predictive control, which require fast convergence. While these packages offer state of the art algorithms, they typically come with simple or no user interface. Their usage is therefore limited due to the effort required to code the model and optimization descriptions.

The JModelica.org platform is positioned to fill the gap left between simulation tools offering optimization capabilities and state of the art numerical algorithms. Primarily, target algorithms are gradient based methods offering fast convergence. Never the less, JModelica.org is well suited for use also with heuristic direct search methods; the requirements with respect to execution interface is typically a subset of the requirements for gradient based methods. The problems addressed by model integration tools is currently beyond the scope of JModelica.org, even though its in-

```
model VDP
  Real x1(start=0);
  Real x2(start=1);
  input Real u;
equation
  der(x1) = (1-x2^2)*x1 - x2 + u;
  der(x2) = x1;
end VDP;
```

Listing 1: A Modelica model of a van Der Pol oscillator.

```
optimization VDP_Opt
   ⑧(objective=cost(finalTime),
         startTime=0,
         finalTime(free=true,
            initialGuess=1))
①  VDP vdp(u(free=true,
   initialGuess=0.0));
②  Real cost (start=0);
equation
③  der(cost) = 1;
constraint
④  vdp.x1(finalTime) = 0;
⑤  vdp.x2(finalTime) = 0;
⑥  vdp.u >= -1;
⑦  vdp.u <= 1;
end VDP_Opt;
```

Listing 2: An Optimica optimization specification based on the van Der Pol Oscillator.

tegration with Python offers extensive possibilities to develop custom applications based on the solution of simulation and optimization problems.

2.2 Optimica

The Optimica extension is discussed in detail [1, 2]. In this paper, a brief overview of Optimica is given and the extension is illustrated by means of an example.

We consider the following dynamic optimization problem:

$$\min_{u(t)} \int_0^{t_f} 1\,dt \tag{1}$$

subject to the dynamic constraint

$$\begin{aligned} \dot{x}_1(t) &= (1-x_2(t)^2)x_1(t) - x_2(t) + u(t), & x_1(0) &= 0 \\ \dot{x}_2(t) &= x_1(t), & x_2(0) &= 1 \end{aligned} \tag{2}$$

and

$$\begin{aligned} x_1(t_f) &= 0 \\ x_2(t_f) &= 0 \\ -1 \le u(t) &\le 1 \end{aligned} \tag{3}$$

The dynamic model (2) of the problem is a van Der Pol oscillator, and the optimization problem corresponds to bringing the system from initial conditions $x_1(0) = 0$, $x_2(0) = 1$ to the origin in minimum time. In addition, the transition is to be performed with limited control authority.

A Modelica model corresponding to the dynamic system (2) is given in Listing 1. Based on this model, an Optimica specification can be formulated, see Listing 2. Since Optimica is an extension of Modelica, language elements valid in Modelica are also valid in Optimica. In addition Optimica also contains new constructs not valid in Modelica.

The Optimica program corresponding to the van Der Pol example can be seen in Listing 2. In order to specify an optimization problem in Optimica, the new specialized class `optimization` is used. Inside such a class, Optimica constructs, as well as Modelica constructs may be used. An instance of the `VDP` model is created by declaring a corresponding component, ①. In order to express that the input `u` is to be tuned in the optimization, the Optimica-specific variable attribute `free` is set to `true`, and in addition, an initial guess for `u` is provided. In order to define the cost function, a variable, `cost` ②, is introduced along with a defining equation, ③. Further, the constraints are given in the `constraint` section, which is a new Optimica construct. In this section, ④–⑤ correspond to the terminal constraints, whereas ⑥–⑦ correspond to the control variable bounds. Notice how the value of a variable at a particular time instant is accessed using an Optimica-specific function call-like syntax. `finalTime` is a built-in variable of the specialized class `optimization` and is used to refer to the time at the end of the optimization interval. Finally, the objective and the optimization interval is specified, ⑧. The construct introduced in Optimica to meet this end can be viewed as built-in class attributes which are given values through class arguments. Here the variable representing the cost function is bound to the built-in class attribute `objective` and it is specified that `finalTime` is to be free in the optimization.

The Modelica and Optimica specifications are then typically translated by a compiler into a format suitable for compilation with a numerical solver in order to obtain the solution.

3 The JModelica.org platform

In order to demonstrate the feasibility and effectiveness of the proposed Optimica extension, a prototype

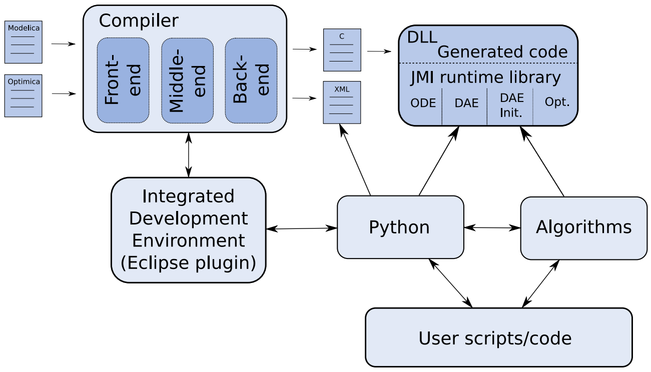


Figure 1: Overview of the JModelica.org platform architecture.

```
model M
  Real x;
equation
  x = 1;
end M;
```

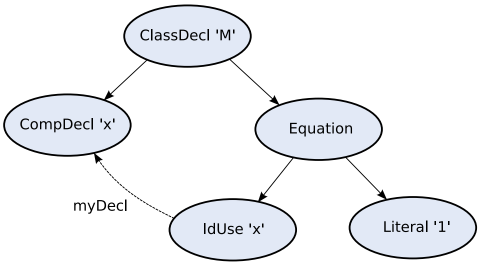


Figure 2: A simple Modelica model (left) and its corresponding abstract syntax tree (right). The dashed arrow represents the reference attribute `myDecl` which binds an identifier to its declaration.

compiler was developed, [1]. Currently, the initial prototype compiler is being developed with the objective of creating a Modelica-based open source platform focused on dynamic optimization.

The architecture of the JModelica.org platform is illustrated in Figure 1. The platform consists essentially of two main parts: the compiler and the JModelica.org Model Interface (JMI) runtime library. The compiler transforms Modelica and Optimica source code into a flat model description and then generates C and XML code. The generated C code contains the actual model equations in a format suitable for efficient evaluation, whereas the XML code contains model meta data, such as variable names and parameter values. The JMI runtime library provides a C interface which in turn can be interfaced with numerical algorithms. There is also an Eclipse plug-in and a Python integration module under development. In this section, the key parts of the JModelica.org platform will be described.

3.1 Compiler Development—JastAdd

Compiler construction has traditionally been associated with intricate programming techniques within the area of computer science. Recent research effort has, however, resulted in new compiler construction frameworks that are easier to use and where it is feasible to develop compilers with a comparatively reasonable effort. One such framework is JastAdd [28, 17]. JastAdd is a Java-based compiler construction framework based on concepts such as object-orientation, aspect-orientation and reference attributed grammars [15]. At the core of JastAdd is an abstract syntax specification, which defines the structure of a computer program. Based on an abstract syntax tree (AST), the compiler performs tasks such as *name analysis*, i.e, finding declarations corresponding to identifiers, *type analysis*, i.e., verifying the type correctness of a program, and code generation.

The JastAdd way of building compilers involves specification of *attributes* and *equations* based on the abstract syntax specification. This feature is very similar to ordinary Knuth-style attribute grammars [32] but enhanced with reference attributes. Accordingly, attributes may be used to specify, declaratively, links between different nodes in the AST. For example, identifier nodes can be bound to their declaration nodes. In Figure 2, an example of a small Modelica program and its corresponding AST is shown. Notice how the reference attribute `myDecl` links an identifier (`IdUse`) to its declaration (`CompDecl`).

JastAdd attributes and equations are organized into separate *aspects*, which form a convenient abstraction for encoding of cross cutting behavior. Typically, implementation of a semantic function, for example type analysis, involves adding code to large number of classes in the AST specification. Using aspects, much like in AspectJ [31], cross cutting behavior can be modularized in a natural way. In addition, this approach is the basis for one of the distinguishing features of JastAdd: it enables development of modularly extensible compilers. This means that it is feasible to develop, with a comparatively moderate effort, modular extensions of an existing JastAdd compiler without changing the core compiler. This feature has been used in the implementation of the JModelica.org Modelica and Optimica compilers, where the Optimica compiler is a fully modular extension of the core Modelica compiler.

The JastAdd compiler transforms the JastAdd specification into pure Java code, where the definition of the abstract grammar translates into Java classes corresponding to Modelica classes, components, functions, and equations. The JastAdd attributes are woven into the Java classes as methods. In addition, methods for traversing an AST and query properties of a particular AST class, e.g., obtain a list of variables contained in a class declaration, are automatically generated. As a result of this approach, compilers produced by JastAdd are in the form of standard Java packages, which in turn can be integrated in other applications. It is therefore not necessary to know the particular details of how to write JastAdd specifications in order to use the JModelica.org compilers, knowledge of Java is generally sufficient.

3.2 The Modelica and Optimica Compilers

At the core of the JModelica.org platform is a Modelica compiler that is capable of transforming Modelica code into a flat representation and of generating C code. In the Modelica compiler, several design strategies, for example name look-up, developed for a Java compiler developed using JastAdd [16], were reused. For additional details on the implementation of the compiler, see [3].

In order to support also the Optimica extension, a modular extension of the core Modelica compiler has been developed. The extended compiler is capable of translating standard Modelica enhanced with the new Optimica syntax presented in Section 2.2. The Optimica extension is reported in more detail in [27].

The JModelica.org Modelica compiler currently supports a subset of Modelica version 3.0. The Modelica Standard Library version 3.0.1 can be parsed and the corresponding source AST can be constructed. Flattening support is more limited, but is being continuously improved.

3.3 Code Generation

The JModelica.org offers a code generation framework implemented in Java as part of the compilers. The framework facilitates development of custom code generation modules and is based on *templates* and *tags*. A template is used to specify the structure of the generated code and tags are used to define elements of the template which is to be replaced by generated code. In order to develop a custom code generation module, the user needs to define a template and a set of tags, and then implement the actual code generation behavior corresponding to each tag. In order to perform the latter, the AST for the flattened Modelica model is typically used, where objects corresponding to declarations, equations and functions are queried for information used to generate the target code.

The JModelica.org platform contains two code generation modules, one for C and one for XML. The generated C code contains the model equations and is intended to be compiled and linked with the JModelica.org Model Interface (see below) in order to offer efficient evaluation of the model equations. The XML output consists of model meta data such as specifications of variables, including their names, attributes and type. Also, the XML output includes a separate file for parameter values. The XML output is similar to what is discussed within the FMI initiative [12], and the intention is for the JModelica.org XML output to be compliant with FMI once finalized. In addition, there is on-going work aimed to develop an XML specification for flattened Modelica models, including variable declarations, functions, and equations [9]. The objective is for JModelica.org to be compliant also with this specification.

3.4 C API

The JModelica.org platform offers a C API, entitled the JModelica.org Model Interface (JMI[1]), suitable for integration with numerical algorithms. The interface provides functions for accessing and setting parameter and state values, for evaluation of the DAE residual function and for evaluation of cost functions and constraints specified in an Optimica model. In addition, Jacobians and sparsity patterns can be obtained for all functions in the interface. To meet this end, a package for automatic differentiation, CppAD [6], has been integrated into JMI. The JMI code is intended to be compiled with the C code that is generated by the compiler into an executable, or into a shared object file.

The JMI interface consists of four parts: the ODE interface, the DAE interface, the DAE initialization interface, and the Optimization interface. These interfaces provide access to functions relevant for different parts of the optimization specification. The ODE and DAE interfaces provide evaluation functions for the right hand side of the ODE and the DAE residual function respectively. The DAE initialization problem provides functions for solving the DAE initialization problem, whereas the Optimization interface provides functions for evaluation of the cost functions and the constraints.

[1] Notice that this acronym is unrelated to Java Metadata interface

3.5 Interactive Environment—Python

Solution of engineering problems typically involves atomization of tasks in the form of user scripts. Common examples are batch simulations, parameter sweeps, post processing of simulation results and plotting. Given that JModelica.org is directed towards scientific computing, Python, see [23], is an attractive option. Python is a free open-source highly efficient and mature scripting language with strong support in the scientific community. Packages such as NumPy [38] and SciPy [20], and bindings to state-of-the art numerical codes implemented in C and Fortran make Python a convenient glue between JModelica.org and numerical algorithms. In addition, IPython [19] with the visualization package matplotlib [29] and the PyLab mode offer an interactive numerical environment similar to the ones offered by Matlab and Scilab.

The JModelica.org Pyhon package includes sub packages for running the compilers, for managing file input/output of simulation/optimization results and for accessing the function provided by the JMI interface. The compilers are run in a Java Virtual Machine (JVM) which is connected to the Python environment by the package JPype, [35]. One of JPype's main features is to enable direct access to Java objects from a Python shell or script. This feature is used to communicate with the compilers, but can also be used to retrieve the ASTs generated by the compilers. The latter feature enables the user to traverse and query the ASTs interactively, see 4.2 for a discussion on example usages of this feature.

The integration of the JMI is based on the ctypes package [22]. Using ctypes, a dynamically linked library (DLL) can be loaded into Python, All the contained functions of the DLL are then exposed and can be called directly from the Python shell. In order to enable use of Numpy arrays and matrices as arguments to the JMI functions, the argument types has been explicitly encoded using standard features of ctypes. In order to provide a more convenient interface to the JMI functions, a Python class, `Model` has been created. This class encapsulates loading of a DLL and typing of the JMI functions, and also provides wrapper functions supporting Python exceptions. In addition, upon creation of a `Model` class, the generated XML meta data files are loaded and parameter values and start attributes are set in the loaded model instance. `Model` objects can then be manipulated, e.g., by setting new parameter values, or passed as an argument to a simulation or optimization algorithm.

3.6 Optimization Algorithms

The JModelica.org platform offers two different algorithms for solving dynamic optimization problems. The first is a simultaneous optimization method based on orthogonal collocation on finite elements [7]. Using this method, state and input profiles are parameterized by Lagrange polynomials which are based on Radau points. This method corresponds to a fully implicit Runge-Kutta method, and accordingly it possesses well known and strong stability properties. By parameterizing the variable profiles by polynomials, the dynamic optimization problem is translated into a non-linear programming (NLP) problem which may be solved by a numerical NLP solver. This NLP is, however, very large. In order to efficiently find a solution to the NLP, derivative information as well as the sparsity patterns of the constraint Jacobians need to be provided to the solver. The simultaneous optimization algorithm has been interfaced with the large-scale NLP solver Ipopt [48], which has been developed particularly to solved NLP problems arising in simultaneous dynamic optimization methods. The algorithm is implemented in C as an extension of JMI, and provides an example of how to implement algorithms based on the JMI functions. In particular, Jacobians computed by CppAD is used, including sparsity patterns.

In addition to the simultaneous optimization algorithm, JModelica.org contains a multiple shooting algorithm, [8]. The algorithm is based on an integrator which is used to simulate the system dynamics and thereby evaluate the cost function, and an optimization algorithm which modifies the optimization variables. Typically, the optimization variables are Modelica parameters in the case of a design or parameter optimization problem, or parameters resulting from discretization of a control input. The multiple shooting algorithm is implemented in Python, and relies on the integrator SUNDIALS [34], its Python interface PySUNDIALS [47], and the optimization algorithm scipy_slsqp, which is included in Scipy. In order to improve the convergence of the optimization algorithm *sensitivities* are computed and propagated to the optimization algorithm. The sensitivities are computed using SUNDIALS. The implementation serves also as an example of how to develop algorithms based on the JModelica.org Python interface. The multiple shooting algorithm is described in more detail in [43].

The above algorithms are both based on the availability of derivatives. For some optimization problems, it is not possible to reliably compute derivatives, and accordingly, numerical optimization algorithms requiring derivative information may fail. This situation may occur for certain classes of hybrid systems. In such cases, heuristic methods which do not require derivative methods may be better suited. Examples of such methods are genetic algorithms, pattern matching, simulated annealing, and simplex (Nelder-Mead). Some methods of this class are freely available for Python, see the OpenOpt project [13] for more information, and may be integrated with the JMI Python interface.

4 Applications

4.1 Dynamic optimization

Prototype versions of the JModelica.org software has been used successfully in applications in different domains. In [26], an application is reported where optimal start-up trajectories are computed for a plate reactor. The problem is challenging not only due to the complexity of the model (approx. 130 differential and algebraic equations), but also due to non-linear and in some conditions unstable dynamics. A main challenge in this project was to obtain trajectories robust to parameter variations. The procedure of finding acceptable solutions was highly iterative in that the cost function and the constraints required several redesigns. The high-level specification framework in combination with automatic code generation offered by Optimica and the JModelica.org platform proved very useful in this respect.

The prototype software has as also been used in a number of other projects involving vehicle systems. For example, in [10] optimal tracks for racing cars were optimized, and in [45], optimal rail profiles were computed for a novel mass transportation system, the NoWait train concept. Other examples where Optimica has been used are reported in [25] where minimum time optimization for an industrial robot is considered and in [24] where an optimal control application of a pendulum system is reported.

4.2 Using ASTs

As described above, the JModelica.org compilers provide direct access to abstract syntax trees (ASTs). The ASTs are abstract representations of Modelica models, and provides a means to access the contents of a Model in a structured and programmatic way. Three different types of ASTs are used during the procedure of producing a flat Modelica representation: the *source* AST, the *instance* AST, and the *flat* AST. The ASTs in the JModelica.org org compilers consists of standard Java objects instantiated from the AST classes produced by the JastAdd compilers. Also, means to traverse the AST are provided automatically, in addition to the methods corresponding to attributes defined in the compiler.

The source AST results from parsing of a Modelica source file. Its structure corresponds precisely to the actual structure of the code, but the details of the concrete syntax has been removed. Given the source AST, a number of queries can be performed. For example, the AST may be traversed and for each class declaration, the documentation annotation and the signatures of the public parameters may be extracted and pretty printed according to a HTML template. Another example would be to traverse the AST and output the default values of all parameters in XML format.

Based on a particular class declaration in the source AST, an instance AST may be constructed. The instance AST differs from the source AST in that in the instance AST the components structurally contained in a component declaration is explicitly represented. Also, in the instance tree, modifications, e.g., class and component redeclarations take effect. In fact, the key to constructing the instance tree is to handle *modification environments* consisting of an ordered set of modifications applicable to a particular class or component instance. Construction of the instance AST is described in [3]. Access to the instance AST enables several analyses to be performed. For example, the instance AST may be traversed, and for each primitive variable encountered the corresponding modification environment may be retrieved and output to file.

The flat AST, corresponding to a flattened Modelica model, is constructed by traversing the instance AST and collecting all primitive variables, equations, algorithms and functions. The flat AST offers a predefined API for retrieving variables of the primitive types. For example, Java methods are provided for retrieving all parameters of Real type, for retrieving all differentiated variables, for querying if a variable occurs linearly in the model equations etc. The flat AST is typically used as a basis for code generation.

5 An Example

We consider the Continuously Stirred Tank Reactor (CSTR) process depicted in Figure 3. The temperature and the reactant concentration of the inlet flow are constant and the control input of the process, u corresponds to the coolant flow. The coolant temperature,

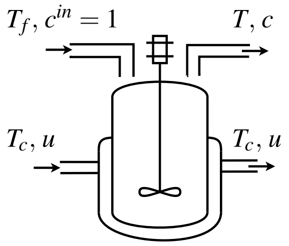


Figure 3: A schematic figure of the CSTR process.

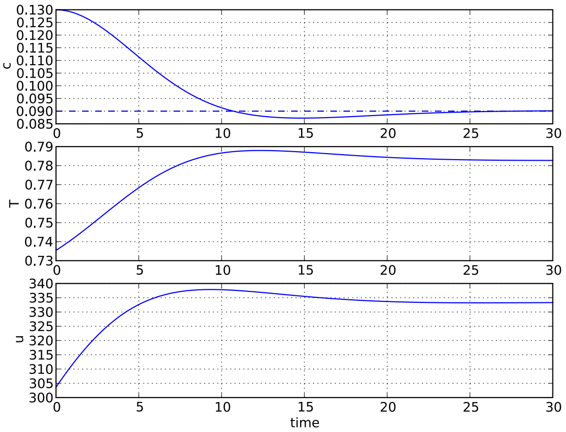


Figure 4: Optimization result for the CSTR example.

T_c, is constant. A dynamic model for an exothermic reaction is then given by

$$\begin{aligned} \dot{c} &= \beta(1-c) - ke^{-N/T}c \\ \dot{T} &= \beta(T_f - T) + ke^{-N/T}c - \alpha(T - T_c)u \end{aligned} \tag{4}$$

where c is the normalized concentration in the reactor, T is the normalized reactor temperature, and β, k, N, and α are physical parameters for the process.

Based on the CSTR model, the following dynamic optimization problem can be formulated:

$$\min_{u(t)} \int_0^{t_f} q_1(c_r - c)^2 + q_2(T_r - T)^2 + r(u_r - u)^2 dt \tag{5}$$

subject to the dynamics (4). The cost function penalizes deviations from a desired operating point given by target values c_r, T_r and u_r for c, T and u respectively. Starting at fixed initial conditions, the optimal solution transfers the system from one operating point to another.

In this case, the numerical solver IPOPT [48] is used to solve the transcribed NLP resulting from direct collocation. The result of the optimization is shown in Figure 4.

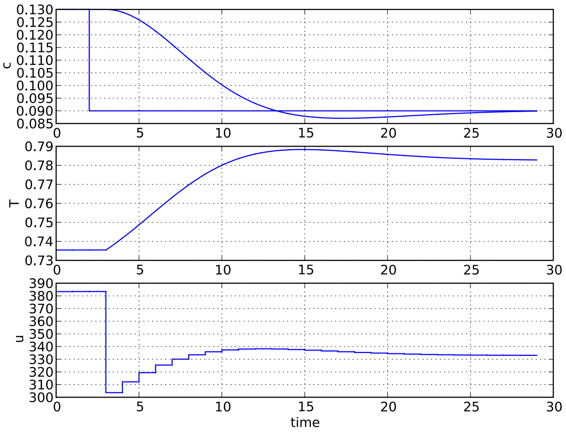


Figure 5: Simulation result of the MPC.

The optimal control problem formulated above can also be used in conjunction with other algorithms available in Scipy. To demonstrate this, a simple model predictive controller (MPC) has been implemented. The MPC control strategy is based on the receding horizon principle, where an open loop optimal control problem is solved in each sample. Simulation of an MPC requires joint simulation of the plant and solution of the optimal control problem. Such operations are easily encoded in Python. The result of executing the MPC is shown in Figure 5.

6 Summary and future work

In this paper, the JModelica.org open source platform has been presented. The platform features compilers written in JastAdd/Java, Optimica support, a C model API, and XML export. The compilers and the C API for evaluation of the model equations have been interfaced with Python, in order to provide an environment for scripting and development of custom applications. In addition, the abstract syntax trees representing the Modelica source code, the instance hierarchy and the flattened model are accessible.

Future plans include improvement of the Modelica compliance of the compiler front-end, integration of additional numerical optimization algorithms, simulation support, and support for heuristic optimization methods such as simulated annealing and genetic algorithms. Also, an Eclipse plug-in is under development where current research on custom IDE development based on JastAdd is explored.

References

[1] Johan Åkesson. *Tools and Languages for Optimization of Large-Scale Systems*. PhD thesis, Department of Automatic Control, Lund University, Sweden, November 2007.

[2] Johan Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *In 6th International Modelica Conference 2008*. Modelica Association, March 2008.

[3] Johan Åkesson, Torbjörn Ekman, and Görel Hedin. Implementation of a modelica compiler using jastadd attribute grammars. *Science of Computer Programming*, July 2009. doi:10.1016/j.scico.2009.07.003.

[4] Mats Andersson. *Object-Oriented Modeling and Simulation of Hybrid Systems*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, December 1994.

[5] Karl Johan Åström, Hilding Elmqvist, and Sven Erik Mattsson. Evolution of continuous-time modeling and simulation. In *Proceedings of the 12th European Simulation Multiconference, ESM'98*, pages 9–18, Manchester, UK, June 1998. Society for Computer Simulation International.

[6] B. M. Bell. CppAD Home Page, 2008. `http://www.coin-or.org/CppAD/`.

[7] L.T. Biegler, A.M. Cervantes, and A Wächter. Advances in simultaneous strategies for dynamic optimization. *Chemical Engineering Science*, 57:575–593, 2002.

[8] H.G. Bock and K. J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In *Ninth IFAC world congress*, Budapest, 1984.

[9] F. Casella, D. Filippo, and J. Åkesson. n XML Representation of DAE Systems Obtained from Modelica Models. In *In 7th International Modelica Conference 2009*. Modelica Association, 2009.

[10] Henrik Danielsson. Vehicle path optimisation. Master's Thesis ISRN LUTFD2/TFRT--5797--SE, Department of Automatic Control, Lund University, Sweden, June 2007.

[11] Dassault Systèmes. iSIGHT Home Page, 2009. `http://www.simulia.com/products/isight.html`.

[12] DLR, Dynasim, ITI and QTronic. The functional model interface. Draft.

[13] Dmitrey L. Kroshko. OpenOpt Home Page, 2009. `http://openopt.org/Welcome`.

[14] Dynasim AB. Dynasim AB Home Page, 2008. `http://www.dynasim.se`.

[15] T. Ekman and G. Hedin. Rewritable Reference Attributed Grammars. In *Proceedings of ECOOP 2004*, volume 3086 of *LNCS*, pages 144–169. Springer-Verlag, 2004.

[16] Torbjön Ekman and Görel Hedin. The jastadd extensible java compiler. In *Proceedings of OOPSLA 2007*, 2007.

[17] Torbjörn Ekman, Görel Hedin, and Eva Magnusson. JastAdd, 2008. http://jastadd.cs.lth.se/web/.

[18] H. Elmqvist, H. Olsson, S.E. Mattsson, D. Brück, C. Schweiger, D. Joos, and M. Otter. Optimization for design and parameter estimation. In *In 7th International Modelica Conference 2009*. Modelica Association, 2005.

[19] Inc. Enthought. IPython FrontPage, 2009. `http://ipython.scipy.org/moin/`.

[20] Inc. Enthought. SciPy, 2009. `http://www.scipy.org/`.

[21] ESTECO. modeFRONTIER Home Page, 2009. `http://www.esteco.com/`.

[22] Python Software Foundation. ctypes: A foreign function library for Python, 2009. `http://docs.python.org/library/ctypes.html`.

[23] Python Software Foundation. Python Programming Language – Official Website, 2009. `http://www.python.org/`.

[24] P. Giselsson, J. Åkesson, and A. Robertsson. Optimization of a pendulum system using optimica and modelica. In *In 7th International Modelica Conference 2009*. Modelica Association, 2009.

[25] M. Hast, J. Åkesson, and A. Robertsson. Optimal Robot Control using Modelica and Optimica. In *In 7th International Modelica Conference 2009*. Modelica Association, 2009.

[26] Staffan Haugwitz, Johan Åkesson, and Per Hagander. Dynamic start-up optimization of a plate reactor with uncertainties. *Journal of Process Control*, 2009. doi:10.1016/j.jprocont.2008.07.005.

[27] Görel Hedin, Johan Åkesson, and Torbjön Ekman. Building DSLs by leveraging base compilers—from Modelica to Optimica. *IEEE Software*, 2009. Submitted for publication.

[28] Görel Hedin and Eva Magnusson. JastAdd: an aspect-oriented compiler construction system. *Science of Computer Programming*, 47(1):37–58, 2003.

[29] J. Hunter, D. Dale, and M. Droettboom. matplotlib: python plotting, 2009. http://matplotlib.sourceforge.net/.

[30] IEEE. Standard VHDL Analog and Mixed-Singnal Extensions. Technical report, IEEE, 1997.

[31] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. *LNCS*, 2072:327–355, 2001.

[32] Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, June 1968. Correction: *Mathematical Systems Theory* 5, 1, pp. 95-96 (March 1971).

[33] Y.D. Lang and L.T. Biegler. A software environment for simultaneous dynamic optimization. *Computers and Chemical Engineering*, 31(8):931–942, 2007.

[34] Center for Applied Scientific Computing Lawrence Livermore National Laboratory. SUNDIALS (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers), 2009. https://computation.llnl.gov/casc/sundials/main.html.

[35] S. Menard. JPype Home Page, 2009. http://jpype.sourceforge.net/.

[36] Modelon AB. JModelica Home Page, 2009. http://www.jmodelica.org.

[37] Numerica Technology. Jacobian Home Page, 2009. http://www.numericatech.com/jacobian.htm.

[38] T. Oliphant. Numpy Home Page, 2009. http://numpy.scipy.org/.

[39] OPTEC K.U. Leuven. ACADO Home Page, 2009. http://www.acadotoolkit.org/.

[40] OptiY. OptiY Home Page, 2009. http://www.optiy.de/.

[41] Phoenix Integration. ModelCenter Home Page, 2009. http://www.phoenix-int.com/software/phx_modelcenter.php.

[42] Process Systems Enterprise. gPROMS Home Page, 2009. http://www.psenterprise.com/gproms/index.html.

[43] J. Rantil, J. Åkesson, C. Führer, and M. Gäfvert. Multiple-Shooting Optimization using the JModelica.org Platform. In *In 7th International Modelica Conference 2009*. Modelica Association, 2009.

[44] The Modelica Association. The Modelica Association Home Page, 2007. http://www.modelica.org.

[45] Jan Tuszynskia, Mathias Persson, Johan Åkesson, Johan Andreasson, and Magnus Gäfvert. Model-based approach for design and validation of a novel concept of public mass transport. In *21st International Symposium on Dynamics of Vehicles on Roads and Tracks*, 2009. Accepted for publication.

[46] University of Heidelberg. MUSCOD-II Home Page, 2009. http://www.iwr.uni-heidelberg.de/~agbock/RESEARCH/muscod.php.

[47] Triple-J Group for Molecular Cell Physiology University of Stellenbosch. PySUNDIALS: Python SUite of Nonlinear and DIfferential/ALgebraic equation Solvers, 2009. http://pysundials.sourceforge.net/.

[48] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–58, 2006.

Building and Solving Nonlinear Optimal Control and Estimation Problems

Jan Poland Alf J. Isaksson
ABB Corporate Research
<jan.poland@ch.abb.com> <alf.isaksson@se.abb.com>

Peter Aronsson
MathCore Engineering AB
< peter.aronsson@mathcore.com>

Abstract

We introduce a tool for obtaining optimal control and estimation problems from graphical models. Graphical models are constructed by combining blocks that can be implemented in Modelica or taken from a palette. The models can be used for predictive control, moving horizon estimation, and/or parameter estimation. We show that the solution time and robustness of the resulting nonlinear program strongly depends on the way the model was built and translated. These observations yield modeling guidelines for increasing robustness and efficiency of the optimization. In particular, we find out that eliminating as many variables as possible from the optimization problem may help a lot.

Keywords: Modelica; Optimization; Optimal Control; State Estimation; Receding Horizon; MPC; MHE

1 Introduction

Model-based methods have been important in many industrial applications for a long time, and their importance still increases today. One typical application field is simulation, where computer models are used to approximate physical processes to great accuracy. Today's models used for simulation are often very complex.

In contrast, computational limits are reached much earlier if a model is used for (online) *optimization.* Nevertheless, Model Predictive Control (MPC) [1] is nowadays a widely applied optimal control method which works by translating the model to an optimization problem, with the help of a performance measure ("cost function") defined in terms of the variables in the model. As opposed to other optimal control methods such as the linear quadratic regulator (LQR), this allows to accommodate constraints, which is very important in practice. In most applications, the optimization problem is formulated and solved for a fixed horizon in time, and the resulting first control move is applied to the plant. This procedure is applied repeatedly ("receding horizon control").

Most MPC applications work with models that are discrete in time or discretized. Depending on the type of the model (linear, nonlinear, involving continuous or discrete variables or both) and the cost functions, different types of optimization problems can arise: linear programs (if both the model and the cost function are linear), quadratic programs (linear model and quadratic cost function), mixed integer linear / quadratic programs (linear model with discrete variables), or (mixed integer) nonlinear programs (NLP) (resulting from a nonlinear model without or with discrete variables). We restrict our attention to the online optimization approach, where the full optimization problem has to be solved in each time step (as opposed to approaches where this is avoided, such as explicit MPC). Hence, the question if MPC can be used efficiently and robustly for an application can be answered in the affirmative if an appropriate solver for the optimization is available.

Consequently, most existing industrial applications use MPC based on *linear* models. Furthermore, since reliable and efficient mixed integer linear solvers have been available for some time now, also models with discrete variables become increasingly popular [2]. On the other hand, nonlinear models result in nonlinear programs (NLPs) which are much harder to solve in general. In particular, solving to global optimality is not possible unless the problem is small or additional structure is given (e.g. convexity holds). Consequently, for nonlinear MPC (NMPC), proving guarantees on the performance, stability, etc.

DOI: 10.3384/ecp09430004

is often impossible. Still, there have been successful applications of NMPC, e.g. [3].

Recently, both computational hardware and nonlinear solvers have become more powerful, making nonlinear MPC applicable for more complex models in principle. In this paper, we will solve NMPC problems without further considering provable performance guarantees, stability, etc. All results in this work have been obtained with IpOpt [4]. We will see later on that it happens quite easily that models are translated to NLPs that are highly multimodal and very difficult to solve. Hence, it is important to construct models in a favorable way. Showing how to do so is one focus of this paper.

Design of the model is rightly considered to be the most difficult and involved task when constructing a model predictive controller. In industrial applications in particular, it is highly desirable that engineers with a moderate mathematical background are able to do so. For this aim, *graphical modeling environments* are especially appropriate: Models are constructed by using blocks and connecting them by lines. Each line represents a signal that leaves one block and enters another block[1].

Blocks should be intuitively understandable functions, e.g. summation of signals, some (nonlinear) function of a signal, or an integrator. A model library or palette should be available which contains a sufficiently flexible collection of predefined blocks, while it must be possible to implement customized blocks. In the present tool, this is done in Modelica. In short, the main graphical modeling functionality of existing Modelica environments (e.g. Dymola, MathModelica) is desirable.

A corresponding graphical modeling environment for linear models with continuous and discrete variables has been realized in ABB's commercial control and optimization platform Expert Optimizer [8], [5]. Combination of blocks is based on matrix multiplication in principle, and the resulting optimization problems (linear programs, quadratic programs, mixed integer linear programs, or mixed integer quadratic programs) are internally represented as matrices. Blocks can be implemented in a description language for this kind of hybrid systems, HYSDEL [6].

We will see that for nonlinear optimization, the way how blocks are implemented and combined can really make a computational difference. Standard graphical Modelica environments (e.g. Dymola, MathModelica) treat a model as a DAE system, and each block that is added to the system typically adds to the number of variables in the system. For instance, if the squared difference of some process variable x to some other signal is of interest, one could attach a corresponding difference to the signal x and then a square function to the difference. Usually, all these quantities will become extra variables in the DAE system. For simulating the system, this will not introduce particular difficulties. For optimization however, it can be very important that these extra variables do not enter the system, but are eliminated. In our framework, still Modelica code written in MathModelica is used to realize user-defined blocks[2]. When connecting blocks however, the chain rule is the crucial instrument that we use to eliminate variables.

In the tool we present in this work, graphical models are formulated (stated) in Matlab/Simulink. They are then used to state objective, constraints, derivatives, Jacobian and Hessian of the associated optimization problem by recursively parsing the model for each evaluation.

So far, we have been talking only of optimal control problems. Typical MPC needs starting values for all states of the model in order to compute and optimize future trajectories. If not all states are observed, then the model can be conveniently used for estimating them, using the *moving horizon estimation* (MHE, [10]) approach. Here, for a fixed number of time steps in the past, an optimization problem is formulated and solved in order to find those values for the state variables that are most in accordance with the observations and the model dynamics. Technically, the MHE task translates to a similar optimization problem as MPC. MHE can be extended to estimating some of the model parameters by adding them as states to the model.

The optimization framework discussed in this paper is similar to ABB's Dynamic Optimizer described in [7], but has a different scope: It is located at a higher level of a plant's automation system and to be integrated into Expert Optimizer, and it focuses on highly customizable modeling by offering a direct access to the graphical modeling environment. As opposed to other modeling frameworks, it *directly* translates a graphical model into an optimization problem, where care is taken to perform the transla-

[1] Note that this is less expressive than standard Modelica, which is object-oriented, and where signals can represent causal structure. However, since our graphical modeling environment allows importing Modelica blocks from MathModelica (see below), we do not lose Modelica's full expressiveness.

[2] Actually, MathCore and ABB have developed together the ABB edition of MathModelica, which the optimization framework we discuss is based on.

tion in a way that robust and fast-to-solve optimization problems are obtained. One key for this is variable elimination, as we will see below.

2 Translating a Graphical Model

Consider, for an illustration, the well-known inverted pendulum model

$$\ddot{\theta} = \frac{g \sin(\theta)}{l} + u,$$

where θ is the angle of the pendulum (the upright position is at $\theta = 0$), g is the gravitational force, l is the length of the pendulum (in the experiments below, we used g / l = 0.5), and u is (a constant multiple of) the torque applied. A graphical model representing this is shown in the red framed part of Figure 1. Here, the constants are hidden in the "plus" block, and the evolution of θ and $\dot{\theta}$ is implemented by single input single output linear time-continuous dynamics

$$\begin{aligned} \dot{x} &= Ax + Bu, \\ y &= Cx + Du. \end{aligned} \quad \text{(eq. 1)}$$

Each of the blocks "theta" and "theta_dot" has one own state, and the parameters are set to A = 0, B = 1, C = 1, and D = 0, respectively.

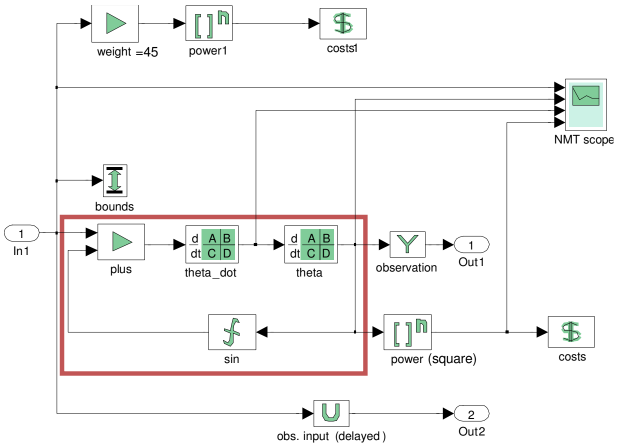


Figure 1: An inverted pendulum model

The implementation shown in Figure 1 is the Simulink model, which allows for a maximum variable elimination. A functionally equivalent model can be implemented graphically in MathModelica shown in Figure 2. Also, the following Modelica listing represents the same model:

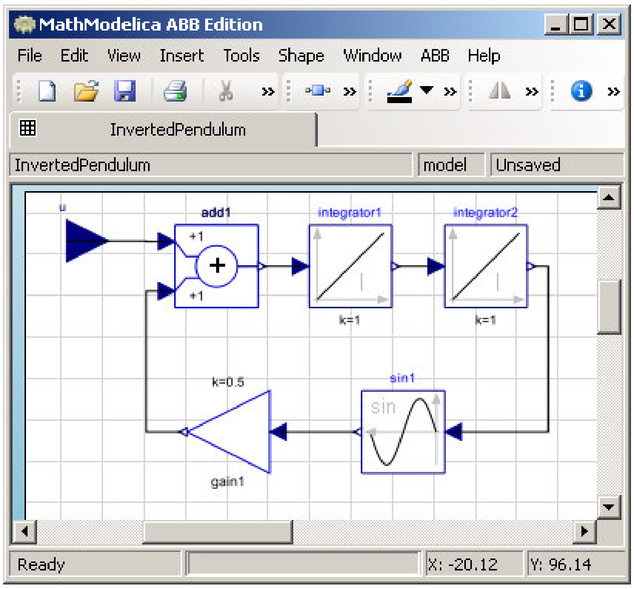


Figure 2: The inverted pendulum model graphically implemented in MathModelica

```
block InvertedPendulum
 parameter Real g;
 parameter Real l;
 Real theta;
 Real th_dot;
 output Real y;
 input Real u;
equation
 der(theta) = th_dot;
 der(th_dot)= g*sin(theta)/l+u;
 y=theta;
end InvertedPendulum;
```

2.1 The Optimal Control Problem

Translating a dynamic model into an optimal control problem including user-defined constraints is a well-established method: Let $x_t = (x_t^j)_{j=1}^n$ be the vector of variables in the model *realized* in a single time step *t*. A *realized* variable is a model variable that is present in the optimization problem, as opposed to a *virtual* variable, which is not present. Then an optimal control problem, discrete in time and for a horizon of $M > 0$, is stated as

$$J^{opt}(x) = \sum_{t=0}^{M} \text{cost}_t(x_t)$$

$$s.t.$$

$$x_t = \text{dynamics}(x_{t-1}) \; for\ all\ 1 \le t \le M,$$

$$\text{equations}(x_t) = 0 \; for\ all\ 0 \le t \le M,$$

$$\text{constraints}_t(x_t) \le 0 \; for\ all\ 0 \le t \le M,$$

$$x_0^j = x_{start}^j \; for\ all\ j \in states.$$

Here, the realized variables comprise at least the inputs to be optimized (and typically they comprise more variables). Costs and constraints may be time dependent, and x_{start} contains the starting values of the *states*. Note that this framework also permits the formulation of a static optimization problem, where $M = 0$ and $states = \varnothing$. Figure 1 shows how cost functions can be graphically stated.

This optimization problem is solved, in the present work, always with IpOpt [4]. We provide all derivative information up to the Hessian, computed analytically, to the solver.

Translation of Modelica components is done by MathModelica ABB Edition.

If we are dealing with a time continuous system, the system must be discretized. Here and in the following we assume that discretization is performed according to implicit Euler, i.e. a differential equation

$\dot{x} = f(x,u,...)$ is translated to

$\mathrm{x}_t = x_{t-1} + \Delta t \cdot f(x_t, u_t, ...)$.

A crucial question to be posed here is which respectively how many variables should be realized as part of x_t. It is clear that at least the inputs to be optimized and the states need to be realized (unless we are able to and desire to recursively solve the evolution equations for the states). However, we can include more or less of the variables defined by the equations into our optimization problem[3]. This decision has a significant impact on the computational time required to solve the resulting optimization problem, as well as the robustness of the solving. We show two numerical examples at this point and draw some conclusions.

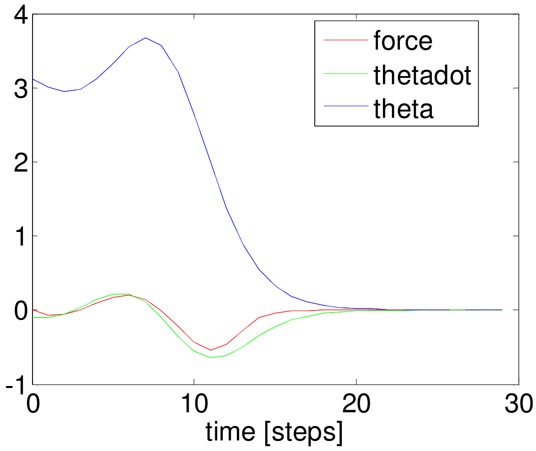


Figure 3: Swing-up trajectory for the inverted pendulum

For the first example, we consider the inverted pendulum model. The weights and bounds on the input are tuned in a way that the optimum is a swing-up, as shown in Figure 3. In the following table, we evaluated the probability that IpOpt finds this optimal solution, starting from randomly initialized values (uniformly in [0, 10]) for all realized variables. We further show the time IpOpt requires on average, as well as the average quality of the successful solutions: the number of time steps after which the predicted trajectory converges to the target. The averages of 500 runs each are shown.

Realized variables	Success rate[%]	Quality [steps]	Time [s]
States and input only (3 per step)	41.2	22.7	0.27
States, input + input and output of θ (5 per step)	42.4	23.0	0.37
As above plus input and output of $\dot{\theta}$ (6 per step)	46	22.6	0.42
As above plus output of the "sin" block (7 per step)	37.4	22.7	0.55

The second numerical example we show is a static nonlinear optimization problem defined by the model in Figure 4, which represents the equation

$$y = u_2^2(u_1-u_2)^3\left(2(u_1-8)^2+1+8.649\cdot10^{-9}\exp(2u_1)-4.65\cdot10^{-5}\exp(u_1)\right)+5(u_1-8)^2$$

$s.t.$

$5 \le u_1 \le 9.8707$

$1 \le u_2 \le 4.$

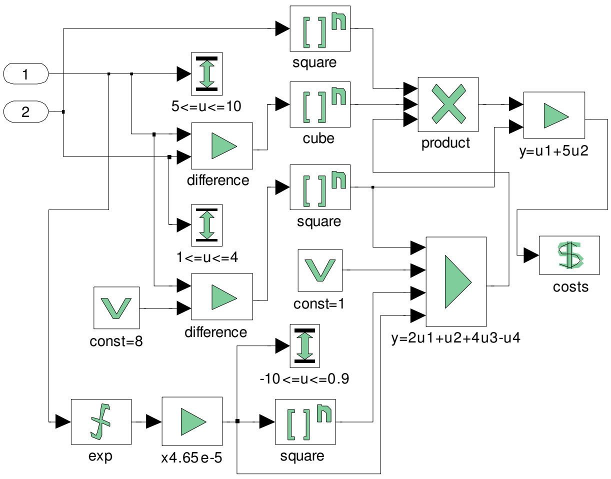


Figure 4: A static nonlinear optimization problem

[3] Not including, i.e. eliminating a variable that appears in a cost function or a constraint implies that we will need to use the chain rule to compute its derivatives.

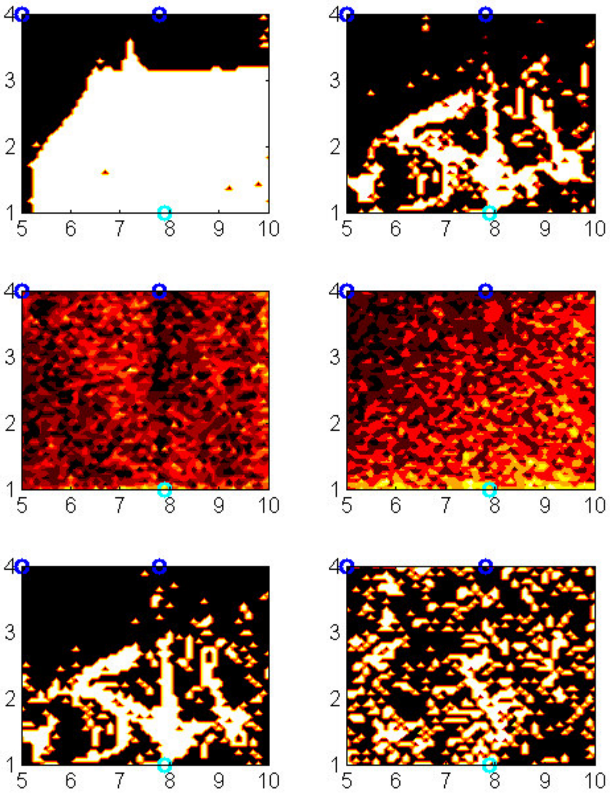


Figure 5: Static nonlinear optimization problem: Probability of converging to the global optimum for six different scenarios, shown in dependence of the 2-dimensional starting points. Hot colors indicate high probabilities, the large area in the top left plot is probability one. The globally optimal solution is shown as a cyan circle, there are two local optimal shown as blue circles. Top left: only the inputs are realized (1), top right: the inputs and the output of the product are realized (2), middle left: the inputs and in- and outputs of the product are realized (3), middle right: all signals are realized (4), bottom: scenarios (2a) and (3a) with consistent initialization.

The first input variable is plotted over [5, 10], the second input variable over [1, 4]. We test four different scenarios: (1) realizing only the input variables (i.e. 2 variables), (2) realizing in addition the output of the product (i.e. 3 variables), (3) realizing in addition the inputs of the product (i.e. 6 variables), and (4) realizing all signals in Figure 4. The resulting computation time and probability of finding the global optimum are summarized in the table below. Figure 5 shows the probabilities of convergence as a function of the starting point for the two inputs (all other realized variables have been initialized with Gaussian distributed values). We observe that for starting points around the global optimum, we reliably converge in scenario (1), while this is no longer true in scenario (2) and seems almost completely arbitrary in scenarios (3) and (4). If we modify scenario (2) and (3) such that the realized variables are initialized with the values corresponding to the inputs instead of Gaussian, we get scenarios (2a) and (3a), and the computation times and probabilities can be seen below.

Scenario	Success rate [%]	Time [s]
(1)	62.0	0.004
(2)	22.1	0.004
(3)	10.9	0.021
(4)	20.2	0.01
(2a)	22.1	0.005
(3a)	27.4	0.022

2.2 Consequences and Recommendations for realizing or eliminating variables

The two examples clearly show that the time required for solving the optimization problem increases with the number of realized variables. Although this has been observed for IpOpt only here, it is realistic to say that the statement can be generalized to any solver. In particular, more variables require more equality constraints to be satisfied and hence typically increase the number of iterations needed by the solver. In the second of the above examples, scenarios (3) and (4) need about 20 times more CPU times than (1) and (2). Looking a bit closer into what happens when IpOpt optimizes scenarios (3) and (4), what can be observed is the following: In the first step of the optimization, the equality constraints are non-satisfied to a high degree, causing the actual solution to move very far away, where in particular the input variables leave the box. Then, it takes quite long for the solution to get back into the region which is feasible for the input variables. The dependence on the exact starting points is very sensitive (chaotic), which causes the rough behavior of the middle two graphs in Figure 5. This happens to a much lesser degree in scenario (2), and not at all in scenario (1). Even when we use a consistent initialization, i.e. one that satisfies the constraints, we see in the lower two graphs that we do not improve the roughness much.

In the inverted pendulum example, we see that similar things happen (not as strongly as in the other example). However, introducing additional variables may to a certain extent even help the optimizer to find the global optimum.

In general, we expect that the optimization problem will be not only more efficiently, but also more robustly solved, the *less* variables are realized. In a MPC situation, typically, we start already close to the optimal solution, since the starting solution is obtained by the optimum from one step before. Hence, the roughness observed for the scenarios (2), (3) and (4) of the second example is a very undesirable property here: it increases the probability that even in the absence of major disturbances, we will not be able to track the optimal trajectory.

We summarize: with IpOpt as underlying solver, our results indicate that for both robustness and solution time it is favorable in general to realize as few variables as possible. For the robustness, there are exceptions where realizing *some* variables may be good. The solution time is always expected to increase with more variables, this should also hold for other solvers.

2.3 The Estimation Problem

The optimal control problem needs in particular starting values for all states. If they are not directly observed, they can be estimated with a model-based observer that uses the same model as MPC. To this aim, the following cost function ("moving horizon estimator") can be minimized [10]:

$$J^{est}(x) = \left\|\varepsilon^{initial}\right\|_2^2 + \sum_{t=-N+2}^{0}\left\|\varepsilon_t^{state}\right\|_2^2 + \sum_{t=-N+1}^{0}\left\|\varepsilon_t^{obs}\right\|_2^2$$

$$s.t.$$

$$x_t = \text{dynamics}(x_{t-1}) + \sigma_t^{state}\varepsilon_t^{state} \;\; for\ all\ -N+2 \le t \le 0,$$

$$\text{equations}(x_t) = 0 \;\; for\ all\ -N+1 \le t \le 0,$$

$$\text{constraints}_t(x_t) \le 0 \;\; for\ all\ -N+1 \le t \le 0,$$

$$y_t^{obs} = y(x_t) + \sigma_t^{obs}\varepsilon_t^{obs} \;\; for\ all\ -N+1 \le t \le 0,$$

$$x_{initial}^{j} = x_{-N+1}^{j} + \sigma_{initial}^{j}\varepsilon_{initial}^{j} \;\; for\ all\ j \in states.$$

Here, we minimize the quadratic state noise ε_t^{state}, measurement noise ε_t^{obs}, and initial state noise $\varepsilon_{initial}$, with weighting factors that are standard deviations (i.e. scales of the noise) σ_t^{state}, σ_t^{obs}, and $\sigma_{initial}$, respectively. For the standard deviations, the valid range is [0, ∞]. The quantity y_t^{obs} contains the observations available at time *t*, while $x^j{}_{initial}$ is the target for the initial state (if desired). The constraints may but need to be the same as in the definition of J^{opt}, in general it makes sense to consider a subset here.

It is just a matter of notational convenience to include the noise variables ("slack variables") ε_t^{state}, ε_t^{obs}, and $\varepsilon_{initial}$ explicitly in the formulation of the optimization problem. It not necessary to realize them, in fact, it may be more efficient not to do so. In the following simulations, we do not realize the noise variables.

Figure 6 shows an example trajectory of the estimation (and then optimization) with the inverted pendulum model from Figure 1. There are two observations defined, namely the input force and θ. The speed $\dot{\theta}$ is not observed, but is reliably estimated.

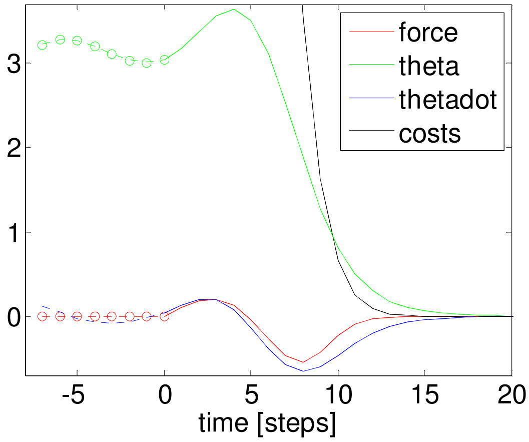


Figure 6: Estimation and optimization trajectory for the inverted pendulum model. Observations are marked by circles, dotted lines are estimations.

2.4 Reducing Interactions in Case of Poor Observability

The toolbox presented in this work facilitates combining models of a large plant from its parts, and optimizing the plant as a whole. In practice however, the whole plant may not be sufficiently observable to allow a reliable state estimation based on the complete model. Namely, if there are relatively poor measurements available in the parts of the plant, the estimation will use the model in order to obtain the numerically most likely estimator for the joint state of the plant. If the model does not very well match the plant, this procedure can easily result in estimates that are poor, or that drastically change from one time step to the next. Of course, this is highly undesirable, as it prevents the model-based controller from stabilizing the plant.

Here, we describe two techniques for cutting the model in several parts that do not interact in the estimation. Hence, the measurements available within one part of the plant are used only within that part, and do not interact with different parts.

2.4.1 Cutting with dynamics

If the parts to be separated are connected by one or more blocks that evolve dynamically, e.g. a delay or a filter, then it is possible to not estimate the states of these blocks, but rather provide their values by an external simulation. In the estimation problem, the dynamics of these blocks are removed from the cost function / constraints.

2.4.2 Cutting without dynamics

If two parts to be separated are connected without dynamics, then the following method can be used: The connection is broken, the estimation is executed with open connection, and some constant value is used as input for the destination of the broken connection. After the estimation, the states in the part connected to the source of the broken connection have attained their values. Now, also the value the broken signal should have is known. Hence, we need to use this value as an input for the destination of the broken connection and rerun the estimation. In general, if the model is divided into *n* parts, we need to run the estimation for *n* times. Note that this method may not converge if the dependence of the parts is circular, hence it should be used only for tree structures (and we should run the estimation *n* times, where *n* is then length of the longest branch of the tree). Also, a signal to be cut must have a clear direction, providing its value from the source to the destination. It must not be used as an implicit input, which enters the dynamics or equations at the source and is defined by some constraint at the destination.

2.5 Parameter Estimation

If observability permits, parameters can be coded as states and estimated by the state estimation. One standard procedure to do so is coding a parameter as a state which evolves as a constant and admits state noise.

3 A Building Block Library

Here we describe a basic set of blocks that is sufficient to model a broad variety of plants.

3.1.1 Single input single output linear time-continuous dynamics, as described in (eq. 1).

3.1.2 Single input single output linear time-discrete dynamics:

$$x_t = Ax_{t-1} + Bu_{t-1},$$
$$y_t = Cx_t + Du_t.$$

3.1.3 Constant

3.1.4 Adaptive constant / bias term, see 2.5.

3.1.5 Nonlinear function, to be chosen from {exp, log, sin, cos, tan, sinh, cosh, tanh, asin, acos, atan}

3.1.6 Power, i.e. $y = u^n$

3.1.7 Weighted sum (linear combination), i.e. $y = a_1u_1 + \ldots + a_nu_n$. Also implements gains.

3.1.8 Product: $y = u_1 \times \ldots \times u_n$

3.1.9 *n*-step delay

3.1.10 Generalized absolute function. The function y = abs(u) can be implemented using an auxiliary variable z to be minimized, and including $z \geq u$ and $z \geq -u$ to the constraint set. This block implements a general convex piecewise linear function with the same principle.

3.1.11 Upper and lower bounds

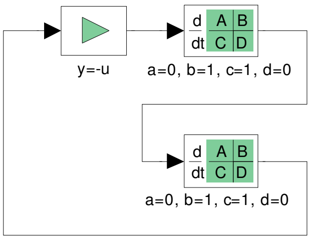
3.1.12 Marks a signal as a cost function

3.1.13 Marks a signal as an observation

3.1.14 Cut for estimation without dynamics as described in 2.4.2.

3.1.15 Users can implement own Modelica blocks.

Note that these basic blocks are sufficient to express a wide range of coupled ODE systems. For example, the following model encodes a harmonic oscillator.



4 Applications, Discussion and Conclusions

We have presented several simulations with toy examples in this paper. The modeling framework presented here will become part of ABB's advanced process control and optimization platform Expert Optimizer, which is described e.g. in [8]. This article and [7], [9] report some results from application of similar modeling techniques to industrial plants. Our modeling framework has been based on the results of ongoing collaboration of ABB and MathCore, [11] describes an early application of related methods.

We have presented a graphical modeling framework and a procedure to directly translate graphical models into optimization problems. We have collected some evidence that, for formulating the optimization problems, it is favorable for speed and robustness to realize as few variables as possible.

References

[1] J. Maciejowski: Predictive Control with Constraints. Pearson Education Limited, Prentice-Hall, Essex, United Kingdom (2002).

[2] A. Bemporad and M. Morari: Control of systems integrating logic, dynamics, and constraints, Automatica 35(3) (1999), 407-427.

[3] R. Franke, M. Rode, K. Krüger: On-line Optimization of Drum Boiler Startup. Proceedings of the 3rd International Modelica Conference, Linköping, November 3-4, 2003.

[4] A. Wächter and L. T. Biegler: On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25-57, 2006.

[5] E. Gallestey, D. Castagnoli, and A. Stothert: Method of generating optimal control problems for industrial processes, European Patent EP1607809 (2006).

[6] F.D. Torrisi and A. Bemporad: HYSDEL - A tool for generating computational hybrid models for analysis and synthesis problems. IEEE Transactions on Control Systems Technology 12(2) (2004), 235-249.

[7] R. Franke, B. S. Babji, M. Antoine, A. Isaksson: Model-based online applications in the ABB Dynamic Optimization framework. Proceedings of the Modelica Conference, 2008, pp. 279-285.

[8] K. Stadler, E. Gallestey, J. Poland, G. Cairns: Optimal Trade-Offs. ABB Review, 2/2009, pp. 17-22.

[9] E. Dahlquist, F. Wallin, H. Ekwall: Dynamic simulators for process control and optimization as well as for operator training in pulp and paper industry, SIMS - 43rd Conference on Simulation and Modeling, Oulu, 2002

[10] C.V. Rao: Moving Horizon Strategies for the Constrained Monitoring and Control of Nonlinear Discrete-Time Systems, Ph.D. Thesis, University of Wisconsin, 2000.

[11] J. Pettersson, L. Ledung and X. Zhang: Decision support for pulp mill operations based on large-scale on-line optimization, Preprints of Control Systems 2006, Tampere 6-8 June, 2006.

Improving Convergence of Derivative-Based Parameter Estimation with Multistart Parameter Clustering Based on DAE Decomposition

Atya Elsheikh[*] Katharina Nöh Eric von Lieres[†]
Research Center Jülich, Institute of Biotechnology 2
{a.elsheikh, k.noeh, e.von.lieres}@fz-juelich.de

Abstract

Derivative-based optimization methods for parameter estimation require good start values in order to converge to the global optimum. A conventional multistart strategy is often not practical for identifying such start values, especially for high dimensional problems. Moreover, the computational efforts for each iteration of the optimizer are significantly increased by the computation of parameter sensitivities. We hence present a multistart recursive clustering strategy that utilizes DAE decomposition algorithms, in particular Tarjan's and tearing algorithms. These algorithms are also used by standard Modelica compilers for improving the performance of solving large DAE systems. Our key concept is to provide a natural decomposition of the parameter estimation problem into smaller clusters (i.e. subproblems), each of which requires fewer start values and less computation. The resulting local minima are taken as start values for enlarged subproblems, and so forth until good start values for the original problem are found. This approach serves to improve global convergence and computational speed of multistart derivative-based optimization strategies for large sparse DAE systems.

Keywords: Parameter estimation, global optimization, cluster methods, DAE decomposition algorithms

1 Introduction

1.1 Problem Specification

Differential algebraic equations (DAE) are widely used in modeling and simulation applications, for example in Electrical Engineering, Biochemical Engineering, Mechanics and Thermodynamics. In most applications the values of some model parameters are not known a priori and must hence be estimated from measured data. A parametrized DAE system is formally given by:

$$F(\dot{x},x,p,t)=0, \quad x(0)=x_0 \tag{1}$$

with a function $F : R^{2N+M+1} \rightarrow R^N$ that is sufficiently smooth with respect to the state variables $x \in R^N$ and parameters $p \in R^M$. Typical parameter estimation problems aim at minimizing the distance between simulation results $x(p,t)$ and measurement data $\tilde{x}(t_j) \in R^N$ at discrete time points t_j with $j = 1..T$ in the sense of least squares:

$$\begin{aligned} r &= \frac{1}{2}||Q||_2^2 \in R && \text{(2a)} \\ Q &= [q_1,..,q_T] \in R^{N\cdot T} && \text{(2b)} \\ q_j &= \tilde{x}(t_j)-x(p,t_j) \in R^N, \quad j=1..T && \text{(2c)} \end{aligned}$$

Note that $Q \notin R^{N\times T}$ is a vector and not a matrix. For start values $p^0 \in R^M$ that are chosen sufficiently close to a local optimum $p^*_{loc} \in R^M$, the Gauss-Newton algorithm converges to p^*_{loc} by iterating the following scheme [1, 2]:

$$\begin{aligned} p^{i+1} &= p^i - (r^i_{pp})^{-1}\cdot r^i_p \\ &\approx p^i - \left(\left[x^i_p\right]^T \cdot x^i_p\right)^{-1} \cdot \left[x^i_p\right]^T \cdot Q \end{aligned}$$

where

$$\begin{aligned} r^i_p &= \frac{\partial r}{\partial p}(p^i) \in R^M \\ r^i_{pp} &= \frac{\partial^2 r}{\partial p^2}(p^i) \in R^{M\times M} \end{aligned}$$

are first and second order partial derivatives of the residual r with respect to the unknown parameters, and

$$x^i_p = \frac{\partial x}{\partial p}(p^i) \in R^{N\cdot T\times M}$$

[*] Evonik Industries are acknowledged for financial support within the BMBF co-funded project SysMAP (project no. 0313704)

[†] To whom correspondence should be addressed.

DOI: 10.3384/ecp09430039

are first order partial derivatives of the model solution x with respect to the unknown parameters, also referred to as parameter sensitivities.

1.2 Common Problems in Derivative-Based Optimization

Practical application of derivative-based optimization methods (for instance Gauss-Newton) for obtaining a global optimum

$$p^* = \underset{p \in \{p^*_{loc}\}}{\arg\min}\, r(p)$$

is typically hindered by the following problems:

1. *Good start values are hard to obtain:* The resulting optimization problem can be efficiently solved when good start values are known. However, from most arbitrarily chosen start values the algorithms either diverge or converge only to a local optimum. This problem will be discussed in more depth in section 1.3.

2. *The Hessian r_{pp} is usually approximated by $(x^i_p)^T \cdot x^i_p$ which is often semi-singular:* In this case the inverse cannot be exactly computed but is usually approximated by a pseudo-inverse using singular value decomposition. This approximation is however inaccurate, in particular for high dimensional matrices in large optimization problems.

3. *Parameter sensitivities x_p are expensive to compute:* These sensitivities are usually computed by either solving the original DAE system (equation 1) together with the associated computationally expensive sensitivity equations:

$$F_{\dot{x}} \cdot \dot{x}_p + F_x \cdot x_p + F_p = 0, \quad x_p(0) = 0 \qquad (3)$$

   or by using less precise finite difference methods.

1.3 Identification of Successful Start Values

The determination of suitable start values for high dimensional parameter estimation problems that are located within the global convergence area of a derivative-based optimization algorithm is not trivial because:

1. The space $S_p \subseteq R^M$ of parameter values that are physically admissible is typically large.

2. The DAE system extended with parameter sensitivities (equations 1 and 3) is usually solvable only in a subspace $S_{sol} \subseteq S_p$.

3. Some parameter values can cause numerical difficulties that are associated with the numerical solver. We denote the subspace covering such parameter values by $S_{diff} \subseteq S_{sol}$. For example, assume that the DAE system (equation 1) together with the sensitivity equations 3 are not stiff in the global optimum p^*, but the optimizer may iterate over parameter values for which these equations are stiff, and hence numerically harder to solve.

4. Nonlinear optimization problems usually have numerous local optima, and global convergence is guaranteed only for start values $p_0 \in N_\varepsilon(p^*)$ from a subspace $S_{N_\varepsilon(p^*)} \subseteq S_p$ around the global optimum p^*.

Start values should hence ideally be chosen within the subspace $S_{conv} = S_{N_\varepsilon(p^*)} \bigcap (S_{sol}/S_{diff})$ in order to efficiently find the global optimum p^*. Conventional multistart optimization strategies are not practical for high dimensional problems in which S_{conv} is notably smaller than S_p. Moreover, computational efforts are significantly increased by the computation of parameter sensitivities.

1.4 Multistart Recursive Clustering

We here present a multistart recursive clustering strategy that is specifically designed to reduce the number of starts required for identifying values in S_{conv} from which the applied derivative-based optimization method globally converges. This multistart strategy heuristically decomposes the optimization problem into smaller clusters (i.e. subproblems), and has similarities with cluster optimization methods. Each cluster C_i defines a parameter estimation subproblem that is characterized by:

- a parameter subset $p_i \in R^{M_i}$ with $\sum_i M_i = M$,
- a variable subset $x_i \in R^{N_i}$ with $\sum_i N_i = N$,
- the corresponding subset of measurement data $\tilde{x}_i \in R^{N_i \times T}$,
- a residual r that is a function only of p_i, x_i and $\tilde{x}_i$ (compare equation 2).

Each cluster C_i is recursively decomposed into subclusters $C_{i,j}$ and so forth. In this way, rather small clusters are created and our strategy begins with separately estimating the respective parameter subsets. Once optima are found, the resulting parameters are taken as new start values and the cluster size is enlarged, and so forth until the complete original system is reconsidered.

In the present study we aim to optimally decompose the parameter estimation problem into smaller subproblems specifically for DAE systems. A related problem has already been researched in the context of speeding up the solution of high dimensional DAE systems, namely finding optimal decompositions of a DAE system into smaller DAE subsystems, each of which solves a subset of state variables. We here determine the clusters C_i with the same DAE decomposition algorithms [7, 8]. These algorithms use intuitive but powerful clustering heuristics to naturally decompose the optimization problem into smaller subproblems with well defined dependencies. The details of this approach will be explained in section 3.

1.5 Practical Aspects

The proposed strategy can effectively solve the previously discussed problems of parameter estimation with derivative-based optimization algorithms:

1. *Fewer starts are required.* The recursive multistart parameter clustering strategy helps to rationally identify good start values from which derivative-based optimization algorithms converge to the global optimum. This optimum can hence be identified with significantly fewer starts for the full system as compared to conventional multistart strategies.

2. *Smaller subproblems are processed most of the time.* The parameter estimation problem for each individual cluster C_i involves fewer parameters p_i, fewer state variables x_i, and mostly also fewer model and sensitivity equations. The full set of system equations is solved only for few iterations from start values that are already close to the global optimum. Moreover, the approximations to the Hessian are often more accurate and numerically better conditioned for small clusters.

3. *Automatic differentiation enables efficient computation of parameter sensitivities.* We apply ADModelica (Automatic Differentiation of Modelica), a self developed tool that exploits high level Modelica compiler techniques for generating Modelica code for efficiently and precisely computing the required parameter sensitivities. Automatic differentiation involves significantly less equations as explicit differentiation and finite difference methods [3, 4, 5]. Moreover, we exploit the known structure of the sensitivity equations for reducing compilation and simulation time on both serial and parallel computers [6].

1.6 Outline

The remainder of this contribution is structured as follows: In section 2 we give an overview of the DAE decomposition algorithms that we reuse in the context of parameter estimation. The implementation of the proposed strategy is then presented in section 3 in a simplified form without low level tricks, although fine tuning could further improve the results that are reported in section 4. In section 5 we conclude with discussion of potential limitations as well as future developments and applications of the presented method.

2 DAE Decomposition

2.1 Structure Digraph of DAE Systems

Directed graphs are used as intermediate representations for symbolical handling and simulation of DAE systems that are automatically generated from Modelica models. Figure 1 shows the simple example of a linear reaction chain, that is mathematically represented by equation 4 with initial values $X_i(0) = X_i^0$ for $i = 1..5$.

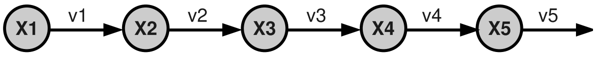


Figure 1: Five nodes with capacities X_i that are connected by flows v_i and subjected to mass conservation.

$$\dot{X}_1 = -v_1 \qquad (4a)$$

$$\dot{X}_i = v_{i-1} - v_i, \quad i = 2..5 \qquad (4b)$$

$$v_j = v_{j,max} \cdot \frac{X_j}{k_j + X_j}, \quad j = 1..4 \qquad (4c)$$

$$v_5 = d \cdot X_5 \qquad (4d)$$

The differential equations 4a to 4b are also referred to as rate equations, and the algebraic equations 4c to 4d determine the flow rates v_j as functions of the maximal flow rates $v_{j,max}$, saturation parameters k_j with $j = 1..4$, and of the degradation rate d. Figure 2a shows the *bipartite graph representation* of the DAE system 4 according to the following definition:

Definition (Bipartite Graph Representation) A bipartite graph representation $G = (V,E)$ of a DAE system (equation 1) consists of a set of vertices:

$$\begin{aligned} V &= V_e \cup V_x \\ V_e &= \{e_i : \textit{equation number } i\} \\ V_x &= \{x_i : \textit{variable number } i\} \end{aligned}$$

and a set of edges:

$$E = \{(e_i, x_j) : x_j \textit{ occurs in equation } e_i\}$$

with $i, j \in \{1, ..., N\}$.

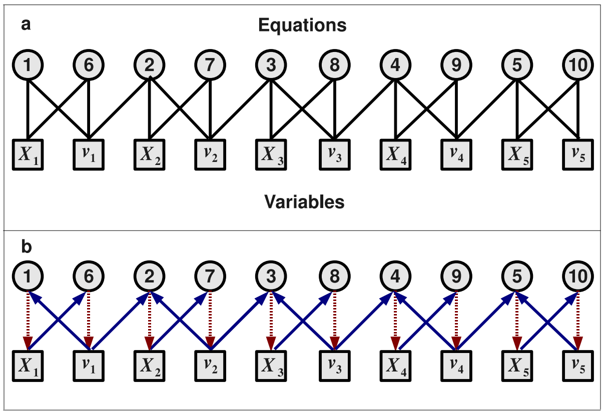


Figure 2: Bipartite graph representation (a) and structure digraph (b) of the example DAE system (equation 4). Red dashed arrow: equation e is explicitly solved for variable x. Blue arrow: variable x is required for solving equation e for another variable.

The bipartite graph representation can be transformed into a *structure digraph* [7], as illustrated in figure 2b for the DAE system from equation 4. This directed graph reveals the causality relations among variables and equations. Ideally each equation e is explicitly solved for one variable x. In practice, however, algebraic loops can occur when a group of equations must be concurrently solved for a group of variables, for instance the ODE system $\dot{x} = y$ and $\dot{y} = x$.

2.2 DAE Decomposition Algorithms

DAE systems that are generated from descriptive Modelica models usually consist of many components and connectors. They are hence high-dimensional and sparse, meaning that only few variables appear in each equation. Instead of solving these equations at once, one can often decompose such systems into smaller blocks of equations. Sequential solution of these blocks reduces overall complexity and consequently speeds up the computation.

The example DAE system (equation 4) can be decomposed into sorted blocks of equations. Application of Tarjan's algorithm [9] to the directed graph in figure 2b yields a set of Strongly Connected Components (SCCs) as shown in figure 3. Each SCC C_i represents an equation block that is solved for a subset of the variables. The SCCs are subjected to a *topological sorting* "<", imposing an order in which the blocks must be solved. $C_i < C_j$ if x_i is required for determining x_j. In figure 3, $C_i < C_j$ if $i < j$, resulting in the solution scheme shown in figure 4.

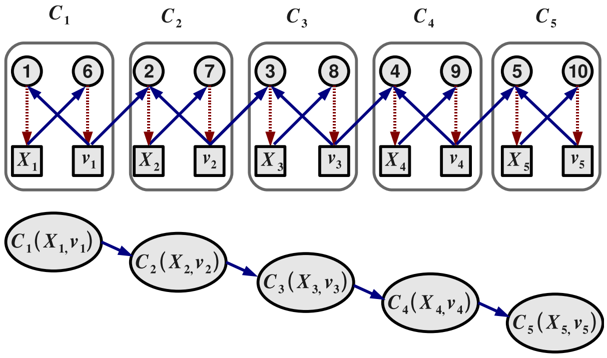


Figure 3: Strongly Connected Components (SCCs) of the structure digraph of the example DAE system.

The block C_1 in figure 3, representing the equations 4a and 4c for $j = 1$, is solved for X_1 and v_1. The value of v_1 is then used to solve the equations represented by the block C_2, and so forth until all variables are determined. A sequential system decomposition as in figure 3 cannot always be achieved. For example consider figure 1 with an additional flow connection from X_5 to X_1. In contrast to figure 3, the resulting digraph now has a cyclic structure, as shown in figure 5.

In this example all equations must be concurrently solved for all variables. Such problems can be solved with Tearing methods [10, 11]: First initial guesses are provided for certain state variables in order to decouple the corresponding SCCs. For example, specification of X_5 would make C_5 independent from C_4. The SCC structure is made acyclic by tearing the graph apart through the *tear variable* X_5 at the connection between C_4 and C_5. The DAE system is then decomposed as before, and the remaining state variables X_i with $i = 1..4$ are sequentially determined. With these solutions the initial guess of X_5 is corrected through v_4, and this process is repeated until X_5 does not significantly change any more. The choice of suitable tear variables generally is a key question of tearing methods. In the present study we apply physical knowledge about the modeled system.

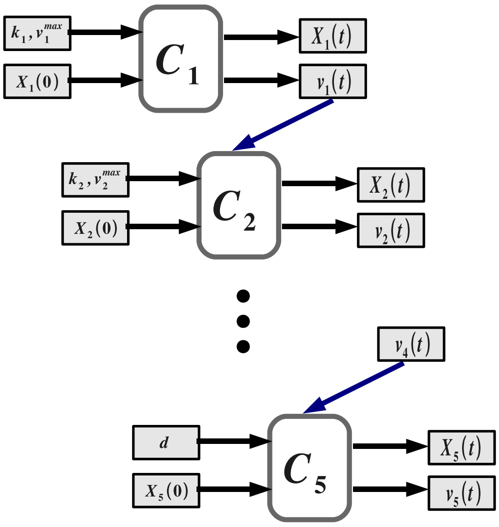


Figure 4: Solution scheme of the equation blocks for the example DAE system (equation 4).

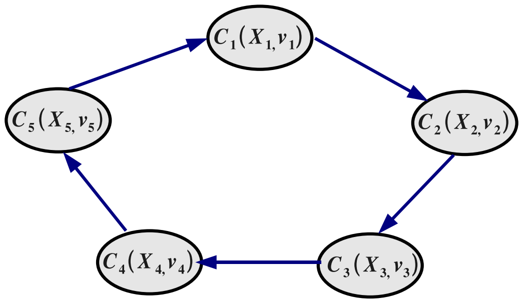


Figure 5: SCCs of the structure digraph of an example DAE system with cyclic structure.

3 Parameter Estimation

We now address the estimation of unknown model parameters by minimizing the distance between simulation results and measurement data. Our basic idea is to decompose the DAE system into SCCs as described in the previous section, and to treat the individual clusters C_i as separate parameter estimation subproblems.

3.1 Example with Linear Structure

Reconsider the model from equation 4 (figure 1), and suppose that $\tilde{X}(t_j) \in R^5$ are corresponding measurement data at discrete time points t_j with $j = 1..T$. Each of the parameter estimation subproblems is characterized by:

- the parameters k_i and $v_{i,max}$ for $i = 1..4$ or, respectively, d for $i = 5$,
- the variables $X_i(t)$ and $v_i(t)$ that equation block C_i is solved for,
- the corresponding measurements $\tilde{X}_i(t_j)$ at times t_j with $j = 1..T$.

The dependency between the subproblems is well-defined, because the fluxes $v_{i-1}(t)$ that are required in an equation block C_i are always determined from the previous equation block C_{i-1} (see figure 4). The relation $C_i < C_j$ for $i < j$ enables to estimate parameters k_j and $v_{j,max}$ from measurements X_j after k_i and $v_{i,max}$ have been estimated for $i < j$.

3.2 Algorithm

Algorithms 1 and 2 formally describe the procedure of our multistart recursive parameter clustering strategy. Three inputs are required:

1. C: A cluster specifying the processed parameter estimation subproblem, in particular the involved parameter, variable and data subsets, as well as the space of feasible parameter values (see section 1.3).
2. *OptAlg*: The optimization strategy to be applied, for example Gauss-Newton algorithm.
3. N: The number of start values for estimating the involved parameter set with a multistart strategy. Alternatively, algorithm 2 can be called with a specific set of start values.

Algorithm 1 first attempts to optimize the parameters in a given cluster C. Algorithm 2 is called for processing a conventional multistart strategy (line 1). If an acceptable residual r is achieved, the solution is returned (line 2), and otherwise the optimization problem is further decomposed.

Algorithm 1 Multistart Clustering Strategy

```
ClusterOptAlg($C$, $OptAlg$, $N$)
input Cluster: $C$
      Optimization Algorithm: $OptAlg$
      Number of Start Values: $N$
output Optima: $p_C^{1..M}$ with $M \leq N$
 1: $p_C^{1..M}$ = Multistart($C$, $OptAlg$, $N$)
 2: if IsAcceptable($p_C^{1..M}$) then
 3:   return
 4: end if
 5: if IsDecomposable($C$) then
 6:   $[C_k, <]$ = SCC($C$)
 7:   for all $k$ do
 8:     $p_{C_k}^{1..M_k}$ = ClusterOptAlg($C_k$, $OptAlg$, $N$)
 9:     for all $j$ do
10:       if $C_k < C_j$ and adjacent($C_k$,$C_j$) then
11:         $C_j \Leftarrow x_k(p_{C_k}^{1..M_k})$
12:       end if
13:     end for
14:   end for
15:   ▷ construct start values from local optima
16:   $p_0^{1..N} = \cup_k p_{C_k}^{1..M_k}$
17:   $p_C^{1..M}$ = Multistart($C$, $OptAlg$, $p_0^{1..N}$)
18: else
19:   ▷ Decompose using a tear variable $x_t$
20:   $[C_k, <]$ = Tearing($C$,$x_t$)
21:   ▷ Start values of all parameters
22:   $C_1 \Leftarrow p_0^{1..N}$
23:   ITR = 0
24:   while ITR < MAX_ITERATIONS do
25:     for all $k$ do
26:       $p_{C_k}^{1..M_k}$=Multistart($C_k$, $OptAlg$, $N$)
27:       for all $j$ do
28:         $C_j \Leftarrow p_{C_k}^{1..M_k}$
29:       end for
30:     end for
31:     $p_0^{1..N} = \cup_k p_{C_k}^{1..M_k}$
32:     $p_C^{1..M}$ = Multistart($C$, $OptAlg$, $p_0^{1..N}$)
33:     if IsAcceptable($p_C^{1..M}$) then
34:       return
35:     end if
36:     ITR = ITR + 1
37:   end while
38: end if
```

Algorithm 2 Multistart Strategy

```
Multistart($C$, $OptAlg$, $N$ or $p_0^{1..N}$)
input Cluster: $C$
      Optimization Algorithm: $OptAlg$
      (Number of) Start Values: $N$ or $p_0^{1..N}$
output Optima: $p_C^{1..M}$ with $M \leq N$
if 3rd Input is $N$ then
  $p_0^{1..N}$ = generateStartValues($C$, $N$)
end if
▷ compute optima from start values
$p_C^{1..N}$ = OptAlg($C$, $p_0^{1..N}$)
▷ Remove poor local optima
$p_C^{1..M}$ = Filter($p_C^{1..N}$)
```

The type of decomposition in algorithm 1 depends on the structure of the underlying DAE system: Linear structures are processed in lines 6-17, and cyclic structures in lines 19-37. The constructed subproblems are sequentially solved (lines 8,26), according to the operator "<". The results of solved clusters are passed to dependent clusters (lines 9,27). Finally, the original problem is reconsidered with the resulting local optima as start values (lines 15-17,31-32).

3.3 Implementation Details

Algorithm 2 applies the optimization algorithm $OptAlg$ to compute local optima $p_C^{1..N}$ of a cluster C_i from N start values that are either self-generated or passed as $p_0^{1..N}$. In our actual implementation, the resulting local optima of each cluster are sorted with respect to the quality of their corresponding residuals r_i. A χ^2-test is then performed, and the worst local optima are removed such that the remaining optima follow a uniform distribution with prespecified confidence level. DAE systems with linear structure are processed with higher confidence levels than DAE systems with cyclic structure.

Moreover, we apply tearing heuristics not for seeking decompositions that are most optimal for efficiently solving the DAE systems, but rather aim at constructing clusters that are small and require only few start values. However, automatic recognition of decompositions that are optimal for the parameter estimation problem is not yet implemented, and tear variables are manually chosen on the basis of physical knowledge about the model topology.

3.4 Example with Linear Structure (continued)

Figure 6 illustrates a run of our algorithm for the example DAE system with linear structure (equation 4, figure 1). First, the parameters of cluster C_1 are estimated with the Gauss-Newton algorithm and a conventional multistart strategy. This strategy generates

N start values for k_1 and $v_{1,max}$, respectively. From $M \leq N$ of these start values the optimization algorithm converges to local optima $k_1^{1..M}$ and $v_{1,max}^{1..M}$.

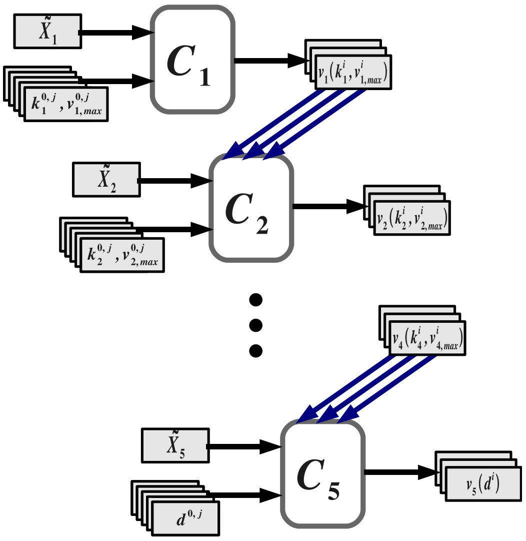

$\tilde{X}_1$
C_1
$k_1^{0,j}, v_{1,max}^{0,j}$
$v_1(k_1^i, v_{1,max}^i)$
$\tilde{X}_2$
C_2
$k_2^{0,j}, v_{2,max}^{0,j}$
$v_2(k_2^i, v_{2,max}^i)$
$v_4(k_4^i, v_{4,max}^i)$
$\tilde{X}_5$
C_5
$d^{0,j}$
$v_5(d^i)$

Figure 6: Multistart clustering strategy for the example DAE system with linear structure (equation 4).

In case the optimization algorithm has not converged for all start values, $N-M$ optima are randomly chosen and copied in order to maintain a full set of N optima. The corresponding N solutions for the state variable v_1 are passed to the next cluster, C_2. The same process is then repeated for C_2 with N start values for k_2 and $v_{2,max}$, respectively, that are randomly paired with the N solutions for v_1, and so on until all parameters are estimated.

Next, larger clusters are constructed by merging smaller clusters. For instance C_1 is merged with C_2, and C_3 with C_4. We apply a binary tree data-structure in order to always merge clusters that are adjacent to each other. Start values are not newly generated at this stage, but the optima from the merged clusters are used. For instance, the optima that were individually estimated for clusters C_1 and C_2 are now together used as start values for the cluster that is merged from C_1 and C_2. This process is recursively performed until the original problem is reconsidered with very good start values for global optimization.

3.5 Example with Cyclic Structure

Figure 7 illustrates a run of our algorithm for the example DAE system with cyclic structure (figure 5). The clusters are defined as before, but the clustering strategy is somewhat different, because the whole DAE system must be concurrently solved for the state variables of all clusters when the parameters of one cluster are estimated. This explains the difference between lines 9 and 27 in algorithm 1.

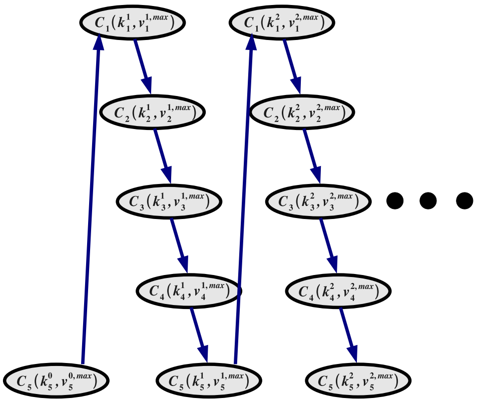

$C_1(k_1^1, v_1^{1,max})$
$C_1(k_1^2, v_1^{2,max})$
$C_2(k_2^1, v_2^{1,max})$
$C_2(k_2^2, v_2^{2,max})$
$C_3(k_3^1, v_3^{1,max})$
$C_3(k_3^2, v_3^{2,max})$
$C_4(k_4^1, v_4^{1,max})$
$C_4(k_4^2, v_4^{2,max})$
$C_5(k_5^0, v_5^{0,max})$
$C_5(k_5^1, v_5^{1,max})$
$C_5(k_5^2, v_5^{2,max})$

Figure 7: Multistart clustering strategy for the example DAE system with cyclic structure.

We treat parameter estimation problems with cyclic structure similarly to the decomposition of DAE systems with cyclic structures for fast solution. Again, N start values are chosen for each unknown parameter (line 21). The N initial guesses for one cluster are fixed, for instance k_5 and $v_{5,max}$ in cluster C_5. Then the parameters of all clusters are sequentially estimated with a conventional multistart strategy. The resulting optima are passed to all clusters (line 27), and the values of those parameters that were originally fixed are refined. Finally the parameters of the original DAE system are estimated with the combined optima of the clusters as start values. This procedure is repeated for several rounds, until either the residual r of the original DAE system drops below a prespecified threshold or the number of rounds exceeds a predefined maximum.

4 Benchmark

Equation 5 defines an abstract example that we use as benchmark for comparing the performance of our multistart clustering strategy with the conventional multistart strategy. The benchmark is again a reaction network model that consists of a linear pathway ($X_1 - X_6$) and a cyclic pathway ($X_6 - X_{12}$). Simulation results without additional noise are used as synthetic data for re-estimating the model parameters. We applied the Gauss-Newton optimization algorithm, and initially choose the interval $(\frac{p^*}{2}, 2p^*)$ around the known global optimum as admissible region S_p for each parameter.

$$\begin{aligned}
\dot{X}_1 &= -v_1 && && \text{(5a)}\\
\dot{X}_i &= v_{i-1} - v_i, && i = 2..5 && \text{(5b)}\\
\dot{X}_6 &= v_5 + v_{12} - v_6 && && \text{(5c)}\\
\dot{X}_i &= v_{i-1} - v_i, && i = 7..12 && \text{(5d)}\\
v_j &= v_{j,max} \cdot \frac{X_j}{k_j + X_j}, && j = 1..12 && \text{(5e)}
\end{aligned}$$

Table 1 documents that we used ten times fewer attempts for our clustered multistart strategy than for the conventional approach with random start values. However, most attempts of the clustered strategy converged to the global optimum, whereas the conventional multistart strategy did not at all converge from 1000 start values that were arbitrarily chosen in S_p.

Table 1: Conventional vs. clustered multistart strategy

Multistart	Conventional	Clustered
# of attempts	1000	100, $N = 10$
Best Quality of r	∞	10^{-5}
Success Rate	0%	96%
# of Simulations	2223	980 / attempt

Table 2 shows that the success rate decreases when the admitted parameter region S_p is enlarged to $(\frac{p^*}{4}, 4p^*)$, even if the number of attempts per cluster is increased from 10 to 20. More attempts improve the probability of finding successful start values at the cost of computational effort.

Table 2: Effect of start value region and number

Start Values	$S_p : p_0 \in (\frac{p^*}{4}, 4p^*)$	
# of attempts	100	
N	10	20
Success Rate	16%	44%
# of simulations / attempt	2550	4906

Table 3 illustrates how the residual r improves with each iteration in the cyclic part of the model for $N = 20$ and $S_p \subset (\frac{p^*}{4}, 4p^*)$. The residual r is significantly reduced in the very first iteration, since the average residual for the initial guesses varied around 890. In attempts 1 to 5 start values close to the global optimum were found after several iterations. After each iteration, the strategy attempts to improve the solution of the tear variable (X_{12} in this example). In attempts 2 and 6 the start values diverged, though small residuals were achieved.

Table 3: Residuals in cyclic DAE part

Attempt	1	2	3	4	5	6
Iter 1	2.59	1.76	1.91	4.83	2.85	3.64
Iter 2	1.19	1.47	0.00	0.89	0.90	2.55
Iter 3	0.79	2.66	-	0.41	0.26	1.47
Iter 4	0.33	-	-	0.00	0.00	1.19
Iter 5	0.00	-	-	-	-	0.81

5 Conclusions and Outlook

In this contribution we have presented a multistart recursive clustering strategy for efficiently estimating the parameters of DAE systems with derivative-based optimization algorithms. The algorithm has been illustrated by two simple examples with and without cyclic dependencies in the solved DAE system. A slightly more complex example that combines linear and cyclic structures has been used as benchmark for comparing the performance of our clustering strategy with the conventional multistart strategy.

At some points we have dealt with special cases, and the algorithms and examples can be generalized and extended in various directions:

- In the presented examples each parameter occurs only in one equation, and clear physical causality relation exist between the variables and the parameters. In general, however, the parameters must be included in the causality analysis.
- The benchmark was manually constructed. System analysis and clustering can be implemented using the ADModelica tool, which provides a suitable infra-structure for analyzing Modelica models and for implementing DAE decomposition algorithms.
- Real measurements are afflicted with errors, which complicates the evaluation and comparison of residual values for different optima.
- Redundant computations can be avoided by storing and recognizing local optima that were already found in earlier computations [13].
- Other optimization algorithms can be applied, for example Levenberg-Marquardt. Different clus-

ters might even be optimized with different algorithms, which might be particularly useful for large models with many variables and parameters. Hyper-heuristics [12] can help to identify the suitable algorithms for each subproblem.

References

[1] A. Antoniou, W. Lu. Practical Optimization, Algorithms and Engineering Applications, Springer Verlag, 2007.

[2] J. Nocedal, S. J. Wright. Numerical Optimization, Springer Series in Operations Research, 2000.

[3] A. Elsheikh and W. Wiechert. Automatic sensitivity analysis of DAE systems generated from equation-based modeling languages. Pages 235-246 in C. H. Bischof, et al. (editors). Advances in Automatic Differentiation. Springer, 2008.

[4] A. Elsheikh, S. Noack and W. Wiechert. Sensitivity analysis of Modelica applications via automatic differentiation. In 6th International Modelica Conference, Bielefeld, Germany, March 3-4, 2008.

[5] A. Griewank. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Frontiers in Applied Mathematics, SIAM, 2000.

[6] X. Ke. Tools for sensitivity analysis of Modelica models, Master Thesis, University of Siegen, 2009.

[7] F. E. Cellier. Continuous System Modeling. Springer Verlag, 1991.

[8] K. Murota. Systems Analysis by Graphs and Matroids, Springer Verlag, 1987.

[9] R. Tarjan. Depth-First Search and Linear Graph Algorithms, SIAM Journal on Computing 1 (1972): 146-160.

[10] H. Elmqvist and M. Otter. Methods for tearing systems of equations in object oriented modeling. In ESM'94 European Simulation Multiconference, Barcelona, Spain, June 1-3 1994.

[11] G. Kron. Diakoptics - The piecewise solution of large-scale systems. MacDonald & Co. London, 1963.

[12] E. Özcan, B. Bilgin, and E. E. Korkamz. A comprehensive analysis of hyper-heuristics, Intelligent Data Analysis 12 (2008): 3-23.

[13] C. Voglis and I.E. Lagaris. Towards ideal multistart. A stochastic approach for locating the minima of a continuous function inside a bounded domain. Applied Mathematics and Computation 213 (2009): 216-229.

Model Library of Polymer Electrolyte Membrane Fuel Cells for System Hardware and Control Design

Kevin L. Davies
Georgia Institute of Technology, Woodruff School of Mechanical Engineering
Atlanta, Georgia USA

Robert M. Moore   Guido Bender
Hawaii Natural Energy Institute
Honolulu, Hawaii USA

Abstract

The trade-offs among dynamic response, efficiency, and robustness to external factors are fundamental to the optimization of hardware and controls for fuel cell systems. No previously published model of polymer electrolyte membrane fuel cells (PEMFCs) has the capability to simultaneously provide dynamic modeling capabilities, a clear representation of physical configurations, adjustable fidelity, and flexible interfaces. This paper presents the first such library, explains key aspects of the library's architecture, and demonstrates simulations under representative scenarios. The models, implemented in the acausal Modelica modeling language, are quasi-three-dimensional (quasi-3D), discretizing the fuel cell and its layers along the directions from the anode to the cathode and down the channel length. *Keywords: fuel cell; system design; hardware; control*

1 Introduction

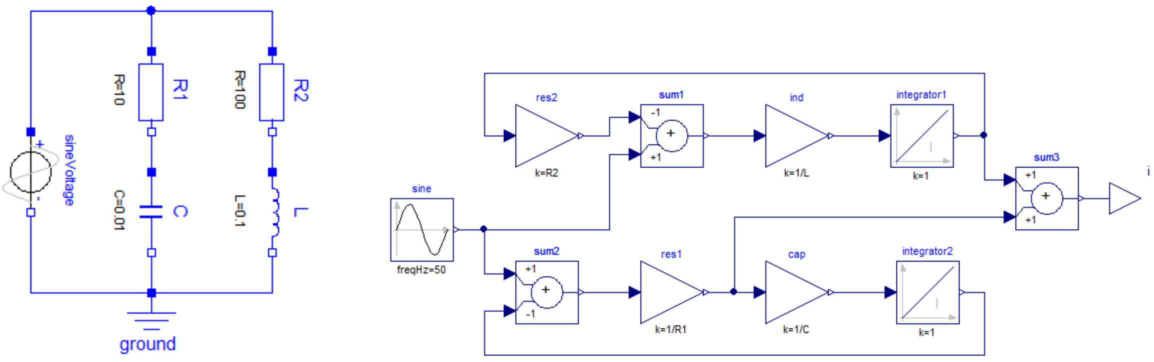
Computer-based models are most useful in the design of hardware and controls for fuel cell (FC) systems if they provide the capability for dynamic simulation, clarity, adjustable fidelity, and acausal interfaces. Control design inherently concerns the dynamic response of a system, as the design of a FC system often requires a compromise between transient performance and efficiency. Thus the model's suitability is dependent on its ability to determine the system's dynamic response from the description of system components.

Hardware design often involves the consideration of multiple configurations. Because the configurations are typically chosen by humans, a model is most useful if it is represented in a clear, intuitive form. As illustrated in Figures 1a and 1b, the form of the model depends strongly on the modeling paradigm. The acausal schematic in Figure 1a directly corresponds to the physical connections between discrete components, therefore it is more understandable. Acausal models exist for all domains in which interactions among components are dominated by energy flow, including domains of mechanical, fluid, and thermal systems.

Typical design involves a top-down process of specification, from the system level to the component level, followed by a bottom-up process of validation (i.e., the system "Vee" [6]). Models used for all stages of design require adjustable fidelity, or the capacity for multiple levels of detail. During the process of specification at the system level, many configurations must be evaluated with limited resources, so that excessively detailed models are time-consuming, unfeasible, and unjustified. At the component level, however, the accurate representation of physical processes demands more detail.

Models should also have acausal interfaces so that they can be evaluated with different simulation inputs and outputs. The underlying equations for potentials and flows in physical systems are bidirectional, with no explicit specification of the order of calculations. The current through a FC, for instance, is determined by bidirectional interactions between the FC and its electronic load (see Zenith et al. [14]). Most physical systems do not inherently have directionality and thus are best represented by acausal models. Acausal models allow inputs and outputs to be defined dur-

DOI: 10.3384/ecp09430107



(a)

(b)

Figure 1: An electrical system represented as an (a) acausal diagram and a (b) causal diagram [7].

ing the final preparation for simulation, whereas causal and noncausal models have predefined inputs and outputs. Noncausal models have inputs and outputs that are opposite relative to a given causal model and should also be clearly distinguished from acausal models.

The Modelica language [8] supports acausal models. The models are expressed using differential algebraic equations (DAEs) and discrete events. Within the DAE formalism, the dynamic behavior of components is captured in terms of implicit ODE combined with algebraic constraints. The behavior of components is explicitly discretized over space, in contrast to a PDE formalism.

2 Background

In the early 1990s, Springer et al. [11], [10] and Bernardi et al. [1] established groundwork for modeling PEMFCs and related key model parameters to FC test results. The models continued to increase in sophistication, up to the inclusion of two-phase flow in three-dimensional (3D)s. In 2004, Weber and Newman [13] reviewed the rapidly growing body of literature on the transport mechanism within PEMFCs. The models range from the detailed description of specific processes to the coverage of the entire cell and take both first-principle and empirical approaches. However, as noted by Zenith et al. [14], many FC models consider the electronic current as an input. In fact, only four of the more than 250 published models [13] are known to be acausal. The model of Rubio et al. [9] provides an acausal and detailed representation of cell polarization. However, it lacks robustness, clear correlation to previous literature and the physical configuration of the FC, fidelity with respect to the concentration gradient down the flow channel (it is one-dimensional (1D)) and consideration of the reactant streams as a mixed fluid, and only has electrical interfaces (no fluid or thermal interfaces). The model of Ungethüm [12] describes the entire FC system and has an acausal fluid network, but no details of the FC stack model have been published. The model described in this paper is a continuation of the work of Davies et al. [3, 4].

The approach and goal of the research described in this paper is to utilize state-of-the-art acausal modeling tools and methods to develop a robust model of a PEMFC that provides the capability for dynamic modeling, clarity, adjustable fidelity, and flexible interfaces.

3 Model Components

3.1 FC Components

The equations of the models in this section and subsequent sections are described in schematic form. At each physical interface node in the schematic (whether it is fluid, or thermal, or electrical), the flow variables sum to zero. This represents the balance equations for energy and chemical species. In the electrical case, for instance, this corresponds to Kirchhoff's Current Law. In addition, the across variables (e.g., electric potential, pressure, temperature) are set equal at every node. This implies that potential energy differences sum to zero around loops, which is analogous to Kirchhoff's Voltage Law.

3.1.1 Flow Plate

Figure 2 provides a schematic for the `FlowPlate`, which consists of a network of `PortVolume`, `PressureLoss`, `DiffusionSurface`, `ThermalConductor`, `ThermalConvection`, `HeatCapacitor`, and `HeatingResistor` elemental models. The large interface icons along the boundary of the Figure represent the outside connections to the `FlowPlate` and the smaller icons in the interior represent the interfaces of the elemental models within the `FlowPlate`. The equations of the fluid network describe the pressure loss down the length of the channel, the storage of fluid in the channel volume, and the interface to the gas diffusion layer (GDL). The equations of the thermal network describe the heat conduction across the flow plate, the resistance heating and thermal energy storage in it, and the heat convection between the fluid and the flow plate. The equations of the electrical network describe the voltage loss across the flow plate, which may or may not be negligible, depending on the values of the parameters that represent the properties and thickness of the material.

The model captures pressure loss due to bulk flow along the reactant/product stream and down the channel (the f direction). The contents of the dotted area are repeated for each of N_f segments ($N_f = 1$ for a 1D `FC` and $N_f > 1$ for quasi-3D). Where the connection "wire" loops back on itself in Figure 2, the elements included in the loop are connected in series. The model assumes perfect electrical and thermal conduction in the f direction, which is appropriate because the electronic flow and heat flow are predominately in the z direction.

3.1.2 Catalyst Layers

Figure 3 is a schematic for the model of a cathode catalyst layer, `CatLayer_ca`, which contains of a network of `ORR`, `MembraneSurface`, `Capacitor`, `ThermalConductor`, `HeatingResistor`, `HeatCapacitor`, `PoreVolume`, `TransportPorous`, `DiffusionMembrane`, `ElectroOsmoticDrag`, and `MembraneVolume` elemental models. The physical meaning of the networks is similar to that of the `FlowPlate`. The contents of the outer dotted rectangle are duplicated N_f times, and the contents of the inner dotted rectangles are duplicated $N_z + 1$ and N_z times. In this manner, the oxygen reduction reaction (ORR) is distributed over $N_f\,(N_z + 1)$ instantiations of the `ORR` . The model assumes perfect fluid, thermal, and electrical insulation in the f direction, which is appropriate because the transport within the flow plate dominates that of the other layers in the f direction. The model of an anode catalyst layer, `CatLayer_an`, is identical except that it contains the `HOR` instead of the `ORR` and the media model represents hydrogen (H_2) and carbon monoxide (CO) instead of oxygen (O_2) and nitrogen (N_2).

3.1.3 GDL and PEM

All of the elements of the `GDL` and `PEM model` are also contained within the `CatLayer_ca`. Like the `CatLayer_ca`, both of these models contain N_z lumped segments through the thickness of the cell and N_f segments in the direction along the length of the flow channel. The network of elemental models in the upper right of Figure 3 (`ThermalConductor`, `HeatingResistor`, `HeatCapacitor`, `PoreVolume`, and `TransportPorous`) represents the `GDL` . The thickness and porosity of each of the N_z lumped segments through the GDL can be independently specified, making it possible to model a GDL with a microporous layer (MPL). Likewise, the network of elemental models in the lower right of Figure 3 (`DiffusionMembrane`, `ElectroOsmoticDrag`, and `MembraneVolume`), as well as `ThermalConductor` and `HeatCapacitor`, is representative of the `PEM` . The transport processes due to diffusion and electro-osmotic drag act in parallel. Electro-osmotic drag transports H_2O from the anode to the cathode, in the direction of protonic flow. Both the electro-osmotic drag process and generation of H_2O from the ORR generally lead to a higher molar concentration of H_2O on the cathode side, so there is a net diffusion of H_2O from the cathode to the anode.

3.2 FC

Figure 4 represents the `FC` . It is a set of `FlowPlate`, `GDL`, `CatLayer_an`, `CatLayer_ca`, and `PEM` s connected through fluid, heat, and electrical interfaces. Each of these interconnections contains N_f interfaces in parallel (one for each channel segment). There are alternative methods to describe numerous processes within the FC. These options allow the model's fidelity to be adjusted to suit

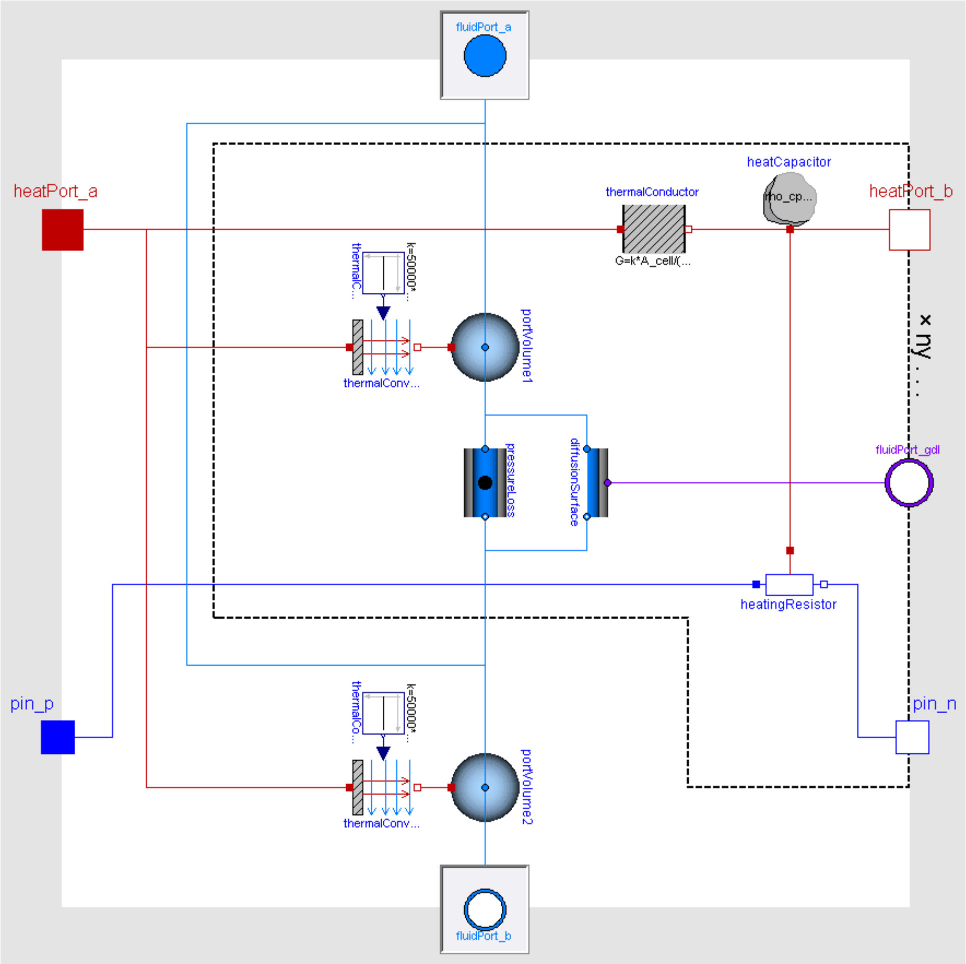


Figure 2: Schematic of the FlowPlate model.

the available simulation time and computational resources. Table 1 lists all of the options and notes the baseline options with asterisks.

3.3 Test Scenarios

Figure 5 shows the baseline test scenario used to evaluate the polarization curve of the model of a FC. The test scenario is essentially a model of an ideal FC test stand. The anode and cathode reactant streams are generated by the FluidSource_an and FluidSource_cas and are dissipated into the Boundary_an and Boundary_ca components, respectively. The H_2O content of the reactant streams is specified in terms of relative humidity (RH) or, alternatively, mole fraction of H_2O. The molar concentration of O_2 in the dry cathode gas can be adjusted, allowing the FC to be simulated with air or O_2. In the baseline scenario, the reactant flow rates are specified as constants in terms of equivalent current (i.e., the current corresponding to 100 % utilization). Alternatively, the flow can be variable, specified in terms of an equivalence ratio and a current measurement. Concurrent or counter flow scenarios can be modeled by switching the cathode flow direction with the Reverser. The pressures at the anode and cathode outlets, as well as the temperatures at the exterior surface of the flow plates, are specified by constant signal sources. The FC is wired to an electrical circuit where the current is specified by a signal source and the electric potential is measured as the dependent variable. The signal source is a ramp function, and the ramp rate is small ($1\,\mathrm{mA\,cm^{-2}\,s^{-1}}$) so that transient effects are negligible.

The test scenario is the highest level of the model. It provides the simulation stimuli as an input to the model under test (i.e., the FC). Therefore, the test scenario contains causal assignments, as shown by the arrows in Figure 4. In the baseline scenario, the signal sources are constants and ramp functions but could instead be data files,

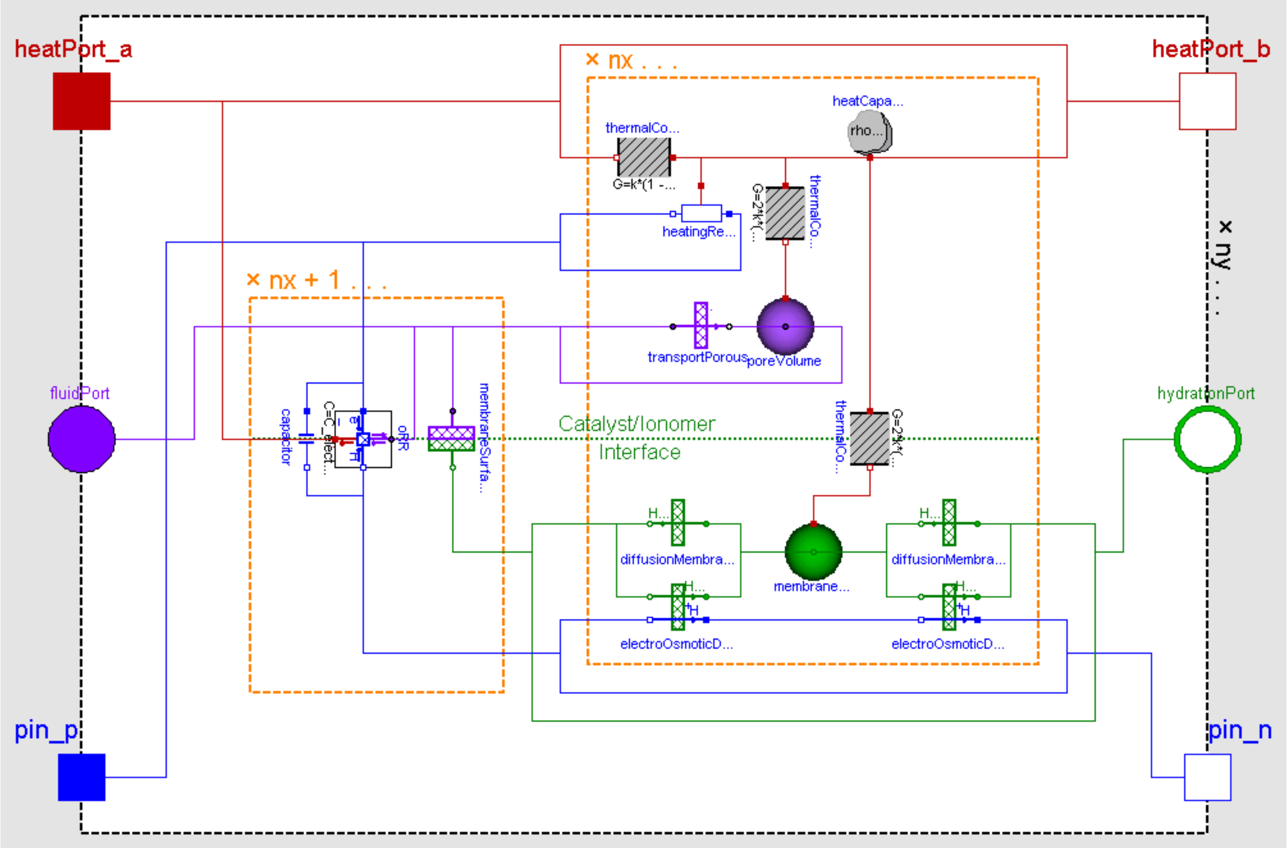


Figure 3: Schematic of the CaLayer_ca model.

inputs from a control system, or real-time signals from hardware-in-the-loop (HiL) equipment. Note that the acausal model formulation extends to the physical boundary of the system of interest, and causal stimuli are applied at the boundary. If the boundary was expanded to encompass a larger system, the causal stimuli would be applied at the new boundary (e.g., a vehicle drive profile applied to a FC vehicle model).

4 Simulation Results

The FC is evaluated by varying conditions and settings relative to a baseline scenario. Except where noted, the baseline conditions are used. Under the baseline scenario, the flow plates are held at 80 °C, and the anode|cathode reactant conditions are: H_2|Air, 1.0|1.0 atm, 80|80 °C, 100|100 %RH, and 2.0|3.0 A cm^{-2} equivalent current. The cell is 1D ($N_f = 1$) and the baseline model options are noted by the asterisks in the third column of Table 1. Figure 6 shows the polarization of individual channel segments of a quasi-3D cell ($N_f = 10$) under anode|cathode reactant flows at constant flow rates of 1.4|1.4 A cm^{-2} equivalent current and at fixed equivalence ratios of 1.25|1.25, with minimum equivalent current density of 0.18|0.18 A cm^{-2}. Here, the fluid is assumed to be non-condensing (option 1.1a in Table 1), and the flow rates are low in order to exaggerate the difference in polarization between upstream and downstream segments. In both scenarios, as current density is increased, the upstream segments pass an increasingly disproportionate share of the current. This occurs due to a positive feedback mechanism within the cell, whereby increased current from the upstream segments leads to lower O_2 molar concentration in the downstream segments, decreasing their performance and requiring increased current from the upstream segments in order to meet the current demand on the entire cell. As shown in Figure 6a, this mechanism can even cause the polarization of the downstream segments to fold back when the flow rate is fixed.

Figure 7a shows the effect of several model options from Table 1. The polarization curves shown in the plot are the outliers among the set of all the polarization curves produced by varying only one option from the baseline.

Table 1: Selectable model options

Model Element	Characteristic	Option
Media	H_2O behavior	1.1a: Non-condensing
		1.1b: *Condensing
	Pressure-diffusivity products	1.2a: *Constant
		1.2b: Dependent on temperature
PressureLoss	Flow regime	2.1a: *Laminar (valid for $Re < 1000$)
		2.1b: Quadratic turbulent (valid for $Re > 4000$)
		2.1c: Laminar and quadratic turbulent (valid for $Re < 1000$ and $Re > 4000$)
		2.1d: Detailed (valid over full Re range)
	Fluid density and dynamic viscosity	2.2a: *Nominal
		2.2b: Calculated from the media model
TransportPorous	Liquid H_2O transport	3.1a: *Not included
		3.1b: Included
HOR	Voltage	4.1a: *Constant
		4.1b: Thermodynamic (modified Butler-Volmer equation)
ORR	Product H_2O	5.1a: Liquid
		5.1b: Vapor
		5.1c: *Two-phase
PEM	Protonic resistance	6.1a: Constant
		6.1b: Dependent on current density
		6.1c: *Dependent on hydration
	Diffusion coefficient	6.2a: *Constant
		6.2b: Dependent on hydration, specified as a polynomial
		6.2c: Dependent on hydration, specified as a lookup table

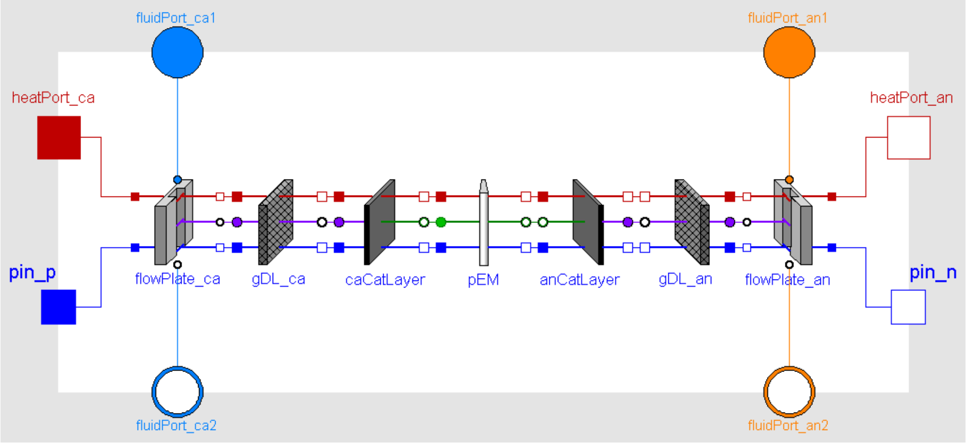


Figure 4: Schematic of the FC model.

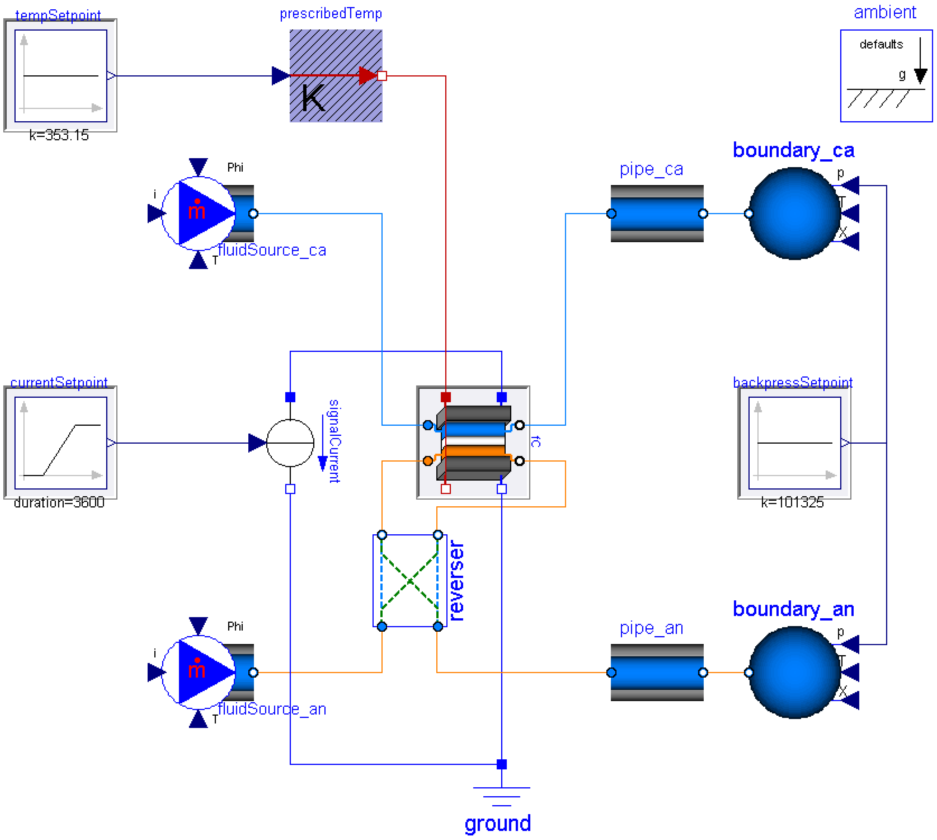


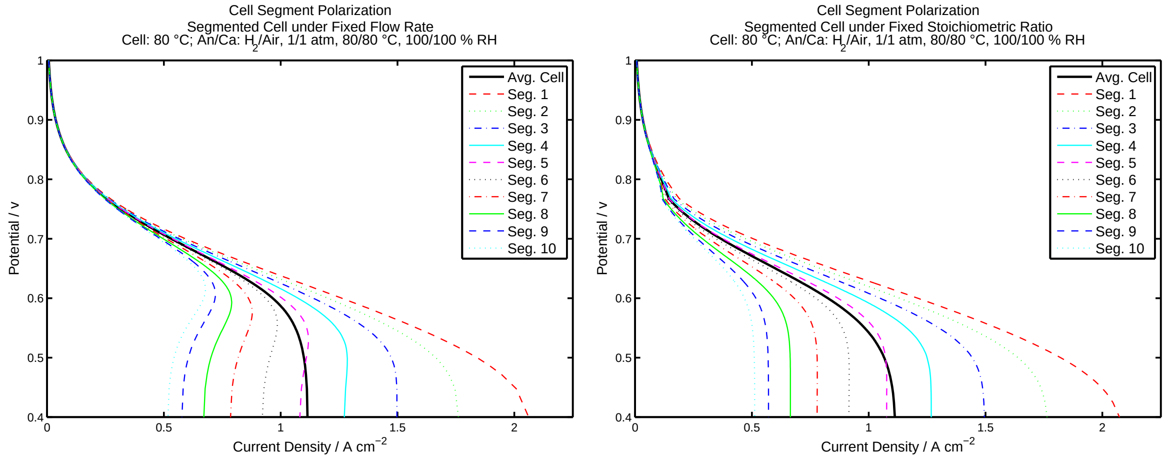
Figure 5: Schematic of the baseline test scenario.

The number of channel segments (N_f) is another important model setting. As N_f is increased, the model becomes more representative, but the simulation takes more time. Figure 7a shows the effect of N_f on the polarization curve. As N_f is increased, there is an apparent change in performance because the O_2 molar concentration in each segment is calculated as the average of that at the inlet and outlet of the segment. The profile of H_2O and O_2 concentration is nonlinear down the channel length due to unequal current through the segments (as demonstrated in Figure 6). These concentrations affect the polarization via the overpotential in the cathode catalyst layer and the resistance loss in the polymer electrolyte membrane (PEM), and the effect is more pronounced where there is a large concentration gradient down the flow channel (e.g., at high currents). However, there is a limit beyond which there is negligible effect from increasing N_f, as shown in Figure 7a.

Figure 8 demonstrates the dynamics of the FC . The electronic load is cyclical, with a sinusoidal current from 0 to 1.5 A cm^{-2} with a 30 s period. Figure 8a shows the voltage losses of the cell. The cathode activation and concentration losses are dominant, and a small loop appears in the polarization curve due to the hysteresis in temperature at the cathode reaction site and the hydration of the PEM. Figure 8b shows that the temperature vs. current hysteresis occurs counterclockwise. As current increases, the internal losses increase, increasing the temperature at the cathode catalyst layer. There is a lag in the temperature response, however, due to the heat capacities of the materials within the cell. The PEM hydration vs. current hysteresis occurs clockwise. As current increases, more H_2O flows from the anode to the cathode due to electro-osmotic drag, resulting in a net drying effect of the PEM. There is a lag in the PEM hydration response since H_2O is stored in the PEM material.

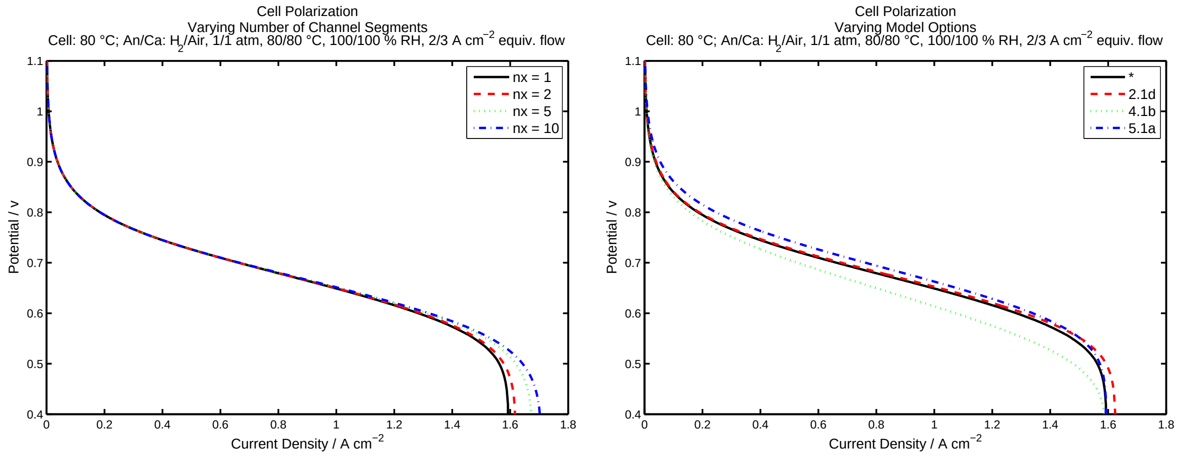
5 Discussion

The models simulate quickly enough to be manageable for design studies of the FC or combined with other models for studies of larger systems. However, any real-time application of the model,



(a) (b)

Figure 6: Cell segment polarization with anode/cathode reactants provided at (a) fixed flow rate of $2.0|2.0\,\mathrm{A\,cm^{-2}}$ and (b) fixed equivalence ratio of $1.25|1.25$.

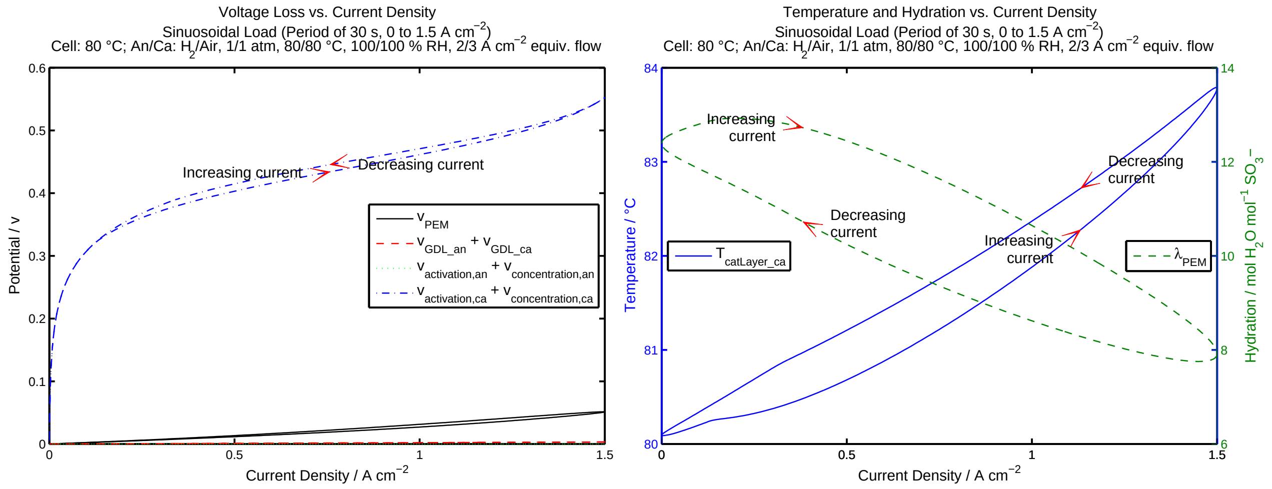


(a) (b)

Figure 7: Effect of model options on the FC polarization curve: (a) selected model options from Table 1 and (b) number of channel segments.

e.g., for HiL or model based control (MBC), would likely require simplification or other adjustments. The baseline simulation has 31 time-varying state variables and 1595 nontrivial equations. Each additional channel segment in the quasi-3D case adds 23 states and 985 equations. As translated by the Dynamic Modeling Laboratory (Dymola) [5] and simulated with the Differential/Algebraic System Solver Library (DASSL), the baseline simulation requires 1.6 s to run on an Intel Centrino Duo T7300 2.0 GHz based computer.

The model translator performs symbolic manipulation on the model so that the simulation is manageable. Several guidelines have been used in the design of the model library so that this simplification is possible. First, storage elements (e.g., capacitors or media storage volumes) are placed at nodes where processes are connected to reduce the size of nonlinear systems of implicit DAEs. This introduces state variables, and the model translator is then able to break the equations into smaller sets which can be calculated independently over each simulation time step. Second, media storage volumes are generally not connected directly, but rather through processes, to avoid index reduction steps in the model translation [2]. Third, the models provide provisions to manage model stiffness (where the time constants of the model vary widely) via the model options listed in Table 1. Fourth, the model equations are carefully



(a) (b)

Figure 8: FC response with a sinusoidal load: (a) losses due to resistance, activation, and concentration in the PEM, GDL, and catalyst layers and (b) cathode catalyst layer temperature and PEM hydration.

formulated to reduce the sensitivity to numerical precision, as in the `TransportPorous`, for example.

The model avoids or carefully handles discrete events where the structure of the model equations changes abruptly, as in the two-phase behavior of H_2O. This gives the model translator more flexibility in the symbolic manipulation of the model equations, and it avoids the need for the simulator to reinitialize when the events occur. When a discrete event occurs, the simulation solver must reinitialize, and by default it begins a guess process with the values of the state variables before the event occurred. However, the discrete events that occur when switching between one-phase and two-phase regions still slow down the simulation. When the quasi-3D model of a FC is used with $N_f = 10$, events occur frequently enough to increase the simulation time significantly. For that reason, H_2O condensation is not included unless absolutely necessary.

6 Conclusion

As described in the introduction, computer-based models are most useful for the hardware and control design of FCs if they provide the capability for dynamic simulation, clarity, adjustable fidelity, and acausal interfaces.

The model library presented here addresses these requirements by modeling the FC as a system of acausal, object-oriented models specified in terms of DAEs in the Modelica language. This paper provides a high-level view of how the models are structured and presents results from several usage scenarios. The research demonstrates the power of combining the empirical and theoretical knowledge in the field of FCs with a state-of-the-art tool for acausal modeling.

In the hardware design of a FC system, there is often a trade-off between attributes of the FC (e.g., efficiency or performance metric) and the balance of plant (e.g., parasitic loss or economic cost). The results demonstrate the ability of the model to evaluate the efficiency of the FC, even with variable causality.

7 Acknowledgments

This research was supported in its initial stages by the Office of Naval Research under grant #N00014-04-0682 to the Hawaii Natural Energy Institute and later by the Robert G. Shackelford Fellowship of the Georgia Tech Research Institute. The authors wish to acknowledge the support of the Modelica Association and the Modelica community by the open sharing of the Modelica Standard Library and `Modelica_Fluid`. Although it is not currently possible to offer the library described within this paper online as open source, the authors request that any interested readers contact them.

References

[1] D. M. Bernardi and M. W. Verbrugge. A mathematical model of the solid-polymer-electrolyte fuel cell. *Journal of The Electrochemical Society*, 139(9):2477–91, 1992.

[2] F. E. Cellier and E. Kofman. *Continuous System Simulation.* Springer, New York, NY, 2006.

[3] K. Davies and R. Moore. Object-oriented fuel cell model library. *ECS Transactions*, 11(1):797, 2007.

[4] K. Davies and R. Moore. PEMFCSim: A fuel cell model library in Modelica. In *Fuel Cell Seminar, Austin, TX*, 2007.

[5] Dynasim AB. Dymola: Dynamic modeling laboratory, 2007. v6.2.

[6] K. Forsberg and H. Mooz. System engineering for faster, cheaper, better. *The Center for Systems Management*, Reprinted by SF Bay Area Chapter of INCOSE, http://www.incose.org/sfbac/, 1998.

[7] Modelica Association. *Modelica: A Unified Object-Oriented Language for Physical Systems Modeling: Tutorial.* Linköping, Sweden, ver. 1.4 edition, December 2000.

[8] Modelica Association. Modelica: A unified object-oriented language for physical systems modeling, February 2 2005.

[9] M. A. Rubio, A. Urquia, L. GonzÃ¡lez, D. Guinea, and S. Dormido. FuelCellLib: A modelica library for modeling of fuel cells. In *4th International Modelica Conference*, Hamburg-Harburg, Germany, March 2005. Modelica Association.

[10] T. E. Springer, M. S. Wilson, and S. Gottesfeld. Modeling and experimental diagnostics in polymer electrolyte fuel cells. *Journal of The Electrochemical Society*, 140(12):3513–3526, 1993.

[11] T. E. Springer, T. A. Zawodzinski, and S. Gottesfeld. Polymer electrolyte fuel cell model. *Journal of The Electrochemical Society*, 138(8):2334–2342, 1991.

[12] J. Ungethüm. Fuel cell system modeling for real-time simulation. In *4th International Modelica Conference*, Hamburg-Harburg, Germany, March 2005. Modelica Association.

[13] A. Z. Weber and J. Newman. Transport in polymer-electrolyte membranes III: Model validation in a simple fuel-cell model. *Journal of The Electrochemical Society*, 151(2):326–339, 2004.

[14] F. Zenith, F. Seland, O. E. Kongstein, B. Borresen, R. Tunold, and S. Skogestad. Control-oriented modelling and experimental study of the transient response of a high-temperature polymer fuel cell. *Journal of Power Sources*, 162(1):215–227, 2006.

Modeling Reaction and Diffusion Processes of Fuel Cells within Modelica

Kevin L. Davies Comas L. Haynes Christiaan J.J. Paredis
Georgia Institute of Technology, Woodruff School of Mechanical Engineering
Atlanta, Georgia USA

Abstract

The field of fuel cell (FC) technology offers a challenging and rewarding application for the Modelica language because it is highly multi-disciplinary and it entails physical phenomena (e.g., catalysis) that are not fully understood. Modelica is a valuable platform from which to explore FCs because it is appropriate for the representation of physical interactions. This paper describes elements of a FC library which has been developed in Modelica. The goal of the modeling effort is to take full advantage of the physically representative nature of the Modelica language. To this end, it is important for the models to be consistent and explicit in terms of energy and species balances. The paper emphasizes the representation of diffusion and electrochemical processes. In these areas, the traditional approach is to represent empirically observed behavior, and this is not necessarily rigorous from the standpoint of energy and species balances. To describe the diffusion and electrochemical processes in a form that is suitable for Modelica, alternative and possibly more physically fundamental model equations have been developed.
Keywords: media; streams; diffusion; fuel cell

1 Introduction

Traditionally, FC models have been developed with the goal of capturing behavior that has been observed empirically [13]. However, a model that is empirically representative under a certain range of conditions may not be representative under others. Moreover, empirically-based models are not necessarily consistent in terms of energy and species balances.

The goal of the research described in this paper is to create a robust, flexible, and dynamic model of FCs in the Modelica language. In order to take full advantage of the Modelica language, it is important for the model to be consistent and explicit in terms of energy and species balances. This presents both a challenge and an opportunity. Problems may occur if empirically derived equations have inconsistencies and are implemented in Modelica. When these problems are solved, however, a deeper understanding of the fundamental phenomenon may be uncovered.

This paper describes elements of a `FC` library in Modelica. It emphasizes two areas that have not been fully explored in Modelica: the transport of chemical species by diffusion and electrochemical reactions. Most FC models in the literature describe diffusion via the dusty-gas model or derivatives of it, which are empirically based and may lead to singularities [14]. Also, most FC models describe the anode and cathode jointly, with only the net reaction and a single electric potential. However, this approach is not suitable for the FC library because the anode and cathode catalyst layers are separate and are connected to opposite sides of the proton exchange membrane (PEM).

2 Model Equations

2.1 Media Library

The `Media` library defines the relationship between the properties of chemical mixtures. The medias within the library are based on the ideal gas assumption. This assumption is appropriate because FCs typically operate at low pressures and high temperatures with respect to the critical pressures and temperatures of the gases that are present, with the exception of H_2O. Depending on a model option, H_2O is either described as an equilibrium mixture of an incompressible liquid and an ideal gas or as a non-condensing ideal gas. A media is specified within every interface and

DOI: 10.3384/ecp09430106

higher-level model of the FC library. In the anode, the media represents a mixture of H_2O, hydrogen (H_2), and carbon monoxide (CO). In the cathode, it represents H_2O, oxygen (O_2), and nitrogen (N_2). At each instantiation, a minimal set of media properties is defined by the conditions within the interface or element. This set, the state, is sufficient to uniquely determine all other intensive properties of the fluid. As Gibb's phase rule [7] states, $N_F = 2 + N_S - N_P$, where N_F is the number of degrees of freedom, N_S is the number of species, and N_P is the number of phases present. For example, the cathode fluid contains H_2O, O_2, and N_2, which means that $N_F = 3$. Therefore, $N_D = 3$ if liquid and gas are both present, but $N_F = 4$ if the fluid is entirely gaseous.

The varying degrees of freedom require careful attention. The Modelica language currently does not support models with variable structure [4]. In addition, the media and the element with which it interacts are separate classes within the object-oriented model structure, and the interface between the two should be the same regardless of the number of phases that are present in the fluid. Since Gibb's phase rule only applies to the intensive properties of the fluid, $N_P - 1$ additional equations must be given to specify both the intensive and extensive properties of the fluid. For example, even though either pressure or temperature is sufficient to specify the intensive properties of saturated H_2O (i.e., on or within the two-phase boundary), an additional property such as water quality (the fraction of H_2O that is vaporized) must be given in order to specify the extensive volume of the H_2O.

The following equations allow the simulation of both one-phase and two-phase regions. Within the mixture, the molar concentration of H_2O is the sum of the molar concentrations of liquid H_2O and H_2O vapor: $C_{H_2O} = C_{H_2O(l)} + C_{H_2O(g)}$. Water is assumed to be the only condensing chemical species within the mixture, so the fraction of H_2O within the gas phase is $C_{g(H_2O)} = C_{H_2O(g)}/(1 - C_{H_2O(l)})$. Water quality is expressed as $x = C_{H_2O(g)}/C_{H_2O}$. In the liquid/gas region, the H_2O is saturated, that is, $C_{H_2O(g)}\,p = p_{l|g}(T)$, where $p_{l|g}(T)$ is the saturation pressure as a function of temperature, defined in the Modelica Standard Library (Modelica library) [6] as $p_{l|g}(T) = 611.657\exp(17.2799 - 4102.99/(T - 35.719))$. From the preceding equations, $C_{H_2O(g)} = 1 - (p\,C_{H_2O(g)})/p_{l|g}(T)$ in the liquid/gas region.

If $C_{H_2O}\,p > p_{l|g}(T)$, then liquid is present. If $p > p_{l|g}(T)$ and $C_{H_2O} = 1$ (i.e., the media is entirely H_2O, without any non-condensing gases present), then the mixture is entirely liquid. The behavior of the phases of the fluid can be summarized as:

$$C_{H_2O(l)} = \begin{cases} C_{H_2O}, \\ \quad \text{if } C_{H_2O}\,p > p_{l|g}(T) \text{ and } C_{H_2O} = 1 \\ \\ 1 - p\,C_{H_2O(g)}/p_{l|g}(T), \\ \quad \text{if } C_{H_2O}\,p > p_{l|g}(T) \text{ and } C_{H_2O} < 1 \\ \\ 0, \\ \quad \text{if } C_{H_2O}\,p \le p_{l|g}(T) \end{cases} \tag{1}$$

The state variables of the media are typically chosen to be the amounts of the chemical species present within a control volume (e.g., expressed in total moles and the molar concentrations of all but one chemical species in the mixture) and temperature. These variables are sufficient to specify all the intensive and extensive properties of the fluid, regardless of whether the fluid is in the one-phase or two-phase region. The continuity of these variables across the discrete events generated by the Boolean conditions in Eq. 1 helps the simulation solver run more efficiently.

The enthalpy and entropy of the chemical species in their pure forms at standard pressure are calculated based on the models in the Modelica library, which implement the equations and empirical data presented by McBride et al. [3]. Dynamic viscosity is also calculated based on the models in the Modelica library. Several other properties including relative humidity (RH), molar mass, Gibbs free energy of chemical species in their pure form at reference pressure, and molar volume or specific volume of the phases and entire mixture are determined from thermodynamics [7]. The properties of the entire mixture are determined by a weighted average of the intensive properties of each chemical species [7].

2.2 Interfaces

Acausal interfaces describe fluid, thermal, and electronic interactions within the FC. Each interface has an equal number of flow (through) and property or potential (across) variables, but no equations.

FluidPort The FluidPort interface describes the flow of chemical mixtures. The flow variables represent chemical species and enthalpy. There are two versions of FluidPort interface. The first version is that of the Modelica_Fluid library [5], which uses a mass basis in order to be compatible with other components such as pumps and valves. The second version, which has been developed for the FC library, uses a mole basis so that processes such as diffusion and chemical reactions may be described directly. The property variables are pressure, specific enthalpy, and the molar concentrations of all but one of the chemical species. Other intensive properties of the mixture passing through the interface are specified by the Media library described above in Section 2.1, and the extensive properties are specified by the model of the control volume, described below in Section 2.3.1.

HydrationPort The HydrationPort interface is similar to the mole based version of the FluidPort interface, but is specific to the flow of H_2O as an absorbed medium through a bulk material (e.g., the PEM). The HydrationPort interface has been developed for the FC library. The flow variables are the molar flow rate of H_2O and the enthalpy flow rate. The property variables are temperature and hydration. In the case of the PEM, hydration is expressed as the ratio of moles of H_2O to moles of sulfonic acid (SO_3^-) groups.

HeatPort The HeatPort interface describes heat flow; the potential variable is temperature. The HeatPort interface and the following Pin interface are instantiated directly from the Modelica library.

Pin The Pin interface describes electronic flow or protonic flow, and the potential variable is electric potential.

2.3 Elemental Models

All of the elemental models include the applicable terms of the energy balance and species balance equations. In the following equations, i is the index of the fluid interfaces, j is the index of the thermal interfaces, and k is the index of the chemical species.

$$\frac{\mathrm{d}u}{\mathrm{d}t} = i\,\Delta\phi + \sum_i \dot{h}_{\rightarrow i} + \sum_j \dot{u}_{\rightarrow j} \tag{2}$$

$$\frac{\mathrm{d}n_k}{\mathrm{d}t} = \dot{n}_{k,react} + \sum_i \dot{n}_{i,k} \tag{3}$$

The storage models (Section 2.3.1) are dynamic, and include the storage term on the left hand side of Eqs. 2 and 3. The process models (Sections 2.3.2 and 2.3.3) are static and do not include the storage term. The enthalpy flow rate through interface i is specified as a sum of the enthalpy of the chemical species:

$$\dot{h}_k = \sum_k \bar{h}_{i,k}\, \dot{n}_{i,k} \tag{4}$$

2.3.1 Storage Models

The storage models integrate the energy and species, flowing into or out of a control volume over time in order to determine the present state, according to Eqs. 2 and 3. Each storage model has at least one time-varying state variable, which describes the condition of the element at a given time.

PoreVolume The PoreVolume model describes the storage of internal energy and chemical species. The properties are determined by the selected media. The time-varying state variables depend on the usage scenario, but are typically temperature, total moles, and the volumetric concentrations of all but one of the chemical species.

MembraneVolume The MembraneVolume model describes the storage of internal energy and H_2O in the PEM. The states are temperature and total moles of H_2O. The hydration of the PEM (λ) is related to n_{H_2O}, the amount of H_2O in the control volume, by the following equation.

$$\lambda = n_{H_2O}\,\mathbf{M}_{dryPEM}/(\rho_{dryPEM}\,A\,\mathrm{d}L_{\cdot z}) \tag{5}$$

PortVolume The PortVolume model is instantiated directly from the Modelica_Fluid library. It is similar to a PoreVolume model, but it accounts for the storage of fluid on a mass basis rather than a mole basis.

Capacitor and HeatCapacitor The Capacitor and HeatCapacitor models are instantiated directly from Modelica library to describe electrical and thermal storage, respectively.

2.3.2 Reaction Models

The reactions of the anode and cathode are modeled separately. The rates of energy (in electrical, chemical, and thermal forms) are balanced by Eq. 2. The flow rates of the species (in electronic, protonic, and chemical forms) are balanced by the stoichiometric ratio of the reactions and Eq. 3. The reaction models take direct advantage of the media properties from McBride et al. [3], which are available in the `Modelica` library.

The reaction models use a new, and possibly more fundamental, form of an electrochemical rate equation which can be contrasted with the widely-used Butler-Volmer equation. The new form is based on the hypothesis that the need for a symmetry factor in the Butler-Volmer equation is actually an artifact of the assumption of equal and opposite biases in the forward and backward directions. The traditional approach is to calculate the Nernst potential and add an "activation overpotential". The Nernst potential is dependent on temperature but not pressure or current, because it assumes standard pressure ($p = p^0$) and an open-circuit condition ($i = 0$). The overpotential accounts for the actual pressure and current. It is typically negative because it decreases with increasing current, decreasing partial pressure of the reactants, and increasing partial pressure of the products. The new electrochemical rate equation is simpler to implement in the Modelica language than a Nernst potential/overpotential approach because it does not separate factors that are actually coupled in reality.

There are two traditional methods of calculating the activation overpotential. The first method, which is seemingly the simplest, is the Tafel equation. It combines the forward and backward currents of the reaction by assuming that the forward current dominates under typical operating conditions. This results in a singularity (a natural logarithm of zero at zero net current), which is unfortunate because the open-circuit condition ($i(t = 0) = 0$) is a typical and convenient initial condition (IC). The Tafel equation is not acceptable for an acausal reaction model, which certainly must be able to simulate at open-circuit conditions. In reality, a FC and its electronic load are separate components, and the FC is not a singularity before the load is connected to it, when $i = 0$.

The Butler-Volmer equation for the activation overpotential handles the open-circuit condition by accounting for the fact that while the net current is zero under the open-circuit condition, there are equal and opposite, but nonzero, diffusion currents in the forward and backward directions. However, the Butler-Volmer equation makes the default assumption that the forward and backward currents are driven by opposite signs of the same bias. That bias is a combination of the Gibbs free energy of the reactants and products, as well as electronic and protonic energies. An "electrode symmetry factor" is then included to account for the fact that the forward and backward bias may not actually be symmetric. The symmetry factor is typically found to be approximately 0.5 for the net reaction. However, in the present FC model, the anode and cathode reactions must be modeled separately since the anode and cathode catalyst layers are separate models. During the development of the model, it was found that the anode symmetry factor of 0.5 produces an unrealistic anode electric potential, which results in an incorrect cell electric potential when the anode and cathode are electrically connected in series to represent the FC.

Therefore, the Butler-Volmer equation is modified under the premise that the power of one of the exponential terms in the Butler-Volmer equation should be associated only with the energies of the reactants needed for the forward direction of the reaction. Accordingly, the power of the second exponential term should be associated with the energies of the reactants for the backward direction of the reaction, i.e., the products of the forward direction of the reaction. It is also realized that the electric potential of the reaction is related to the sum of an electronic energy and a protonic energy. The electrons (e^-) and protons (H^+) are either reactants or products, depending on the reference direction of the reaction.

The exponential factors of the electrochemical rate equation are of the form $\exp(g/\mathbf{R}\,T)$, but are typically written in the form $(p/p^0)\exp(g^0/\mathbf{R}\,T)$, which is equivalent under the assumption of an ideal gas. This equivalence can be shown by realizing that Gibbs free energy can be defined as $g = h - T\,s$. Enthalpy is defined via $\mathrm{d}h = T\,\mathrm{d}s + \mathrm{v}\,\mathrm{d}p$, which can be reduced under the assumption of an ideal gas and integrated under isothermal conditions to give $s - s^0 = -\mathbf{R}\ln(p/p^0)$, where s^0 is the entropy at standard pressure (p^0).

Gibbs free energy can therefore be represented as a value at standard pressure ($g^0 = h - T\,s^0$) and an offset, such that $g = g^0 - T\,(s - s^0)$, or $g = g^0 + \mathbf{R}\,T\,\ln(p/p^0)$. Therefore, $\exp(g/\mathbf{R}\,T) = (p/p^0)\,\exp(g^0/\mathbf{R}\,T)$ if the media behaves as an ideal gas.

When the reaction models are formulated under these premises, it is no longer necessary to calculate a Nernst potential at all. The new electrochemical rate equation results in the electrode electric potential directly. Also, the need for two model parameters (the symmetry factors of the anode and cathode) is eliminated. This offers the advantage that fewer parameters must be specified to utilize the FC model. The new electrochemical rate equations are more robust than the Tafel equation because the natural log of zero cannot occur unless the media properties are miscalculated.

`ORR` An `ORR` model describes the oxygen reduction reaction (ORR) whereby oxygen is consumed and H_2O is produced: $\frac{1}{2}\,O_2 + 2H^+ + 2e^- \rightarrow H_2O$. Bockris et al. give an equation [2, Eq. 7.7] for the forward current density of an electrochemical reaction involving Ag^+:

$$i''\!\rightarrow = \mathbf{F}\,\frac{\mathbf{k}\,T}{\mathbf{h}}\left[n''_{Ag^+}\right]\exp\left(-\frac{\Delta\bar{g}^0}{\mathbf{R}\,T}\right)\exp\left(-\frac{\beta\,\Delta\phi\,\mathbf{F}}{\mathbf{R}\,T}\right) \tag{6}$$

The equation is adapted here for the ORR and modified to a more fundamental form. It is noted that the concentration factor should strictly include all of the reactants—not only O_2 and H^+, but also e^-. The concentration factor is the product of the molar concentrations of the reactants raised to the power of the corresponding stoichiometric ratios and divided by the molar volume of the mixture. In Eq. 6, n''_{Ag+} is the surface concentration of Ag+. Here, it is desirable to relate the equation to the molar concentration of the chemical species and volumetric concentration of the mixture. A coefficient, γ_{ca}, is introduced to relate the effective surface concentration of the reactants on the electrode to the volumetric concentration of the chemical species in the catalyst layer. This coefficient is dependent on the geometry of the electrode (e.g., catalyst surface area) and other factors.

The coefficients of the energy terms in the exponential and the powers of the molar concentrations are the stoichiometric ratios of the reaction, so the coefficient of the electronic term is 2. The voltage loss is only applied to the reaction (e^- only appears as a reactant, not as a product), so the electrode symmetry factor (β) is removed. The factors are reordered and regrouped to be clearer in terms of energy.

$$\begin{aligned} i''_{ca}\!\rightarrow = {} & \mathbf{F}\,\frac{\mathbf{k}\,T}{\mathbf{h}\,\gamma_{ca}\,\bar{\mathrm{v}}}\,[C_{O_2}]^{0.5}\,[C_{H^+}]^2\,[C_{e^-}]^2 \\ & \cdot\exp\left(\frac{0.5\bar{g}_{O_2}{}^0 + 2\bar{g}_{H^+}{}^0 - 2\mathbf{F}\,\Delta\phi_{ca}}{\mathbf{R}\,T}\right) \end{aligned} \tag{7}$$

Following the same logic, the backward current is given by the equation below, which is a departure from Eq. 7.11 of Bockris et. al [2].

$$i''_{ca}\!\leftarrow = \mathbf{F}\,\frac{\mathbf{k}\,T}{\mathbf{h}\,\gamma_{ca}\,\bar{\mathrm{v}}}\,[C_{H_2O}]\,\exp\left(\frac{\bar{g}_{H_2O}{}^0}{\mathbf{R}\,T}\right) \tag{8}$$

The net current is the forward current minus the backward current. Unlike the strict form of Eq. 7, C_{H^+} and C_{e^-} are assumed to be unity. The model assumes that the molar concentration of H^+ is not limiting because, as noted by Wang et al. [11], the molar concentration of H^+ is high in acidic liquid media. The model also assumes that the molar concentration of e^- is not limiting; the voltage losses due to electronic resistance are included separately elsewhere.

$$\begin{aligned} i''_{ca} = {} & \mathbf{F}\,\frac{\mathbf{k}\,T}{\mathbf{h}\,\gamma_{ca}\bar{\mathrm{v}}} \\ & \cdot\left[[C_{O_2}]^{0.5}\exp\left(\frac{0.5\bar{g}_{O_2}{}^0 - 2\mathbf{F}\,\Delta\phi_{ca}}{\mathbf{R}\,T}\right)\right. \\ & \left. - [C_{H_2O}]\exp\left(\frac{\bar{g}_{H_2O}{}^0}{\mathbf{R}\,T}\right)\right] \end{aligned} \tag{9}$$

`HOR` An `HOR` model describes the hydrogen oxidation reaction (HOR) whereby H_2 is consumed and e^- and H^+ are produced: $H_2 \rightarrow 2H^+ + 2e^-$. Following a similar derivation to that of the `ORR` model:

$$\begin{aligned} i''_{an} = {} & \mathbf{F}\,\frac{\mathbf{k}\,T}{\mathbf{h}\,\gamma_{an}\,\bar{\mathrm{v}}}\left[\exp\left(\frac{2\mathbf{F}\,\Delta\phi_{an}}{\mathbf{R}\,T}\right)\right. \\ & \left. - [C_{H_2}]\exp\left(\frac{\bar{g}_{H_2}{}^0}{\mathbf{R}\,T}\right)\right] \end{aligned} \tag{10}$$

The `HOR` and `ORR` models each have only one parameter. The values of γ_{an} and γ_{ca} can be uniquely determined by the cathode and cathode electric potentials and the intensive properties of the fluid at a single value of current. Experimentally, it is difficult to distinguish the anode and cathode contributions to the IR-free voltage; however, the anode overpotential is typically much lower [1]. It may not be necessary to determine the exact ratio between γ_{an} and γ_{ca} because the goal of the model library is to describe the overall FC electric potential-current relationship rather than the electric potential-current relationships of the anode and cathode separately.

2.3.3 Transport Models

The flow process models describe the flow of energy and species through the FC, but not the storage of energy or species.

`PressureLoss` The model of pressure loss describes advection due to a gradient in total fluid pressure. The fluid is assumed to be uniformly mixed at the molar concentrations C_j of the upstream interface. The flow rate of chemical species i through the downstream interface j, $\dot{n}_{i,j}$, is given by:

$$\dot{n}_{i,j} = C_j \, \dot{n}_j \tag{11}$$

The relationship between the flow rate and pressure difference depends on the fluid properties and the flow regime (e.g., laminar or turbulent), which are either assumed to be constant or determined based on the operating conditions and the fluid properties. The flow rate versus pressure loss equations are given in the `WallFriction` within the `Modelica_Fluid` library [5]. The `PressureLoss` model does not affect the behavior of the FC model significantly because the pressure difference across the flow channels of a FC are often negligible with respect to the absolute pressure. Nevertheless, the model is included to separate the flow channel into discrete storage volumes with varying molar concentrations of the species.

`DiffusionSurface` The `DiffusionSurface` model represents the surface at the boundary between advection-dominated flows (e.g., the flow of reactants down the flow channel due to a gradient in pressure) and diffusion-dominated flows (e.g., the flow of reactants through the gas diffusion layer (GDL) due to gradients in molar concentration). The molar concentration of chemical species at the diffusive interface is the average of that across the advective flow path.

`TransportPorous` The `TransportPorous` model describes flow through a porous material due to gradients in molar concentration and pressure. The following equation is has been developed to describe the flow of gas from Fick's law [17], Darcy's law, and an analogy to laminar flow through a pipe. The gas phase of the chemical species of interest, i, is coupled in a binary manner to the $N_S - 1$ other chemical species.

$$\dot{n}_{i(g)} = -D_i \, A \, \frac{\partial n'''_i}{\partial L_{\cdot z}} + \frac{\sum\limits_{\substack{j=1 \\ j \neq i}}^{N_S} C_j \, \dot{n}_{j(g) \nabla p} / D_{i,j}}{\sum\limits_{\substack{j=1 \\ j \neq i}}^{N_S} C_j / D_{i,j}} \tag{12}$$

Optionally, the advective flow of liquid water can be added in parallel to the flow of gas. The flow of liquid water is described according to Eq. 4 in [18]. If the flow of liquid water is included, then the porosity available for the flow of gas is reduced according to $\varepsilon_{vg} = \varepsilon_v \, \mathrm{v}_g / (\mathrm{v}_l + \mathrm{v}_g)$.

In the literature within the field of proton exchange membrane fuel cells (PEMFCs) [12], the transport process is often described with a viscous flow equation such as Darcy's law and $N_S - 1$ repetitions of an equation consisting of momentum, Stefan-Maxwell diffusion, and Knudsen diffusion terms. However, this approach is not rigorous in terms of energy. Weber and Newman [14] note that the typical modeling approach, the dusty-gas model, can lead to a singular matrix. The traditional equations for flow through a porous material have been modified to derive Eq. 12, which is suitable for the energy-based Modelica formalism.

The development of Eq. 12 starts with Darcy's law to describe the advective flow of gases through a porous medium over a nominal or superficial cross-sectional area (A^0) [15]:

$$\dot{\mathrm{v}} = -\frac{\kappa \, A^0}{\mu} \, \frac{\partial p}{\partial L_{\cdot z}} \tag{13}$$

Darcy's law is modified to use kinematic viscosity instead of dynamic viscosity ($\mu = \rho \, \nu$) and molar flow rate instead of volumetric flow rate ($\dot{v} = \bar{\mathrm{v}} \, \dot{n}$). Since density is related to molar mass and molar volume ($\rho = \mathbf{M} / \bar{\mathrm{v}}$), the previous equation can be rewritten as:

$$\dot{n}_{g\nabla p} = -\frac{\kappa A^0}{\mathbf{M}\,\nu_g}\frac{\partial p}{\partial L_{\cdot z}} \tag{14}$$

The equation for laminar flow in a circular pipe is modified in a similar way:

$$\dot{n}_{\nabla p} = -\frac{\pi\, d^4}{128\mathbf{M}\,\nu_g}\frac{\partial p}{\partial L_{\cdot z}} \tag{15}$$

The following equation is obtained by substituting the definition of hydraulic diameter in terms of the effective cross-sectional area and wetted perimeter ($d = 4A^{eff}/P$)and recognizing that the perimeter is the circumference of a circle ($P = 2\pi\, r$):

$$\dot{n}_{\nabla p} = -\frac{\left(A^{eff}\right)^4}{P^3\, r\,\mathbf{M}\,\nu_g}\frac{\partial p}{\partial L_{\cdot z}} \tag{16}$$

Comparing Eqs. 15 and 16, the permeability can be written as $\kappa = (A^{eff})^4/(P^3\, r\, A^0)$. The effective cross-sectional area for flow, A^{eff}, is related to the nominal cross-sectional area by $A^{eff} = \varepsilon_A\, A^0$. The areal porosity, ε_A, is related to the volumetric porosity by $(\varepsilon_A)^{1/2} = (\varepsilon_v)^{1/3}$, assuming that the geometry of the pores is isotropic. This assumption is consistent with the Bruggeman correction for tortuosity which leads to ${D_{i,j}}^{eff} = (\varepsilon_v)^{3/2}\, D_{i,j}$ (Eqs. 41 and 42 in [12]). Assuming that the cross-section of the pores is circular with radius r and volumetric porosity ε_{vg} available for gas, then the permeability of the medium to gas is given by $\kappa = r^2\,(\varepsilon_{vg})^{8/3}/8$.

The total advective flow can be split into flows of the individual chemical species by accounting for the coupling between the chemical species. The coupling is assumed to be binary, so the development starts with the Stefan-Maxwell equation for binary diffusion [16]. Here, i denotes the chemical species of interest and j denotes the other chemical species.

$$\nabla C_i = \sum_{\substack{j=1\\ j\neq i}}^{N_S} \frac{C_i C_j}{D_{i,j}}(v_j - v_i) \tag{17}$$

The concentration gradient is set to zero here because the diffusive flow is handled separately. The velocity of the flow is related to the molar flow rate ($v = \dot{n}\,\bar{\mathrm{v}}/A$). Making these adjustments and solving for the molar flow rate of chemical species i:

$$\dot{n}_{i(g)\nabla p} = \frac{\sum\limits_{\substack{j=1\\ j\neq i}}^{N_S} C_j\, \dot{n}_{j(g)\nabla p}/D_{i,j}}{\sum\limits_{\substack{j=1\\ j\neq i}}^{N_S} C_j/D_{i,j}} \tag{18}$$

Only $N_S - 1$ equations in the form of Eq. 18 are unique; the ${N_S}^{th}$ equation is redundant and consistent. In summary, the $N_S + 1$ variables characterizing the advective flow of the `TransportPorous` model ($\dot{n}$, $\dot{n}_1$,..., $\dot{n}_{N_S}$) are described by Eq. 16 (with $\kappa = r^2\,(\varepsilon_v)^{8/3}/8$), $N_S - 1$ equations in the form of Eq. 18, and an additional equation stating the molar flow rate ($\dot{n} = \sum \dot{n}_j$).

The diffusive flow of each chemical species is given by Fick's law, stated in terms of volumetric concentration of the chemical species and the molar volume of the mixture:

$$\dot{n}_{i\nabla C} = -D_i A \frac{\partial n'''_i}{\partial L_{\cdot z}} \tag{19}$$

The advective and diffusive flow rates are added for each chemical species to obtain the net flow rate within the `TransportPorous` model, Eq. 12.

`DiffusionMembrane` The `DiffusionMembrane` model represents the diffusion of H_2O through the PEM. The rate of diffusive flow is proportional to the hydration gradient, as described by Eq. 20 in [10]. The diffusion coefficient is either held constant or related to PEM hydration according to Eq. 22 in [10] and the software code in developed by Springer [8].

`ElectroOsmoticDrag` The `ElectroOsmoticDrag` model describes both electro-osmotic drag and resistance to protonic flow in the PEM. The flow of H_2O through the PEM and the voltage loss across it are both related to the protonic current. As described by Eq. 18 in [10], electro-osmotic drag carries H_2O through the PEM at a rate that is proportional to protonic current. The coefficient of proportionality depends on PEM hydration. The difference in electric potential is also proportional to protonic flow, as described by Ohm's law. The resistance to protonic flow is either constant, described as an empirical function of protonic current, or related to PEM hydration according to Eq. 25 in [10]. The energy balance within `ElectroOsmoticDrag` model accounts for heating due to resistance.

`MembraneSurface` The `MembraneSurface` model relates hydration to humidity, or the activity of H_2O, at the surface of the PEM as Springer et al. determined empirically (Eqs. 16 and 17 in [10]).

`HeatingResistor` A `HeatingResistor` model describes heat generation and voltage loss due to a constant resistance.

`ThermalConductor` and `ThermalConvection` The `ThermalConductor` model and `ThermalConvection` models describe heat flow and are instantiated directly from the `Modelica` library.

3 Simulation Results

Figure 1 compares polarization curves with varying test conditions, specifically anode and cathode pressure, flow plate temperature, cathodic reactant flow rate, and anode and cathode RH. The baseline scenario is shown as the solid curve in each plot. Figure 1a shows that cell electric potential increases with increased operating pressure due to higher reactant volumetric concentration (i.e., higher reactant partial pressure), but that the gain decreases with increasing pressure. Figure 1b shows that the cell electric potential decreases as the flow plate temperatures and thus the catalyst layer temperatures are increased, due to the decreased change in Gibbs free energy of the reaction (i.e., decreased Nernst potential). Figure 1c shows that electric potential increases as the cathode reactant flow rate is increased, especially in the high current density region, due to increased O_2 molar concentration. Figure 1d demonstrates the net effect of two underlying consequences of varying reactant RH. Higher RH leads to increased hydration of the PEM and lower protonic resistance in the PEM, which causes the slope of the curve to become less negative. However, the increase in RH also decreases the molar concentration of the reactants, so the limiting current density decreases. Qualitatively, the trends shown in Figure 1 are consistent with the experimental test results presented in the literature [9] .

4 Summary

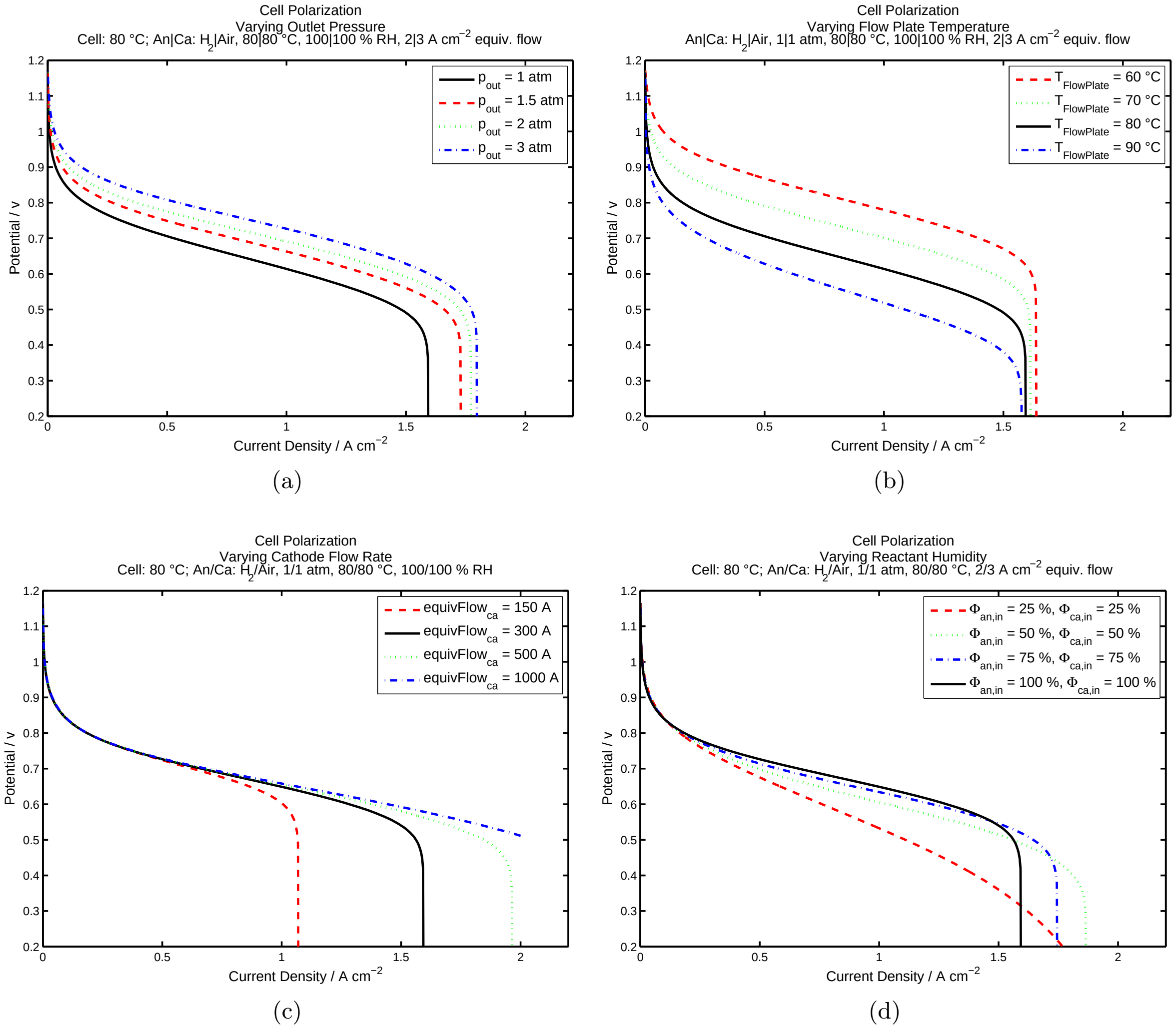
Most previous FC models have been created with the primary goal of matching the empirical observations of FC operation. In some cases, this has resulted in model equations that are not rigorous on an energy basis. The research presented within this paper modifies some of the traditional FC model equations to be suitable for the acausal, energy-based representation. Modelica is used as a platform to resolve the differences between empirical representations and first-principle relationships pertaining to fuel cells.

Two areas have been emphasized in this paper: the transport of chemical species by diffusion and electrochemical reactions. These areas are central to fuel cell modeling and have not previously been explored to a full extent in Modelica. Most models in the FC literature describe diffusion via the dusty-gas model or derivatives of it, which are empirically based and may lead to singularities [14]. An alternative has been proposed which is explicit in terms of energy balances. Most FC models also describe the anode and cathode jointly. However, this approach is not suitable for the FC library because the anode and cathode catalyst layers are separate and are connected to opposite sides of the PEM. The proposed alternative addresses this need by modifying the traditional Butler-Volmer equation.

5 Nomenclature

Symbols

A	Area ($/\mathrm{m}^2$).
C	Molar concentration ($/\mathrm{mol\,mol}^{-1}$).
D	Diffusion coefficient ($/\mathrm{m}^2\,\mathrm{s}^{-1}$).
d	Diameter (/m).
$\mathbf{F}$	Faraday's constant ($/\mathrm{C\,mol}^{-1}$).
g	Gibbs free energy (/J).
$\mathbf{h}$	Planck's constant (/J s).
h	Enthalpy (/J).
i	Current (/A).
$\mathbf{k}$	Boltzmann's constant ($/\mathrm{J\,K}^{-1}$).
L	Length (/m).
$\mathbf{M}$	Molar mass ($/\mathrm{kg\,mol}^{-1}$).
N	Number (/1).
n	Amount (/mol).
P	Perimeter (/m).



(a)

(b)

(c)

(d)

Figure 1: Polarization curves with varying (a) pressure at cathode and anode outlets, (b) temperature of anode and cathode flow plates, (c) reactant flow rate through the cathode flow channels, and (d) relative humidity at anode and cathode inlets.

p Pressure (/Pa).

$\mathbf{R}$ Universal gas constant (/J mol^{-1}K^{-1}).

R Resistance (/Ω).

r Radius (/m).

s Entropy (/J K^{-1}).

T Temperature (/K).

t Time (/s).

u Internal energy or heat (/J).

v Volume (/m^3).

v Velocity (/m s^{-1}).

x Water quality (i.e., the fraction of water that is vaporized) (/1).

β Electrode symmetry factor (i.e., charge transfer coefficient).

ε Porosity (fraction of free space to total space) (/1).

γ Surface area per volume (/m^{-1}).

κ Permeability (/m^2).

λ PEM hydration (/mol$_{\mathrm{H_2O}}$ (mol$_{\mathrm{SO_3^-}}$)$^{-1}$).

μ Dynamic viscosity (/Pa s).

ν Kinematic viscosity (/m^2 s^{-1}).

ϕ Electric potential, (/V).

ρ Density (/kg m^{-3}).

Accents

$\bar{(\ )}$ Per amount (/mol^{-1}).

$\dot{(\ )}$ Per time (e.g., flow rate or velocity) (/s^{-1}).

$(\ )''$	Per area ($/\mathrm{m}^{-2}$).
$(\ )'''$	Per volume ($/\mathrm{m}^{-3}$).

Subscripts

$(\ )_{(x)}$	As x.
$(\ )_{\cdot x}$	Along x.
$(\ )_{\nabla x}$	Due to the gradient in x.
$(\ )_{x,y}$	Of x and y.
$(\ )_x$	Of x.
A	On an areal basis.
an	The anode.
ca	The cathode.
dry	Without or excluding H_2O.
F	Degrees of freedom.
g	The gas phase.
i	The interface or chemical species denoted by i.
j	The interface or chemical species denoted by j.
k	The interface or chemical species denoted by k.
l	The liquid phase.
P	Phases.
$react$	The reaction.
S	Species.
v	On a volumetric basis.
z	The dimension from the anode to the cathode (parallel to charge flow).
$\leftarrow$	In the backward direction.
$\rightarrow$	In the forward direction.

Superscripts

$^{+}$	In the positive state.
$^{-}$	In the negative state.
^{0}	In the initial, nominal, or reference state (e.g., standard pressure).
^{eff}	Effective.
^{mod}	Modified.

6 Acknowledgements

This research was supported in its initial stages by the Office of Naval Research under grant #N00014-04-0682 to the Hawaii Natural Energy Institute and later by the Robert G. Shackelford Fellowship of the Georgia Tech Research Institute. The authors wish to acknowledge the support of the Modelica Association and the Modelica community by the open sharing of the `Modelica` library and the `Modelica_Fluid` library. Although it is not currently possible to offer the library described within this paper online as open-source code, the authors request that any interested readers contact them.

References

[1] D. M. Bernardi and M. W. Verbrugge. A mathematical model of the solid-polymer-electrolyte fuel cell. *Journal of The Electrochemical Society*, 139(9):2477–91, 1992.

[2] J. O. M. Bockris, A. K. N. Reddy, and M. Gamboa-Aldeco. *Modern Electrochemistry 2A: Fundamentals of Electrodics.* Kluwer Academic/Plenum Publishers, New York, 2nd edition, 2000.

[3] B. McBride, M. Zehe, and S. Gordon. NASA Glenn coefficients for calculating thermodynamic properties of individual species. NASA report TP-2002-211556, 2002.

[4] Modelica Association. Modelica: A unified object-oriented language for physical systems modeling, February 2 2005.

[5] Modelica Association. Modelica Fluid Library, August 2006. v1.0 Beta 1.

[6] Modelica Association. Modelica Standard Library, March 2006. v2.2.1.

[7] M. J. Moran and H. N. Shapiro. *Fundamentals of Engineering Thermodynamics.* John Wiley & Sons, Inc., Hoboken, NJ, 5th edition, 2004.

[8] Private communication from T. E. Springer, LANL. Fortran code of Springer 1991 polymer electrolyte fuel cell model, 2007.

[9] T. E. Springer, M. S. Wilson, and S. Gottesfeld. Modeling and experimental diagnostics in polymer electrolyte fuel cells. *Journal of The Electrochemical Society*, 140(12):3513–3526, 1993.

[10] T. E. Springer, T. A. Zawodzinski, and S. Gottesfeld. Polymer electrolyte fuel cell model. *Journal of The Electrochemical Society*, 138(8):2334–2342, 1991.

[11] J. X. Wang, T. E. Springer, P. Liu, M. Shao, and R. R. Adzic. Hydrogen oxidation reaction

on Pt in acidic media: Adsorption isotherm and activation free energies. *Journal of Physical Chemistry C*, 111(33):12425–12433, 2007.

[12] A. Z. Weber and J. Newman. Modeling transport in polymer-electrolyte fuel cells. *Chemical Reviews*, 104(10):4679 – 4726, 2004.

[13] A. Z. Weber and J. Newman. Transport in polymer-electrolyte membranes III: Model validation in a simple fuel-cell model. *Journal of The Electrochemical Society*, 151(2):326–339, 2004.

[14] A. Z. Weber and J. Newman. Modeling gas-phase flow in porous media. *International Communications in Heat and Mass Transfer*, 32(7):855 – 860, 2005.

[15] Wikipedia. Darcy's law, November 2008.

[16] Wikipedia. Maxwell-Stefan-diffusion, November 2008.

[17] Wikipedia. Fick's law of diffusion, April 2009.

[18] Z. Zhan, J. Xiao, Y. Zhang, M. Pan, and R. Yuan. Gas diffusion through differently structured gas diffusion layers of PEM fuel cells. *International Journal of Hydrogen Energy*, 32(17):4443–4451, 2007.

Dynamic Modelling of *CO_2*-removal units for an IGCC power plant

Sindy Heil[1] Christian Brunhuber[2] Kilian Link[2] Julia Kittel[1] Bernd Meyer[1]

[1]Institute of Energy Process Engineering and Chemical Engineering
TU Bergakademie Freiberg
09596 Freiberg
Sindy.Heil@iec.tu-freiberg.de,

[2]Siemens AG, Energy Solutions
Freyeslebenstraße 1
91058 Erlangen

Abstract

This article describes dynamic models of the carbon dioxide (*CO_2*) -removal units which are coupled with conventional models to form a complete model of an IGCC power plant with *CO_2* capture.

Therefore some components of the Modelica_Fluid 1.0 library and packages of the Modelica.Media library from Modelica 3.0 were used. Not yet available components were developed.

The results obtained with Dymola 7.1 were compared with steady state simulations calculated with other tools (ChemCAD and Aspen Plus) and a very good agreement was found.

Keywords: IGCC, Rectisol Wash, CO Shift, *CO_2*-removal

1 Introduction

The object of interest is an Integrated Gasification Combined Cycle (IGCC) Carbon Capture & Storage (CCS) power plant with Siemens Fuel Gasifier Technology (SFGT). This is a climate-friendly power plant where a gas island consisting of gasification and a gas treatment is connected with a Combined Cycle (gas and steam turbine) to generate electricity.

The interactions between the several plant units are very complex and require a dynamic analysis to predict bottlenecks, to react to planned revisions (e.g. load changes), unplanned outages (gasifier trip, gas turbine trip, etc.) and to ensure the correct and safe operation behaviour of the plant. Furthermore the dynamic model is the basis for the development of an optimised control system. The overall object of the research is to raise the availability of IGCC power plants (Figure 1) because this is inevitably connected with the operating and therefore economic efficiency of the plant.

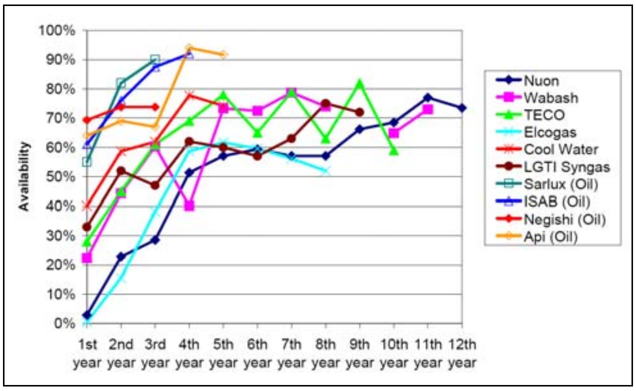


Figure 1: Availability statistics for IGCC first-of-a kind plant [1]

As illustrated in Figure 1 the availability rises over the years of operation. The aim is to start already with a higher availability and of course to operate the plant with a high availability. This demands an exact process knowledge which can be gained with dynamic modelling.

DOI: 10.3384/ecp09430021

Making use of dynamic modelling for analysing IGCC processes gets more and more relevant.

Schoen for example used a dynamic model to control the performance of the Buggenum IGCC [2]. The U.S. Department of Energy's (DOE) of the National Energy Technology Laboratory (NETL) works on an IGCC dynamic plant simulator for a research and training center [3].

This contribution deals with the simplified modelling of the transient behaviour of an IGCC power plant with Modelica and Dymola with the focus on the gas path of the plant.

The introduction provides a short review of the IGCC power plant with CCS technology and the interaction of the sub-units.

The main part of the article describes the dynamic modelling of the CO_2-removal units: CO shift and Rectisol wash.

In the last part of the paper results of the modelled sub-units are demonstrated and an outlook of further challenges is given.

2 IGCC power plant

In Figure 2 the sub-units of an IGCC power plant and their main interaction flows are shown.

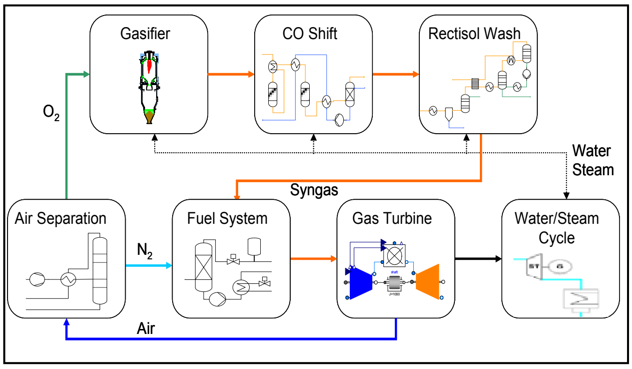


Figure 2: Simplified scheme of an IGCC power plant

In the gasifier the coal is gasified with oxygen (O_2) to produce a synthesis gas (syngas). The main components of the syngas are carbon monoxide (CO) and hydrogen (H_2).

In the next step the CO of the syngas is converted in the CO shift together with steam into CO_2 and H_2.

The formed CO_2, sulphur compounds like hydrogen sulphide (H_2S) and carbonyl sulphide (COS) and other impurities like nitrogen compounds are removed from the syngas by means of a physical wash, e.g. Rectisol or Selexol. In the presented contribution the Rectisol wash is chosen which uses methanol (CH_3OH) as solvent. This physical scrubbing process separates highly purified CO_2 which allows the application of CCS technologies.

In the fuel system the cleaned syngas is diluted with nitrogen (N_2) to produce an utilisable fuel for the gas turbine to generate electricity.

Parts of the compressed air from the gas turbine compressor can be routed to the air separation unit. There the air is separated into O_2 for the gasification process and N_2 as fuel diluting agent.

The gasifier und gas turbine waste heat is used to generate steam. This steam is routed to a steam turbine for electrical power generation.

3 Developed Models

The motivation of using Modelica for this application is based on its multi-purposed, object-oriented background, which allows the user an equation-based approach. In contrast, tools like Aspen Dynamics offer already most of the required components and also more detailed media models, but are not that flexible for user specific developments.

For the implementation some models from the Modelica.Media and the Modelica_Fluid library were used. These libraries provide components to model thermo-hydraulic systems, but do not focus on gas dynamic problems [4]. Consequently components which are not yet available in the Modelica libraries, like the shift reactors or the absorber columns, were modelled. Further physical properties of methanol as physical solvent of the Rectisol wash and its mixture with CO_2 or water (H_2O) were defined as incompressible media.

Because of the complexity of the process many assumptions and simplifications were necessary to ensure a simulation in real time. For example the chemical water gas shift reaction and the sour gas absorption are approximated by interpolation functions depending on temperature. The developed models are based on the equations obtained from thermodynamic literature and assume equilibrium conditions with ideal behaviour in liquid phase and vapour phase.

Generally only the gas path is modelled and the water / steam cycle is neglected. The gas vector consists of the following 8 components:

$$N_2, H_2, CO, CO_2, CH_4, H_2S, COS, H_2O.$$

3.1 CO Shift

The CO shift is an equilibrium-limited reaction. *CO* reacts exothermally with steam at elevated temperatures according to:

$$CO + H_2O \rightarrow CO_2 + H_2. \quad (\Delta H_{298} = -41 \text{ kJ/mole}) \quad (1)$$

Figure 3 shows a simplified process flow diagram for this application.

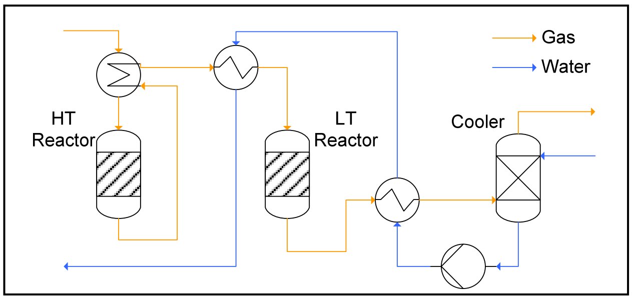


Figure 3: Process flow diagram of a CO shift

In the presented example the CO shift is carried out in two adiabatic reactors in series with intercooling.

Because of the already adequate moisture content after the gasifier with quench there is no saturation step necessary after the gasification island.

The heat of the exit gas from the first reactor (high temperature CO shift, HT) is recovered as high-pressure superheated steam. The steam produced after the second reactor (low temperature CO shift, LT) is recovered by further heat integration [5].

The *CO* concentration in the exit gas depends on the temperature and the mixture composition of the syngas which is provided by the gasifier.

In the first reactor the bulk of *CO* is converted. The LT reactor, which is installed downstream of the HT shift realises a *CO*-conversion down to parts per million (ppm) levels at the reactor outlet.

The reactors include a catalyst bed to promote the CO shift reaction. This catalyst is capable for the conversion of *CO* in sour gas. This means it is active in the presence of sulphur compounds. The positive side effect of this catalyst is the simultaneously executed COS hydrolysis with the following chemical reaction:

$$COS + H_2O \rightarrow CO_2 + H_2S. \quad (\Delta H_{298} = -35 \text{ kJ/mole}) \quad (2)$$

Down-stream of the CO shift reactors the shifted syngas is cooled down and the condensed water is separated and used for the intercooling.

3.1.1 CO Shift Reactor

In the reactor model the mass, component mass, energy and momentum equations have to be considered.

The balance equations were all effected by the CO shift and the COS hydrolysis. These chemical reactions can be considered as instantaneous that means that the chemical equilibrium is attained. The reactions are modelled with the help of conversion rates for *CO* and *COS* which are calculated by linear equations depending on the temperature T in a predefined interval as described as follow:

$$conversionrate(T) = a + bT. \qquad (3)$$

The values for these linear equations were obtained by a sensitivity analysis of a CO shift reactor in Aspen Plus by varying the temperature in the corresponding interval. With these conversion rates the component mass balances are calculated. Based on this knowledge the energy balance can be specified with the exothermal heat of the CO shift and COS hydrolysis reaction. Therefore the heat values of the gas at the inlet and outlet are used. The energy balance also includes the reactor mass as heat storage.

The pressure drop depends on the mass flow. For the design case default parameters for both reactors are given. The following relation is used [6]:

$$\frac{\dot{m}^2}{\Delta p} = \frac{\dot{m}_0^2}{\Delta p_0}, \qquad (4)$$

where $\dot{m}_0$ is the mass flow and Δp_0 the pressure drop in the design case.

Important for the design and the dynamic behaviour is also the space velocity, which has to be between 1,000 h^{-1} and 3,000 h^{-1} for this application [7]. In this example the space velocity is set to 2,600 h^{-1}.

3.1.2 Heat Exchanger

The heat exchanger is built on several heat nodes to realise more than one heat transmission point.

Every heat node consists of two vessels called ClosedVolume taken from the Modelica_Fluid library. They are connected by their heat ports with a given heat transfer coefficient. With the valves between the heat nodes the pressure drop, given as a constant parameter, is taken into account.

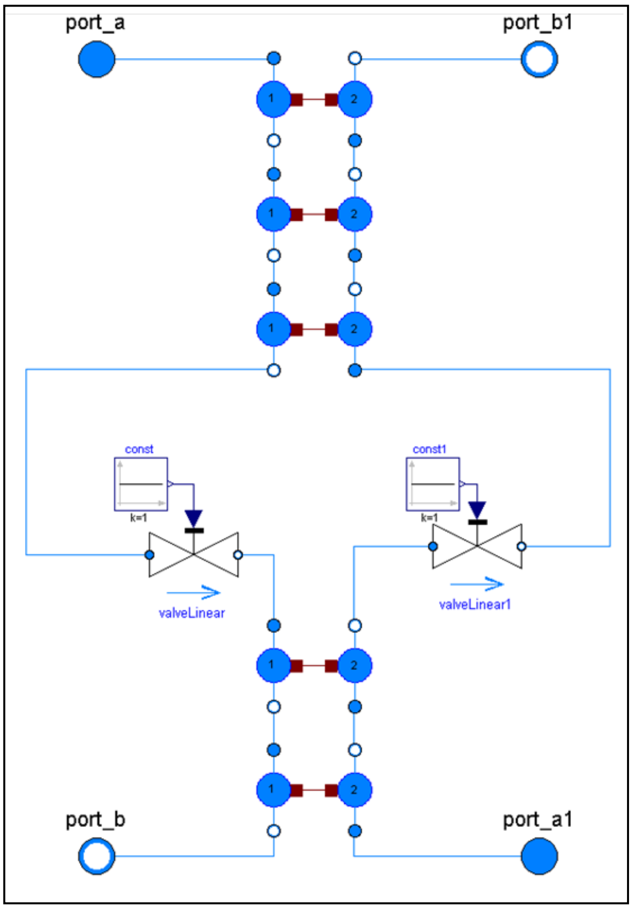


Figure 4: Schematic of the heat exchanger with 5 heat nodes in Dymola

3.1.3 Cooler

The cooler is located between the CO shift and the Rectisol wash.

In the cooler the gas is cooled down and the condensed water leaves the column.

To account for the temperature and moisture gradient the column is divided into theoretical stages, where the conservation laws are derived for each theoretical stage. The mass balance includes the gas and the water flows. For the water content in the gas the saturation state is calculated. The heat of condensation is considered in the energy balance. The pressure drop is assumed as constant for the complete column.

3.1.4 Specific challenges

The process flow diagram of the complete CO shift diagram in Dymola is shown in Figure 5.

The validation of the dynamic model is another challenge, because relevant dynamic data (*T, X, p*) from existing plants is not yet available. Nevertheless, steady state performance was validated with the help of simulation result in Aspen Plus and ChemCAD. The dynamic behaviour could only be validated via plausibility check.

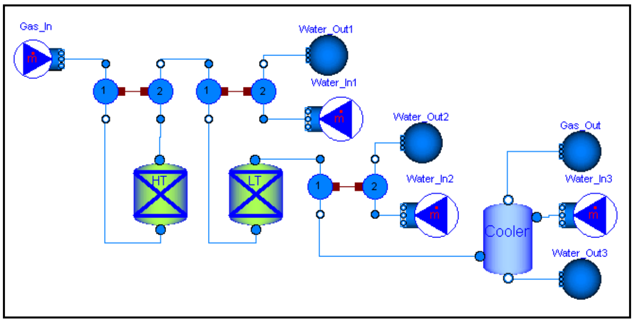


Figure 5: Process flow diagram of the CO shift diagram in Dymola

3.2 Rectisol Wash

The Rectisol process is a physical wash process which uses cold methanol as physical solvent. The undesired components of the raw gas, that are produced in gasification with coal, such as CO_2, H_2S, *COS, HCN*, NH_3 and other traces are physically absorbed by methanol. In the regeneration part these components are desorbed by reducing the pressure of the solvent, stripping or reboiling the solvent.

The different solubilities of the components allow a selective removal of H_2S and CO_2 dependent on the temperature. Also the solubility of the trace components, which is much higher than those of H_2S, allows removing them separately in the prewash stage. This gives the ability to achieve very high gas purity with H_2S concentration of typically 0.1 ppm and CO_2 concentrations in the range of 2 – 4 Vol.-% down to few ppm [7].

In Figure 6 the process flow diagram of the Rectisol plant in Dymola is presented. The process flow diagram shows a selective two-step design. This means that H_2S is removed in the first step followed by the CO_2-removal in the second step.

The raw gas entering the plant in the prewash stage is cooled. There trace components are removed at a very small cold solvent rate. The gas is first chilled by heat exchange with process off-gas and then by refrigeration.

Thereafter, in the H_2S absorber the sulphur is removed from the gas using a relatively small flow of CO_2-rich solvent to a residual sulphur content of below 0.1 ppm.

The CO_2 is removed in a two-stage CO_2 absorber with the main methanol flow. In the lower section, the CO_2 content of the gas is reduced to about 5 % using flash-regenerated methanol. The remaining

CO_2 is removed using regenerated, cold methanol in the upper section.

The refrigeration balance of the system is maintained by an ammonia refrigeration plant.

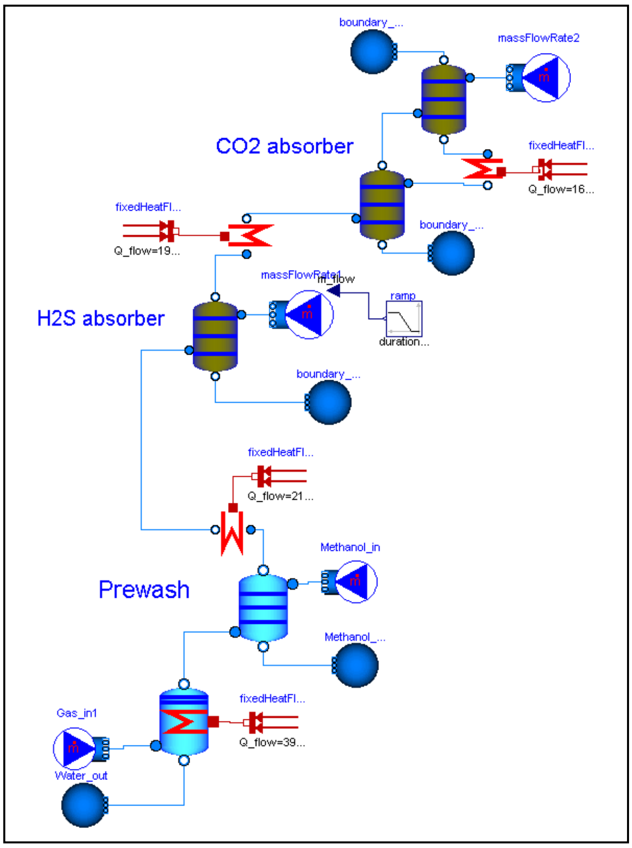


Figure 6: Process flow diagram of the modelled Rectisol plant in Dymola

For the simulation of the Rectisol plant only the gas path is modelled. The regeneration of the methanol solvent and the interaction with the water steam cycle are neglected. Furthermore only the absorption of *CO_2* into methanol is taken into account. There is no transfer of *H_2S* and other trace components considered.

3.2.1 Mixture of Methanol and CO_2

For the modelling of the Rectisol plant the solvent methanol and its mixtures with *CO_2* or water are necessary. These media are not yet available in the Modelica.Media library. Therefore they were created as incompressible media with the help of tables. The minimal data set needed to describe the thermodynamic states is tables of the density ρ and the specific heat capacity c_p as functions of the temperature. For these values data from the NIST Chemistry WebBook was included [8].

For the mixture M of *CO_2* and methanol ideal properties were assumed. This leads to the following equations, where the properties of the mixture follows from the properties of the components in respect of there mass fraction *X*:

$$\rho_M = 1 / \left(\frac{X_{CH_3OH}}{\rho_{CH_3OH}} + \frac{X_{CO_2}}{\rho_{CO_2}} \right), \tag{5}$$

$$c_{p,M} = X_{CH_3OH} \cdot c_{p,CH_3OH} + X_{CO_2} \cdot c_{p,CO_2}. \tag{6}$$

The same relations are used for the mixture of *CH_3OH* and *H_2O* needed in the prewash section.

3.2.2 Prewash

The Prewash consists of a cooler and a prewash column. In the cooler with condensate trap a predefined heat flow is released. In the prewash column a small methanol flow cools the gas down again and is derived together with the condensed water flow.

For the calculation of the saturated gas properties the same equations like in the cooler are used.

3.2.3 *CO_2* Absorber

The raw gas enters the absorber column at the bottom section and is contacted with the scrubbing methanol introduced at the top of the column. The methanol leaves the column at the bottom together with the absorbed *CO_2*.

The modelling of the absorber column is based on the equilibrium stage model, which divides the column into theoretical stages and calculates the balance equations for each several stage (Figure 7).

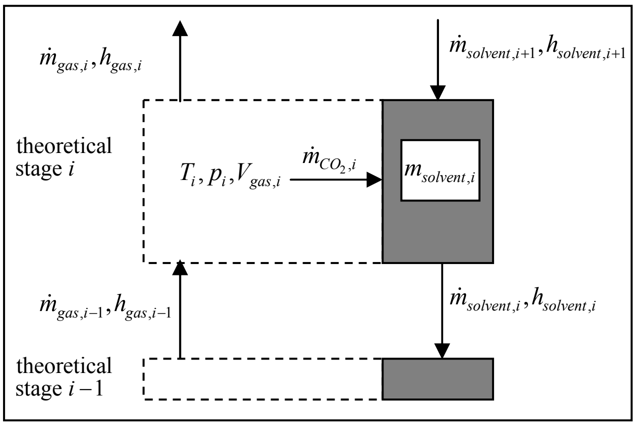

$\dot{m}_{gas,i}, h_{gas,i}$
$\dot{m}_{solvent,i+1}, h_{solvent,i+1}$
theoretical stage i
$T_i, p_i, V_{gas,i}$
$\dot{m}_{CO_2,i}$
$m_{solvent,i}$
$\dot{m}_{gas,i-1}, h_{gas,i-1}$
$\dot{m}_{solvent,i}, h_{solvent,i}$
theoretical stage $i-1$

Figure 7: Schematically illustration of in- and output streams of a theoretical stage

In the CO_2 absorber there are two different media: the gas and the solvent. For each medium a mass balance is considered but only one energy balance is implemented.

The following modelling assumptions are used:

1) Each column theoretic stage is considered as an adiabatic system.
2) In the energy balance the wall material is regarded as a heat storage system and the exothermic process heat of the CO_2 absorption in the polar solvent is implemented.
3) On the liquid side methanol does not vaporise and hence does not go into the gas phase. Against on the gas side only CO_2 is transferred into the liquid phase.
4) This solubility of CO_2 in CH_3OH is a function of temperature at a partial pressure of one atmosphere.
5) For the gas / liquid equilibrium the ideal Henry law is used, even though this is completely reliable only at low molar fraction and at moderate pressure where no real gas behaviour is to be considered. The Henry law can be described with the following equation [9]:

$$Y_{gas,i} \cdot p = HK \cdot Y_{solvent,i}\,, \tag{7}$$

where $Y_{gas,i}$ is the molar fraction of the component i in the gas, $Y_{solvent,i}$ the molar fraction of the component i in the solvent, p the pressure and HK the Henry coefficient. In this case CO_2 is meant by the component i.

6) To calculate the Henry coefficient experimentally investigated values [10] were interpolated and expressed as polynomial of the temperature T in a predefined interval:

$$HK(T) = a + bT + cT^2. \tag{8}$$

7) For the pressure loss Δp only the hydrostatic part is considered [11]. Therefore it depends only on the solvent level h_{CH_3OH} on the theoretical stages:

$$\Delta p = \rho_{CH_3OH} \cdot g \cdot h_{CH_3OH}\,, \tag{9}$$

where ρ_{CH_3OH} is the density of methanol and g the standard gravity.

3.2.4 Validation of the CO_2 absorber

The steady state results of the models were validated with calculations simulated with tools like Aspen Plus and ChemCAD. Therefore the CO_2 absorber is connected with sources and sinks of gas or solvent to compare the results for several cases between Dymola, Aspen Plus and ChemCAD (Figure 8).

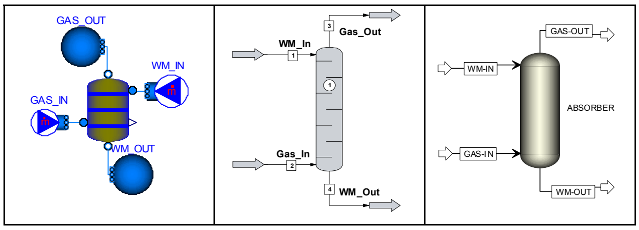


Figure 8: CO_2 absorber in Dymola (left), ChemCAD (middle) and Aspen Plus (right)

For the reference case the following input values are used.

	solvent_in	gas_in
$\dot{m}$ $[kg/s]$	568.07	225.80
T $[K]$	223.15	238.85
p $[bar]$	24.25	24.35
X_{N_2} $[-]$	0	0.090626
X_{H_2} $[-]$	0	0.058889
X_{CO} $[-]$	0	0.047220
X_{CO_2} $[-]$	0	0.803203
X_{CH_4} $[-]$	0	0.000062
X_{H_2S} $[-]$	0	0
X_{COS} $[-]$	0	0
X_{H_2O} $[-]$	0	0
X_{CH_3OH} $[-]$	1	0

Table 1: Input values in the CO_2 absorber for the reference case

In this table $\dot{m}$ is the mass flow, T the temperature, p the pressure and X the mass fraction.

The CO_2 absorber was simulated with 8 theoretical stages and the input values listed in Table 1 in ChemCAD, Dymola and Aspen Plus.

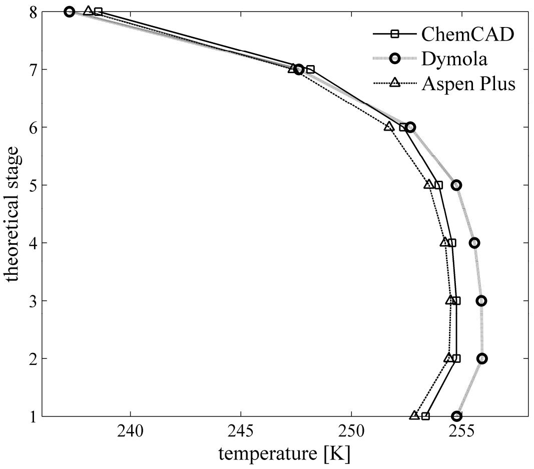


Figure 9: Temperature profile of CO_2 absorber

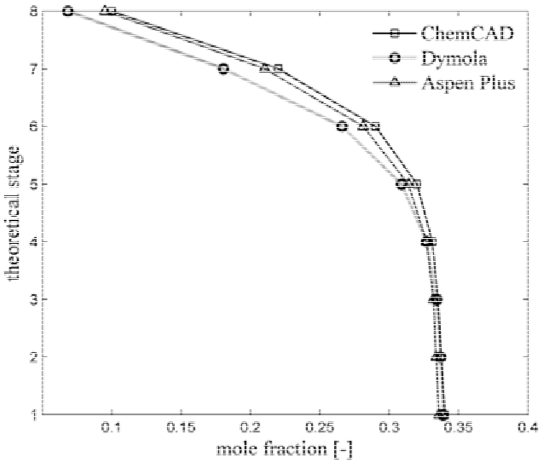


Figure 10: CO_2 content profile in the gas of the CO_2 absorber

case 1	Reference case (Table 1)
case 2	$T_{gas\_in} = 228.85\,K$
case 3	$\dot{m}_{gas\_in} = 425.8\,kg/s$
case 4	$X_{CO_2,gas\_in} = 0.603203$ $X_{N_2,gas\_in} = 0.290626$
case 5	$\dot{m}_{solvent\_in} = 368.0345\,kg/s$
case 6	$T_{solvent\_in} = 243.15\,K$
case 7	$X_{CO_2,solvent\_in} = 0.15$ $X_{N_2,solvent\_in} = 0.85$
case 8	$p_{gas\_in} = 34.35\,bar$ $p_{solvent\_in} = 34.25\,bar$

Table 2: Variation of the input values of the CO_2 absorber

Figure 9 and Figure 10 show the absorber profiles of the temperature and the CO_2 content of the gas. The theoretical stage 8 is the head and stage 1 is the sump of the column. The profiles correlate very well with each other.

As shown in Table 2 the input values were varied for 8 cases.

Figure 11 to Figure 14 show the results of comparing the CO_2 content in the outlet gas and in the solvent and the associated temperatures of the gas and the solvent from the simulation in Dymola with the steady state results calculated in ChemCAD and Aspen Plus.

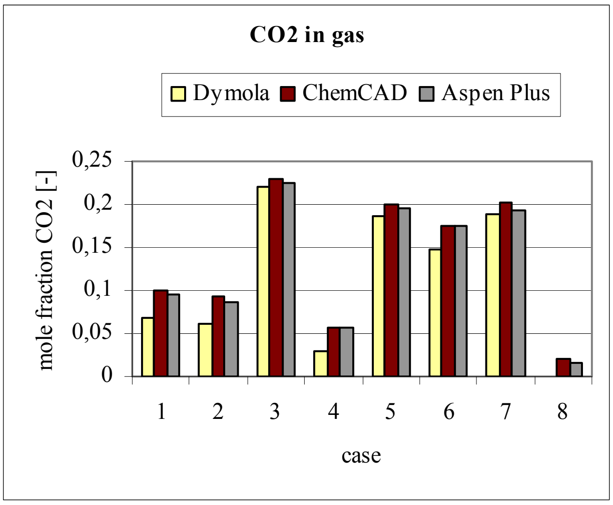


Figure 11: Mole fraction of CO_2 in outlet gas of CO_2 absorber compared between Dymola, ChemCAD and Aspen Plus

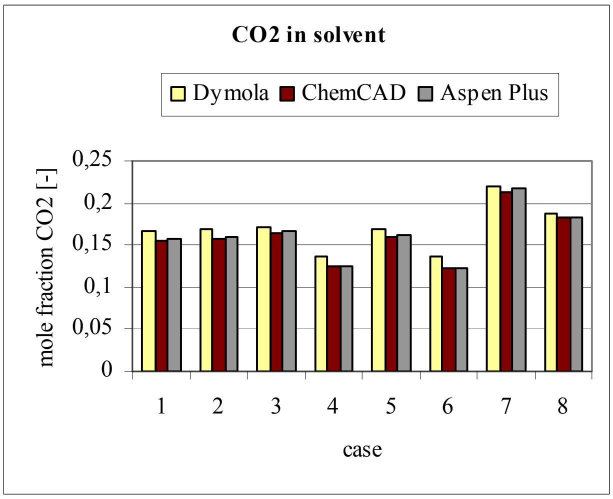


Figure 12: Mole fraction of CO_2 in solvent after CO_2 absorber compared between Dymola, ChemCAD and Aspen Plus

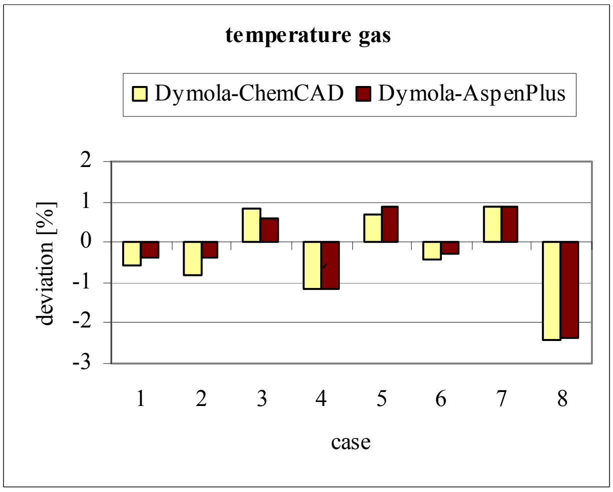


Figure 13: Deviation of temperature of the outlet gas of CO_2 absorber compared between Dymola and ChemCAD and between Dymola and Aspen Plus

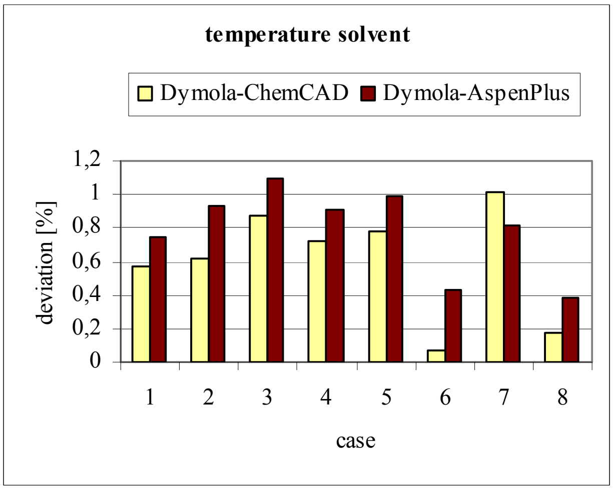


Figure 14: Deviation of temperature of the solvent after CO_2 absorber compared between Dymola and ChemCAD and between Dymola and Aspen Plus

As physical property model in ChemCAD the extended Soave-Redlich-Kwong method and in Aspen Plus the Predicted Redlich Kwong-Soave method were used. The results obtained in the Dymola model show similar results compared to the other simulation tools.

The main differences appear in the CO_2-fraction in gas in case 1 and 2 with 3.2 mole-% between Dymola and ChemCAD (Figure 11) and for the temperature in gas in case 8 with a deviation of 2.4 % between Dymola and ChemCAD as well as between Dymola and Aspen Plus (Figure 13).

4 Conclusions and Outlook

Dynamic models for the CO_2 removal were presented. Because of the ambition to guarantee a computing time faster than real time the resulting DAE systems were solved by the variable time step solver DASSL in Dymola.

The developed simulation models of the CO shift and the Rectisol wash have proven their capability to simulate complex power plant components.

A good agreement was observed for the steady state results of Dymola simulations compared to ChemCAD and Aspen Plus calculations.

Future work will concentrate on power block models, a model for the air separation unit and the gasifier. At the end the overall ambition is to couple the dynamic models with each other in order to build a complete model of an IGCC with CO_2 capture. When all developed models have been sufficiently validated and connected various process studies of control concepts can be performed.

References

[1] Holt, N.: Gasification Technology Status. Electric Power Research Institute EPRI, California USA, 2006

[2] Schoen, P.: Dynamic Modeling and Control of Integrated Coal Gasification Combined Cycle Units. PhD thesis, Faculty of Mechanical Engineering and Marine Technology, Technical University Delft, 1993

[3] Provost, G. T.; Erbes, M. R.; Zitney, S. E.; Phillips, J. N.; McClintock, M.; Stone, H. P.; Turton, R.; Quintrell, M.; Marasigan, J.: Generic Process Design and Control Strategies Used to Develop a Dynamic Model and Training Software for an IGCC Plant with CO_2 Sequestration. International Pittsburgh Coal Conference, Pittsburgh, USA, 2008

[4] Casella, F.; Otter, M.; Proelss, K.; Richter, Ch.; Tummescheit, H.: The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks. Proceedings of the 5th International Modelica Conference, Wien, 2006

[5] Haldor Topsoe: Sulphur resistant water-gas shift/sour shift, www.topsoe.com, 2009

[6] Perez, A. A. G.: Modelling of a Gas Turbine with Modelica. Department of Automatic Control, Lund Institute of Technology, 2001

[7] Kohl, A.; Nielsen, R.: Gas Purification: 5th Edition, Gulf Publishing Company, Houston, Texas, 1997

[8] National Institute of Standards and Technology (NIST) Chemistry WebBook, http://webbook.nist.gov/chemistry/form-ser.html (call date: 11-08-09)

[9] Felli, V.: Appendix C: Analysis and simulation of a rectisol-based acid gas purification process. In: A cost-benefit Assessment of Gasification-Based Biorefining in the Kraft Pulp and Paper Industry, 2006

[10] Schroedter, F.; Melzer, W.-M.; Knapp, H.: Investigation of phase equilibria in multi-component mixtures consisting of propane, carbon dioxide, water and various organic solvents. Gas Separation and Purification, Vol. 5, 1991

[11] Sattler, K.: Thermische Trennverfahren, Grundlagen, Auslegung, Apparate. 3. Auflage, WILEY-VCH, Weinheim, 2001

Mixed Quantitative and Qualitative Simulation in Modelica

François E. Cellier
ETH Zürich
Switzerland
FCellier@Inf.ETHZ.CH

Victorino Sanz
UNED Madrid
Spain
VSanz@DIA.UNED.ES

Abstract

This article introduces a new Modelica library, FIRlib, developed for the mixed quantitative and qualitative simulation of physical systems. Qualitative submodels are built using the *Fuzzy Inductive Reasoning (FIR)* paradigm.

Whereas Modelica has been designed for modeling physical systems from first principles, some systems do not lend themselves to this kind of modeling, either because they are too poorly understood (no meta-knowledge is available yet) or because they are so complex that capturing their behavior in a detailed fashion would be a hopeless undertaking.

Use of the new library is demonstrated by means of two examples, a simple hydraulic control system (a textbook example) and a model of the human cardiovascular system.

Keywords: fuzzy inductive reasoning; inductive modeling; qualitative modeling; mixed quantitative and qualitative simulation; cardiovascular system

1 Introduction

Modelica has been designed as an environment for modeling physical systems from first principles in an object-oriented fashion.

Yet, there exist systems that don't lend themselves easily to this type of modeling, either because the meta-laws governing their dynamic behavior are not fully understood, or because these systems are too complex to be described with complete details.

In both of these cases, we need a tool that can capture dynamic behavior *inductively*, i.e., from observations, rather than *deductively*, i.e., from first principles.

Typical tools that are used for such purposes include *artificial neural networks* and *fuzzy modelers*. In this paper, we propose the use of a fuzzy modeling approach called *Fuzzy Inductive Reasoning (FIR)* [2,5].

In FIR, observations of input/output behavior of an unknown system are fuzzified (discretized with fuzzy membership functions associated with each class). A fuzzy rule base of dynamic relations between inputs and outputs is then automatically synthesized. The fuzzy rule base constitutes the qualitative model of the system. It is subsequently used to infer qualitative behavior of the system in a qualitative simulation step. The qualitative simulation results, so-called episodes, are then defuzzified (quantified) to trajectory behavior using the information contained in the fuzzy membership functions.

We sometimes encounter systems that are partially understood, i.e., the meta-laws describing some of its subsystems are well-known, whereas those describing other subsystems are unknown or only incompletely known.

In such cases, it is useful to be able to simulate such systems using a mixed quantitative and qualitative simulation environment. A (synthetic) example model is shown in Fig.1.

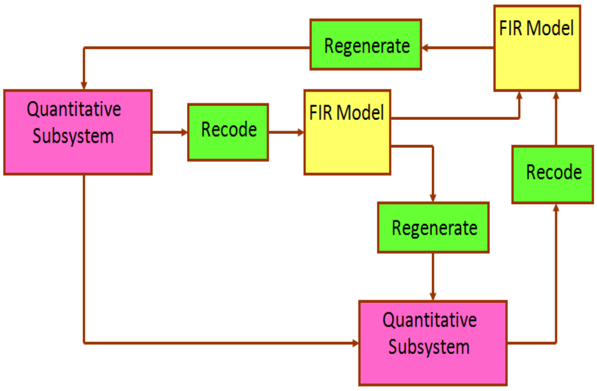


Fig.1: Mixed quantitative and qualitative model

The pink boxes of Fig.1 represent quantitative subsystems, whereas the yellow boxes represent qualitative subsystems. Quantitative signals can be converted to qualitative signals (i.e., fuzzified) using the green *Recode* block, whereas qualitative signals can be converted to quantitative signals (i.e., defuzzified) using the green *Regenerate* block.

DOI: 10.3384/ecp09430050

2 Qualitative Variables

Qualitative variables are variables that assume qualitative values. Variables of a dynamical system are functions of time. The behavior of a dynamical system is a description of the values of its variables over time. The behavior of quantitative variables is usually referred to as trajectory behavior, whereas the behavior of qualitative variables is commonly referred to as episodical behavior. Qualitative simulation can thus be defined as the process of inferring the episodical behavior of a qualitative dynamical system or model.

Qualitative variables are frequently interpreted as an ordered set without distance measure [1]. It is correct that 'warm' is "larger" (warmer) than 'cold,' and that 'hot' is "larger" (warmer) than 'warm.' Yet, it is not true that:

$$\text{'warm'} - \text{'cold'} = \text{'hot'} - \text{'warm'}$$

or, even more absurdly, that:

$$\text{'hot'} = 2 \cdot \text{'warm'} - \text{'cold'}$$

No subtraction operator is defined for qualitative variables.

Whereas many qualitative simulation engines treat also the independent variable, *time*, as a qualitative variable, FIR does not. FIR simulates the behavior of qualitative states as functions of a quantitative *time* variable.

Without this feature, FIR would not be capable of dealing with mixed quantitative and qualitative models.

3 Fuzzy Inductive Reasoning

The Fuzzy Inductive Reasoning (FIR) methodology consists of four primary modules. The *Recode* module converts (fuzzifies) quantitative variables into qualitative variables; the *Optmask* module determines inductively a qualitative model relating sets of observations of input and output behavior; the *Forecast* module performs a qualitative simulation by inferring the episodical (qualitative) future behavior of a set of output variables given a set of input variables and a qualitative model; and finally the *Regenerate* module converts (defuzzifies) qualitative variables into quantitative variables.

3.1 Fuzzification

Recoding denotes the process of converting a quantitative variable to a qualitative variable. In general, some information is lost in the process of recoding. Obviously, a temperature value of 97°F contains more information than the value 'hot.' Fuzzy recoding avoids this problem. Fig.2 shows the fuzzy recoding of a variable called "systolic blood pressure."

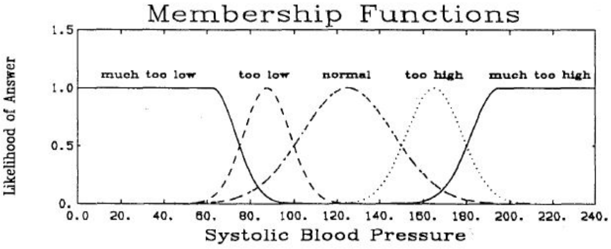


Fig.2: Fuzzy recoding

For example, a quantitative systolic blood pressure of 135.0 is recoded into a qualitative class value of 'normal' with a fuzzy membership value of 0.895, and a side value of 'right.' Thus, a single quantitative value is recoded into a qualitative triple. Any systolic blood pressure with a quantitative value between 100.0 and 150.0 will be recoded into the qualitative value 'normal.' The fuzzy membership function denotes the value of the bell-shaped curve shown on Fig.2, always a value between 0.5 and 1.0, and the side function indicates whether the quantitative value is to the left or to the right of the maximum of the currently active fuzzy membership function. Obviously, the qualitative triple contains the same information as the original quantitative variable. The quantitative value can be regenerated accurately from the qualitative triple, i.e., without any loss of information.

The shape of the fuzzy membership functions can be chosen either Gaussian or triangular, and the landmarks, i.e., the values of the variable to be recoded that separate neighboring classes from each other, can be either user-specified, or they can be determined by the FIR software itself using a variety of different approaches, such as the equal partitioning method [8], whereby the landmarks are chosen such that each class of the recoded variable contains the same number of samples.

3.2 Fuzzy Modeling

A qualitative model determines a relationship between the class values of a set of input variables and that of an output variable. FIR encodes the qualitative model using a so-called *optimal mask*.

A mask denotes a relationship between a set of variables. For example, let us consider the following raw data model consisting of five variables, namely two input variables, u_1 and u_2, and three output variables, y_1, y_2, and y_3, that are recorded at different instants of time:

$$\begin{array}{l c}
\textit{time} & \begin{array}{ccccc} u_1 & u_2 & y_1 & y_2 & y_3 \end{array} \\
\begin{array}{l} 0.0 \\ \delta t \\ 2 \cdot \delta t \\ 3 \cdot \delta t \\ \vdots \\ (n_{\text{rec}} - 1) \cdot \delta t \end{array} &
\begin{pmatrix} \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix}
\end{array}$$

Each column of the raw data model lists the class values of one qualitative variable recorded at different instants of time, and each row lists the class values of all qualitative variables recorded simultaneously. The raw data matrix is accompanied by a fuzzy membership matrix and a side matrix of identical dimensions.

A *mask* denotes a relationship between these variables. For example, the mask:

$$\begin{array}{l c}
{}^{t}\backslash^{x} & \begin{array}{ccccc} u_1 & u_2 & y_1 & y_2 & y_3 \end{array} \\
\begin{array}{l} t-2\delta t \\ t-\delta t \\ t \end{array} &
\begin{pmatrix} 0 & 0 & 0 & 0 & -1 \\ 0 & -2 & -3 & 0 & 0 \\ -4 & 0 & +1 & 0 & 0 \end{pmatrix}
\end{array}$$

denotes the following relationship pertaining to the five variable system:

$$y_1(t) = f(y_3(t\text{-}2\delta t), u_2(t\text{-}\delta t), y_1(t\text{-}\delta t), u_1(t))$$

The single positive element in the mask, always located in the last row, denotes the position of the model output. The negative elements denote the positions of the model inputs. The example mask has four inputs. The sequence in which they are enumerated is immaterial. They are usually enumerated from left to right and top to bottom. Thus, the mask is simply a matrix representation of the qualitative relationship relating model inputs to the model output.

The mask must have the same number of columns as the raw data matrix. The number of rows of the mask is called the *depth* of the mask. The mask can be used to map a dynamic relationship onto a static relationship. To this end, the mask is shifted over the raw data matrix. Selected inputs and outputs can be read out from the raw data matrix and can be written on a single row next to each other. Fig.3 illustrates this process.

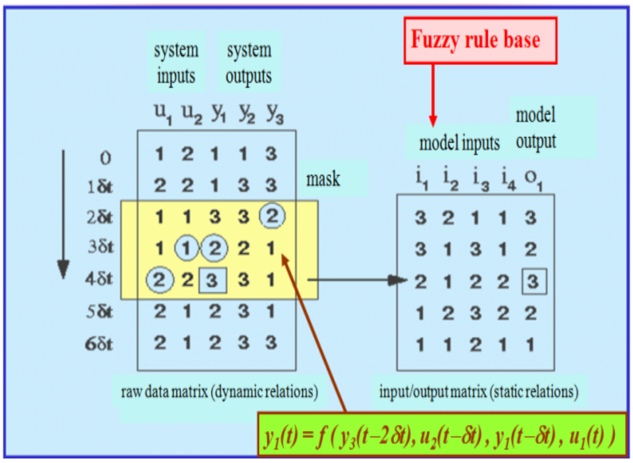

system inputs
system outputs
u_1 u_2 y_1 y_2 y_3
0
1δt
2δt
3δt
4δt
5δt
6δt
1 2 1 1 3
2 2 1 3 3
1 1 3 3 2
1 1 2 2 1
2 2 3 3 1
2 1 2 3 1
2 1 2 3 3
mask
Fuzzy rule base
model inputs
model output
i_1 i_2 i_3 i_4 o_1
3 2 1 1 3
3 1 3 1 2
2 1 2 2 3
1 2 3 2 2
1 1 2 1 1
raw data matrix (dynamic relations)
input/output matrix (static relations)
$y_1(t) = f\,(\,y_3(t-2\delta t), u_2(t-\delta t)\,, y_1(t-\delta t)\,, u_1(t)\,)$

Fig.3: Flattening dynamic relationships

After the mask has been applied to the raw data matrix, the formerly dynamic episodical behavior has become static, i.e., the relationship is now contained within a single row:

$$o_1(t) = f(i_1(t), i_2(t), i_3(t), i_4(t))$$

The resulting matrix is called *input/output matrix*. Each row of the input/output matrix represents a fuzzy rule.

How is the mask selected? A *mask candidate matrix* is constructed, in which negative elements denote positions of potential model inputs, and the single positive element denotes the position of the model output. A good mask candidate matrix for the aforementioned five variable system might be:

$$\begin{array}{l c}
{}^{t}\backslash^{x} & \begin{array}{ccccc} u_1 & u_2 & y_1 & y_2 & y_3 \end{array} \\
\begin{array}{l} t-2\delta t \\ t-\delta t \\ t \end{array} &
\begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & +1 & 0 & 0 \end{pmatrix}
\end{array}$$

A mask candidate matrix is an ensemble of all acceptable masks. The optimal mask selection algorithm determines the best among all masks that are compatible with the mask candidate matrix. The mask shown before is one such mask. The optimal mask is the one mask that maximizes the forecasting power of the inductive reasoning process. To this end, the mask selection algorithm optimizes a mask quality metric that is a combination of a Shannon entropy reduction metric (making the input/output matrix as deterministic as possible) and an observation ratio metric (ensuring that most input/output patterns have been observed at least five times) [5].

3.3 Fuzzy Simulation

Once the optimal mask has been determined, it can be applied to the given raw data matrix resulting in a particular input/output matrix. Since the input/output matrix contains functional relationships within single rows, the rows of the input/output matrix can now be sorted in alphanumerical order. The result of this operation is called the *behavior matrix* of the system. The behavior matrix is a finite state machine. For each combination of input values, it shows, which output is most likely to be observed.

Forecasting (simulation) is now a straightforward procedure. The mask is simply shifted further down beyond the end of the raw data matrix, future inputs are read out from the mask, and the behavior matrix is used to determine the future output, which can then be copied back into the raw data matrix. In fuzzy forecasting, it is essential that, together with the qualitative output, also a fuzzy membership value and a side value are forecast. Thus, fuzzy forecasting predicts an entire qualitative triple, from which a quantitative variable can be regenerated whenever needed.

3.4 Defuzzification

Once the qualitative output episode has been determined, a quantitative trajectory can easily be constructed by the reverse operation of fuzzy recoding. The class, membership, and side values are simply recombined to produce a real-valued signal.

4 FIR Software

Originally, the FIR algorithms had been coded in Fortran and were made available as a CTRL-C library [6]. Mixed quantitative and qualitative simulations were performed in ACSL, which could invoke the Fortran routines of the *Recode*, *Forecast*, and *Regenerate* subroutines directly, i.e., the qualitative models were constructed in CTRL-C, but mixed simulations were run in ACSL [5].

When CTRL-C died, the algorithms were recoded in C, and CTRL-C was replaced by Matlab as the interactive matrix manipulation environment.

There are currently two separate Matlab toolboxes available implementing the FIR algorithms. SAPS-II [5] offers a command-driven interface. The user invokes the four blocks of the FIR methodology by writing m-files. Visual-FIR [9] offers a menu-driven interface. Here, the user doesn't write any code, but selects combinations of algorithms from a set of pull-down menus. SAPS-II is more general, but Visual-FIR is easier and faster to use.

Both toolboxes can be used to run purely qualitative simulations directly under Matlab. Yet, mixed quantitative and qualitative simulations cannot be run in this fashion. Thus when ACSL died, we lost our ability to run mixed simulations.

This is where the new FIRlib fits in. The software allows us to once again run mixed quantitative and qualitative simulations, replacing ACSL by Modelica.

Just like the former ACSL implementation, FIRlib currently offers *Recode*, *Forecast*, and *Regenerate* modules only. There is no need to offer an *Optmask* module in the software, as the qualitative model is being generated off-line. Hence also with FIRlib, the qualitative models are being created using either SAPS-II or Visual-FIR. Future versions of FIRlib may offer an *Optmask* module also for convenience.

FIRlib offers currently two implementations of the FIR algorithms. In one of them (native SAPS), the formerly C-coded algorithms were translated into Modelica. The other (external SAPS) invokes C-coded routines.

For small examples, there is little difference between the two versions. Yet, the cardiovascular system model will not run efficiently in Dymola using the native SAPS modules. The FIR algorithms operate on very large data tables that Dymola converts to individual variables. Hence the cardiovascular system model, when using native SAPS, generates 200,000 scalar variables, whereas only 4000 variables are generated when the external SAPS modules are invoked.

5 A Simple Textbook Example

We shall demonstrate the use of FIRlib by means of a simple position control system involving a hydraulic motor with a servo-valve. The control system is shown in Fig.4.

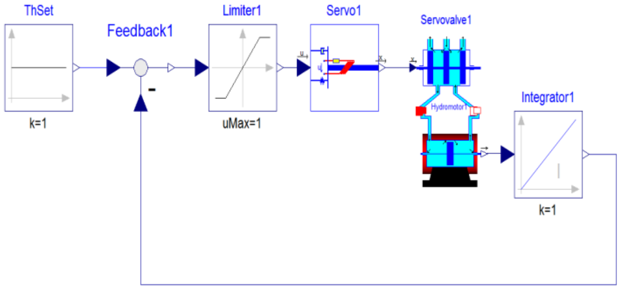


Fig.4: A position control system

The servo-valve and the hydraulic motor models were composed using the hydraulic sub-library of BondLib [3]. The hydraulic motor model is shown in Fig.5.

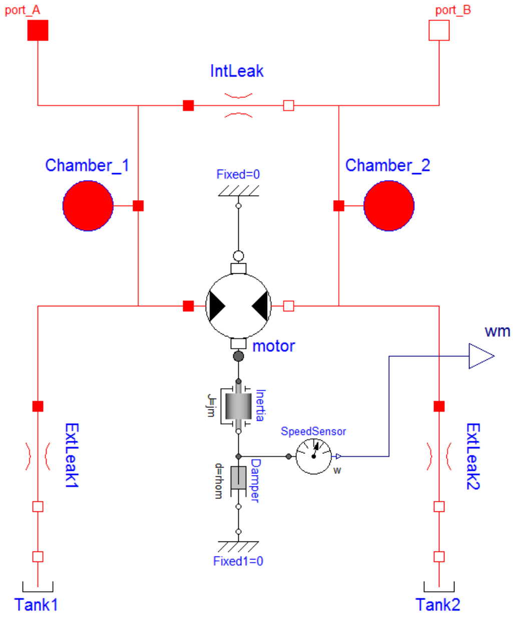


Fig.5: Hydraulic motor

We want to replace the entire hydraulic part of the model by a qualitative model. We assume that the mechanical torque, τ, of the hydraulic motor is a function of the actuator signal, u, and the angular velocity of the motor, ω.

$$\tau = f(u, \omega)$$

Therefore, we need to recode (fuzzify) these three signals. This is done in Modelica using FIRlib, as shown in Fig.6.

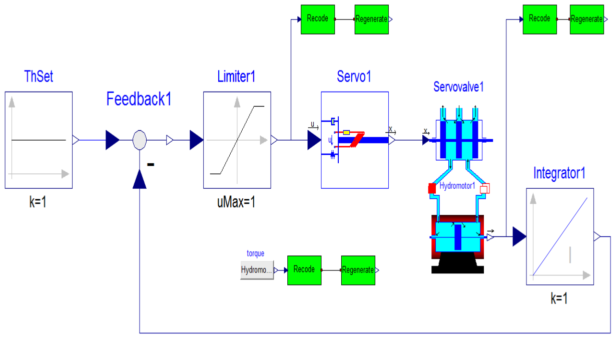


Fig.6: Fuzzification of three signals

The *Recode* block converts a real-valued signal to a qualitative triple. The FIR connector, at the output of the *Recode* block, contains three signals representing the class, membership, and side values of the fuzzified signal. The fuzzified signals were immediately defuzzified (regenerated) again so that Dymola can then be used to plot the signal and verify that fuzzification/defuzzification were done correctly. This is shown in Fig.7.

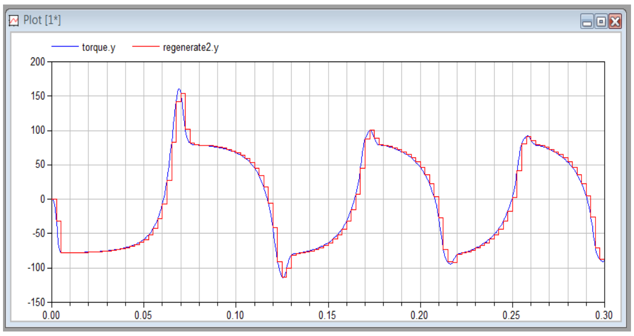


Fig.7: Torque signal, original and regenerated

The three recoded signals were then exported to Matlab. In Matlab, the raw data matrix (or rather, the three matrices containing the class, membership, and side values) was constructed, and the SAPS-II toolbox was used to generate the optimal mask and, from it, the input/output matrix and the behavior matrix.

The optimal mask and the corresponding behavior matrix (actually three matrices) were then reimported into Dymola to be used in a mixed quantitative and qualitative simulation. The mixed model is shown in Fig.8.

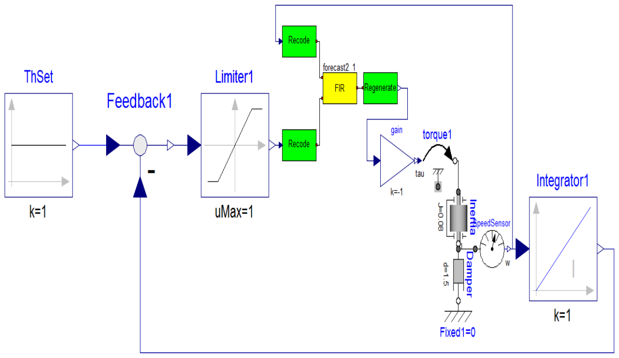


Fig.8: Mixed quantitative and qualitative model

The yellow *FIR* block represents the qualitative simulation (forecasting) engine. It takes the recoded (fuzzified) actuator and angular velocity signals and, using table look-up and interpolation in the behavior matrix, estimates the correct value of the torque in the form of a qualitative signal. The green Regenerate block then converts the qualitative triple back to a real-valued quantitative signal that can be used by regular Modelica models.

Fig.9 shows the motor angle trajectories of the original purely quantitative simulation and the mixed quantitative and qualitative simulation.

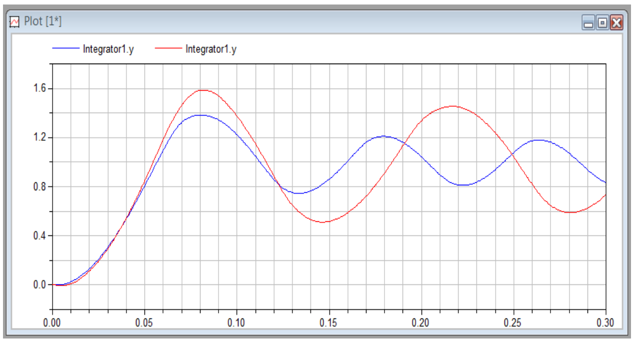


Fig.9: Motor angle trajectories

The (red) motor angle trajectory computed by the mixed simulation is a little more sluggish and a little less stable than that of the purely quantitative simulation (blue), because we didn't sample fast enough.

6 The Cardiovascular System

The human cardiovascular system is composed of two parts.

The hemodynamics describe the blood flow through the heart and the blood vessels. This part is well understood. It functions like any other hydraulic system with a pump and some pipes, with valves and containers of liquid. The hemodynamics can thus be well described by differential equations, and consequently, a quantitative model of the hemodynamics is adequate.

On the other hand, we need to describe also the control signals that operate on the hemodynamics. Control signals determine how fast the heart beats, how much the chambers contract, etc. The functioning of these nervous control signals is less well understood, and consequently, a qualitative model of the central nervous system control of the cardiovascular system may be more suitable.

6.1 Hemodynamics

The hemodynamics model has been presented at a previous conference [4]. It is built in BondLib using encapsulated bond graphs [3,7]. Although the bond graphs themselves are only seen at the bottom layer of the hierarchy, whereas all higher layers are built using symbols that medical professionals understand, the connectors are bond graph connectors everywhere. In order for this to work, all container models end in junctions, whereas all transporter models end in bonds. In this way, by following the rule that container and transporter models must always toggle, there is no need to fully wrap [7] the bond graph models, as was done in the hydraulic sub-library.

The heart model is shown in Fig.10.

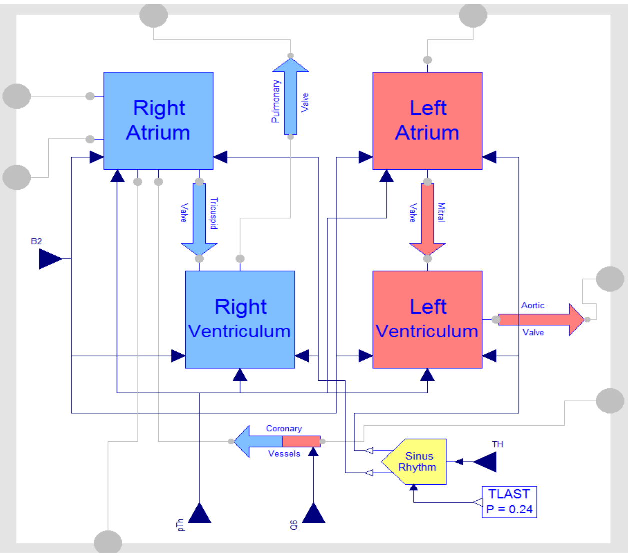


Fig.10: Model of the human heart

The model contains four container models representing the four heart chambers, as well as five transporter models. Four of them represent the four heart valves, the tricuspid and pulmonary valves carrying (blue) venous blood, and the mitral and aortic valves carrying (red) arterial blood. Also included is a model of the coronary blood vessels that are responsible for the oxygenation of the heart muscle. The yellow sinus rhythm block calculates the trigger impulses that lead to the contraction of the four heart chambers. It is controlled by the heart rate controller, one of the central nervous system control functions of the heart. The left chambers are shown on the right side of the graph, because this is what a heart surgeon experiences when he or she operates on a patient.

The heart is embedded in the thorax, shown in Fig.11.

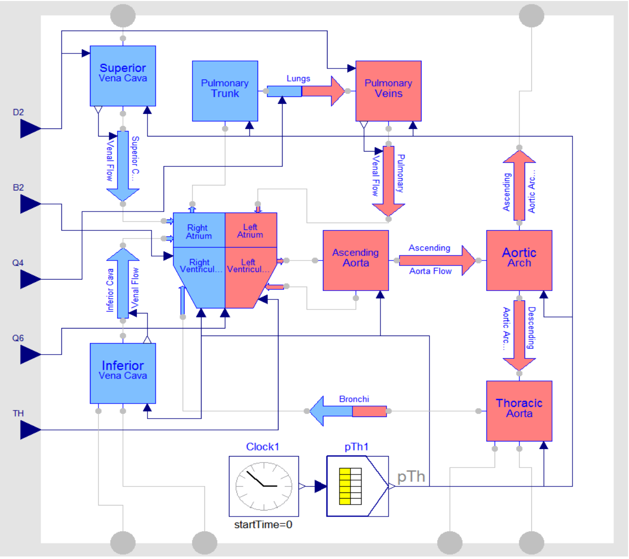


Fig. 11: Model of the thorax

The thorax model contains the heart and all of its external connections. Also included are models of the lungs and the bronchi. The tabular block at the bottom calculates the thoracic pressure that stems from the breathing. As the lungs expand, there is less space available for the heart and the blood vessels, and consequently, they experience an external pressure.

The overall hemodynamics model is shown in Fig.12.

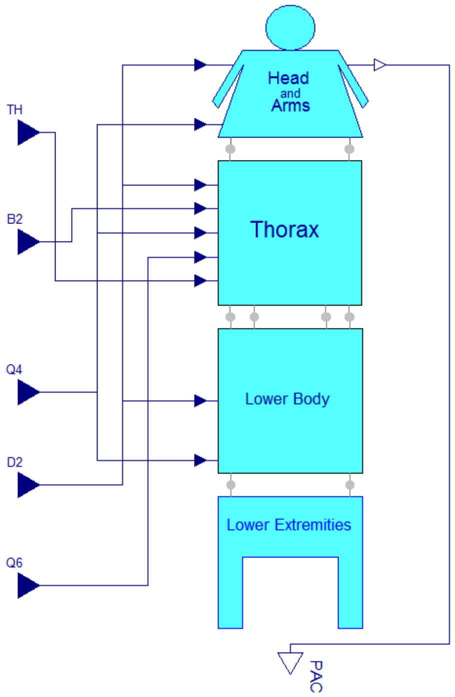


Fig.11: Model of the hemodynamics

The hemodynamics model contains models of the thorax, the head and arms (brain and blood vessels of the upper extremities), the lower body (abdomen and stomach), as well as the blood vessels of the legs.

The hemodynamics are controlled by five control signals denoting the heart rate, the myocardiac contractility, the peripheric resistance, the venous tone, and the coronary resistance.

It is assumed that all five control signals are functions of the same variable, namely the carotid sinus blood pressure, *PAC*, i.e., the arterial pressure in the brain.

6.2 Central Nervous System Control

Five separate single-input/single-output (SISO) qualitative FIR models are to be identified that each calculate one of the five control signals as a function of the carotid sinus blood pressure.

The data needed for the identification of the five FIR models are here not collected from a fully quantitative simulation (as in the previous case), but rather, they are obtained through measurements from real patients having a heart catheter for some reason or other (invasive procedure). The patients gave their consent to perform a number of so-called Valsalvæ maneuvers [10,11,12], a breathing test that excites the entire cardiovascular system. Data were collected from 10 different patients, each performing five Valsalvæ maneuvers. In the experiments described here, we used the data of one patient only. Four of the five Valsalvæ maneuvers (4800 data records) were used to identify the five controller models, and the final 1200 data records (the final maneuver) were used for model validation.

Fig.12 shows the recorded data of the venous tone controller signal of one patient during one Valsalvæ maneuver.

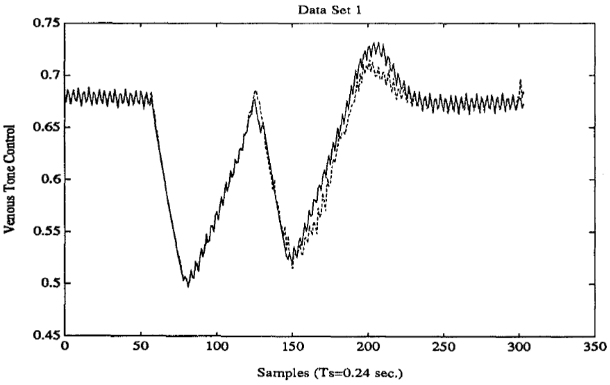


Fig.12: Venous tone controller signal

The large and low-frequency oscillation is caused by the breathing pattern of the Valsalvæ maneuver, whereas the superposed small and high-frequency oscillation is caused by the beating of the heart.

The Valsalvæ maneuver shown is the one that was not used for model identification. Superposed with the measurement data is the forecast obtained by the qualitative FIR model.

The five models were identified using the SAPS-II toolbox, and also the simulation was performed using the same Matlab toolbox. As this is a purely qualitative simulation, there was no need to perform the simulation in Modelica using FIRlib.

6.3 Mixed Quantitative and Qualitative Simulation of the Human Cardiovascular System

The top-level model is shown in Fig.13. The pink box on the right-hand side represents the hemodynamics model, whereas the green box on the left-hand side represents the central nervous system control containing the five (yellow) qualitative FIR models representing the five controllers.

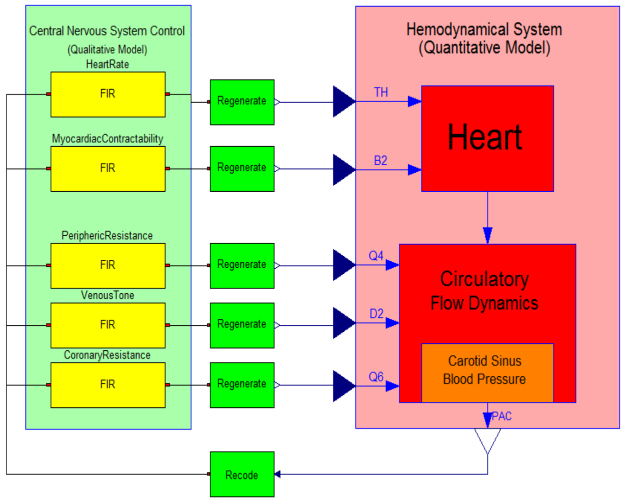
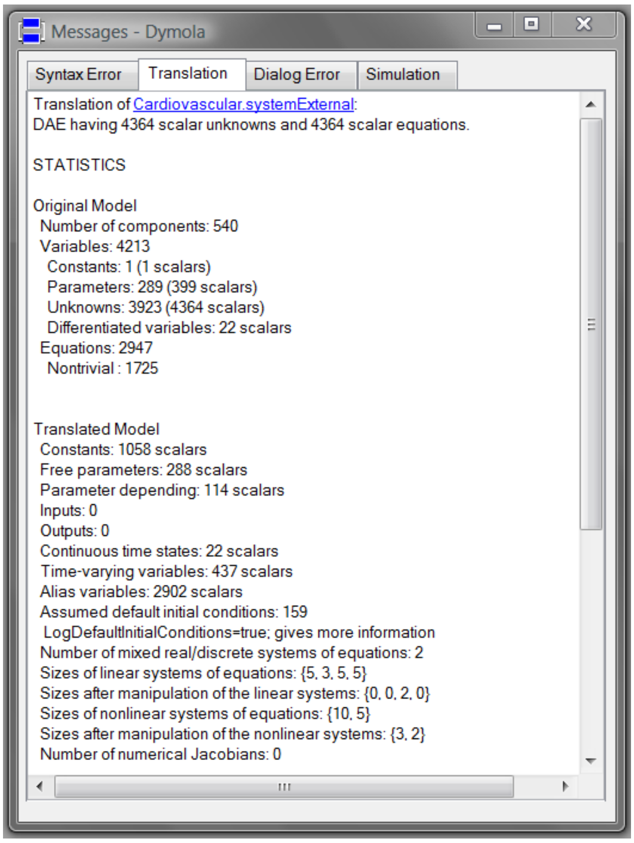


Fig.13: Cardiovascular system model

The green (Recode, Regenerate) and yellow (FIR) blocks are those of the external SAPS sub-library, i.e., the C-coded FIR algorithms are being used.

The model was then compiled. The translation log is shown in Fig.14.

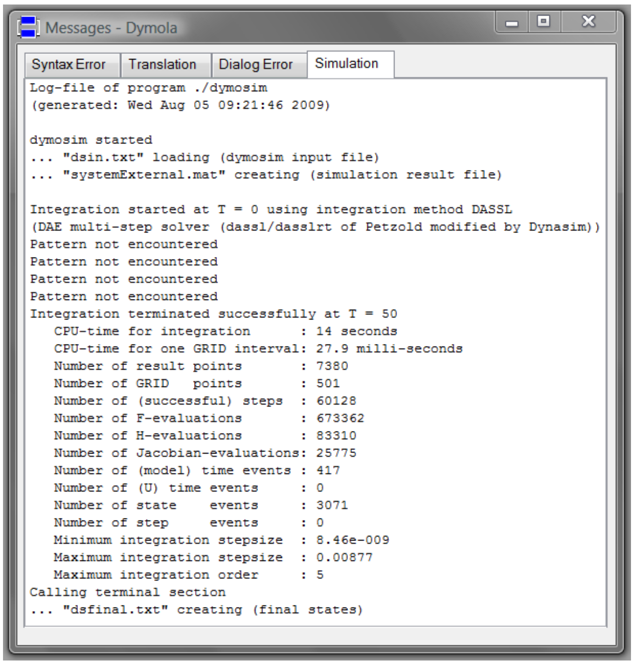


```
Translation of Cardiovascular.systemExternal:
DAE having 4364 scalar unknowns and 4364 scalar equations.

STATISTICS

Original Model
 Number of components: 540
 Variables: 4213
   Constants: 1 (1 scalars)
   Parameters: 289 (399 scalars)
   Unknowns: 3923 (4364 scalars)
   Differentiated variables: 22 scalars
 Equations: 2947
   Nontrivial : 1725


Translated Model
 Constants: 1058 scalars
 Free parameters: 288 scalars
 Parameter depending: 114 scalars
 Inputs: 0
 Outputs: 0
 Continuous time states: 22 scalars
 Time-varying variables: 437 scalars
 Alias variables: 2902 scalars
 Assumed default initial conditions: 159
  LogDefaultInitialConditions=true; gives more information
 Number of mixed real/discrete systems of equations: 2
 Sizes of linear systems of equations: {5, 3, 5, 5}
 Sizes after manipulation of the linear systems: {0, 0, 2, 0}
 Sizes of nonlinear systems of equations: {10, 5}
 Sizes after manipulation of the nonlinear systems: {3, 2}
 Number of numerical Jacobians: 0
```

Fig.14: Translation log

The flattened model contained originally 4364 scalar variables and equations. After code optimization, 22 state variables and 437 algebraic variables remained.

Fig.15 shows the simulation log.



```
Log-file of program ./dymosim
(generated: Wed Aug 05 09:21:46 2009)

dymosim started
... "dsin.txt" loading (dymosim input file)
... "systemExternal.mat" creating (simulation result file)

Integration started at T = 0 using integration method DASSL
(DAE multi-step solver (dassl/dasslrt of Petzold modified by Dynasim))
Pattern not encountered
Pattern not encountered
Pattern not encountered
Pattern not encountered
Integration terminated successfully at T = 50
   CPU-time for integration      : 14 seconds
   CPU-time for one GRID interval: 27.9 milli-seconds
   Number of result points       : 7380
   Number of GRID   points       : 501
   Number of (successful) steps  : 60128
   Number of F-evaluations       : 673362
   Number of H-evaluations       : 83310
   Number of Jacobian-evaluations: 25775
   Number of (model) time events : 417
   Number of (U) time events     : 0
   Number of state    events     : 3071
   Number of step     events     : 0
   Minimum integration stepsize  : 8.46e-009
   Maximum integration stepsize  : 0.00877
   Maximum integration order     : 5
Calling terminal section
... "dsfinal.txt" creating (final states)
```

Fig.15: Simulation log

The simulation took 14.0 seconds of real time to complete 50 seconds of simulated time. Four times during the simulation, one of the controllers encountered a pattern that had not been recorded in the training database. In those cases, no prediction was possible, and therefore, the software simply retained the previous prediction value.

Fig.16 shows the thoracic pressure, *pTh*, generated by a table look-up function inside the thorax model.

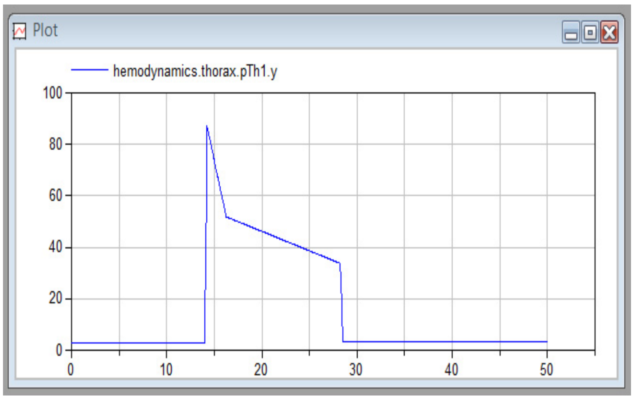


Fig.16: Thoracic pressure

The graph shows the simulated "Valsalvæ" maneuver. The "patient" is not breathing during 14 seconds, then "he" inhales sharply, holds "his" breath more or less for another 14 seconds, then exhales sharply again.

The resulting carotid sinus pressure, *PAC*, as calculated by the hemodynamics model, is shown in Fig.17.

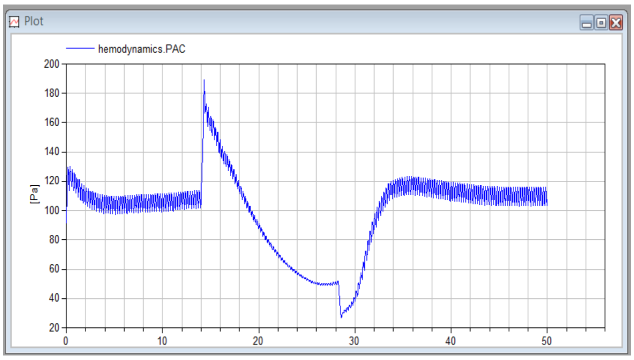


Fig.17: Carotid sinus pressure

The high-frequency oscillation is caused by the heartbeat, calculated in the sinus rhythm box of the heart model. The low-frequency oscillation is the hemodynamic response to the simulated breathing pattern.

Fig.18 shows the venous tone control signal as calculated by the corresponding FIR model in response to the simulated breathing pattern.

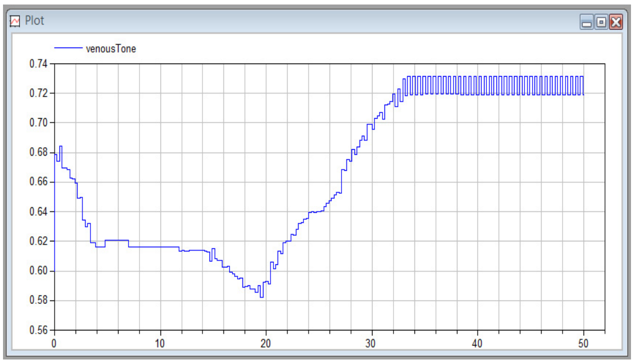


Fig.18: Venous tone control signal

7 Conclusions

In this paper, we have demonstrated how mixed qualitative and quantitative models can be simulated in Modelica using the new FIRlib library. The qualitative models make use of fuzzy inductive reasoning, a non-parametric inductive approach to modeling continuous-time systems by means of fuzzy logic. The approach was demonstrated by a small textbook example involving a hydraulic position control system. A model of the human cardiovascular system served as a larger example. In that model, the hemodynamics were described using quantitative models derived from first principle, whereas the nervous central system control functions were modeled by use of qualitative FIR models.

References

[1] Babbie, E.: *The Practice of Social Research*, 5th Edition, Wadsworth Publishing Company, Belmont, CA, 1989

[2] Cellier, F.E.: *Continuous System Modeling*. Springer-Verlag, New York, 1991

[3] Cellier, F.E. and Nebot, A.: The Modelica Bond Graph Library, In: *Proc. 4th International Modelica Conference*, Hamburg, Germany (2005) Vol.1, 57-65

[4] Cellier, F.E. and Nebot, A.: Object-oriented Modeling in the Service of Medicine, In: *Proc. 6th Asia Simulation Conference*, Beijing, China (2005) 33-40

[5] Cellier, F.E., Nebot, A., Mugica, F., and de Albornoz, A.: Combined Qualitative/Quantitative Simulation Models of Continuous-time Processes Using Fuzzy Inductive Reasoning Techniques. In: *International Journal of General Systems* (1996) Vol. 24(1-2), 95-116

[6] Cellier, F.E. and Yandell, D.W.: SAPS-II: A New Implementation of the Systems Approach Problem Solver, In: *Intl. J. General Systems* (1987) Vol. 13(4), 307-322

[7] Cellier, F.E. and Zimmer, D.: Wrapping Multi-bond Graphs: A Structured Approach to Modeling Complex Multi-body Dynamics, In: *Proc. 20th European Conference on Modeling and Simulation*, Bonn, Germany (2006) 7-13

[8] Escobet, A., Huber, R.M., Nebot, A., and Cellier F.E.: Enhanced Equal Frequency Partition Method for the Identification of a Water Demand System, In: *Proc. AI, Simulation and Planning in High Autonomy Systems*, Tucson, Arizona (2000) 209-215.

[9] Escobet, A., Nebot, A., and Cellier, F.E.: Visual-FIR: A Tool for Model Identification and Prediction of Dynamical Complex Systems, In: *Simulation Modeling Practices and Theory* (2008) Vol. 16(1), 76-92

[10] Nebot, A.: *Qualitative Modeling and Simulation of Biomedical Systems Using Fuzzy Inductive Reasoning*, Ph.D. Dissertation, Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Barcelona, Spain, 1994

[11] Nebot, A., Cellier, F.E., and Vallverdú, M.: Mixed Quantitative/Qualitative Modeling and Simulation of the Cardiovascular System, In: *Computer Methods and Programs in Biomedicine* (1998) Vol. 55(2), 127-155

[12] Vallverdú, M.: *Modelado y Simulación del Sistema de Control Cardiovascular en Pacientes con Lesiones Coronarias*, Ph.D. Dissertation, Institut de Cibernètica, Universitat Politècnica de Catalunya, Barcelona, Spain, 1993

Author Biographies

François E. Cellier received his BS degree in electrical engineering in 1972, his MS degree in automatic control in 1973, and his PhD degree in technical sciences in 1979, all from the Swiss Federal Institute of Technology (ETH) Zurich. Dr. Cellier worked at the University of Arizona as professor of Electrical and Computer Engineering from 1984 until 2005. He then returned to his home country of Switzerland. Dr. Cellier's main scientific interests concern modeling and simulation methodologies, and the design of advanced software systems for simulation, computer aided modeling, and computer-aided design. Dr. Cellier has authored or co-authored more than 200 technical publications, and he has edited several books. He published a textbook on Continuous System Modeling in 1991 and a second textbook on Continuous System Simulation in 2006, both with Springer-Verlag, New York.

Victorino Sanz received his MS degree in computer science in 2004 from the Universidad Politécnica de Madrid. He is currently a Ph.D. student in systems engineering and automatic control at the Universidad Nacional de Educación a Distancia (UNED) in Madrid. His research focuses on the development of several Modelica libraries for discrete-event system and hybrid system modeling and simulation.

Parallel DEVS and Process-Oriented Modeling in Modelica

Victorino Sanz Alfonso Urquia Sebastian Dormido
Dpto. Informática y Automática, ETSI Informática, UNED
Juan del Rosal 16, 28040, Madrid, Spain
{vsanz,aurquia,sdormido}@dia.uned.es

Abstract

This manuscript presents a new free Modelica library, named DESLib and composed of four packages: RandomLib, DEVSLib, SIMANLib and ARENALib. DESLib has been designed and implemented to facilitate the description of discrete-event models using the Parallel DEVS formalism (using DEVSLib), and to facilitate the process-oriented modeling of logistic systems (using SIMANLib and ARENALib). SIMANLib and ARENALib models are designed as DEVS models, and implemented using DEVSLib, to facilitate its development, comprehension and maintenance. RandomLib includes functionalities to generate random numbers and random variates, and facilitate the development of stochastic models. The communication mechanism used to transport information between models in DESLib is presented. This mechanism facilitates the combination of DEVS and process-oriented models to describe discrete-event systems at multiple levels. DESLib also includes interfaces to combine its components with other Modelica libraries, facilitating the composition of multi-formalism and multi-domain hybrid models. DESLib can be downloaded from `http://www.euclides.dia.uned.es`.

Keywords: discrete-event systems, hybrid modeling, Parallel DEVS, process-oriented modeling, random number generation, stochastic simulation, logistic model

1 Introduction

Modelica provides language constructs to describe the trigger conditions of time and state events, and also the actions associated to the events [1]: (1) update the value of discrete-time variables and reinitialize continuous-time state variables, using *when* clauses; and (2) change the mathematical description of equations and assignments, using the *if* statement.

These features have facilitated the development of state machine models [2, 3] and also of libraries supporting different formalisms for discrete-event system modeling. Some of these libraries are StateCharts [4], StateGraph [5], HyAuLib [6], PetriNets [7] and ExtendedPetriNets [8]. Other approach is described in [9], in which the discrete-event system is described using an external tool that generates the corresponding Modelica code. The use of Modelica language to describe discrete-event models of communication networks is presented in [10].

1.1 Parallel DEVS Formalism

The support of the Modelica language to the DEVS (Discrete EVent System specification) formalism [11] is an open research area. The feasibility of constructing basic atomic and coupled DEVS models in Modelica was demonstrated in [12]. Another implementation of this formalism was performed in the ModelicaDEVS library [13], designed to simulate continuous-time systems using the Quantized State System (QSS) integration methods [14, 15].

An atomic model is the simplest component that can be defined using Parallel DEVS (PDEVS) [16], and can be formally described with the tuple $M = (X, S, Y, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$, where X is the set of input ports and values, Y is the set of output ports and values, S is the set of sequential states, δ_{ext}, δ_{int} and δ_{con} are the transition functions, λ is the output function and ta is the time advance function. An atomic model updates its state with δ_{int} every time a given amount of time (defined by ta) is elapsed without any external input, thus triggering an internal event. An output can be generated, using the λ function, before executing δ_{int}. Inputs are stored in a *bag*, which is a set with possible multiple occurrences of its elements. When any input is received, the external event is triggered and the state is updated by δ_{ext}, that manages the elements in the bag. The simultaneous occurrences of an external and an internal event trigger a confluent event, and the state is updated by δ_{con}.

DOI: 10.3384/ecp09430104

A coupled model can be described as a composition of other atomic or coupled models. It is specified in the PDEVS formalism with a tuple $M = (X,Y,D,\{M_d|d \in D\},EIC,EOC,IC)$, where X is the set of input ports and values, Y is the set of output ports and values, D is the set of component names, M_d is the set of DEVS components, EIC is the set of connections between input ports and components, EOC is the set of connections between components and output ports and IC is the set of internal connections between components.

1.2 Process-Oriented Modeling

According to the process-oriented "world view", systems are described from the point of view of the entities that flow through them using the available resources [17]. This modeling methodology, widely used for describing complex logistic systems, is supported by several modeling languages (e.g., GPSS/H, SLAM II, SIMSCRIPT II.5, SIMAN, SIMULA and SIMPL/1) and integrated environments (e.g., Arena, AutoMod, ProModel, Witness and SIMPROCESS).

Arena [18] is a widely used process-oriented simulation environment. It includes components to describe the flowchart diagram of the system, that represents the flow of entities. That diagram includes the processes and actions performed to the entities along the simulation run. Other components allow to describe the characteristics of those processes and actions (such as, the organization policy of the queues, the number of available resources, etc.). Components in Arena are internally described by the lower-level components of the SIMAN language [19].

Process-oriented modeling with Modelica is an attractive research field. An introduction of operations management modeling with Modelica is described in [20], where the authors present a case study of an inventory system and describe the problems encountered when modeling these kind of systems using Modelica. Almost no other work has been performed in this field.

2 The DESLib Modelica Library

The first objective of the research work presented in this manuscript is to facilitate the description of discrete-event models using the PDEVS formalism [16] and also facilitate the connection of these models to other hybrid models developed using other Modelica libraries.

The second objective of our work is to facilitate the process-oriented modeling of logistic systems using Modelica and also to facilitate the connection of these logistic system models to hybrid models developed using other Modelica libraries.

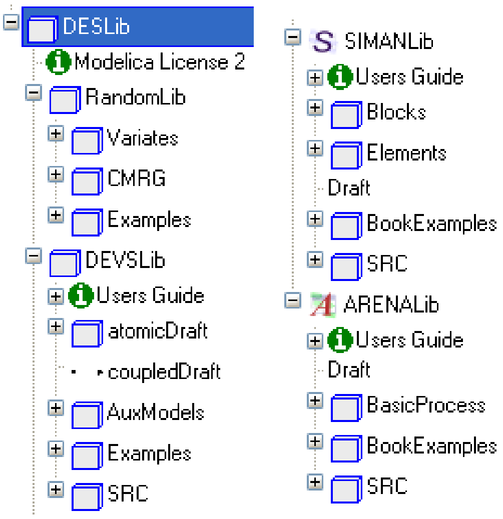


Figure 1: DESLib library.

In order to achieve these two objectives, a new Modelica library, named DESLib, has been designed and programmed using Dymola [21]. DESLib, which is freely available under the terms of the Modelica License 2, can be downloaded from `http://www.euclides.dia.uned.es/`.

DESLib is composed of the following four packages: RandomLib, DEVSLib, SIMANLib and ARENALib. The general architecture of the library is shown in Fig. 1.

- The RandomLib package includes an implementation of the CMRG random number generator, used by Arena, and some functionalities for random variates generation.
- The DEVSLib package facilitates the description of discrete-event models in Modelica following the Parallel DEVS formalism.
- The SIMANLib and ARENALib packages facilitate the description of process-oriented models. The components in these packages have been designed as atomic and coupled PDEVS models, and implemented using DEVSLib.

DEVSLib, SIMANLib and ARENALib use the same communication mechanism to transport information between models. This makes all their components compatible and can be combined to develop models at multiple description levels. The implemented communication mechanism is detailed in Section 4.1.

The organization of the manuscript is as follows. A description of the RandomLib package is included in Section 3. The architecture and design of the DEVS-

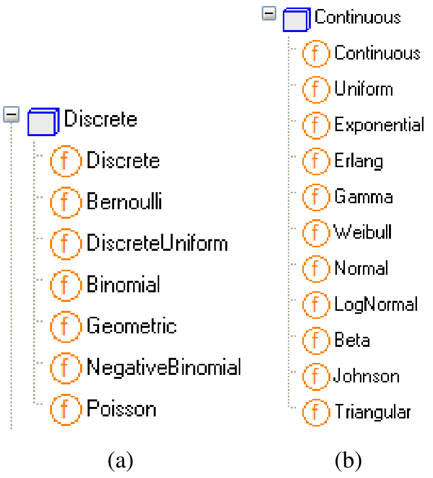
Lib package, together with its functionalities to describe Parallel DEVS models, are described in Section 4. The components and functionalities of the SIMANLib and ARENALib packages are described in Section 5. Two case studies are discussed in Section 6: the model of a restaurant constructed using SIMANLib and the model of an electronic factory constructed using ARENALib. Finally, some conclusions are given in Section 7.

3 RandomLib

Process-oriented models are usually stochastic [22]. The RandomLib package is used in conjunction with DEVSLib, SIMANLib and ARENALib to model discrete-event and process-oriented stochastic models of logistic systems.

RandomLib contains a Modelica implementation of the Combined Multiple Recursive Generator (CMRG) which is used in Arena [23, 24]. The implemented random number generator (RNG) gives the possibility of creating multiple random streams, and sub-streams, that can be considered as independent RNGs [24]. The generator period is close to 2^{191}, and can be divided into disjoint streams of length 2^{127}. At the same time, each stream can also be divided into 2^{51} adjacent sub-streams, each of length 2^{76}.

RandomLib is composed of three packages (see Fig. 1): CMRG, Variates and Examples. The CMRG package includes the implementation of the CMRG uniform random number generator. Although available in C, this generator has been implemented in Modelica in order to facilitate its use, comprehension and reutilization in other Modelica libraries.



(a) (b)

Figure 2: Discrete and continuous probability distribution functions included in RandomLib.

The Variates package includes several functions to generate random variates from continuous and discrete probability distributions. The included probability distribution functions are shown in Fig. 2. These functions use by default the CMRG generator as source of uniform random numbers. However, any other Modelica library for uniform random number generation can be used, redeclaring the record that represents the generic generator, its initialization function and the generic uniform random number generation function.

The Examples package contains several examples of random uniform and random variates generation in order to facilitate the use of the library. One of the included examples is shown in Listing. 1

```
model VariatesSimple
  "generates 5 random variates with
   Expo(5) distribution"
  Variates.Generator g "RNG";
  Real u[5] "vector of random variates";
algorithm
  // initialization of the RngStream
  when initial() then
    g := Variates.initGenerator();
  end when;
  when time <= 0 then
    for i in 1:5 loop
      // generation of variates.
      (u[i],g) :=
      Variates.Continuous.Exponential(g,5);
    end for;
  end when;
end VariatesSimple;
```

Listing 1: Random variates generation using RandomLib.

4 DEVSLib

The DEVSLib package, as shown in Fig. 1, contains the following packages and models: the UsersGuide that contains the documentation about the structure and use of DEVSLib; the atomicDraft and coupledDraft models that are used to construct new DEVS models; the AuxModels package that includes several useful auxiliary models; the Examples package that contains several examples of systems modeled using DEVSLib and; the SRC package that includes all its internal implementation and documentation.

DEVSLib supports the definition of models using the PDEVS formalism. Atomic and coupled models can be constructed with DEVSLib following their DEVS formal specification, similarly to how it is performed by other DEVS tools, such as DEVSJAVA [25], CD++ [26], or adevs [27].

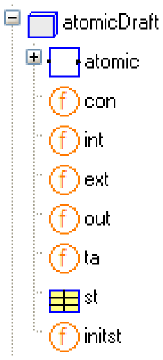


Figure 3: DEVSLib atomic model structure.

The structure of an atomic model in DEVSLib is shown in Fig. 3. The transition, output and time advance functions (*con*, *int*, *ext*, *out* and *ta* in Fig. 3) are described as Modelica functions. The state is described using a Modelica record (*st*), and initialized using the *initst* function. Any variable required to describe the state of the model can be added to that record. An example of new atomic model construction using DEVSLib is given in Section 4.3.

The development of coupled models with DEVSLib also follows its formal specification. Using the object-oriented modeling capabilities of Modelica, coupled models are constructed connecting previously developed components and including the required input/output ports.

The interconnection of the components in a coupled model may include algebraic loops. Due to the characteristics of the Modelica language, these algebraic loops are not allowed. This problem can be avoided by redefining the behavior of the models of the loop into one single atomic model, or breaking the loop inserting the *breakloop* model, included in DEVSLib, in any of its connections.

4.1 DEVSLib Model Communication

Model communication in PDEVS follows a message passing mechanism. The output function generates a new message and sends it through an output port. The message will be received, triggering an external event, by the models connected to that output port. The message may contain any kind of information, called the “value” of the message.

The model communication mechanism in Modelica is based in the definition of ports, called “connectors”, and connections between ports, using “connect-equations”. Variables defined in two connected connectors are either equaled, or summed up and the sum equaled to zero.

The Modelica model communication and the DEVS message passing mechanisms are conceptually different. The former equals values of variables while the latter transports information between models.

In order to allow the description of DEVS models in Modelica, a message passing mechanism has to be developed [28]. The development of this mechanism has been the most challenging problem solved during the development of the DEVSLib package. Several approaches were studied and developed, in order to implement the message passing mechanism in Modelica.

The direct implementation of the message passing mechanism in DEVSLib using Modelica connectors was studied. The mentioned DEVS implementations in Modelica [12, 13] use boolean variables inside the connectors to detect external events – i.e. received messages. However, the connectors does not allow the simultaneous reception of messages, because their variables can not be assigned with several values at the same time. Also, Modelica does not allow a variable number of objects in a model, so the message transmission can not be directly implemented.

Other approaches for implementing the message passing mechanism in DEVSLib, based in an intermediate storage for the transmitted messages, were studied and implemented. The first approach was to use a text file to store the messages, so the sender writes the message to the file and notifies it to the receiver, that reads it. This approach allows simultaneous reception of messages, because several messages can be written to the file, but its performance and versatility are poor. The other approach substitutes the text files by dynamic memory space. This increases the performance and the versatility of the mechanism, allowing to manage different types of messages without redefining the message management operations.

The dynamic memory approach for message passing is the mechanism implemented in DEVSLib. This approach is combined with the standard Modelica connectors to provide a transparent communication mechanism to the user. At the end, DEVSLib models are connected using standard Modelica connectors and connect-equations.

4.2 Interface Models with Other Modelica Libraries

DEVSLib includes several interface models to translate messages into continuous-time signals, and viceversa. The use of these interface models allows to combine models developed using DEVSLib with the components of other Modelica libraries.

There are two mechanisms used in the continuous-to-discrete translation: the cross-functions and the quantization. The former translates the value of a continuous-time signal into a message every time the signal crosses a given threshold, in one direction (upwards or downwards). The models "crossUP" and "crossDOWN" implement this behavior in DEVSLib. The quantization mechanism is implemented by the "quantizer" model. This model generates a message every time the value of the continuous-time signal changes in a predefined quantum, similarly to the behavior of the QSS first-order method [15].

On the other hand, the discrete-to-continuous translation is performed generating a piecewise-constant signal whose value is the value of the last message received. The model "DICO" (DIscrete-to-COntinuous) implements this behavior in DEVSLib.

These interface models are implemented to manage the standard DEVSLib message type. However, the message type in DEVSLib can be redefined by the user. In this case, the interface models can be adapted to the new message type.

4.3 Model Development with DEVSLib

The development of new models using DEVSLib requires the following steps:

- Declare the input and output ports of the model.
- Define the state variables of the model and their initialization.
- Define the transition, output and time advance functions.

A "bank teller" system is described in this section as an example of model construction. In this system the customers arrive to the bank and wait their turn in the queue. If the teller is idle, the customer is served immediately. Otherwise, the teller will serve the first customer in the queue. When finished, the customer leaves the bank and the teller serves another customer if anyone else is waiting, or waits for a new arrival.

The model of this system is composed of two atomic models: the *customers* and the *teller*. The *customers* model represents the arrivals of new customers. The inter-arrival time follows an exponential probability distribution with mean 10 mins. The *teller* model represents the person serving customers and the queue. The time spent by a customer with the teller follows an exponential probability distribution with mean 8 mins.

The Parallel DEVS specification of the *customers* model is the following:

$$M = (X_M, S, Y_M, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$$

where:

$X_M = \emptyset$

$S = \Re_0^+$

$Y_M = \{"out", \{1\}\}$

$\delta_{int}(sigma) = interArrivalTime$

$\delta_{ext}() =$ nothing since $X_M = \emptyset$

$\delta_{con}() =$ nothing since $X_M = \emptyset$

```
model customers "Customers arrival"
  replaceable Real interarrival = 1;
  extends AtomicDEVS(numIn=1,numOut=1,
    redeclare record State = st);
  redeclare function Fint =
    int(iat=interarrival);
  redeclare function Fout = out;
  redeclare function Fta = ta;
  redeclare function initState = initst;
  Interfaces.outPort outPort1;
equation
  iEvent[1] = 0;
  // OUTPUT PORTS
  oEvent[1] = outPort1.event;
  oQueue[1] = outPort1.queue;
end customers;

record st "state of the model"
  Real sigma;
end st;

function initst "state initialization func."
  output st out;
algorithm
  // first internal transition at time = 1
  out.sigma := 1;
end initst;

function int "Internal Transition Function"
  input st s;
  input Real iat //inter-arrival time;
  output st sout;
algorithm
  sout := s;
  sout.sigma := iat;
end int;

function out "Output Function"
  input st s;
  input Integer queue[nports];
  input Integer nports;
  output Integer port[nports];
protected
  stdEvent y;
algorithm
  y.Value := 1;
  // send output event with message y
  sendEvent(queue[1],y);
  port[1] := 1;
end out;

function ta "Time Advance Function"
  input st s;
  output Real sigma;
algorithm
  sigma := s.sigma;
end ta;
```

Listing 2: DEVSLib code for the customers model of the Bank Teller system.

$$\lambda(sigma) = 1$$
$$ta(sigma) = sigma$$

The *interArrivalTime* is a continuous-time input of the model that represents the time between customer arrivals, similarly to the continuous-time inputs described in the DEV&DESS formalism [11].

The implementation of the *customers* model using DEVSLib is shown in Listing 2. The code has been adapted from the atomicDraft model, and follows its structure (see Fig. 3). The input port has been removed, and the iEvent array is set to 0 because the model will never receive a message. The state record contains a variable that represents the interval for the next internal transition (sigma). The model will execute its first internal transition at time 1, due to the initialization of sigma. The external and confluent transition functions (*ext* and *con*) have been removed because the model has no input ports. The internal transition function (*int*) sets the value of sigma with the input “iat”, which is a continuous-time input that represents the probability distribution for the inter-arrival times. The output function generates a new message, that represents a new customer, and sends it through the output port. The time advance function only returns the value of sigma, set by δ_{int}.

The Parallel DEVS specification of the *teller* model is the following:

$$M = (X_M, S, Y_M, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$$

where:

$$X_M = \{"in", \{1\}\}$$
$$S = \{"active", "passive"\} \times \Re_0^+ \times \mathbb{N}$$
$$Y_M = \{"out", \{1\}\}$$
$$\delta_{int}(phase, sigma, nqueue) = \begin{cases} ("active", PT, nqueue - 1) & if\ nqueue > 0 \\ ("passive", \infty, 0) & otherwise \end{cases}$$
$$\delta_{ext}(phase, sigma, nqueue, u, e, X) = \begin{cases} ("active", PT, nqueue) & if\ "passive" \\ (phase, sigma - e, nqueue + 1) & otherwise \end{cases}$$
$$\delta_{con}(phase, sigma, nqueue, u, e, X) = \delta_{ext}(\delta_{int}(phase, sigma, nqueue), u, 0, X)$$
$$\lambda(phase, sigma, nqueue) = 1$$
$$ta(phase, sigma, nqueue) = sigma$$

PT is a continuous-time input of the model that represents the service time for each customer.

The implementation of the *teller* model is similar to the *customers* model, and also follows the structure of the atomicDraft model. The *teller* has one input and one output ports. Its state record includes the operational mode of the teller (phase, initialized to 1 == “idle”), the interval for the next internal transition (sigma, initialized to infinity) and the queue (nqueue, initialized to 0 == “empty”). Since the arrival of a new customer does not include additional information (like its name, age, etc.), the queue only stores the number of customers waiting. At external events, when a new customer arrives it is either serviced (teller “idle” with phase == 1) or waits in queue (teller “busy” with phase == 2). The value of sigma is set with the service time (also received as a continuous-time input) or the rest of the service time of the customer being processed, if the teller is “idle” or “busy” respectively. At internal events, when the serviced customer leaves, the teller checks the value of the queue. If any other customer is in the queue, the teller starts its service and sets sigma to the new service time. If no customers are waiting, the teller becomes “idle” and waits for a new arrival setting the sigma to infinity. The output function generates a new message that represents the customer leaving the bank.

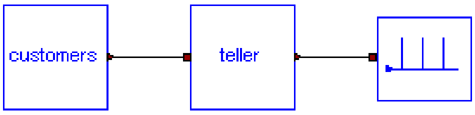


Figure 4: Bank teller system modeled using DEVSLib.

The system constructed using DEVSLib is shown in Fig. 4. It includes the code for generating the random inter-arrival and service times, using RandomLib. The model connected to the output of the *teller* only shows the departure of the customers. The results after simulating the model during 20 time units are shown in Fig. 5, containing the arrival, the departure and the number of customers in queue over the simulation. The average number of customers in queue over a long simulation (10^6 time units) is shown in Fig. 6. That

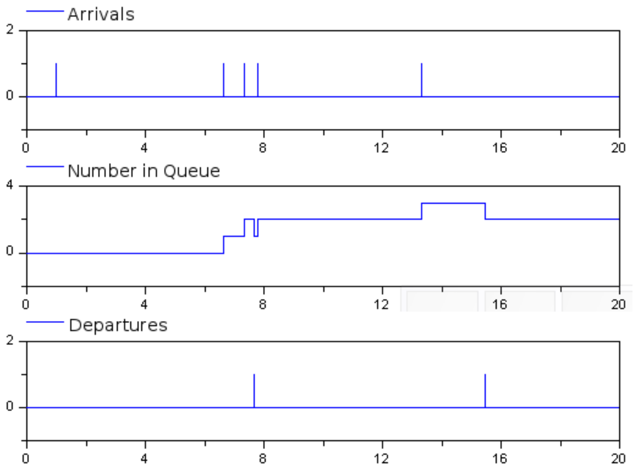


Figure 5: Bank teller system simulation results.

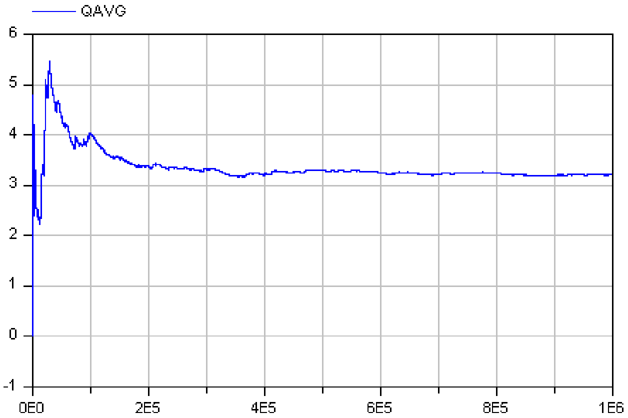


Figure 6: Average number of customers in queue from the simulation of the bank teller system.

result tends to the analytical result (an average of 3.2 jobs in queue) of the equivalent M/M/1 queue system.

5 SIMANLib and ARENALib

SIMANLib and ARENALib support the process-oriented modeling methodology, in a similar fashion to SIMAN and Arena, but with limited capabilities. Systems modeled using SIMANLib and ARENALib are composed of two parts: the flowchart diagram and the experimental data. The flowchart diagram describes the flow of entities through the system. It is defined by the *blocks* in SIMANLib and the *flowchart modules* in ARENALib. The experimental data describes the particular information of an experiment to be performed with the system. It corresponds to the amount of available resources, the characteristics of the queues, the statistical information to be recorded, etc. It is defined by the *elements* in SIMANLib and the *data modules* in ARENALib.

The structure of both packages (see Fig. 1) is similar, and are divided into two areas: the users area and the developers area. The users area in SIMANLib is composed of the Blocks (containing flowchart components), Elements (containing data components), Draft (used to construct new models) and BookExamples (containing the implementation of several examples described in [19]) packages. The users area in ARENALib is composed of the BasicProcess (containing both flowchart and data modules) and BookExamples (containing the implementation of several examples described in [18]) packages. Both, SIMANLib and ARENALib, contain a UsersGuide package that includes a description of their characteristics, structure and use. The developers area in both packages contains the SRC package, with the internal implementation of their components and the developers documentation.

The communication between flowchart components also follows a message passing mechanism, where the value of the messages are the entities. Entities are created by the Create blocks or modules, and sent to the next component. Each component performs a process (or action) to the received entity and sends it to the next component. Entities leave the system at Dispose blocks or modules.

5.1 SIMANLib Components

SIMANLib reproduces several elements of the SIMAN language [19]. The included components are shown in Fig. 7.

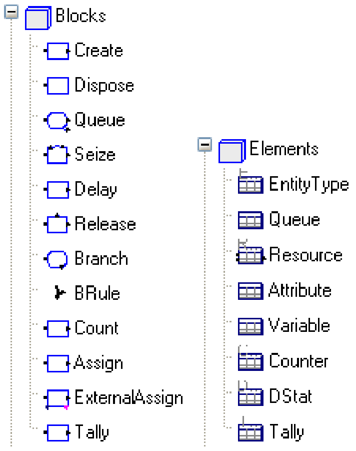


Figure 7: SIMANLib components: blocks and elements.

In order to facilitate the development and maintenance of the SIMANLib package, the SIMANLib blocks have been specified using the DEVS formalism [29] (i.e., as atomic PDEVS models) and have been implemented using the DEVSLib package. Also the Resource element has been modeled as an atomic PDEVS model, in order to manage the seize and release petitions to the resource.

5.2 ARENALib Components

ARENALib reproduces several elements of the Basic Process panel of the Arena Simulation environment [18]. The included components are shown in Fig. 8.

A previous implementation of the ARENALib package was presented in [30]. That implementation, which was directly coded in plain Modelica, was difficult to understand, maintain and modify. Arena com-

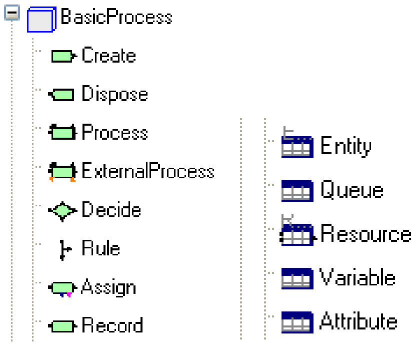


Figure 8: ARENALib components: flowchart and data modules.

ponents are developed using SIMAN constructs [18]. Analogously to SIMANLib, the components of ARENALib have been reconstructed as coupled PDEVS models, using the SIMANLib package. As an example, the internal structure of the Process flowchart module is shown in Fig. 9. The use of DEVS to describe SIMANLib and ARENALib constructs facilitates the understanding of the behavior of each library component, the development of new models and its implementation.

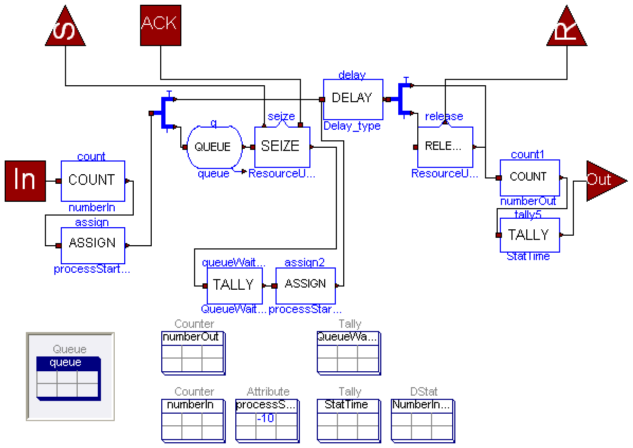
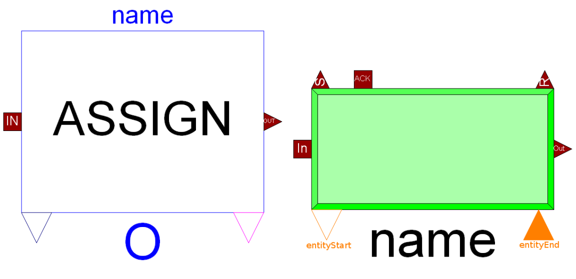


Figure 9: Internal structure of the ARENALib process module.

5.3 Hybrid System Modeling

SIMANLib and ARENALib include models that facilitate combining a process-oriented model with a continuous-time model. These models are: the External Assign block in SIMANLib and the External Process in ARENALib (which is not present in Arena). These models are shown in Fig. 10. Also, due to the compatibility between ports, any model developed using DEVSLib can be combined with SIMANLib and ARENALib, taking into account the values of the transmitted messages.



(a) (b)

Figure 10: Hybrid system modeling components: a) SIMANLib External Assign; and b) ARENALib External Process.

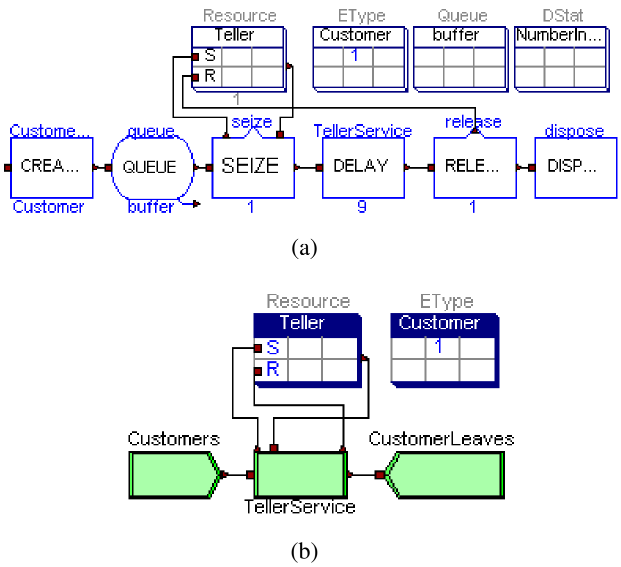
The External Assign block behaves like an Assign block, setting the value of a variable or attribute, and also includes two continuous-time output ports. These ports can be used to detect the changes in the value of the variable, and use that value in a continuous-time model.

The External Process behaves like a normal Process module, representing a process performed to the entities, but the processing time (delay) is calculated by an external continuous-time model (named “ext-process”). Every time an entity arrives to the module, and after seizing the resources, if needed, the External Process changes the value of the “entityStart” port to the reference of the received entity. The ext-process has to detect this change as a notification to start processing an entity. When the process is finished, the ext-process changes the value of the “entityEnd” port to the reference of the previously received entity. With that change, the External Process module detects the end of the process, identifies the entity and lets it continue through the flowchart diagram.

5.4 Model Development with SIMANLib and ARENALib

The “bank teller” system constructed using SIMANLib and ARENALib is presented in this section. The flowchart diagrams of both models are shown in Fig. 11

The model constructed using SIMANLib contains the following blocks: Create (that represents the arrival of customers), Queue, Seize (that together with the Release manages the availability of the teller), Delay (that represents the delay due to the service time), Release and Dispose (that represents the departure of customers). It also includes the following elements:



(a)

(b)

Figure 11: Bank teller system modeled using: a) SIMANLib; and b) ARENALib.

Resource (that represents the teller), EType (that represents the customers), Queue (that describes the organization of the queue) and DStat (that calculates the statistics for the number of customers in queue).

The model constructed using ARENALib contains the following flowchart modules: Create (that represents the arrivals of customers), Process (that represents a customer serviced by the teller) and Dispose (that represents the departure of customers). The data modules included are: Resource (representing the teller) and Entity (representing the customers).

Both models are equivalent, but as shown in Fig. 11 the components in SIMANLib perform simpler actions and more components are required to model the same behavior.

Both systems have been simulated for 10^6 time units to study the steady-state behavior. The statistical indicators are automatically calculated during the simulation run by the DStat elements. The results are shown in Table 1, including the average number of customers in queue and the half-width intervals calculated by Arena (SIMAN and Arena obtain the same results because in both cases the same seed is used to initialize the RNG).

6 Case Studies

Two case studies are presented to show the functionalities of SIMANLib and ARENALib: a restaurant and an electronic factory. The first model is analyzed running independent terminating simulations and the second one performing one long (steady-state) simulation. The results are equivalent to the ones obtained using SIMAN and Arena, respectively.

Table 1: Bank teller system simulation results using SIMANLib, ARENALib, Arena and SIMAN.

Model	Avg. Customers in Queue	Half-Width
SIMANLib	3.3073	-
ARENALib	3.1212	-
Arena	3.2089	0.22
SIMAN	3.2089	0.22

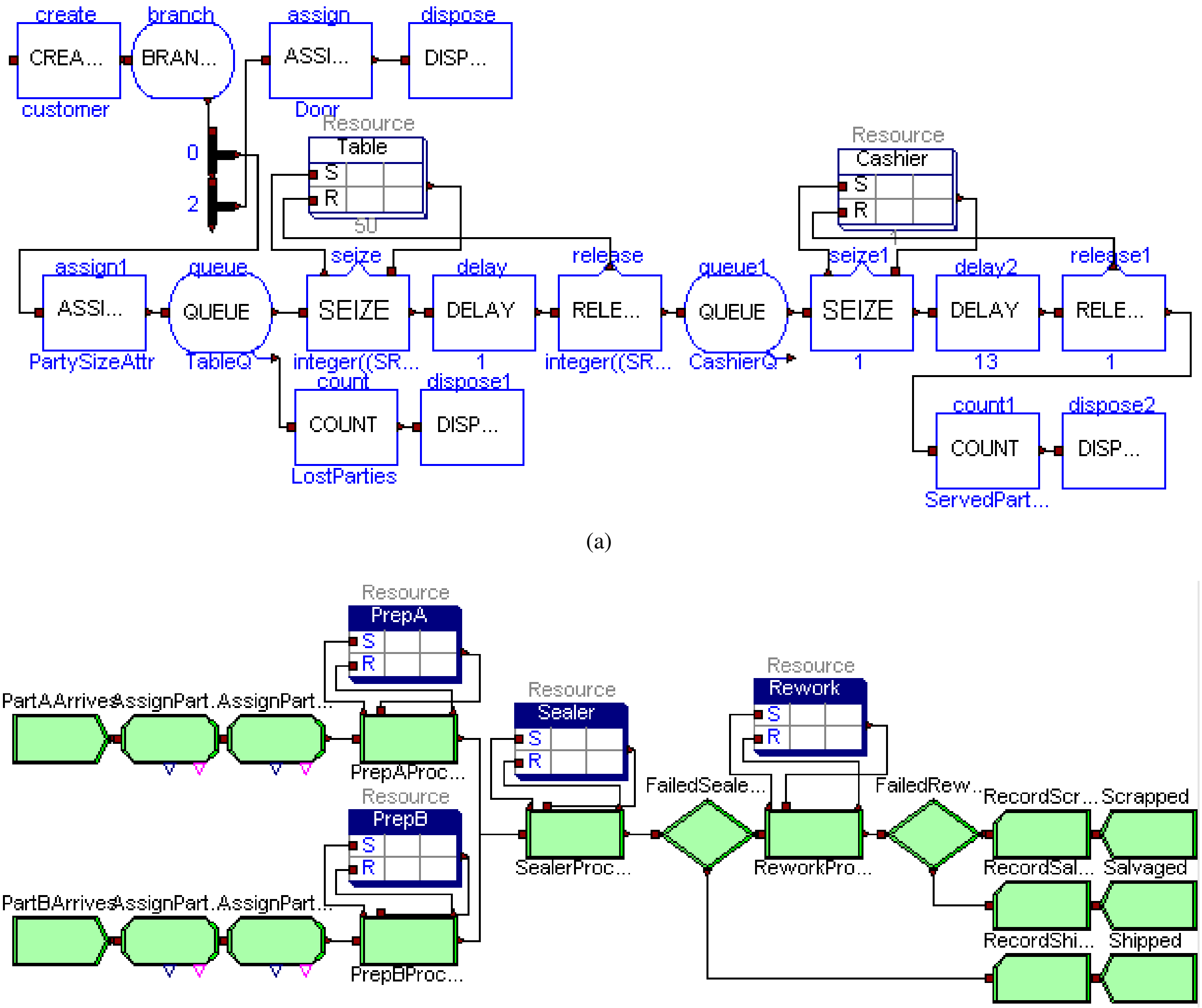
6.1 Restaurant Model

The restaurant model described in [19] has been composed using SIMANLib (see Fig. 12a). Customers arrive in groups from 2 to 5 persons and wait for an available table. If there are already 5 groups waiting, the new group leaves without waiting. The restaurant has 50 tables. Each table is for two people, so several tables may be needed for each group. When seated, the group is served and eats. At the end, the group pays the check to the cashier and leaves. The restaurant receives customers from 5 p.m. to 9 p.m., and, after that, waits until all the customers leave.

Table 2: Restaurant simulation results, comparing SIMANLib and SIMAN.

Indicator (avg.)	SIMANLib	SIMAN
groups served	136.13	135.43
groups lost	15.83	14.00
busy tables	24.49	24.25
groups waiting	0.62	0.72
cashier util.(%)	42.51	41.94

In order to analyze the system, 30 independent simulation runs, each of 480 time units, have been performed. Each run record statistics about the number of customers served, the number of busy tables, the number of waiting customers, the number of groups that left without entering and the utilization of the cashier. The simulation results, comparing the SIMANLib and SIMAN models are shown in Table 2 (average values).



(a)

(b)

Figure 12: Case studies: a) restaurant modeled using SIMANLib; and b) electronic assembly system modeled using ARENALib.

6.2 Electronic Factory Model

The electronic assembly and test system described in [18] has been composed using ARENALib (see Fig. 12b). Two types of electronic parts (A and B) are received in the system, are pre-processed and sealed. Each type has a different pre-processing and sealing time. After that, the sealed parts are inspected. Correct parts are shipped, and the rest need to be reworked. After the rework process, they are inspected again and classified into salvaged and scrapped.

The system has been simulated during 50000 time units, in order to evaluate its steady-state behavior. Multiple statistical indicators are automatically calculated by ARENALib. Some of these indicators (average values) are shown in Table 3 and the are compared with the results obtained with Arena, including the half-width (H-W) interval.

Table 3: Electronic factory simulation results, comparing ARENALib and Arena.

Indicator (avg.)	ARENALib	Arena	H-W
Shipped	18.650	19.774	2.273
Salvaged	89.348	81.522	8.715
Scrapped	88.564	78.125	(Insuf)
Sealer.WaitTime	0.447	0.453	0.035
Sealer.ProcessTime	2.609	2.617	(Corr)
Sealer.Utilization	0.601	0.605	0.011
Sealer.NumberInQueue	0.103	0.105	0.007
Rework.WaitTime	40.272	32.974	8.020
Rework.ProcessTime	30.033	28.452	1.823
Rework.Utilization	0.622	0.583	0.043
Rework.NumberInQueue	0.834	0.675	0.180

7 Conclusions

A new free Modelica library, named DESLib, has been designed and programmed to facilitate the development of discrete-event and process-oriented models. The library is composed of four packages: RandomLib, DEVSLib, SIMANLib and ARENALib. DEVSLib supports the development of discrete-event models following the Parallel DEVS formalism. SIMANLib and ARENALib facilitate the development of process-oriented models of logistic systems, with functionalities similar to the SIMAN language and the Arena simulation environment. RandomLib contains an implementation of the CMRG random number generator, used in Arena, that combined with the other packages of DESLib facilitates the development of stochastic discrete-event models.

The hierarchic description of components in DESLib (SIMANLib compopnents constructed using DESLib, and ARENALib components constructed using SIMANLib) and the use of the DEVS formalism simplifies its understanding, maintenance, reuse and further development.

The communication mechanism developed and included in DESLib allows to transport structured information between models. This mechanism can also be easily adapted to other applications and libraries.

The description and study of stochastic discrete-event and logistic models with Modelica is supported by DESLib. Due to the interface models included in the library, the combination of DESLib with other Modelica libraries facilitates the description of complex hybrid models.

References

[1] Modelica Association. Modelica - A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification (v. 3.1). Available at http://www.modelica.org/documents, 2009.

[2] Mattsson S. E, Otter M, Elmqvist H. Modelica Hybrid Modeling and Efficient Simulation. In Proc. of the 38th IEEE Conf. on Decision and Control, pp. 3502–3507, 1999.

[3] Otter M, Elmqvist H, Mattsson S. E. Hybrid Modeling in Modelica Based on the Synchronous Data Flow Principle. In Proc. of the 10th IEEE Intl. Symposium on Computer Aided Control System Design, pp. 151–157, 1999.

[4] Ferreira J, de Oliveira J. E. Modelling Hybrid Systems Using Statecharts and Modelica. In Proc. of the 7th IEEE Intl. Conf. on Emerging Technologies and Factory Automation, pp. 1063–1069, 1999.

[5] Otter M, Årzén K.-E, Dressler I. State-Graph - a Modelica Library for Hierarchical State Machines. In Proc. of the 4th Intl. Modelica Conf., pp. 569–578, 2005.

[6] Pulecchi T, Casella F. HyAuLib: Modelling Hybrid Automata in Modelica. In Proc. of the 6th Intl. Modelica Conf., pp. 239–246, 2008.

[7] Mosterman P. J, Otter M, Elmqvist H. Modelling Petri Nets as Local Constraint Equations for Hybrid Systems Using Modelica. In Proc. of the Summer Computer Simulation Conf., pp. 314–319, 1998.

[8] Fabricius S. M. O. Extensions to the Petri Net Library in Modelica. ETH Zurich, Switzerland, 2001.

[9] Remelhe M. A. P. Combining Discrete Event Models and Modelica - General Thoughts and a Special Modeling Environment. In Proc. of the 2nd Intl. Modelica Conf., pp. 203–207, 2002.

[10] Färnqvist D, Strandemar K, Johansson K. H, Hespanha J. P. Hybrid Modeling of Communication Networks Using Modelica. In Proc. of the 2nd Intl. Modelica Conf., pp. 209–213, 2002.

[11] Zeigler B. P, Kim T. G, Prähofer H. Theory of Modeling and Simulation. Academic Press, 2000.

[12] Fritzson P. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEEE Computer Society Pr., 2003.

[13] Beltrame T, Cellier F. E. Quantised State System Simulation in Dymola/Modelica Using the DEVS Formalism. In Proc. of the 5th Intl. Modelica Conf., pp. 73–82, 2006.

[14] Cellier F. E, Kofman E. Continuous System Simulation. Springer, 2006.

[15] Kofman E. Discrete Event Simulation of Hybrid Systems. SIAM Journal on Scientific Computing, 25(5):1771–1797, 2004.

[16] Chow A. C. H. Parallel DEVS: a Parallel, Hierarchical, Modular Modeling Formalism and its Distributed Simulator. Trans. of the Society for Computer Simulation Intl., 13(2):55–67, 1996.

[17] Derrick E. J, Balci O, Nance R. E. A comparison of selected conceptual frameworks for simulation modeling. In Proc. of the 1989 Winter Simulation Conf., pp. 711–718, 1989.

[18] Kelton W. D, Sadowski R. P, Sturrock D. T. Simulation with Arena. McGraw-Hill, 4^{th} ed., 2007.

[19] Pegden C. D, Sadowski R. P, Shannon R. E. Introduction to Simulation Using SIMAN. McGraw-Hill, 1995.

[20] Mikler J, Engelson V. Simulation for Operation Management: Object Oriented Approach using Modelica. In Proc. of the 3^{rd} Intl. Modelica Conf., pp. 207–214, 2003.

[21] Dynasim AB. Dymola Dynamic Modeling Laboratory User's Manual. http://www.dymola.com, 2009.

[22] Law A. M. Simulation Modelling and Analysis. McGraw-Hill, 4^{th} ed., 2007.

[23] L'Ecuyer P. Software for Uniform Random Number Generation: Distinguishing the Good and the Bad. In Proc. of the 33^{rd} Conf. on Winter Simulation, pp. 95–105, 2001.

[24] L'Ecuyer P, Simard R, Chen E. J, Kelton W. D. An Object-Oriented Random-Number Package With Many Long Streams and Substreams. Oper. Res., 50 (6):1073–1075, 2002.

[25] Zeigler B. P, Sarjoughian H. S. Introduction to DEVS Modeling & Simulation With JAVA: Developing Component Based Simulation Models. Available at http://www.acims.arizona.edu/PUBLICATIONS/, 2003.

[26] Wainer G. CD++: A Toolkit to Develop DEVS Models. Software: Practice and Experience, 32(13):1261–1306, 2002.

[27] Nutaro J. ADEVS - A Discrete Event System Simulator. Arizona Center for Integrative Modeling & Simulation (ACIMS), University of Arizona, Tucson. Available at http://www.ece.arizona.edu/˜nutaro/index.php, 1999

[28] Sanz V, Urquia A, Dormido S. Introducing Messages in Modelica for Facilitating Discrete-Event System Modeling. In Proc. of 2^{nd} Intl. Workshop on Equation-Based Object-Oriented Languages and Tools, pp. 83-93, 2008.

[29] Sanz V, Urquia A, Dormido S. DEVS Specification and Implementation of SIMAN Blocks Using Modelica Language. In Proc. of the Winter Simulation Conf., pp. 2374–2374, 2007.

[30] Sanz V, Urquia A, Dormido S. ARENALib: A Modelica library for Discrete-Event System Simulation. In Proc. of the 5^{th} Intl. Modelica Conf., pp. 539–548, 2006.

Stream Connectors – An Extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena

Rüdiger Franke, ABB AG, Germany, Ruediger.Franke@de.abb.com
Francesco Casella, Politecnico di Milano, Italy, Casella@elet.polimi.it
Martin Otter, DLR Institute for Robotics and Mechatronics, Germany, Martin.Otter@dlr.de
Michael Sielemann, DLR Institute for Robotics and Mechatronics, Michael.Sielemann@dlr.de
Hilding Elmqvist, Dassault Systèmes (Dynasim), Sweden, Hilding.Elmqvist@3ds.com
Sven Erik Mattson, Dassault Systèmes (Dynasim), Sweden, SvenErik.Mattsson@3ds.com
Hans Olsson, Dassault Systèmes (Dynasim), Sweden, Hans.Olsson@3ds.com

Abstract

Modelica 3.0 as well as other physical modeling languages have two basic variable types to describe the interaction of physical components: “Potential” (or across) variable and “flow” (or through) variable. It is shown that with these variable types it is not possible to describe in a numerically sound way bi-directional flow of matter. Other alternatives based on signal flow oriented modeling have severe restrictions how components can be connected together.

This fundamental problem is addressed in Modelica 3.1 by introducing a third type of connector variable for physical systems, called stream variable, declared with the prefix stream.

This article motivates and introduces stream variables. Examples are given for their utilization in basic fluid models.

Keywords: thermo-fluid, stream variable, convection, potential/flow variable, across/through variable

1 Introduction

Connectors and connector designs are crucial for the modular modeling of complex physical systems. The understanding of simulation models is generally simplified if the modular model structure corresponds to the structure of actual physical devices. Connectors of such device-oriented models shall represent actual ports, like flanges for fluid flow.

Modelica provides different kinds of connection variables for the definition of connectors. Input and output signals are used for the connection of control blocks. Connections between physical devices are generally treated with pairs of “potential” (or across) and “flow” (or through) variables. Electrical systems are a typical example, where voltage differences are treated as potential variables and currents are treated as flow variables.

Thermo-fluid systems deal with convective transport phenomena. The two basic variable kinds in a physical connector – potential variable and flow variable – are not sufficient to describe in a numerically sound way the bi-directional flow of matter with convective transport of specific quantities, such as specific enthalpy and chemical composition. The values of these specific quantities are determined from the upstream side of the flow, i.e., they depend on the flow direction. When using potential and flow variables, the corresponding models would include nonlinear systems of equations with Boolean unknowns for the flow directions and singularities around zero flow. Such equation systems cannot be solved reliably in general. The model formulations can be simplified when formulating two different balance equations for the two possible flow directions. This is not possible with potential and flow variables though.

The requirements for a good, general-purpose object-oriented modeling framework in this domain are:

- single definition of one fluid connector class;
- intuitive semantics for hierarchical, device-oriented modeling: outer connectors of a functional unit model (e.g., a steam generator) should correspond to physical flanges of the unit itself;
- numerical robustness at zero and reverting flow;
- arbitrary connections between multiple fluid connectors are possible; automatically generated connection equations represent idealized flow junctions.

DOI: 10.3384/ecp09430078

No Modelica fluid library existing so far fulfills all of these requirements. The experience made with different approaches for the modeling of fluid systems during the last years shows that the available kinds of connector variables are not well suited for the description of interfaces and connections for convective transport phenomena, such as thermo-fluid flow.

Without restriction of generality, this article discusses the convective transport of energy. Therefore, the following analysis is formulated in terms of specific enthalpy. Nonetheless, the principle holds for other quantities transported via convection such as substance concentrations as well.

2 Existing Definitions of Fluid connectors

This section discusses two different approaches that are widely used to describe the interaction of thermo-fluid components. Other known approaches are either similar to the discussed ones or are not valid with respect to the connection semantics of Modelica 3.

2.1 Connection Semantics of Balanced Models

In Modelica 3.0 the concept of balanced models was introduced [7]. This approach requires that every model is "locally balanced", which means that the number of unknowns and equations must match on every hierarchical level. It is then sufficient to check every model locally only once, e.g., all models in a Modelica package. Using these models (instantiating and connecting them, redeclaring replaceable models etc.) will then lead to a model where the total number of unknowns and the total number of equations will match - a very important and useful property.

In [7] it is shown that this property can only hold if the number of flow variables in a connector is identical to the number of non-causal, non-flow variables (i.e., variables that do not have a **flow**, **input**, **output**, **parameter**, **constant** prefix). Therefore, this restriction on connectors was introduced in Modelica 3.0 and is utilized below.

2.2 Treatment of Transported Quantities as Signals

Focusing on the requirement for numerical robustness and simplicity for a Modelica tool, transported quantities can be communicated as signals through connectors. This approach has been implemented successfully in the ThermoPower library, see [2]. The basic idea is to use a pair of input/output variables for each transported intensive quantity (e.g., the specific enthalpy or the composition). Each of them corresponds to a specific direction of flow (into or out of the connector):

```
connector FluidPort_SignalA
  import SI = Modelica.SIunits;
         SI.Pressure         p;
  flow   SI.MassFlowRate     m_flow;
  input  SI.SpecificEnthalpy hBA;
  output SI.SpecificEnthalpy hAB;
end FluidPort_SignalA;
```

It is not possible to connect two connectors of this type together, since then two input variables (hBA) would be connected together and this is not allowed due to block diagram semantics. It is also not possible to just remove the input/output prefixes, because the resulting connector would violate the Modelica 3 restrictions, see section 2.1, since the number of non-causal, non-flow variables (p, hBA, hAB) would not be identical to the number of flow variables (m_flow).

Therefore, a second connector is needed with the same variables but with exchanged input/output prefixes:

```
connector FluidPort_SignalB
  import SI = Modelica.SIunits;
         SI.Pressure         p;
  flow   SI.MassFlowRate     m_flow;
  output SI.SpecificEnthalpy hBA;
  input  SI.SpecificEnthalpy hAB;
end FluidPort_SignalB;
```

The input variables in both connectors refer to the value the intensive quantity would have assuming the flow is entering the component; the output variables refer to the value that the flow would have if going out of the component. These quantities are always well-defined, and do not have any discontinuity around zero mass flow rate.

The use of signals in physical connectors, however, complicates the device-oriented modeling of fluid systems. Modelica signals are not intended for physical connectors and their use restricts the connection semantics. Each connection set on the same hierarchical level must contain exactly one FluidPort_SignalA and one FluidPort_SignalB connector. Adaptor models need to be introduced in all other cases.

2.3 Treatment of Transported Quantities as Potential Variables

Focusing on the requirements for intuitive, device-oriented connection semantics, transported quantities can be modeled as potential variables. Previous versions of Modelica_Fluid treat specific enthalpy as potential variable, complemented with enthalpy flow rate as flow variable, see [4]. The connector definition reads:

```
connector FluidPort_Potential
  import SI = Modelica.SIunits;
       SI.Pressure         p;
  flow SI.MassFlowRate     m_flow;
       SI.SpecificEnthalpy h_mix;
  flow SI.EnthalpyFlowRate H_flow;
end FluidPort_Potential;
```

The pressure `p` and the mass flow rate `m_flow` describe the hydraulic phenomena of fluid flow. The specific enthalpy `h_mix` and the enthalpy flow rate `H_flow` describe the convective transport of energy. Since no input/output prefixes are used in this connector, there are no restrictions how components can be connected together, contrary to the signal flow approach of section 2.2.

A connection set is seen as an infinitesimally small mixing volume and the specific enthalpy in the connector is the mixing enthalpy for arbitrary flow directions. Figure 1 shows an example with three connected components.

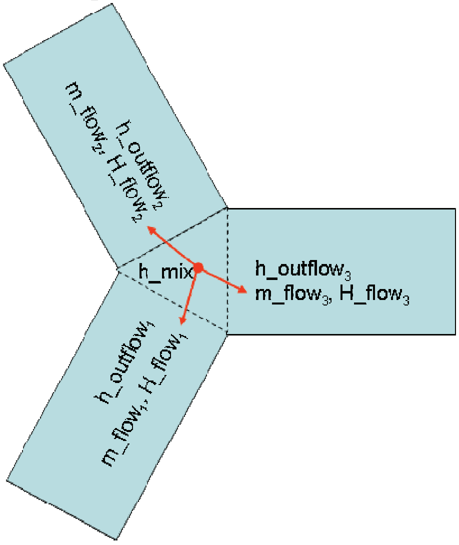


Figure 1: Exemplary connection set with three connected components and a common mixing enthalpy

Consider a connection set with n connectors. The mixing enthalpy is defined by the mass balance

$$0 = \sum_{j=1\ldots n} \dot{m}_j \tag{1}$$

and the energy balance at the infinitesimal small control volume of the connection point

$$0 = \sum_{j=1\ldots n} \dot{H}_j \tag{2}$$

with

$$\dot{H}_j = \dot{m}_j \begin{cases} h_{mix} & \textbf{if } \dot{m}_j > 0 \\ h_{outflow\,j} & \textbf{if } \dot{m}_j \le 0 \end{cases} \tag{3}$$

Herein, mass flow rates $\dot{m}_i$ are positive when entering models (i.e. exiting the connection set). The specific enthalpy $h_{outflow}$ represents the specific enthalpy inside the component, close to the connector, for the case of outflow.

When connecting components together, the Modelica connection semantics for flow variables will result in equations (1) and (2), since `m_flow` and `H_flow` are flow variables. (3) needs to be implemented in a component for every connector of this component.

If the mass flow rates are unknowns, as it is usually the case, then (1)-(3) is a set of non-linear algebraic equations in the mass flow rates $\dot{m}_i$ and the mixing enthalpy h_{mix}. From (3) it can be seen, that the unknown mass flow rates enter this equation set also as Boolean expressions $\dot{m}_j > 0$.

In the most often occurring case of two connected components, it is easy to compute h_{mix} from (1)-(3):

$$h_{mix} = \begin{cases} h_{outflow,2} & \textbf{if } \dot{m}_1 > 0 \\ h_{outflow,1} & \textbf{if } \dot{m}_1 \le 0 \end{cases}$$

Since usually $h_{outflow,1} \neq h_{outflow,2}$, the mixing enthalpy h_{mix} is discontinuous at zero mass flow rate.

If all mass flow rates are zero, $\dot{m}_j = 0$, then (1)-(3) are identically fulfilled for every value of h_{mix}, so these equations do not have a unique solution at this point, which means they are singular. Note, that this singularity is in line with the physics of the problem: The idealized infinitesimally small mixing volume does not have storage; therefore, its thermodynamic state is exclusively defined by the fluid flowing through it. If *no* fluid flows through the connection set, its state is not defined and a singularity arises at zero mass flow.

The above observations can be summarized as follows: Non-linear equation systems might occur, e.g., due to steady-state initialization, steady state models, ideal mixing, or pressure drop components directly connected together. If this is the case and if h_{mix} is an iteration variable of this non-linear equation system, then (1) a singularity is present when all mass flow rates are zero, (2) h_{mix} is usually discontinuous at this singular point, and (3) the directions of the mass flow rates, $\dot{m}_j > 0$, are unknowns, i.e., the equation system consists not only of real but also of Boolean unknowns. No numerical solver is known that is able to compute the solution of such a non-linear algebraic equation system in a reliable way and it is very unlikely that this will ever be the case.

This analysis shows clearly that the mixing enthalpy h_{mix} should not be computed and it should never be used in a connector.

3 Stream Connectors

The goal of stream connectors is to fulfill all requirements on a thermo-fluid connector regarding intuitive device-oriented connection semantics on one hand and numerical robustness on the other.

From the analyses in the previous section, it becomes obvious that, in order to meet the goal of numerical robustness, the plain specific mixing enthalpy should not be computed. Note that the energy balance, together with the piecewise equation given in equation (3), consists of different branches. It can therefore be split. Separate energy balances are the result; each valid for a specific flow direction.

This design decision leads to two primary consequences. First, the corresponding connector variables (called "stream" variables) come in two incarnations, one under the assumption of fluid entering the component, the other under the assumption of fluid leaving the component. The specific enthalpy under the assumption of fluid leaving the component, $h_{outflow}$, is one of the two and has to be established via governing equations of the model. If a model has a "volume/capacity", these equations refer to the corresponding state variables of this volume. If the model does not have a volume/capacity, the specific enthalpy under the assumption of fluid <u>leaving</u> the component at one connector is an algebraic (instantaneous) function of the values of the specific enthalpies under the assumption of fluid <u>entering</u> the component at the other connectors of this component. This is the second consequence: Whenever the mixing enthalpy is *used* in a model it is the mixing enthalpy under the assumption of fluid flowing into said model. We establish this quantity using a dedicated built-in operator

$$inStream\left(h_{outflow,j}\right) = h_{mix}\left(\dot{m}_j \geq 0\right). \qquad (4)$$

This is the definition of the second incarnation of the specific mixing enthalpy on the level of a single flange. This means, the mixing enthalpy is only computed for the case when fluid is entering a component, but not when fluid is leaving. From the perspective of the connection set, this definition leads to *n* different incarnations of the specific mixing enthalpy (with *n* as number of connections in a connection set).

After motivating the concept, we will now describe it in detail. First, using stream variables, a fluid connector is defined as follows.

```
connector FluidPort_Stream
  import SI = Modelica.SIunits;
         SI.Pressure          p;
  flow   SI.MassFlowRate      m_flow;
  stream SI.SpecificEnthalpy h_outflow;
end FluidPort_Stream;
```

In this section, only the most important type of connections is treated, where components are connected on the same "level", so called "inside" connections. The special case of "outside" connections, i.e., connections along a hierarchical model, is sketched in appendix A2.

A connector containing stream variables is called a stream connector. A stream connector must have exactly one scalar variable with the **flow** prefix. The idea is that all stream variables of this connector are associated with this flow variable. A stream variable

- defines a quantity that is transported by a flow through the stream connector (i.e. large-scale motion via, e.g., `m_flow`);
- represents the value of the transported quantity for the case of outflow through the stream connector, i.e. `m_flow < 0`;
- does not lead to the generation of any connection equations (for "inside" connections).

A model using a stream connector must expose the outflow value, which is known from internal storage or from inflow values of other connectors, independently of the flow direction. The inflow value can be obtained by a model on demand by using the new operator

inStream(h_outflow)

In the Modelica Language Specification 3.1 [6], the definition for this operator is given implicitly. It is based on the balance equation for transported quantities, see (2) and (3):

$$0 = \sum_{j=1\ldots n} \dot{m}_j \cdot \begin{cases} h_{mix} & \textbf{if } \dot{m}_j > 0 \\ h_{outflow,j} & \textbf{else} \end{cases} \qquad (5)$$

However, the common enthalpy h_{mix} for ideal mixing shall not be computed. Instead, for <u>every connector *i*</u> the balance equation (5) is written under the assumption that flow is only entering a component via connector *i* (so these are "n" equations):

$$0 = \sum_{j=1\ldots n} \dot{m}_j \cdot \begin{cases} h_{mix\_in,i} & \textbf{if } \dot{m}_j > 0 \textbf{ or } j = i \\ h_{outflow,j} & \textbf{else} \end{cases} \qquad (6)$$

$$\textbf{inStream}(h_{outflow,i}) = h_{mix\_in,i} \qquad i = 1,2,\ldots,n$$

As a result, the "n" balance equations (6) compute "n" mixing enthalpies $h_{mix\_in,i}$ under the assumption that flow enters the respective connector *i*. Similarly to the enthalpy for ideal mixing, a singularity is present if all mass flow rates vanish: If $\dot{m}_j = 0$ then (6) is identically fulfilled for every value of $h_{mix\_in,i}$. It is therefore necessary to approximate the solution of (6) in an open neighborhood of that point. We neglect this issue for the moment and will come back to it in the subsequent section.

As shown in appendix A1, the enthalpy for ideal mixing for "n" connected components can be explicitly computed from (5) by:

$$h_{mix} = \frac{\sum\limits_{j=1\ldots n} h_{outflow_j} \max(-\dot{m}_j, 0)}{\sum\limits_{j=1\ldots n} \max(-\dot{m}_j, 0)} \tag{7}$$

In a similar way, the mixing enthalpy under the assumption of a flow into connector i can be explicitly computed from (6) by:

$$\begin{aligned} \textbf{inStream}(h_{outflow,i}) &= h_{mix}\left(\dot{m}_i \geq 0\right) \\ &= \frac{\sum\limits_{j=1\ldots n,\, j\neq i} h_{outflow,j} \max(-\dot{m}_j, 0)}{\sum\limits_{j=1\ldots n,\, j\neq i} \max(-\dot{m}_j, 0)} \end{aligned} \tag{8}$$

All variables needed for the expansion of the inStream operator are explicitly given in a connection set. The own outflow value $h\_outflow_i$ is not considered as a connector cannot have inflow and outflow at the same time.

Figure 2 visualizes stream variables and the dependencies of the inStream operator for three connected components.

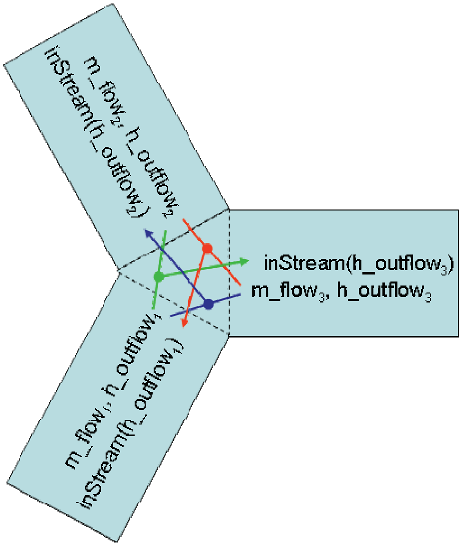


Figure 2: Exemplary connection set with three connected components

The expansion of the inStream operator simplifies for some important special cases.

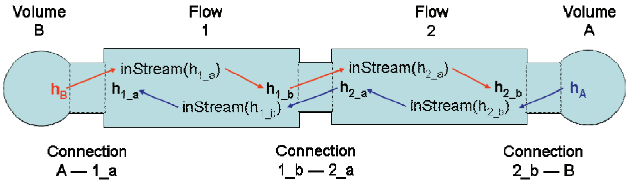


Figure 3: Exemplary series connection of multiple models with stream connectors

Figure 3 shows the series connection of two volumes via two "flow" models that neither store mass nor energy. In this case the inStream operator reduces to a simple propagation of two independent values. This can be easily derived from (8) for the special case of two connected components:

$$\begin{aligned} &\textbf{inStream}(h_{outflow,1}) = h_{outflow,2} \\ &\textbf{inStream}(h_{outflow,2}) = h_{outflow,1} \end{aligned} \tag{9}$$

These equations state that the value of the inflowing specific enthalpy is just the value of the specific enthalpy from the upstream side. This result could have been obtained also directly by inspection of a figure of this situation. Therefore, the following relations hold for Figure 3:

$$\begin{aligned} \textbf{inStream}(h_B) &= h_{1\_a} = \textbf{inStream}(h_{1\_b}) \\ &= h_{2\_a} = \textbf{inStream}(h_{2\_b}) = h_A \end{aligned} \tag{10}$$

In volume B the energy balance, with the internal energy U_B of volume B, might have the following form which can be considerably simplified using (10):

$$\begin{aligned} \frac{dU_B}{dt} &= \dot{m}_B \cdot \begin{cases} \textbf{inStream}(h_B) & \textbf{if } \dot{m}_B > 0 \\ h_B & \textbf{if } \dot{m}_B \leq 0 \end{cases} \\ &= \dot{m}_B \cdot \begin{cases} h_A & \textbf{if } \dot{m}_B > 0 \\ h_B & \textbf{if } \dot{m}_B \leq 0 \end{cases} \end{aligned} \tag{11}$$

Note, that h_A and h_B are either states of the respective volumes or can be directly computed from the states via the media equations, which means that these variables are "known". The equation in (11) still contains a relation of the unknown mass flow rate ($\dot{m}_B > 0$). However, this is uncritical here, because this relation can be directly computed from the states of the volume and from the pressure-drop equations of the two flow models:

$$\begin{aligned} \dot{m}_B &= f_1\left(p_{1\_b} - p_B, \rho_{1ba}, \rho_{1ab}\right) \\ &= f_2\left(p_A - p_{1\_b}, \rho_{2ba}, \rho_{1ab}\right) \\ \rho_{1ab} &= \rho\left(p_B, \mathbf{inStream}(h_{1\_a})\right) = \rho\left(p_B, h_B\right) \\ \rho_{1ba} &= \rho\left(p_{1\_b}, \mathbf{inStream}(h_{1\_b})\right) = \rho\left(p_{1\_b}, h_A\right) \\ \rho_{2ab} &= \rho\left(p_{1\_b}, \mathbf{inStream}(h_{2\_a})\right) = \rho\left(p_{1\_b}, h_B\right) \\ \rho_{2ba} &= \rho\left(p_A, \mathbf{inStream}(h_{2\_b})\right) = \rho\left(p_A, h_A\right) \end{aligned} \qquad (12)$$

Function f_1(..) is the pressure drop relation of component flow 1 and f_2(..) the respective one of component flow 2. These functions depend basically on the pressure difference between their ports and the upstream density ρ, i.e., ρ_{ba} when the flow is from port “b” to port “a” and ρ_{ab} when the flow is from port “a” to port “b” (other dependencies, e.g., from dynamic viscosity η, can be handled in a similar way). Since p_A, p_B, h_A, h_B are “known” quantities from volumes A and B, respectively, (12) are basically a non-linear implicit equation to compute p_{1_b} and then an explicit equation to compute $\dot{m}_B$. Since f_1(..) and f_2(..) are strictly increasing monotonic functions that are at least continuous, the scalar non-linear equation system is easy to solve. Once $\dot{m}_B$ is computed, the relation appearing in the energy balance of volume B, $\dot{m}_B > 0$, can be calculated as well.

To summarize, the system in Figure 3 is easy to solve numerically and only requires to solve one (uncritical) scalar non-linear equation in one unknown and otherwise only a series of explicit equations needs to be solved. The above analysis holds if the medium equations are a function of pressure p and specific enthalpy h. Other medium dependencies are uncritical as well, as discussed in section 4.

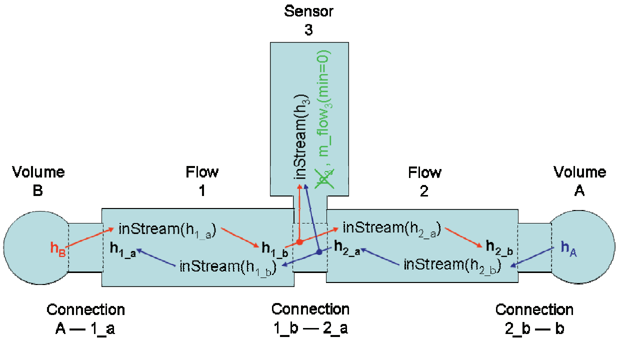


Figure 4: Exemplary series connection of multiple models with stream connectors and a sensor component.

The outflow values of connectors that never provide outflow, like absolute sensors, do not need to be considered for the expansion of inStream operators. Such connectors are defined by using the `min` attribute for the flow variable, see Figure 4.

The inStream value of an unconnected connector is defined to be equal to the outflow value.

The complete specification of stream connectors, for inside and outside connectors is given in [6].

4 Numerical Properties

In this section, the regularization of the inStream() operator is sketched and several other numerical properties are analyzed.

4.1 Regularization of the inStream() Operator

As already analyzed in the preceding section, the return value of the **`inStream`**`()` operator cannot be uniquely computed if all mass flow rates are zero. When using equation (8), even a division by zero occurs. In [6] this issue is resolved by requiring that the **`inStream`**`()` operator is appropriately approximated in an open neighborhood of vanishing mass flow rates and that the approximation must fulfill the following requirements:

1. **`inStream`**($h_{outflow,i}$) must be unique with respect to all values of the flow and stream variables in the connection set, and must have a continuous dependency on them.
2. Every solution of the implicit equation system (6) must fulfill (6) identically (up to the usual numerical accuracy), provided the absolute value of every flow variable in the connection set is greater than a small value ε > 0 ($|\dot{m}_1| > \varepsilon$ **and** $|\dot{m}_2| > \varepsilon$ ... **and** $|\dot{m}_n| > \varepsilon$). This means that the balance equation is approximated only for small mass flow rates and otherwise is exactly fulfilled.

There are several possibilities to fulfill these requirements. In Figure 5, the definition of the inStream() operator is shown for the case of a three-way mixing point before applying any approximations. The region with all mass flow rates but $\dot{m}_i$ zero or positive is left open as the value of the operator inStream($h_{outflow,i}$) is arbitrary by definition here.

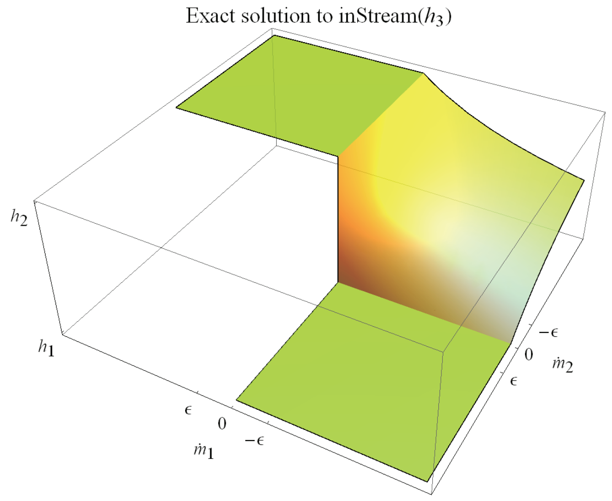


Figure 5: Exact solution for a three-way connection

In the Modelica Language Specification [6], a recommended regularization of the inStream() operator is given. First, let σ_i be the sum of the mass flow rates considered for applying the operator to port *i*.

$$\sigma_i = \sum_{j=1\ldots n,\ j\neq i} \max(-\dot{m}_j, 0)$$

Then, the expressions using the conventional operator max(x, 0) in equation (8) are substituted by a custom operator positiveMax(x, σ_i), which is defined such that it always returns a positive, non-zero value.

$$\text{inStream}(h_{outflow,i}) = \frac{\sum_{j=1\ldots n,\ j\neq i} h_{outflow,j} \cdot \text{positiveMax}(-\dot{m}_j, \sigma_i)}{\sum_{j=1\ldots n,\ j\neq i} \text{positiveMax}(-\dot{m}_j, \sigma_i)} \qquad (13)$$

A suitable definition of the operator is a linear combination of max(*x*, 0) and $\varepsilon \in \Re^+$ along a suitably chosen variable α.

$$\text{positiveMax}(x, \sigma_i) = \alpha \cdot \max(x, 0) + (1-\alpha) \cdot \varepsilon\,.$$

The variable α is a C^1 smooth step function of σ_i.

$$\alpha(\sigma_\mathrm{i}) = \begin{cases} 1 & \text{if } \sigma_i > \varepsilon \\ \left(3 - 2\dfrac{\sigma_i}{\varepsilon}\right)\left(\dfrac{\sigma_i}{\varepsilon}\right)^2 & \text{if } 0 < \sigma_i \leq \varepsilon \\ 0 & \text{if } \sigma_i \leq 0 \end{cases}$$

As a result, the value of the inStream() operator is always well-defined and is a continuous function of the variables entering (8). If all mass flow rates are zero, positiveMax(…) = ε, and

$$\begin{aligned} \text{inStream}(h_{outflow,i}) &= \frac{\sum_{j=1\ldots n,\ j\neq i} h_{outflow,j} \cdot \varepsilon}{\sum_{j=1\ldots n,\ j\neq i} \varepsilon} \\ &= \frac{\varepsilon \cdot \sum_{j=1\ldots n,\ j\neq i} h_{outflow,j}}{\varepsilon \cdot (n-1)} \\ &= \frac{\sum_{j=1\ldots n,\ j\neq i} h_{outflow,j}}{n-1} \end{aligned}$$

that is, the operator returns the arithmetic mean of the stream variables (but without $h_{outflow,i}$). Figure 6 shows an illustration of this regularization for the case of a three-way mixing point. Herein, the arithmetic mean value is shown in blue. Note that outside of the regularization domain points remain, which are not continuously differentiable. This is necessary due to the second requirement to the regularization, which states that the approximation must be exact whenever the absolute values of all flow variables are greater than a given small value.

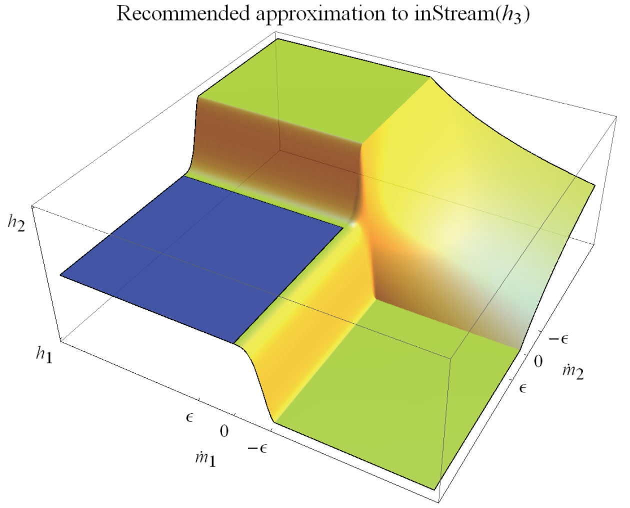


Figure 6: Recommended approximation for a three-way connection

Note, the following trivial implementation of the positiveMax() operator is also allowed according to [6]:

$$\text{positiveMax}(x, \sigma_i) = \max(x, \varepsilon).$$

In this case, the approximation is still exact whenever the absolute values of all flow variables are greater than a given small value. However, in the regularization region the operator is no longer continuously differentiable. See figure 7 for an illustration.

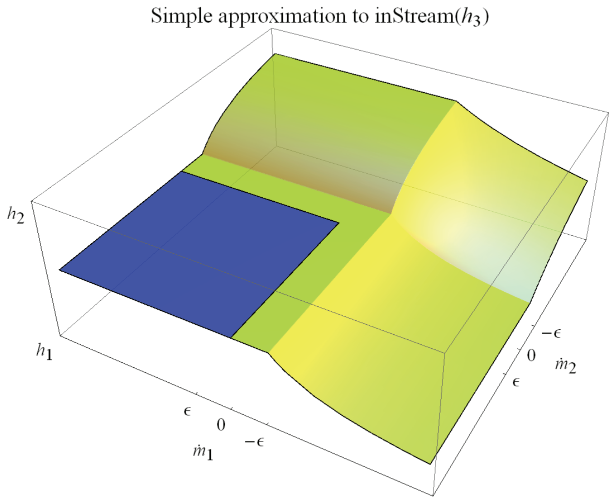


Figure 7: Simplified approximation for a three-way connection

4.2 Iteration Variables for Nonlinear Algebraic Equation Systems

For the robustness of the simulation, the choice and number of iteration variables in non-linear equation systems are crucial. We will discuss this issue at the example of a three-way mixing point with flow models ("detailed wall friction with laminar and turbulent region"), see Figure 8, and in the general case.

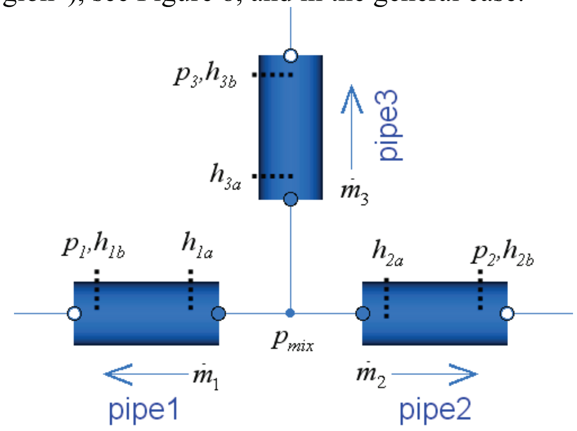


Figure 8: Three way mixing point.

As medium, the fluid "Modelica.Media.IdealGases.-MixtureGases.FlueGasSixComponents" is used which is a mixture of six ideal gas substances. The independent medium variables are pressure p, temperature T and 5 independent mass fractions Xi. This system is basically described by the following set of equations:

$$
\begin{aligned}
0 &= \dot{m}_1 + \dot{m}_2 + \dot{m}_3 \\
\dot{m}_1 &= f_1\left(p_{mix} - p_1, \rho_{1ab}, \rho_{1ba}, \eta_{1ab}, \eta_{1ba}\right) \\
\dot{m}_2 &= f_2\left(p_{mix} - p_2, \rho_{2ab}, \rho_{2ba}, \eta_{2ab}, \eta_{2ba}\right) \\
\dot{m}_3 &= f_3\left(p_{mix} - p_3, \rho_{3ab}, \rho_{3ba}, \eta_{2ab}, \eta_{2ba}\right) \\
\rho_{1ab} &= \rho(p_{mix}, T_{1a\_inflow}, \mathbf{inStream}(Xi_{1a})) \\
T_{1a\_inflow} &= T\left(p_{mix}, \mathbf{inStream}(h_{1a}), \mathbf{inStream}(Xi_{1a})\right) \\
\mathbf{inStream}(h_{1a}) &= \frac{h_{2a} \cdot max(-\dot{m}_2, \varepsilon) + h_{3a} \cdot max(-\dot{m}_3, \varepsilon)}{max(-\dot{m}_2, \varepsilon) + max(-\dot{m}_3, \varepsilon)}
\end{aligned}
$$

...

This set of nonlinear algebraic equations can be basically solved with three iteration variables, namely two mass flow rates and the pressure p_{mix} in the mixing point: If two mass flow rates are given, the third mass flow rate can be computed via the mass balance (the first equation in the set of equations above). Furthermore, the inStream operators each depend on h_outflow's of the components attached to the mixing points and can therefore be computed, once the mass flow rates are known (e.g. h_{2a} = inStream(h_{2b}) which can be, e.g., computed from the states of the volume connected to pipe2; similarly h_{3a}, Xi_{2a}, Xi_{3a} can be computed etc., and then **inStream**(h_{1a}), **inStream**(Xi_{1a}) etc. can be computed).

In order to compute all needed intensive quantities for the ideal gas mixture, the (unknown) temperature T must be computed from pressure, specific enthalpy and mass fractions. This requires in general to solve one non-linear algebraic equation in one unknown. In the Modelica.Media package, this is performed with the algorithm of Brent [1] which is completely reliable and very efficient[1]. Once the temperature is known, densities ρ and dynamics viscosities η may be computed for the wall friction correlations. Also, with the pressure in the mixing point, all pressure differences can be computed and finally all mass flow rates via functions f_1(..), f_2(..), f_3(..). The 3 residual equations are then the differences between the mass flow rates given by the solver and the ones computed from the wall friction correlations.

In the general case of a N-way mixing point, a similar analysis shows that N-1 mass flow rates and

[1] An interval is given in the medium definition, in which the root must be present. If possible, a smaller interval is computed by inverse quadratic interpolation (interpolating with a quadratic polynomial through the last 3 points and computing the zero). If this fails, bisection is used, which always reduces the interval by a factor of 2. The inverse quadratic interpolation method has superlinear convergence. This is roughly the same convergence rate as a globally convergent Newton method, but without the need to compute derivatives of the non-linear function.

the mixing point pressure can be used as iteration variables for a reduced nonlinear algebraic equation system of size N. Note, the number of iteration variables is independent of the number of substances in the medium.

Since the iteration variables – pressure and mass flow rates – are always continuous and in many cases also continuously differentiable, the non-linear algebraic equation system is well-posed and fulfills the requirements of a non-linear algebraic equation solver. For example, in Figure 9 , a typical simulation result of the 3-way mixing point of Figure 8 is shown, where it is clearly seen that the two iteration variables $\dot{m}_1(t)$ and $\dot{m}_2(t)$ are continuous and even continuously differentiable as function of time *t*.

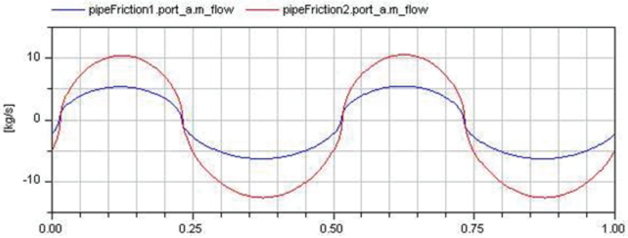


Figure 9: Mass flow rates as iteration variables of the three way mixing point.

The model equations need to be implemented in a specific form, in order to get the desirable properties from above, see section 5.2. The important point is that a function is called to compute the independent variables of the medium (i.e *p*,*T*,*Xi* in the case of the flue gas medium above) from the potential and streams variables of the connector (*p*,*h*,*Xi*) and not vice versa.

In former Beta versions of the Modelica_Fluid library [4], as well as in the ThermoPower library [2], the inverse correlation is used instead: In a “flow” model component a function is present to compute (*p*,*h*,*Xi*) from the medium states (*p*,*T*,*Xi*) via the BaseProperties model which is present in the Modelica.Media models. In such a case, a tool needs to use the temperatures close to the connection points as additional iteration variables. Temperatures are, however, a very bad choice for iteration variables, if bi-directional flow can occur, because they change usually discontinuously when the mass flow changes direction. For example, in Figure 10, the temperatures of Figure 8 are shown that would be used as iteration variables in such a case.

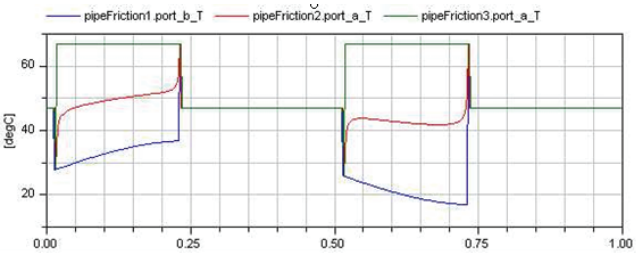


Figure 10: Temperatures as iteration variables of the three way mixing point.

As can be clearly seen, the temperatures are discontinuous and discontinuous iteration variables violate the pre-requisite of every nonlinear algebraic solution method and a reliable solution can therefore not be expected.

The results of the analysis above are summarized in the following table:

Library	*smoothness of iterat. var.*	*number of iterat. var.*
Modelica_Fluid Beta [4]	discontinuous	22
ThermoPower [2]	discontinuous	6
Modelica.Fluid [5]	continuous	3

The above table shows some key results, when simulating the three-way mixing point example from Figure 8 with different Modelica packages with Dymola 7.2 [2]:

- The Beta version of the Modelica_Fluid library [4] uses the connectors from section 2.3 and gives rise to a nonlinear algebraic equation system of 22 iteration variables, where many of the iteration variables are discontinuous when mass flow rate changes direction. So, a reliable solution is not possible.
- The ThermoPower library [2] uses the connectors from section 2.2. If only 1:1 connections are present, the numerical properties of the ThermoPower library and of the streams concept are similar, since a similar technique is used to propagate the specific enthalpy along the connected “flow” models. The main difference is that with the streams concept no longer connection restrictions are present. If an ideally mixing component is added to the ThermoPower library, the three-way mixing example gives rise to a nonlinear algebraic equation system of 6 iteration variables, where 3 of them are discontinuous when mass flow rate changes direction. So, a reliable solution is not possible in this case. The reason is that in the ThermoPower library the connector variables are computed from the medium states and not vice versa.
- The Modelica.Fluid library [5] uses the stream connector concept of section 3. The three-way mixing example gives rise to a nonlinear algebraic

equation system of 3 iterations variables where all of them are continuous. So, a reliable solution is to be expected. This demonstrates that there is a tremendous difference in numerical reliability between the Beta version of Modelica_Fluid and the Modelica.Fluid library released with the Modelica Standard Library 3.1 in August 2009.

The analysis above was performed for an ideal mixing point with a multi-substance medium. One might expect that the described problems are gone if a more detailed model with a volume is used for the mixing point. However, during steady-state initialization and/or for a pure steady-state simulation, the time derivatives of the dynamic states are set to zero and again nonlinear equation systems with similar properties are present. Again, it can be expected that the streams-connector approach leads to equation systems with a small number of continuous iteration variables and can therefore be solved reliably.

Note, if a volume would be used in the connection point of the three-way mixing point, the overall model would have 7 additional states (p, T, 5 mass fractions Xi). If the more precise description is not needed, one can therefore expect that the ideal mixing formulation (leading to a reliably solvable nonlinear algebraic equation system with 3 iteration variables) could result in a more efficient simulation.

4.3 Inverse Function Annotation

The streams connector concept introduced in section 3 is very well suited for media where the potential and stream variables in the connector (e.g., p, h, Xi) are also used as medium states. For media with other media states, there is the disadvantage that for every connection a non-linear algebraic equation has to be solved in order to compute the medium states from the potential and streams variables of the connector. For example, if a medium with independent variables pressure p and temperature T was used for the example in Figure 3, then four scalar non-linear algebraic equations need to be solved to compute (p,T) from (p,h) at points 1a, 1b, 2a, and 2b. This is completely unnecessary for points 1a and 2b, since here the medium states could just be propagated from the volumes.

In order to more efficiently cope with such situations, in Modelica 3.1 [6], section 12.8, a new annotation is introduced to define the inverses of functions. For example, (p,T) media should have the following two basic function definitions:

```
function h_pT
   input  Real p    "pressure";
   input  Real T    "temperature";
   output Real h    "specific enthalpy";
algorithm
   ...
end h_pT;

function T_ph
   input  Real p    "pressure";
   input  Real h    "specific enthalpy";
   output Real T    "temperature";
   annotation(inverse(h = h_pT(p,T));
algorithm
   ...
end T_ph;
```

For a (p,T) media the specific enthalpy h is computed explicitly via function h_pT(..). Additionally, the inverse function T = T_ph(..) is needed which will usually need to call a non-linear algebraic equation solver in order to invert function h_pT(..). In such a case, function T_ph(..) should have the annotation "**inverse**(h = h_pT(p,T)". This annotation states that the inverse of T_ph is function h_pT. The essential requirement is that the inverse function must have the same input/output arguments as the direct function, but the order of the arguments may be permuted. Especially, a variable that was declared as "input", is used as "output" in the inverse function.

The "inverse" annotation signals a tool that it should use this function if possible, because it will be more efficient than using the "direct" function. For example, in Figure 11 two components are connected together and stream variables are used in the connectors. Let us assume that *p*, T_1 are known variables, since they are states of a volume.

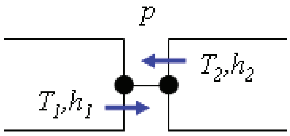


Figure 11: Enhancing efficiency with the inverse annotation when propagating media properties.

This situation is described by the following equations:

$$
\begin{aligned}
h_1 &= h_{pT}(p,T_1) \\
\textbf{inStream}(h_2) &= h_1 \\
T_{2,inflow} &= T_{ph}(p,\textbf{inStream}(h_2)) \\
&= T_{ph}(p,h_1)
\end{aligned}
$$

Since the inverse function h_{pT} is defined as annotation in function T_{ph}, the tool is advised to try whether the usage of the inverse function will simplify the equations. Indeed this is the case here: If the last equation is replaced by the inverse function:

$$h_1 = h_{pT}(p, T_1)$$
$$\textbf{inStream}(h_2) = h_1$$
$$h_1 = h_{pT}(p, T_{2,inflow})$$

it can be detected that the same function is called twice. In both cases, the same arguments p,h_1 are used. The second input arguments T_1, $T_{2,inflow}$ must be identical, in order that this can hold. Therefore, the above equations can be simplified to:

$$h_1 = h_{pT}(p, T_1)$$
$$\textbf{inStream}(h_2) = h_1$$
$$T_{2,inflow} = T_1$$

As a result, function T_{ph} need no longer be called and therefore one scalar nonlinear algebraic equation computation is removed.

The above simplification rule is, e.g., available in Dymola 7.2 [2]. In a few media of Modelica.Media (version 3.1) the inverse annotation is already included. It is planned to include the annotation for all media where this makes sense in the follow-up version of the Modelica Standard Library.

To summarize, with the inverse annotation, appropriate media description and corresponding tool support, the unnecessary scalar nonlinear algebraic equation systems are avoided for 1:1 connections, when using media that do not have (p,h,X) as states.

5 Examples for Basic Fluid Models

Thermo-fluid models are built from components, each representing a control volume. A control volume is defined as a fixed region in space where one studies the masses and energies crossing the boundaries of the region. The boundaries of a control volume usually represent the physical boundaries of the parts through which fluid flow is occurring. Often two basic kinds of component models are distinguished:

- Volume models define a thermodynamic state for a lumped medium.
- Transport or flow models serve to connect volume models. They do not define an own thermodynamic state. Instead they define the transport of fluid for thermodynamic states given at their ports.

Multiple volume and transport models can be assembled to build component models hierarchically. The following examples use the

```
connector FluidPort = FluidPort_Stream;
```

5.1 Volume model

A basic model of an ideally mixed lumped volume is defined below.

```
model Volume "Lumped volume, e.g. vessel"

  replaceable package Medium =
Modelica.Media.Interfaces.PartialMedium;

  parameter Integer nPorts=0
    "Number of ports";
  FluidPort[nPorts] ports;

  parameter Modelica.SIunits.Volume V
     "Volume of device";
  Modelica.SIunits.Mass             m
     "Mass in device";
  Modelica.SIunits.Energy           U
     "Inner energy in device";

  Medium.BaseProperties medium;

equation
  // Definition of port variables
  for i in 1:nPorts loop
    ports[i].p = medium.p;
    ports[i].h_outflow = medium.h;
  end for;

  // Mass and energy balance
  m = V*medium.d;
  U = m*medium.u;
  der(m) = sum(ports.m_flow);
  der(U) = ports.m_flow *
           actualStream(ports.h_outflow);
end Volume;
```

The volume model can be used with an arbitrary medium model out of Modelica.Media. The mass balance and the energy balance sum up the contribution of flows going through nPorts fluid ports.

Alternatively the volume model could be defined with only one fluid port as multiple connections can be made to it. Then, however, ideal mixing would take place in the port outside the volume. Using unary connections to multiple ports ensures that the mixing takes place inside the volume. This is the generally intended behavior for multi-way connections to a volume. Nonlinear systems of mixing equations are avoided.

The actualStream operator used to define the energy balance is a shorthand notation for:

```
actualStream(port.h_outflow) ==
  if port.m_flow > 0 then
    inStream(port.h_outflow) else
    port.h_outflow;
```

Note that with stream variables the Boolean unknown for the flow direction appears in the energy balance, where the flow dependent value of the specific enthalpy is multiplied with the mass flow rate. This is why a reverting flow, i.e. m_flow crossing zero, does not cause jumping values in the model equations.

5.2 Transport model

The flows are defined with transport models. A basic model for isenthalpic flow is given below:

```
model IsenthalpicFlow
  "Flow model without storage of mass or
   energy, e.g. fitting or valve"

  replaceable package Medium =
Modelica.Media.Interfaces.PartialMedium;

  FluidPort port_a, port_b:

  Medium.ThermodynamicState state_a
    "State at port_a if inflowing";
  Medium.ThermodynamicState state_b
    "State at port_b if inflowing";

equation
  // Medium states for inflowing fluid
  state_a = Medium.setState_phX(
              port_a.p,
              inStream(port_a.h_outflow));
  state_b = Medium.setState_phX(
              port_b.p,
              inStream(port_b.h_outflow));

  // Mass balance
  0 = port_a.m_flow + port_b.m_flow;

  // Isenthalpic energy balance
  port_a.h_outflow =
       Medium.specificEnthalpy(state_b);
  port_b.h_outflow =
       Medium.specificEnthalpy(state_a);

  // (Regularized) Momentum balance
  port_a.m_flow = f(
              port_a.p, port_b.p,
              Medium.density(state_a),
              Medium.density(state_b));
end IsenthalpicFlow;
```

The flow model does not define own thermodynamic states as it has no storage of mass or energy. Instead the mass flow rate is defined for thermodynamic states seen through the ports for the case of inflow. These are generally the (transformed) states defined by connected volume models.

The flow model defines steady-state mass, energy and momentum balances. The function f gives the relationship between pressure drop and mass flow rate.

The use of the ThermodynamicState records state_a and state_b also ensures that appropriate equation systems are generated by a Modelica tool for medium models that do not use pressure and enthalpy as independent variables, such as gas models often using pressure and temperature instead (see section 4 above).

5.3 Sensor model

Ideal sensor models are used to pick up fluid properties, such as temperatures. They do not contribute to the fluid flow. A basic model for a temperature sensor is given below.

```
model TemperatureSensor
  "Ideal temperature sensor"

  replaceable package Medium =
Modelica.Media.Interfaces.PartialMedium;

  FluidPort port(m_flow(min=0))
    "Port with no outflow ever";

  Modelica.Blocks.Interfaces.RealOutput T
    "Upstream temperature";

equation
  T = Medium.temperature(
        Medium.setState_phX(port.p,
          inStream(port.h_outflow));

  port.m_flow = 0;

  port.h_outflow =
    Medium.specificEnthalpy(
      Medium.setState_pTX(
        Medium.reference_p,
        Medium.reference_T))
    "Never used, but seen in plots";
end TemperatureSensor;
```

The mass flow rate in the port is defined to be non-negative. This tells the translation tool that the outflow enthalpy defined by the sensor must not be used in any mixing equations, as outflow does never happen; see also Figure 4 above.

6 Conclusions

So far it has not been possible to agree on a common approach for the formulation of fluid ports. Existing approaches have either been based on control signals, which do not allow the device-oriented modeling of fluid systems, or on pairs of potential/flow variables, which lead to numerically unreliable equation systems.

The new stream variables allow the declarative formulation of fluid ports. Stream variables define the convective transport of specific quantities, such as specific enthalpy or chemical composition. The semantics is simple and can easily be supported by a Modelica tool.

Stream connectors have been standardized in Modelica 3.1. They are used in Modelica.Fluid. Currently Dymola 7.2 [2] and SimulationX 3.2 [8] support stream connectors. Other tool vendors already announced to support the concept, too.

The streams concept is a big step forward for fluid modeling in Modelica. However, stream connectors are not yet the ultimate solution for fluid modeling because there are still missing features:

- The used medium has currently to be defined for every component. It would be nicer if the medium was defined at one source and the medium definition would then be propagated through the connection structure.
- When components are connected together, the connection semantics ensures that the mass and the energy balance are fulfilled exactly (in the sense of “ideal” mixing). The momentum balance is exact in case of identical dynamic pressure for each of the connected flanges. If this approximation is not sufficiently accurate, an appropriate explicit junction model has to be used. It would be useful if also the momentum balance was automatically fulfilled when 3 or more components are connected together. This requires including information about the geometry of fluid flanges into the connector description.

7 Acknowledgments

Partial financial support of ABB and of DLR for this work within the ITEA2 project EUROSYSLIB is highly appreciated (BMBF Förderkennzeichen: 01IS07022F).

References

[1] R.P. Brent (1973): **Algorithms for Minimization without derivatives**. Prentice Hall, pp. 58-59.

[2] F. Casella, A. Leva (2003): **Modelica open library for power plant simulation: design and experimental validation**. Proceedings of the Modelica 2003 Conference, editor: P. Fritzson, Linköping, Sweden. www.modelica.org/events/Conference2003/papers/h08_Leva.pdf

[3] Dymola (2009). **Dymola Version 7.2**. Dassault Systèmes, Lund, Sweden (Dynasim). www.dymola.com

[4] H. Elmqvist, H. Tummescheit, M. Otter (2003): **Object-Oriented Modeling of Thermo-Fluid Systems**. Proceedings of the Modelica 2003 Conference, editor: P. Fritzson, Linköping, Sweden. www.modelica.org/events/Conference2003/papers/h40_Elmqvist_fluid.pdf

[5] R. Franke, F. Casella, M. Sielemann, K. Proelss, M.Otter, M. Wetter (2009): **Standardization of thermo-fluid modeling in Modelica.Fluid**. Proceedings of the Modelica 2009 Conference, editor: F. Casella, Como, Italy. www.modelica.org/events/modelica2009

[6] Modelica (2009): **Modelica Language Specification, Version 3.1**. www.modelica.org/documents/ModelicaSpec31.pdf

[7] H. Olsson, M. Otter, S.E. Mattsson, H. Elmqvist (2008): **Balanced Models in Modelica 3.0 for Increased Model Quality**. Proceedings of the Modelica 2008 Conference, editor: B. Bachmann, Bielefeld, Germany. www.modelica.org/events/modelica2008/Proceedings/sessions/session1a3.pdf

[8] SimulationX (2009). **SimulationX Version 3.2**. ITI, Dresden, Germany. www.simulationx.com

Appendix

A1 Determination of the Enthalpy for Ideal Mixing

In previous sections a central formula is an explicit equation to compute the ideal mixing enthalpy of an infinitesimally small connection point. In this section, this formula is derived. For simplicity, it is first derived at hand of 3 model components that are connected together, see Figure 2. The case for N connections follows correspondingly.

The energy and mass balance equations at the infinitesimally small control volume of the connection point of the three connected components are:

$$\begin{aligned}0 = \dot{m}_1 \cdot &\begin{cases} h_{mix} & \textbf{if } \dot{m}_1 > 0 \\ h_{outflow,1} & \textbf{if } \dot{m}_1 \le 0 \end{cases} \\ + \dot{m}_2 \cdot &\begin{cases} h_{mix} & \textbf{if } \dot{m}_2 > 0 \\ h_{outflow,2} & \textbf{if } \dot{m}_2 \le 0 \end{cases} \\ + \dot{m}_3 \cdot &\begin{cases} h_{mix} & \textbf{if } \dot{m}_3 > 0 \\ h_{outflow,3} & \textbf{if } \dot{m}_3 \le 0 \end{cases}\end{aligned} \tag{1a}$$

$$0 = \dot{m}_1 + \dot{m}_2 + \dot{m}_3 \tag{1b}$$

With the max(..) operator:

max(*a*,*b*) = **if** *a* > *b* **then** *a* **else** *b*

the balance equations above can be rewritten:

$$\begin{aligned}0 = &\max(\dot{m}_1,0)\cdot h_{mix} - \max(-\dot{m}_1,0)\cdot h_{outflow,1} \\ &+ \max(\dot{m}_2,0)\cdot h_{mix} - \max(-\dot{m}_2,0)\cdot h_{outflow,2} \\ &+ \max(\dot{m}_3,0)\cdot h_{mix} - \max(-\dot{m}_3,0)\cdot h_{outflow,3}\end{aligned} \tag{2a}$$

$$\begin{aligned}0 = &\max(\dot{m}_1,0) - \max(-\dot{m}_1,0) \\ &+ \max(\dot{m}_2,0) - \max(-\dot{m}_2,0) \\ &+ \max(\dot{m}_3,0) - \max(-\dot{m}_3,0)\end{aligned} \tag{2b}$$

Equation (2a) is solved for h_{mix}

$$h_{mix} = \frac{\begin{pmatrix} \max\left(-\dot{m}_1,0\right)\cdot h_{outflow,1} + \\ \max\left(-\dot{m}_2,0\right)\cdot h_{outflow,2} + \\ \max\left(-\dot{m}_3,0\right)\cdot h_{outflow,3} \end{pmatrix}}{\max\left(\dot{m}_1,0\right)+\max\left(\dot{m}_2,0\right)+\max\left(\dot{m}_3,0\right)}$$

Using (2b), the denominator can be changed to:

$$h_{mix} = \frac{\begin{pmatrix} \max\left(-\dot{m}_1,0\right)\cdot h_{outflow,1} + \\ \max\left(-\dot{m}_2,0\right)\cdot h_{outflow,2} + \\ \max\left(-\dot{m}_3,0\right)\cdot h_{outflow,3} \end{pmatrix}}{\max\left(-\dot{m}_1,0\right)+\max\left(-\dot{m}_2,0\right)+\max\left(-\dot{m}_3,0\right)}$$

This is an explicit formula to compute h_{mix} for three connected components. It is straightforward to generalize this formula to “n” connections:

$$h_{mix} = \frac{\sum\limits_{j=1\ldots n} \max(-\dot{m}_j,0)\cdot h_{outflow\,j}}{\sum\limits_{j=1\ldots n} \max(-\dot{m}_j,0)}$$

A2 Stream Variables in Hierarchical Models

The discussion of stream connectors has been restricted to connections on the same hierarchical level in this paper so far. In this section the handling of stream variables in hierarchical models is considered.

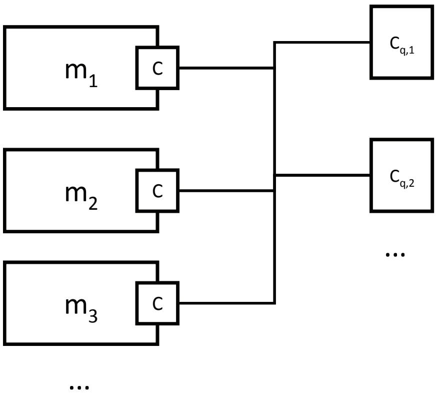


Figure 12: Exemplary fluid system with N=3 inside connectors and M=2 outside connectors

Figure 12 shows an exemplary hierarchical fluid system. If the inStream operator is used in one of the models $m_{i,}$ i=1…N, then the contributions of the inside and the outside connectors of the connection set need to be taken into account. This results in:

```
inStream(houtflow,i)
```

$$= \frac{\sum\limits_{j=1\ldots i-1,i+1\ldots N} h_{outflow\,j} \max(-\dot{m}_j,0) + \sum\limits_{k=1\ldots M} h_{outflow\,k} \max(\dot{m}_k,0)}{\sum\limits_{j=1\ldots i-1,i+1\ldots N} \max(-\dot{m}_j,0) + \sum\limits_{k=1\ldots M} \max(\dot{m}_k,0)}.$$

Note the opposite sign for mass flow rates in the outside connectors, following the sign convention for flow variables in hierarchical models.

The models m_i each explicitly define the stream variables `h_outflow` of their port c_i. These ports are inside connectors in the shown connection set. There are no explicit equations in the hierarchical model that define the stream variables h_outflow in the outside connectors $c_{q,k}$, k=1…M. A Modelica tool needs to define the stream variables of outside connectors c_q by establishing one mixing equation for each stream variable $h_{\text{outflow},q}$, q=1…M:

$$h_{outflfow,q} = \frac{\sum\limits_{j=1\ldots N} h_{outflow\,j} \max(-\dot{m}_j,0) + \sum\limits_{k=1\ldots q-1,q+1\ldots M} h_{outflow\,k} \max(\dot{m}_k,0)}{\sum\limits_{j=1\ldots N} \max(-\dot{m}_j,0) + \sum\limits_{k=1\ldots q-1,q+1\ldots M} \max(\dot{m}_k,0)}$$

This equation considers the contributions of inside and outside connectors in the connection set as well as the sign convention for flow variables in hierarchical models.

The treatment of zero flow for inside and outside connectors is the same as the treatment of zero flow in connections on the same hierarchical level (see Section 4.1).

Standardization of Thermo-Fluid Modeling in Modelica.Fluid

Rüdiger Franke, ABB AG, Germany – Ruediger.Franke@de.abb.com,
Francesco Casella, Politecnico di Milano, Italy – Casella@elet.polimi.it,
Michael Sielemann, DLR Institute for Robotics and Mechatronics – Michael.Sielemann@dlr.de,
Katrin Proelss, TU Hamburg-Harburg, Germany – K.Proelss@tu-harburg.de,
Martin Otter, DLR Institute for Robotics and Mechatronics, Germany – Martin.Otter@dlr.de,
Michael Wetter, LBNL, USA – MWetter@lbl.gov

Abstract

This article discusses the Modelica.Fluid library that has been included in the Modelica Standard Library 3.1. Modelica.Fluid provides interfaces and basic components for the device-oriented modeling of one-dimensional thermo-fluid flow in networks containing vessels, pipes, fluid machines, valves and fittings.

A unique feature of Modelica.Fluid is that the component equations and the media models as well as pressure loss and heat transfer correlations are decoupled from each other. All components are implemented such that they can be used for media from the Modelica.Media library. This means that an incompressible or compressible medium, a single or a multiple substance medium with one or more phases might be used with one and the same model as long as the modeling assumptions made hold. Furthermore, trace substances are supported.

Modeling assumptions can be configured globally in an outer System object. This covers in particular the initialization, uni- or bi-directional flow, and dynamic or steady-state formulation of mass, energy, and momentum balance. All assumptions can be locally refined for every component.

While Modelica.Fluid contains a reasonable set of component models, the goal of the library is not to provide a comprehensive set of models, but rather to provide interfaces and best practices for the treatment of issues such as connector design and implementation of energy, mass and momentum balances. Applications from various domains are presented.

Keywords: Modelica, thermo-fluid; one dimensional fluid flow, single substance, multi substance, trace substances

1 Introduction

Modelica.Fluid was announced together with Modelica Media at the Modelica'2003 conference, after the Modelica Association had made an attempt to standardize the most important interfaces and to provide good solutions for the basic problems of fluid modeling [1]. By now Modelica.Media is widely used. Regarding Modelica.Fluid it has not been possible to meet the ambitious goal for device-oriented modeling in realistic fluid applications so far. Still many different fluid libraries exist, each defining its own basics and each having its own downsides. Based on lessons learned, the Modelica Association has made a second attempt to standardize the basic fluid interfaces during the last year. It turned out that the regular Modelica connection approach with effort and flow variables is not sufficient for device-oriented fluid modeling. The newly introduced stream variables [3] represent properties transported by large-scale motion of a flow, such as specific enthalpy transported via convection by a mass flow. This makes it possible that the significant amount work that went into Modelica.Fluid finally yields fruits; 17 persons have contributed to the development during the last 6 years. Compared to previous beta releases, the code was reorganized and extended to cover the whole range from steady-state models to dynamic energy, mass and momentum balances. The fundamental balance equations for one-dimensional fluid flow and heat flow have been decoupled from the device models based on them. This not only simplifies the readability and understanding, but also the maintenance and further development of the library.

2 Library Structure and Interfaces

2.1 Library Structure

DOI: 10.3384/ecp09430077

Figure 1 shows the package structure of Modelica.Fluid.

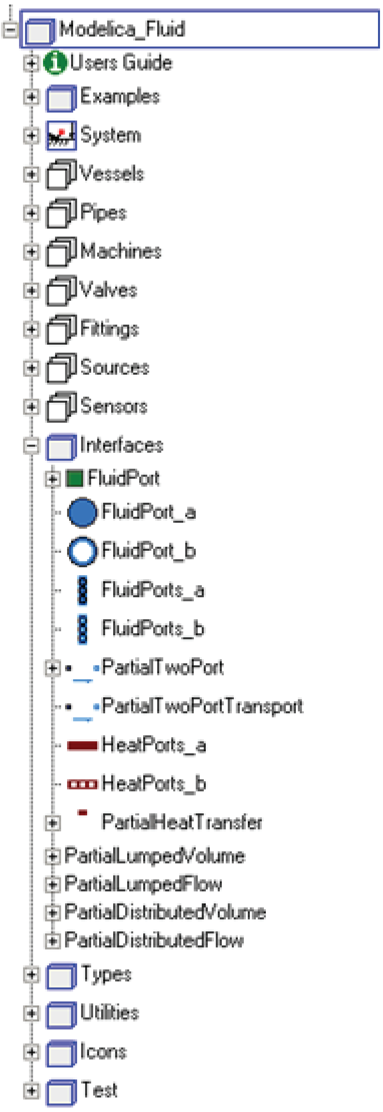


Figure 1: Library structure and interfaces of Modelica.Fluid

The overall library structure is oriented on fluid devices. The main packages are:

- **Vessels** are devices for storing fluid;
- **Pipes** are devices for transporting fluid;
- **Machines** convert between energy held in a fluid and mechanical energy;
- **Valves** regulate fluid flow;
- **Fittings** are adaptors for the connection of fluid components and for the regulation of fluid flow;
- **Sources** define boundary conditions for fluid models;
- **Sensors** are used to measure fluid properties and flow rates.

2.2 Interfaces

The Interfaces package defines both fluid connectors and partial base classes for the implementation of component models. The fluid connectors utilize the new Modelica stream variables, see [3], for the declarative modeling of convective transport of heat and substances. The fluid connector is defined as:

```
connector FluidPort
  replaceable package Medium;

  Medium.Pressure p;
  flow Medium.MassFlowRate m_flow;

  stream Medium.SpecificEnthalpy
    h_outflow;

  stream Medium.MassFraction
    Xi_outflow[Medium.nXi];

  stream Medium.ExtraProperty
    C_outflow[Medium.nC];
end FluidPort;
```

The medium characterizes the fluid passing the port. This ensures that only ports transporting the same fluid can be connected. Moreover, the medium defines numerical ranges, nominal values and equations of state appropriate for the fluid and its application domain at hand.

The purely hydraulic portion of fluid flow is described with the pressure and the mass flow rate in the port. The fluid may transport energy, modeled as specific enthalpy. The mass fractions describe the composition of multi-substance fluids. Extra properties can be used to model trace substances.

The library defines four different versions of the fluid port which all have the same semantics, but different icons. `FluidPort_a` and `FluidPort_b` are intended for fluid models with single flanges such as pipes. `FluidPorts_a` and `FluidPorts_b` shall be used for components with individual flanges for each connection made to them, such as vessels.

2.3 Partial Base Classes

Base classes are used to unify the implementations of similar models and equations. There finds two different kinds of base classes, which are usually combined via multiple inheritance:

- Shell models define interfaces of flow devices, such as fluid ports;
- Balance models define only balance equations, such as the momentum balance or heat transfer with the environment. No connectors are instantiated herein.

The balance models are designed such that they can either be used with inheritance or with replaceable models.

A typical concrete fluid model extends from two base classes: a shell model and a balance model; see e.g. `Fittings.SimpleGenericOrifice` extending from `PartialTwoPortTransport` and from `PartialLumpedFlow`.

Replaceable models are typically used for heat transfer with the environment; see e.g. `Vessels.-BaseClasses.PartialLumpedVessel` conditionally enabling a replaceable heat transfer model with the flag `use_HeatTransfer` on the Assumptions tab.

2.3.1 Shell models

`PartialTwoPort` provides two fluid ports `port_a` and `port_b`, which are common to many components like pipes, machines and valves. `PartialTwoPortTransport` extends from `PartialTwoPort` and additionally defines steady-state mass and substance balances for component models without storage of fluid. An extending class still needs to define the energy balance (correspondingly in steady-state), which might, for example, be based on an isenthalpic state transformation for fittings or a polytropic one for machines.

2.3.2 Balance models

The balance models predefine equations for dynamic and for steady-state simulation. Moreover the initialization and the connection to a medium model are treated in these base classes.

The equations are formulated with variables that represent generic boundary and source terms of the corresponding balances. Extending classes need to define the boundary and source terms.

`PartialDistributedVolume` defines the mass and the energy balance for one-dimensional distributed flow models. The model equations are formulated for `n` flow segments, which are characterized with the variable vectors

```
SI.Volume[n] fluidVolumes;
SI.Mass[n] ms;
SI.Energy[n] Us;
```

for the volume, the mass and the internal energy of the fluid per segment. Moreover

```
SI.Mass[n,Medium.nXi] mXis;
SI.Mass[n,Medium.nC] mCs;
```

model the substance masses and the trace substance masses if a multi component medium is used.

The mass balance is defined as

```
der(ms) = mb_flows;
```

with mb_flows[n] a vector of n boundary and source terms.

The energy balance is defined as

```
der(Us) = Hb_flows + Wb_flows + Qb_flows;
```

distinguishing enthalpy flow rates Hb_flows[n], mechanical power Wb_flows[n] and heat flow rates Qb_flows[n] for boundary and source terms.

The substance mass balances and the trace substance mass balances are defined as

```
der(mXis) = mbXi_flows;
der(mCs) = mbC_flows;
```

with mbXi_flows[n,Medium.nXi] and mbC_flows[n,Medium.nC] the boundary and source terms for mass flow rates of independent substances and trace substances, respectively.

The separate base class P`artialDistributedFlow` defines the momentums

```
SI.Momentum[m] Is;
```

for `m` flow segments with the balance equation

```
der(Is) = Ib_flows - Fs_p - Fs_fg;
```

using the boundary and source terms `Ib_flows[m]` for flow of momentum across boundaries, the pressure forces `Fs_p[m]`, and the friction and gravity forces `Fs_fg[m]`.

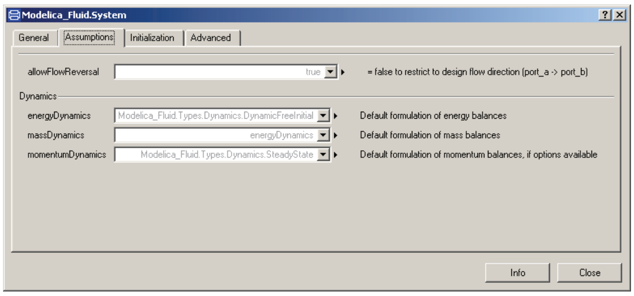
The use of a different base class for the momentum balance allows the flexible utilization of different discretization schemes than for the mass and the energy balances. For instance, following the staggered grid approach, m=n-1 momentum balances are defined between n fluid volumes.

The additional base classes `PartialLumpedVolume` and `PartialLumpedFlow` provide lumped versions of mass, energy and momentum balances.

An interface for heat transfer with the environment is defined in the `PartialHeatTransfer` model. It predefines a vector of heat flows `Q_flows[n]` through a vector of `heatPorts[n]`. Moreover `PartialHeatTransfer` provides a simple consideration of heat losses to the ambient by using the parameter *k* as coefficient of heat transfer. A concrete heat transfer model extending from `PartialHeatTransfer` needs to define `Q_flows` based on the thermodynamic states, flow regime and surface areas of `n` flow segments.

2.4 System Wide Properties

Due to common interfaces and base classes, many models contain similar configuration parameters. It would be tedious to open all individual configuration dialogs in order to change the same setting in all models. This is why Modelica.Fluid defines a System object providing global defaults for all components in a fluid model.



Modelica_Fluid.System

allowFlowReversal: true — = false to restrict to design flow direction (port_a -> port_b)

Dynamics

energyDynamics: Modelica_Fluid.Types.Dynamics.DynamicFreeInitial — Default formulation of energy balances

massDynamics: energyDynamics — Default formulation of mass balances

momentumDynamics: Modelica_Fluid.Types.Dynamics.SteadyState — Default formulation of momentum balances, if options available

Figure 2: Parameter dialog of the System object

Figure 2 shows the configuration of modeling assumptions. The system wide default values cover:

- General parameters that define ambient pressure, temperature and the force of gravity.
- Assumptions that declare dynamics options, ranging from steady-state to dynamic with fixed or free initial values. These assumptions are made per balance type. A typical fluid model would have dynamic energy and mass balances together with steady-state momentum balances. Moreover the analysis of a fluid model can be restricted to only cover the design flow direction from `port_a` to `port_b`.
- Initialization with common start values.
- Advanced settings to provide default values for mass flow rates and pressure drops that shall be considered small for the numerical analysis of reverting flow conditions.

These system-wide settings can, however, be overridden at the component level to allow, for example, the use of steady-state and dynamic components within the same system model.

3 Rigorous Implementation of One-Dimensional Fluid Flow

The Pipes sub-package provides a rigorous implementation for one-dimensional thermo-fluid flow. The governing equations of pipe flow are the mass balance

$$\underbrace{\frac{\partial(\rho A)}{\partial t}}_{der(ms)} = -\underbrace{\frac{\partial(\rho A v)}{\partial x}}_{mb\_flows},$$

the energy balance

$$\underbrace{\frac{\partial(\rho u A)}{\partial t}}_{der(Us)} = -\underbrace{\frac{\partial\left(\rho v\left(u+\frac{p}{\rho}\right)A\right)}{\partial x}}_{Hb\_flows} + \underbrace{vA\frac{\partial p}{\partial x} + vF_F}_{Wb\_flows} + \underbrace{\frac{\partial}{\partial x}\left(kA\frac{\partial T}{\partial x}\right) + \dot{Q}_e}_{Qb\_flows}$$

and the momentum balance

$$\underbrace{\frac{\partial(\rho v A)}{\partial t}}_{der(Is)} = -\underbrace{\frac{\partial(\rho v^2 A)}{\partial x}}_{Ib\_flows} - \underbrace{A\frac{\partial p}{\partial x}}_{Fs\_p} - \underbrace{\left(F_F + A\rho g\frac{\partial z}{\partial x}\right)}_{Fs\_fg}.$$

Below the curly braces the names of corresponding variables that are predefined in the base classes are given; see Section 2.3.2.

Here t is the time and x is the independent spatial coordinate along the direction of fluid flow. The fluid is characterized by the density ρ, the specific internal energy u, the temperature T, the velocity v of the flow, and the thermal conductivity k. The flow device is characterized by the area A perpendicular to the direction x, the Fanning friction factor F_F, and the heat flow $\dot{Q}_e$ exchanged with the environment. Moreover g is the constant of gravity and z is the spatial coordinate along the gravity.

Note that the energy balance does not contain contributions of the momentum. The kinetic energy of fluid flow is treated by the momentum balance; see [1].

`Pipes.BaseClasses.PartialTwoPortFlow` implements the balances. It extends from `Interfaces.PartialDistributedVolume` and applies the finite volume approach to the discretization along the spatial coordinate x with `n` flow segments. Figure 3 gives a graphical overview.

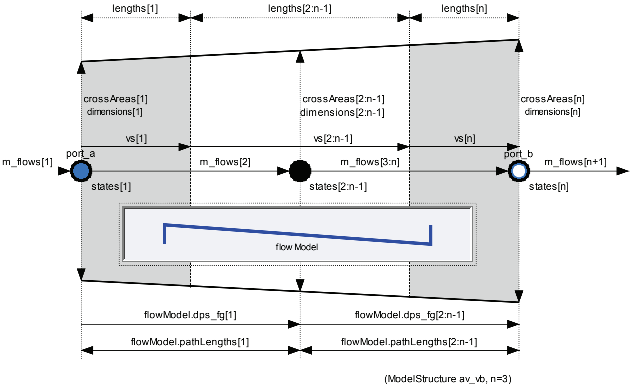


Figure 3: Overview of PartialTwoPortFlow

The replaceable flowModel defines m=n-1 momentum balances. Concrete implementations of flow models are provided in the subpackage Pipes.-BaseClasses.FlowModels. They include laminar, turbulent and detailed one-phase flow. The flow models are either parameterized with pipe geometries or, alternatively, with nominal flow conditions.

The overall model structure follows the staggered grid approach with nNodes=n flow segments. Momentum balance equations are formulated for neighboring flow segments. There are different choices for the formulation of the boundary conditions at the fluid ports, which can be configured with the parameter ModelStructure on the Advanced tab:

- av_vb: Symmetric setting with nNodes-1 momentum balances between nNodes flow segments. The ports port_a and port_b expose the first and the last thermodynamic state, respectively. Connecting two or more flow devices therefore may result in high-index DAEs for the pressures of connected flow segments;
- a_v_b: Alternative symmetric setting with nNodes+1 momentum balances across nNodes flow segments. Half momentum balances are placed between port_a and the first flow segment as well as between the last flow segment and port_b. Connecting two or more flow devices therefore results in algebraic pressures at the ports. The specification of good start values for the port pressures is essential for the solution of large nonlinear equation systems;
- av_b: Unsymmetric setting with nNodes full momentum balances, one between the n-th volume and port_b, potential pressure state at port_a;
- a_vb: Unsymmetric setting with nNodes full momentum balances, one between the first volume and port_a, potential pressure state at port_b.

The model structure influences the equations that result from the interconnection of multiple flow models. The connection of models that expose steady-state momentum balances through their ports (model structure a_v_b) generally results in a nonlinear equation system for the connection point. The connection of models that expose states of fluid volumes with storage through their ports (model structure av_vb) results in high-index DAEs for pressure states. Note that the states representing enthalpy or temperature still stay separated due to the used stream connectors.

A specific pipe flow model still needs to add the source terms Qb_flow and Wb_flow for heat and work exchanged with the environment. The component model Pipes.DynamicPipe extends from PartialTwoPortFlow and defines these terms.

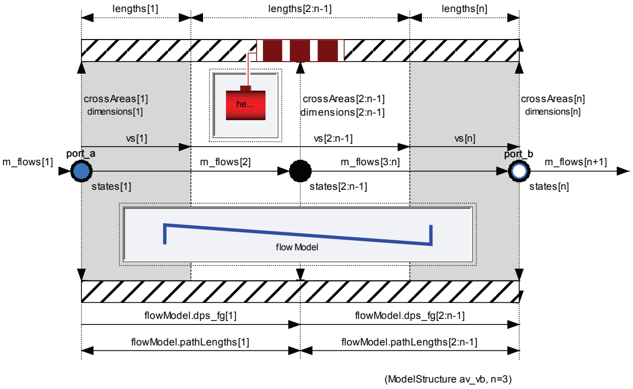


Figure 4: Overview of Pipes.DynamicPipe that extends from Pipes.BaseClasses.PartialTwoPortFlow and defines the terms Wb_flows and Qb_flows

Figure 4 gives an overview of Pipes.DynamicPipe. The term Wb_flow has been implemented according to the energy balance given above. The term Qb_flow is defined by a replaceable wall heat transfer model. Some predefined choices can be found in Pipes.BaseClasses.HeatTransfer.

4 Treatment of Wall Friction and Flow Reversal

In a one-dimensional model of fluid dynamics, the wall shear stress cannot be established directly as product of dynamic viscosity and gradient of fluid velocity perpendicular to the fluid surface, as all changes perpendicular to the bulk flow velocity are not resolved in such a model. Consequently, the transfer of momentum between the fluid and an adjacent surface is modeled using a lumped approach. The pressure drop due to wall friction is a computed as product of dynamic pressure and a loss factor ζ ,

$$\Delta p_t = \zeta \frac{\rho v|v|}{2}.$$

If a component models a domain of zero “length” (no extension with respect to the flow direction) then the loss factor is established directly via an appropriate correlation (e.g. a valve). For pipes and other components of finite length, the loss factor is defined using a wall friction coefficient λ, pipe length *l*, and diameter *d*,

$$\zeta = \lambda \frac{l}{d}.$$

All correlations have been regularized to be continuous and have a finite, non-zero, and smooth first derivative. The functions are all guaranteed to be strictly monotonic, which guarantees that a unique inverse exists. Wherever possible, the correlations are provided in two versions, one to calculate mass flow as a function of pressure drop and one to compute pressure drop from mass flow rate. Furthermore, most functions contain a single statement only such that Modelica tools may inline them to avoid function call overhead.

4.1 Pipe Wall Friction

In case of pipe models, the wall friction correlations are provided in form of replaceable packages defined in `Pipes.BaseClasses.WallFriction`. A replaceable instance of the corresponding base class is added to the staggered grid momentum balance in `Pipes.BaseClasses.FlowModels.PartialGenericPipeFlow`. Implementation details are given in [1], regularization is discussed in [2]. Figure 5 provides a Moody chart of an exemplary pipe wall friction correlation.

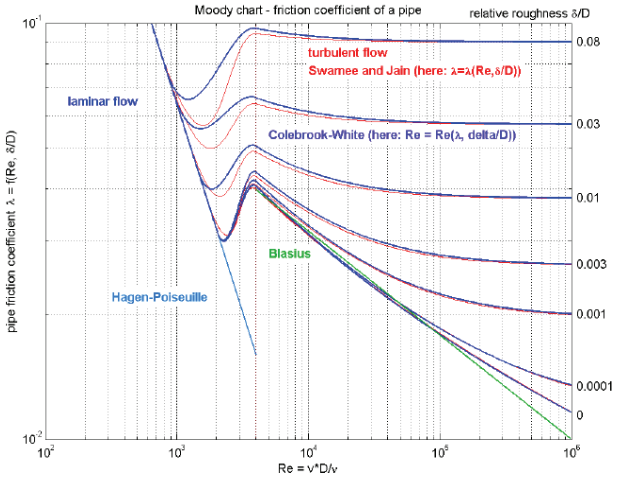


Figure 5: Moody chart of correlations provided in Pipes.BaseClasses.WallFriction

4.2 Pipe Wall Friction and Static Head

When static head is of relevant order of magnitude in a pipe (e.g. natural circulation), properly modeling pipe capacity is important to have a well-defined density and therefore pressure difference due to static head $\Delta p_{sh} = \rho g z$. This is supported by the pipe models using an average of the control volume densities to the left and the right of each momentum balance. In some cases it is reasonable to neglect the pipe capacity however, e.g. when the cost of the associated state variables is high. In this simplification, each pipe segment is filled instantaneously with fluid of the upstream density. The resulting exact solution is not monotonic and thus not bijective if the rate of change of elevation *z* along the domain has the same sign as the rate of change of density. This is shown in blue in figure 6 below.

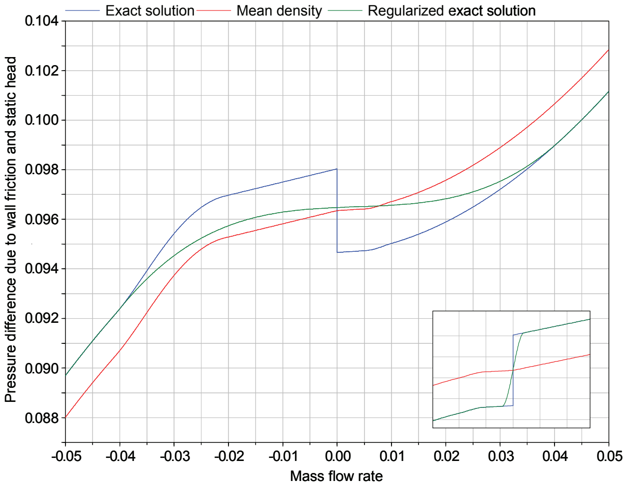


Figure 6: Pressure difference due to wall friction and static head over mass flow rate, non-bijective and bijective case (inset)

This problem cannot be regularized using the Fritsch-Carlson splines employed in [2]. Instead, a modified algorithm was developed based on [5]. It is available under `Utilities.regFun3()` and used to regularize this problem.

Figure 6 illustrates the different options for this particular problem. The exact solution is shown in blue, a simplified approach using a mean density independent of the actual flow direction in red, and the regularized exact version in green. Note in particular the large difference in the necessary width of the regularization interval required to yield a bijective approximation.

4.3 Fittings with Non-Constant Cross Section

According to their definition, loss factors ζ establish a pressure difference in total pressure. For several components, not only the density but also the cross section area is identical on both ends. In this case the dynamic pressure is constant over the component and therefore the difference in total pressure equals the difference in static pressure, i.e. the difference between the port pressures. Additionally to components falling in the latter category, Modelica.Fluid provides component models for fittings with cross section area changing over the device.

Herein, we discuss this type of components at the example of an adaptor model in which the cross section changes abruptly, `Fittings.AbruptAdaptor`. In [6], two different correlations are given, one for a sudden expansion, and a second one for a sudden contraction, which were combined into the given model. Similar to other wall friction correlations for devices with changing cross section area, this type of correlation does not refer to *upstream* density and velocity. Instead, the pressure drop is defined as loss factor times dynamic pressure at the *smallest* cross section area.

Based on the definition of total pressure $p_t = p + 1/2\rho v^2$ and the definition of the loss factor ζ we arrive at the following equation relating the *static* pressure drop and the mass flow rate.

$$\Delta p = \frac{1}{2}\dot{m}^2\left(\frac{\pm\zeta}{\rho_{sca}A_{sca}^2} - \frac{1}{\rho_a A_a^2} + \frac{1}{\rho_b A_b^2}\right)$$

Herein, indices *a* and *b* each refer to one of the ends of the adaptor. Index *sca* refers to the end with smaller cross section area. Furthermore, the positive sign refers to the case with positive mass flow rate.

When substituting the particular correlations mentioned above, the parenthesis turns out to be negative independently of the flow direction. Therefore, the function is not bijective and therefore cannot be inverted.

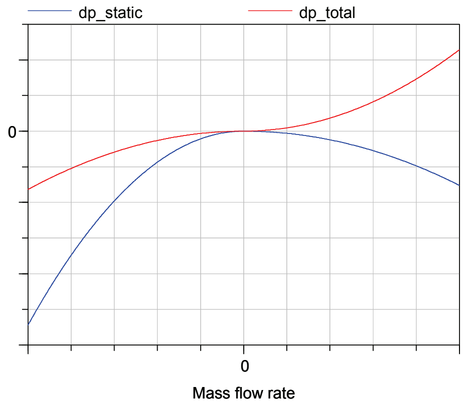


Figure 7: Pressure difference of static pressure and total pressure for flow reversal in an adaptor

Figure 7 provides exemplary results for a particular parametrization of the adaptor (d_a=0.1m, d_b=0.2m). As can be seen in the illustration, the pressure drop in static pressure is not monotone, only the pressure drop in total pressure is.

5 Heat transfer

Due to the underlying one-dimensional approach are temperature gradients perpendicular to the main flow not resolved. Therefore, heat transport between the fluid bulk flow and its environment, as for example a pipe wall, is described by a lumped approach based on a heat transfer coefficient *k*, the heat transfer area A_h and the driving temperature difference between bulk flow and the wall inside.

$$\dot{Q} = k \cdot A_h \cdot (T_{port} - T_{fluid})$$

The computation of the transport coefficient may range from simple constant parameters to complex correlations depending on fluid properties, geometry information and flow position.

The resulting heat flow is used in the energy balance, which looks the same for a wide range of specific component models, while the heat transfer correlation itself needs to be very flexible even at the very top level of a component model which is ready to be used in a larger system. For this reason the heat transfer correlation is implemented as a replaceable model with a defined public interface and can be propagated across several hierarchical levels (see e.g. Fig. 4).

Geometry parameters that are specific to a certain correlation can be entered as a class modification at the top level.

`DistributedPipe` and `LumpedVessel` use each their own constraining type, based on a common model, which are the starting points for example models in the library as well as possible user extensions. Additional heat transfer resistances as walls and insulation materials can be added in each correlation as an option. Thermal capacitance must be covered by an external component connected to the heat ports.

The following correlations are already implemented in the library:

- Zero resistance for lumped and distributed flow. The fluid temperature is then directly exposed to the heat port, which may cause higher index systems if a fixed temperature boundary condition is directly connected.
- Constant heat transfer coefficient for lumped and distributed flow: The heat transfer coefficient is entered as a constant parameter in the user interface
- Nusselt-number (Nu) based, forced convection driven heat transfer for distributed pipe flows: The heat transfer coefficient is deter-

mined from fluid flow properties according to a correlation for laminar and turbulent pipe flow in [7].

The last model inherits from a template class which provides an easy to use interface for other Nu-based correlations, which are e.g. used in heat exchangers.

All models and their corresponding interface classes are found in their respective component category folder, `Vessels.BaseClasses.HeatTransfer` and `Pipes.BaseClasses.HeatTransfer`.

6 Examples

Modelica.Fluid contains a number of simple examples from various application domains. They are intended for the exploration of the library and may serve as a starting point for own developments.

6.1 PumpingSystem

This example models a supply system for drinking water; see Figure 8. It features lumped mass and momentum balances as well as an embedded control.

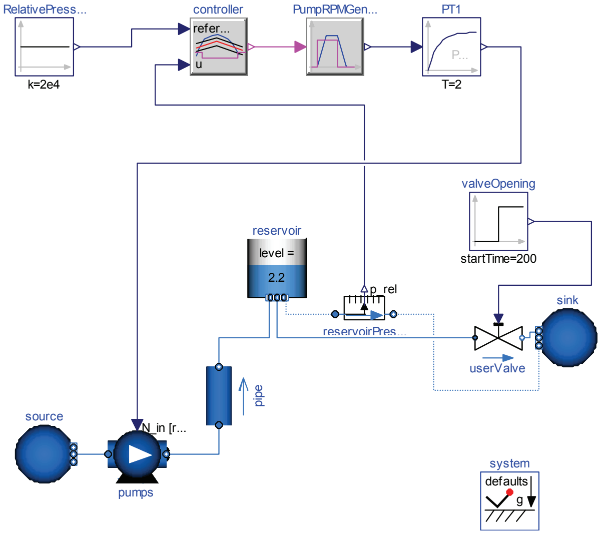


Figure 8: PumpingSystem example

Water is pumped from a source by a pump (fitted with check valves) through a pipe whose outlet is 50m higher than the source, into a reservoir. The users are represented by an equivalent valve, connected to the reservoir.

The water controller is a simple on-off controller, regulating on the gauge pressure measured at the base of the tower; the output of the controller is the rotational speed of the pump, which is represented by the output of a first-order system.

6.2 HeatingSystem

This example models a home heating system; see Figure 9. It features a closed flow cycle and idealized embedded controls in the pump and in the heater.

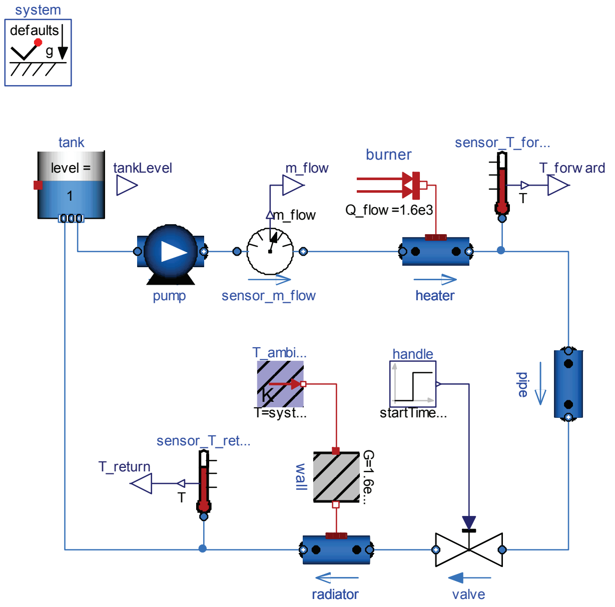


Figure 9: HeatingSystem example

After 2000s of simulation time the handle valve fully opens. A simple idealized control is embedded into the respective components, so that the heating system can be regulated with the valve: the pump controls the pressure, the burner controls the temperature.

The simulation can be turned from dynamic to steady-state or, steady-state initial conditions by selecting the `energy-`, `mass-`, and `momentumDynamics` under the Assumptions tab of the global system object. The selection gets propagated to all component models.

It is important to note that the `massDynamics` of the tank is set to `FixedInitial`, in order to obtain sensible initial conditions for the closed flow cycle.

Also note that the heat transfer model of the tank is enabled, in order to define a thermal insulation against the environment. This way the steady-state initial temperature of the tank is also defined in the case of zero flow.

6.3 DrumBoiler

This example models the drum boiler of a power plant; see Figure 10. It features two phase flow. The locally defined evaporator component explicitly models the phase change from water to steam.

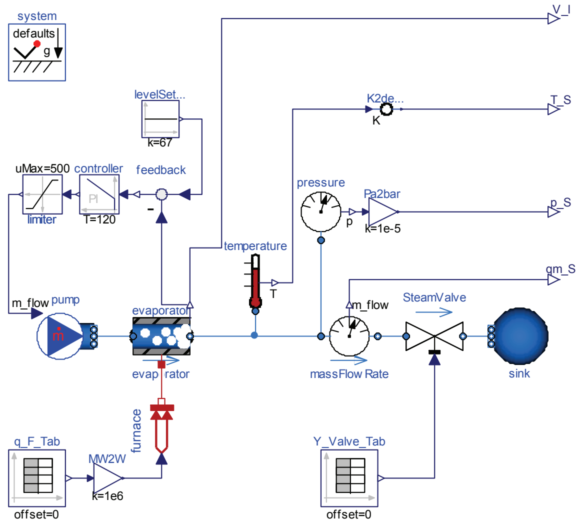


Figure 10: DrumBoiler example

During simulation, the boiler is ramped up from standstill to full load. The prescribed control of fuel flow rate and steam valve position is specified in two lookup tables.

More details about this example, including also the calculation of an optimal start-up control, can be found in [4].

6.4 BranchingDynamicPipes

This example models long pipes; see Figure 11. It features dynamic momentum balances subject to flow reversal.

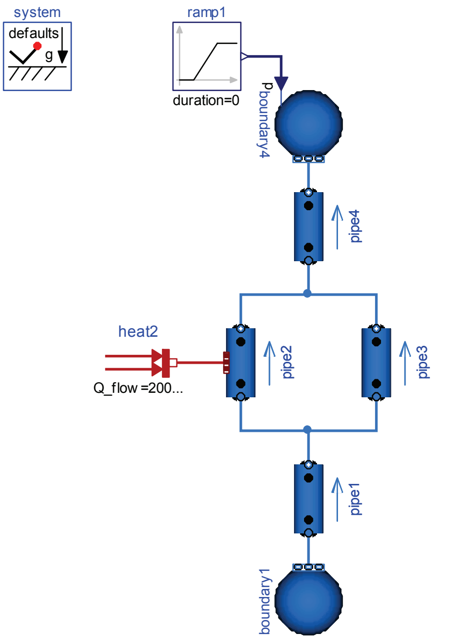


Figure 11: BranchingDynamicPipes example

Applying the default model structure `av_vb` of the `DynamicPipe` models, the idealized junctions are treated as high-index DAEs. The pressure states of connected pipes get lumped together – there is no need to explicitly introduce junction models into the connection points.

At simulation time 2s, the pressure of boundary4 jumps, which causes a pressure wave and flow reversal.

6.5 TraceSubstances

This example models an air conditioning system controlling the CO2 content in a room. It features trace substances.

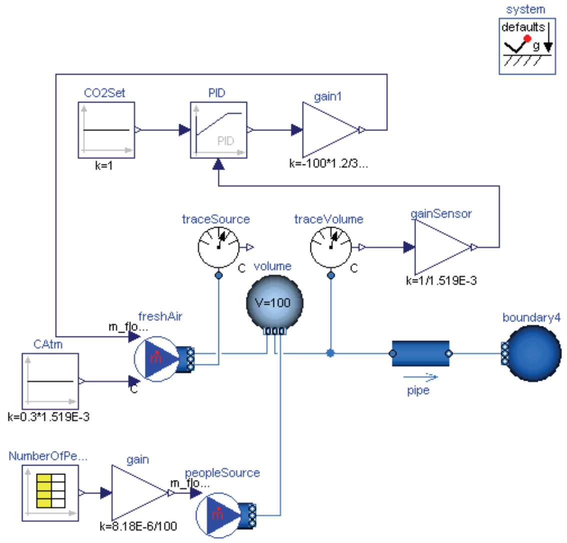


Figure 12: Example model for trace substances.

Figure 12 shows the example model `Fluid.Examples.TraceSubstances.RoomCO2WithControls`. It models a room volume with a CO_2 source and a fresh air supply with feedback control. The CO_2 emission rate is proportional to the room occupancy, which is defined by a schedule. The fresh air mass flow rate is controlled such that the room CO_2 concentration does not exceed 1000 PPM (=1.519E-3 kg/kg). In the model, the implementation of the feedback control normalizes the measured CO_2 concentration so that the controller tracks a set point of one.

The fresh air supply has a CO_2 concentration of 300 PPM, which corresponds to a typical CO_2 concentration in the outside air. The CO_2 emission from the occupants is implemented using a mass flow source. Depending on activity and size, the CO_2 emission rate per person is about 8.18E-6 kg/s. In the model, this value is multiplied by the number of occupants that is read from a time table. Notice that when modeling CO_2 emitted by people, we want to add CO_2 to the room, but no bulk mass flow rate. However, the CO_2 source model at the bottom of Figure 12 outputs a non-zero air mass flow rate with some prescribed CO_2 concentration. Since the air mass flow rate associate with the CO_2 source model contributes to the volume's energy balance, this mass flow rate needs to be kept small. Thus, in the source model, the CO_2 concentration is set to *C=100 kg/kg*, whereas the output of the gain is scaled such that the mass flow rate is *m_flow = 1/100 * nPeo * 8.18E-6 kg/(s*person)*, where *nPeo* is the number of people in the room. This results in an air mass flow rate that is about 5 orders of magnitudes smaller than the mass flow rate of the fresh air supply, and hence its contribution to the volume's energy balance is negligible.

7 Conclusions

Modelica.Fluid attempts to standardize the most important interfaces and to provide good solutions for the basic problems of fluid modeling. Its design is a collaborative effort; 17 persons have contributed to the development during the last 6 years.

Modelica.Fluid uses stream connectors to model the convective transport of energy and substances. It provides rigorous implementations of mass, energy and momentum balances for one-dimensional thermo-fluid flow. Moreover the library provides a detailed implementation of pipe friction.

Modelica.Fluid contains a reasonable set of simple models of flow device, such as vessels, pipes and pumps. The medium models are treated separately in Modelica.Media. They are freely configurable for each device model, covering compressible and incompressible single- and multi-substance media as well as trace substances.

Several examples from various application domains demonstrate the application of Modelica.Fluid. The library does not attempt to provide a complete set of device models for all of these application domains. Application specific libraries shall base on Modelica.Fluid. This simplifies model exchange and the sharing of knowledge.

Modelica.Fluid supports dynamic and steady-state simulations for one and the same model by specifying global assumptions in a system object.

Modelica.Fluid does not yet cover multi-phase flow. Further restrictions are seen in the connection approach that does not allow the propagation of medium models and geometrical information, such as heights and diameters, through fluid ports.

The future development of Modelica.Fluid will be driven by its applications and by the people who contribute.

References

[1] H. Elmqvist, H. Tummescheit, M. Otter: Object-Oriented Modeling of Thermo-Fluid Systems, Modelica 2003 Conference, Linköping, November 2003. www.modelica.org/events/Conference2003/papers/h40_Elmqvist_fluid.pdf

[2] F. Casella, M. Otter, K. Proelss, C. Richter, H. Tummescheit: The Modelica Fluid and Media Library for Modeling of Incompressible and Compressible Thermo-Fluid Pipe Networks, Modelica 2006 Conference, Vienna, September 2006. www.modelica.org/events/modelica2006/Proceedings/sessions/Session6b1.pdf

[3] R. Franke, F. Casella, M. Otter, M.Sielemann, H.Elmqvist, S.E. Mattson, H. Olsson: Stream Connectors, Modelica 2009 Conference.

[4] R. Franke, K. Krüger, M. Rode: On-line Optimization of Drum Boiler Startup, Modelica 2003 Conference, Linköping, November 2003. www.modelica.org/events/Conference2003/papers/h29_Franke.pdf

[5] M. G. Gasparo and R. Morandi: Piecewise cubic monotone interpolation with assigned slopes. Computing 46, pages 355-365, 1991.

[6] I.E. Idelchik: Handbook of Hydraulic Resistance. Jaico Publishing House, 2005.

[7] Verein Deutscher Ingenieure (1997): VDI Wärmeatlas. Springer Verlag, Ed. 8, 1997.

Acknowledgements

The design of the Modelica.Fluid library is a collaborative effort and many have contributed. The authors like to particularily thank Jonas Eborn, Hilding Elmqvist, Manuel Gräber, Carsten Heinrich, Kilian Link, Christoph Richter, and Hubertus Tummescheit for their valuable contributions.

Partial financial support by ABB and by DLR for this work within the ITEA project EUROSYSLIB is highly appreciated (BMBF Föderkennzeichen: 01IS07022F). This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231.

FluidDissipation for Applications - A Library for Modelling of Heat Transfer and Pressure Loss in Energy Systems

Thorben Vahlenkamp Stefan Wischhusen
XRG Simulation GmbH,
Harburger Schlossstraße 6-12, 21079 Hamburg
{vahlenkamp, wischhusen}@xrg-simulation.de

Abstract

The results of a free and open source MODELICA library for convective heat transfer and pressure loss calculations of energy devices called FLUIDDISSIPATION will be presented based on the goals shown at the MODELICA Conference 2008 [1]. The FLUIDDISSIPATION library is developed within the European research project EuroSysLib.

The library delivers a broad range of verified and validated correlations describing convective heat transfer and pressure loss of fluids in energy devices. These correlations are numerically optimised to provide efficient and stable transient simulations. The library also provides convective heat transfer models and flow models of most heat transfer and pressure loss correlations using the also free and open source MODELICA_FLUID library [2] as thermo-hydraulic framework for system simulation.

Scope, implementation concept, numerical challenges, verification and validation of the FLUIDDISSIPATION library will be exemplarily described (e.g. for convective heat transfer and pressure loss of two-phase flow).

Industrial applications for thermo-hydraulic system simulation (e.g. air distribution circuit for supplemental cooling, aircraft engine fuel feeding system) are presented using FLUIDDISSIPATION correlations implemented within MODELICA_FLUID models. A detailed documentation is available in the library itself.

Keywords: convective heat transfer; pressure loss; dissipation; MODELICA_FLUID

1 Introduction

Energy conversion in any thermo-hydraulic process is declined due to unwanted heat transfer (as a result of temperature difference) and pressure losses (as a result of local losses due to geometry and/or frictional losses) of a working fluid [3]. Both physical phenomena increase entropy and decrease exergy of an energy system. Therefore an amount of energy of a working fluid to be transformed into mechanical work is dissipated.

These fluid dissipation effects (e.g. pressure loss of pipe network) have to be compensated by higher energy supply of other system components (e.g. delivery height of pumps). A reduction of fluid dissipation effects is a way to optimise the efficiency of a thermo-hydraulic process with a corresponding minimisation of operation costs.

Thus modelling fluid dissipation effects are necessary for thermo-hydraulic processes to evaluate existing energy systems and to find out optimising potentials.

The following sections deliver insight into the open source MODELICA library FLUIDDISSIPATION:

- **Library content** with verified and validated correlations describing the dissipation effects due to convective heat transfer and pressure losses of fluids used in energy devices
- **Implementation** of the library describing its functional approach, numerical optimisations and the creation of convective heat transfer and flow models (out of MODELICA_FLUID as thermo-hydraulic framework and the dissipation correlations)
- **Verification and validation** of the library
  - Convective heat transfer is exemplarily validated for an even gap as well as for condensation and boiling of two-phase flow

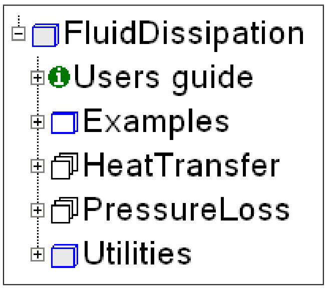
DOI: 10.3384/ecp09430012

  - Pressure loss is exemplarily validated for a curved bend as well as for two-phase flow

- **System simulation** of an environmental control system, a supplemental cooling system and an aircraft engine feeding system as industrial application

2 Library content

The FLUIDDISSIPATION library consists of five major packages shown in Fig.1.



FluidDissipation
Users guide
Examples
HeatTransfer
PressureLoss
Utilities

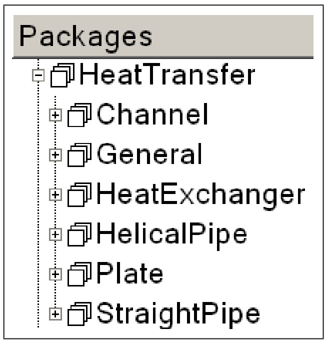
Figure 1: Package content of the FLUIDDISSIPATION library.

The **USERS GUIDE** section provides a GETTINGSTARTED for the usage of the FluidDissipation library and reports important changes w.r.t. previous versions.

The **EXAMPLES** package is divided into three packages. In the VERIFICATIONS package all heat transfer and pressure loss correlations are verified or validated with literature. In the APPLICATION package these correlations are implemented into MODELICA_FLUID as thermo-hydraulic framework. Examples of convective heat transfer models and flow models are provided. In the TESTCASES package these MODELICA_FLUID models are used for system simulation of test applications (e.g. aircraft engine feeding system).

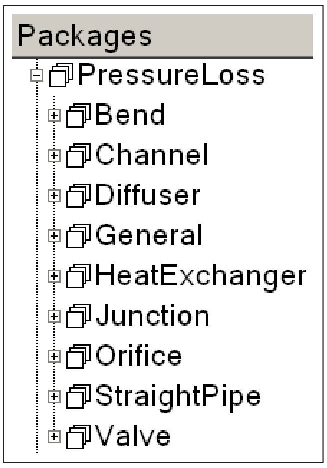
The **HEATTRANSFER** section consists of 6 major energy device packages shown in Fig.2. Each energy device package contains its corresponding correlations for the convective heat transfer coefficient. Every package can provide several correlations for this device (best seen in the library itself). For example a channel can be calculated with a constant wall temperature or constant heat flow rate as boundary condition.

The **PRESSURELOSS** section consists of 9 major energy device packages shown in Fig.3. Each energy device package contains its corresponding correlations for the pressure loss coefficient. Every package can provide several correlations for this device (best seen in the library itself). For example a bend can be calculated with a curved or an edged turning.



Packages
HeatTransfer
Channel
General
HeatExchanger
HelicalPipe
Plate
StraightPipe

Figure 2: Package content of convective heat transfer devices in the FLUIDDISSIPATION library.



Packages
PressureLoss
Bend
Channel
Diffuser
General
HeatExchanger
Junction
Orifice
StraightPipe
Valve

Figure 3: Package content of pressure loss devices in the FLUIDDISSIPATION library.

3 Implementation of library

3.1 Functional approach

The intention of the FLUIDDISSIPATION library is to create a base library for convective heat transfer and pressure loss calculations as dissipation effects of fluid flow. To ensure its interoperability with other thermo-hydraulic libraries (e.g. MODELICA_FLUID) the following main aspects have been realised during modelling:

- Independence of thermo-hydraulic framework (applicable with other libraries)
- Literally use of function calls
- Input and/or output arguments of function calls are delivered by records (e.g. fluid properties, geometry parameters)

Besides its interoperability this functional approach considers the ease of use at a maximum of numerical efficiency during system simulation. For example each function call only needs its target variables (e.g. mass flow rate, heat transfer coefficient or pressure loss) and one record with all other variables as input arguments. The intention for the future is to separate the dynamic variables of the input record from the constant ones (e.g. fluid properties from geometry parameters) due to numerical reasons (see 3.2). In addition functions using output records to monitor extra information like a failure status shall not be used for the integration into fluid models. These functions are best used for functions calls only.

The principles of how to implement convective heat transfer or pressure loss functions from FLUIDDISSIPATION into other thermo-hydraulic libraries can be done according to the following steps (here for pressure loss as example).

1. Use/Create model with missing pressure loss calculation (e.g. fluid interfaces for flow model)
2. Choose pressure loss function of interest
3. Choose corresponding pressure loss records
4. Build function-record construction
5. Assign record variables

This simplified implementation principle is shown in Fig.4.

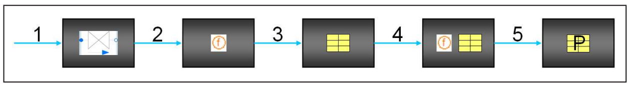


Figure 4: Principle implementation method to integrate FLUIDDISSIPATION functions into a thermo-hydraulic framework.

Integration examples of dissipation functions into MODELICA_FLUID as thermo-hydraulic framework are explained in detail in Sec.3.3. A description can also be found in [1] or in the GETTINGSTARTED section of the USERS GUIDE in the FLUIDDISSIPATION library itself.

3.2 Numerical optimisation

All convective heat transfer and pressure loss functions of the library are numerically optimised to provide efficient transient simulations. Measurement data for dissipation correlations are adapted into **continuous functions**. A smoothing function (see [3]) is used to ensure continuity and **differentiability**. For example the pressure loss of an overall fluid flow in a straight pipe is calculated via the correlations of the laminar and the turbulent regime with the smoothing function in between representing the transition regime.

An additional aspect to improve computational efficiency during system simulation can be to **avoid numerical Jacobians**. Numerical Jacobians are created in the translation process of a MODELICA simulation tool out of an underlying equation system of a simulation model, if the tool is not able to create analytical Jacobians.

For instance the pressure loss in a flow model can be modelled as a compressible case[1] or an incompressible case[2] to avoid additional **nonlinear equations**. An analytical or numerical Jacobians is then needed, if a compressible flow model has got a known mass flow rate at its interfaces instead of a pressure loss. Especially in large system simulations, the **avoidance of nonlinear equations** improves the performance of a simulation system because then there is no need for a nonlinear solver.

For pressure loss the FLUIDDISSIPATION library delivers both a compressible as well as an incompressible calculation for inverse calculations w.r.t. mathematical feasibility of its invertability. In an application flow model **no additional nonlinear equations** are created due to these functions if used correctly. Even if the pressure loss functions are used in a numerically unintended way by the user (e.g. the mass flow rate for a flow model is unknown in an incompressible case), all functions are able to provide **analytical Jacobians** to avoid the creation of numerical Jacobians.

3.3 Integration of library into thermo-hydraulic framework using MODELICA_FLUID library

Convective heat transfer models and flow models have been created using FLUIDDISSIPATION functions and the MODELICA_FLUID library as one example for the implementation into a thermo-hydraulic framework. These models calculate the convective heat transfer coefficient on the one hand and the mass flow rate or pressure loss on the other hand only. These base models have to be enhanced if additional features like balance equations or discretisation are needed.

The actual content of MODELICA_FLUID base

[1] To calculate the mass flow rate out of a given pressure loss
[2] To calculate a pressure loss out of a given mass flow rate

models provided by FLUIDDISSIPATION is shown in Fig.5. All dissipation correlations for convective heat transfer and pressure loss are grouped into its corresponding geometry device. For example if using a BENDFLOWMODEL the user can easily change a pressure loss calculation from an edged to a curved geometry. All dissipation correlations for each geometry device can be easily exchanged by choosing an intended application model via a drop-down menu in a parameter window[3]. After selection a corresponding record can be edited (e.g. for setting geometry parameters).

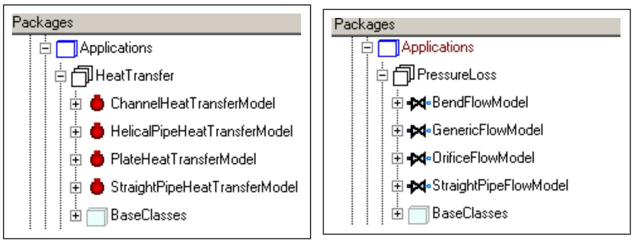


Figure 5: Application models using FLUIDDISSIPATION correlations and MODELICA_FLUID as thermo-hydraulic framework. Convective heat transfer models are shown in the left, whereas flow models are shown in the right.

3.3.1 Heat transfer

An implementation of convective heat transfer functions into usable MODELICA_FLUID models is shown in Fig.6. Generally heat transfer correlations are modelled through replaceable heat transfer models. For example a PLATEHEATTRANSFERMODEL can be chosen to the laminar, turbulent or overall fluid flow regime. All replaceable heat transfer calculations of one energy device are stored separately its BASECLASSES section.

Referring to Fig.6 the calculation of the thermodynamic state (needed for the fluid properties) has to be calculated outside of the provided heat transfer models. Here the thermodynamic state can be assigned through an inner model provided at the same hierarchy as the used instance of a heat transfer model. The convective heat transfer coefficient kc is calculated via a function call using its corresponding record with needed input variables like geometry parameters. The resulting heat flow rate $\dot{Q}_{flow}$ is calculated out of kc, heat transfer area A_{kc} and temperature difference between thermal port and thermodynamic state.

[3] Here using Dymola from Dynasim as IDE

```
model PlateHeatTransferModel
  "Application model for a plate in Modelica_Fluid"

  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a
       thermalPort
    "Thermal port interface";

  replaceable package HeatTransfer=
      FluidDissipation.Examples.Applications.HeatTransfer.
           BaseClasses.Plate.Overall
    constrainedby
    FluidDissipation.Examples.Applications.HeatTransfer.
         BaseClasses.Plate.BasePlateHT
    "Characteristic of convective heat transfer";

  outer FluidDissipation.Examples.TestCases.HeatTransfer.
       StateForHeatTransfer stateForHeatTransfer "Thermodynamic
        state from (missing) volume";

  FluidDissipation.Examples.Applications.HeatTransfer.
       BaseClasses.Plate.BasePlateHT.HeatTransferPlate IN "
       Input record";

equation
  kc = HeatTransfer.coefficientOfHeatTransfer(IN);
  thermalPort.Q_flow = kc*A_kc*dT "Heat transfer rate";
end PlateHeatTransferModel;
```

Figure 6: Relevant source code for a convective heat transfer model using FLUIDDISSIPATION correlations.

3.3.2 Pressure loss

An implementation of pressure loss functions using MODELICA_FLUID is shown in Fig.7. Pressure loss correlations are modelled through replaceable pressure loss models. The modelling referring to Fig.7 allows to replace a flow model instance of each energy device by its alternatives stored in the BASECLASSES.

```
partial model PartialFlowModel
  "Partial flow model for bend functions in Modelica_Fluid"
replaceable model PressureLoss = FD.CurvedBend.
     CurvedBendFlowModel
  constrainedby FD.BaseBendPL.BaseFlowModel;
end PartialFlowModel;
```

```
model CurvedBendFlowModel
  "Curved bend: Application flow model for bend function"
extends FD.Bend.BaseBendPL.BaseFlowModel;
FD.Bend.CurvedBend.PressureLossInput IN "Input record"
equation
  m_flow = FD.Bend.CurvedBend.massFlowRate_dp(dp, IN);
end CurvedBendFlowModel;
```

```
model BendFlowModel
  "Application flow model for bend functions in Modelica_Fluid"
extends FD.BaseBendPL.PartialFlowModel
  (redeclare model PressureLoss = PressureLossUsed)
  "Characteristic of bend pressure loss";
replaceable model PressureLossUsed = FD.BaseBendPL.
     BaseFlowModel
  "1st: choose pressure loss calculation | 2nd: edit record";
PressureLoss flowModel "Instance for chosen bend pressure loss"
     ;
end BendFlowModel;
```

Figure 7: Relevant source code for a flow model using FLUIDDISSIPATION correlations implemented in MODELICA_FLUID as thermo-hydraulic framework.

The calculation of pressure loss in a flow model can be implemented in dependence of targets. Then a compressible case or an incompressible case can be chosen to adjust an optimal numerical behaviour of a simulation model. Thus an inverse calculation without generating analytical Jacobians can be created automatically if supported by the MODELICA simulation tool.

4 Verification and validation of library

All functions in the library are verified against its based literature or validated against other literature if available. These verifications or validations are documented in the online documentation of the library. The validation of convective heat transfer and pressure loss are exemplarily described for one phase and two phase flow afterwards.

For heat transfer all functions in the library delivers a local or mean convective heat transfer coefficient.

$$kc = \frac{Nu \cdot \lambda}{Length} \tag{1}$$

kc	as convective heat transfer coefficient $[\mathrm{W}/\mathrm{m^2 \cdot K}]$,
λ	as heat conductivity of fluid $[\mathrm{W}/\mathrm{m \cdot K}]$,
$Length$	as characteristic length [m],
Nu	as Nusselt number $[-]$.

The Nusselt number Nu results from the actual flow regime, fluid properties and geometry parameters.

The compressible or incompressible pressure loss calculation uses a (total) pressure loss coefficient according to Eq.2.

$$\zeta_{tot} = \frac{2 \cdot \Delta p_{tot}}{\rho_{ref} \cdot v_{ref}^2} \tag{2}$$

ζ_{tot}	as pressure loss coefficient $[\mathrm{W}/\mathrm{m \cdot K}]$,
Δp_{tot}	as total pressure loss [Pa],
ρ_{ref}	as reference density $[\mathrm{kg}/\mathrm{m^3}]$,
v_{ref}	as reference velocity $[\mathrm{m}/\mathrm{s}]$.

The pressure loss coefficient ζ_{tot} is defined as the ratio of the total pressure loss between the inlet and outlet of a device to the dynamic pressure in a reference section. The total pressure loss for overcoming the forces of hydraulic resistance is dissipated. Thus the state of flow undergoes a change. At adiabatic conditions the mechanical work of a fluid flow is converted into heat due to resistance forces[4]. However the temperature of the fluid does not change at a constant velocity then. The reason is that the work of expansion due to pressure loss is entirely converted into the work of overcoming the resistance forces. The heat generated by this mechanical work is compensated by cooling induced through this expansion.

The total pressure loss is arbitrary subdivided into local and frictional pressure losses even though they are physically inseparable.

$$\Delta p_{tot} = \Delta p_{loc} + \Delta p_{fri} \tag{3}$$

Δp_{tot}	as total pressure loss [Pa],
Δp_{loc}	as local pressure loss [Pa],
Δp_{fri}	as frictional pressure loss [Pa].

The total pressure loss can only be assumed to be the difference in static pressures, if there is no change in the cross sectional area of an energy devices, no mixing or splitting occurs and geodetic pressure loss can be neglected. Otherwise total pressures shall be used for modelling because the pressure loss coefficient can also have negative values. For example a negative pressure loss coefficient can occur at the mixing of two fluid flows in a junction having different velocities. In this case the dynamic pressure difference of the fluid flow with the lower velocity is increased between the section before and after mixing.

4.1 Heat transfer in an even gap

The validation of convective heat transfer for an even gap is shown in Fig.8. The convective heat transfer coefficient has been calculated for developed and undeveloped fluid flow with one or two sides of the gap being isothermal. The validation is shown for the mean Nusselt number in an even gap with isothermal walls at a developed fluid flow. There is a good correlation w.r.t. literature for this boundary condition.

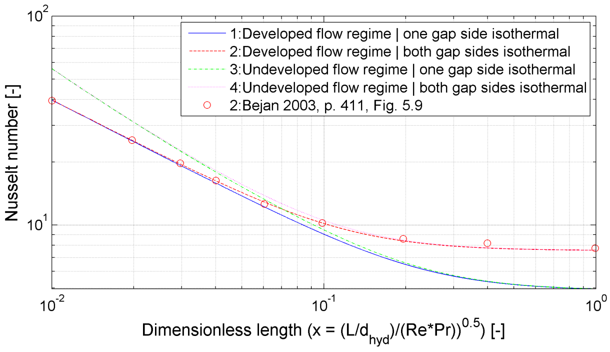


Figure 8: Validation for convective heat transfer of one phase laminar fluid flow in an even gap. The Nusselt number is shown in dependence of a dimensionless length at different boundary conditions. Simulation results are validated against *Bejan* [4].

[4] Total energy as sum of thermal and mechanical energy remains constant.

4.2 Pressure loss in a curved bend

The validation of pressure loss calculations is exemplarily shown for a curved bend in Fig.9.

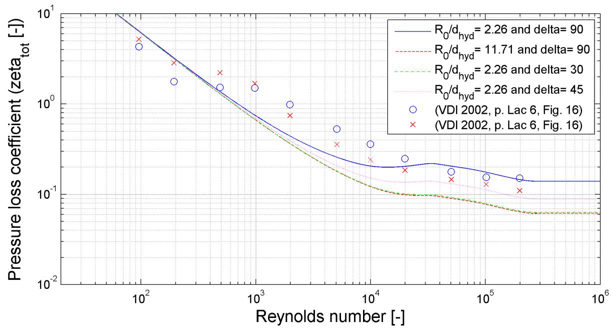
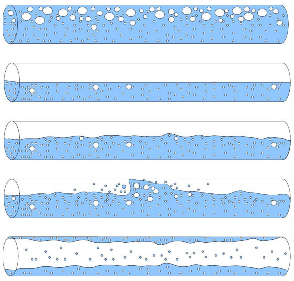


Figure 9: Validation of one phase overall fluid flow in a curved bend. The pressure loss coefficient is shown in dependence of the Reynolds number with relative curvature radii and angle of curvature of turning as parameter. Simulation results are validated against *VDI* [5].

The validation in Fig.9 comparing the pressure loss correlations out of *Idelchik* [6] with data from *VDI* [5] shows the right physical behaviour of fluid flow through a curved bend. The results give a good prediction of pressure loss in the laminar and fully developed turbulent regime. Deviations in the transition regime are in an adequate range of uncertainty.

4.3 Two phase flow in a horizontal straight pipe

The validation of two phase heat transfer and pressure loss calculations is exemplarily shown for a horizontal straight pipe. The various flow regimes during boiling and condensation during two phase flow are shown in Fig.10. For example at stratified flow a fluid is divided into a separated vapour and liquid phase due to gravity. The actual two phase flow regime during boiling and condensation can be determined with known vapour fraction and mass flux out of a flow pattern map (e.g. from *Steiner* in *VDI* [5]). For describing two phase flow usually a heterogeneous or homogeneous approach is used. These simplified approaches differ in the calculation of the cross sectional void fraction. The complete range of two phase flow regimes shown in Fig.10 can be modelled with these approaches.



- Bubble flow
- Stratified flow
- Wavy flow
- Slug flow
- Annular flow

Figure 10: Flow regimes for two phase flow in horizontal straight pipes.

In the **homogeneous** approach gas and liquid phase have the same velocity. The homogeneous void fraction is calculated with Eq.4.

$$\varepsilon_{hom} = \frac{1}{1+\left(\frac{1-\dot{x}}{\dot{x}}\right)\cdot\frac{\rho_g}{\rho_l}} \qquad (4)$$

$\varepsilon_{hom} = {}^{A_g}\!/_{A_g+A_l}$	as void fraction $[-]$,
ρ	as density of a phase $[{}^{kg}\!/_{m^3}]$,
$\dot{x} = {}^{\dot{m}_g}\!/_{\dot{m}_g+\dot{m}_l}$	as vapour fraction $[-]$,
A	as cross sectional area of a phase $[m^2]$,
$\dot{m}_{g/l}$	as mass flow rate of a phase $[{}^{kg}\!/_{s}]$.

The fluid properties of the two phases are averaged using the homogeneous approach. This approach is best applicable for the bubble flow regime with gas bubbles uniformly dispersed in the liquid phase.

The **heterogeneous** approach describes the two phases separately. In this case each phase can flow with a constant but different mean velocity. The separated phases are described with the so called slip ratio s as ratio of gaseous and liquid phase velocity. The heterogeneous void fraction is calculated from Eq.5.

$$\varepsilon_{het} = \frac{1}{1+\left(\frac{1-\dot{x}}{\dot{x}}\right)\cdot\frac{\rho_g}{\rho_l}\cdot s} \qquad (5)$$

ε_{het}	as void fraction $[-]$,
ρ	as density of a phase $[{}^{kg}\!/_{m^3}]$,
$s = {}^{v_g}\!/_{v_l}$	as slip ratio $[-]$,
$v_{g/l}$	as velocity of a phase $[{}^{m}\!/_{s}]$.

4.3.1 Heat transfer at condensation

The correlation of *Shah* [7] in Eq.6 is used to calculate the convective heat transfer during condensation in a horizontal straight pipe. In *Shah* [7] the heat transfer is assumed to take place only in the liquid phase. Therefore this correlation is best used for the annular flow regime, where the pipe wall is completely covered with liquid. The influence of the gaseous phase is described by a two phase multiplier.

$$\alpha_{2ph} \quad = \quad \alpha_l \cdot \left(1 + \frac{3.8}{p_r^{0.38}} \cdot \left(\frac{\dot{x}}{1-\dot{x}}\right)^{0.76}\right) \qquad (6)$$

α_{2ph} as two phase heat transfer coefficient $[{}^{W}/{}_{m^2 \cdot K}]$,
α_l as liquid heat transfer coefficient $[{}^{W}/{}_{m^2 \cdot K}]$,
p as actual pressure [Pa],
p_c as critical pressure [Pa],
$p_r = {}^{p}/{}_{p_c}$ as reduced pressure $[-]$,
$\dot{x}$ as vapour fraction $[-]$.

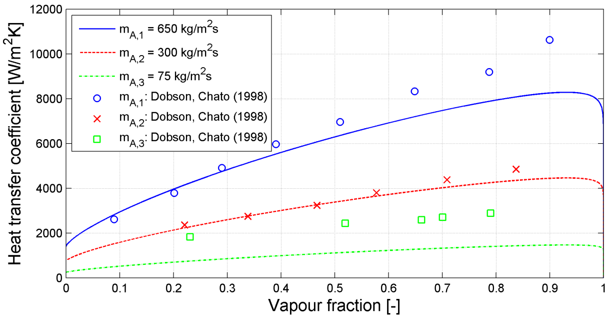


Figure 11: Validation of the local heat transfer coefficient during condensation in a horizontal straight pipe. The correlation of *Shah* [7] is validation against measurement results from *Dobson/Chato* [8] with the refrigerant R134a as medium.

In Fig.11 the *Shah* [7] correlation used for condensation in two phase flow has been validated against measurement results according to *Dobson/Chato* [8]. The refrigerant R134a has been used at a saturation temperature of 35°C and a mass flux between 75 to 650 ${}^{kg}/{}_{m^2 \cdot s}$. Generally these experimental data are underestimated by the correlation of *Shah* [7]. There are significant deviations for a small mass flux as well as for a very high mass flux and high vapour fraction. Deviations at a mass flux smaller than 130 ${}^{kg}/{}_{m^2 \cdot s}$ can be explained by having a different flow regime from the intended annular flow regime underlying the correlation of *Shah* [7]. For a moderate mass flux the validation shows a good prediction of the two phase heat transfer coefficient at annular flow regime.

4.3.2 Heat transfer at boiling

The correlation of *Gungor/Winterton* [9] is used to calculate the convective heat transfer during flow boiling in a horizontal straight pipe. This correlation expresses the physical effects of forced convection and nucleate boiling via a local two phase heat transfer coefficient (see Eq.7). The correlation can be used for subcooled and saturated flow boiling in horizontal and vertical straight pipes.

$$\alpha_{2ph} \quad = \quad E^* \cdot \alpha_{conv} \; + \; S^* \cdot \alpha_{nucl} \qquad (7)$$

$$\alpha_{conv} \quad = \quad \frac{0.023 \cdot [Re_l \cdot (1-\dot{x})]^{0.8} \cdot Pr_l^{0.4} \cdot \lambda_l}{d_{hyd}} \qquad (8)$$

$$\alpha_{nucl} \quad = \quad \frac{55 \cdot p_r^{0.12} \cdot q^{0.67}}{M^{0.5} \cdot (-\log_{10}(p_r))^{0.55}} \qquad (9)$$

α_{2ph} as two phase heat transfer coefficient $[{}^{W}/{}_{m^2 \cdot K}]$,
α_{conv} as factor due to forced convection $[{}^{W}/{}_{m^2 \cdot K}]$,
α_{nucl} as factor due to nucleate boiling $[{}^{W}/{}_{m^2 \cdot K}]$,
λ_l as liquid heat conductivity $[{}^{W}/{}_{m \cdot K}]$,
d_{hyd} as diameter [m],
E^* as convection enhancement factor $[-]$,
S^* as boiling suppression factor $[-]$,
M as molar mass of medium $[{}^{kg}/{}_{mol}]$,
p_r as reduced pressure $[-]$,
Pr_l as liquid Prandtl number $[-]$,
Re_l as liquid Reynolds number $[-]$,
q as heat flux $[{}^{W}/{}_{m^2}]$,
$\dot{x}$ as vapour fraction $[-]$.

In Fig.12 the *Gungor/Winterton* [9] correlation used for flow boiling in two phase flow is validated against measurement results according to *Kattan/Thome* [10]. The refrigerant R134a has been used at a saturation temperature of 4.4°C and a mass flux between 100 to 299${}^{kg}/{}_{m^2 \cdot s}$. The deviations in validation can be explained by considering the broad range of application for the *Gungor/Winterton* [9] correlation. Nevertheless there is an appropriate prediction of the heat transfer during flow boiling (e.g. for wavy to annular flow).

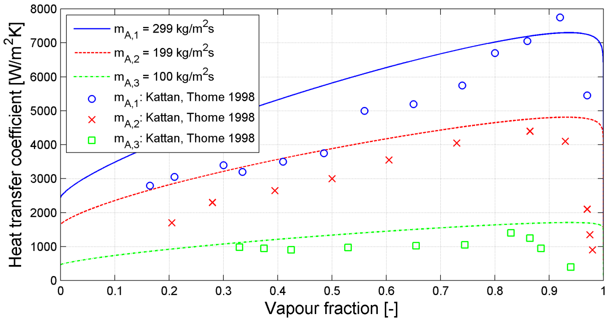


Figure 12: Validation of the local heat transfer coefficient at flow boiling in a horizontal straight pipe. The correlation of *Gungor/Winterton* [9] is validated against *Kattan/Thome* [10] with R134a as medium.

4.4 Pressure loss

The pressure loss in two phase flow for a horizontal straight pipe is calculated out of a momentum pressure loss and a frictional pressure loss.

$$\Delta p_{2ph} = \underbrace{\Delta p_{mom}}_{\text{momentum}} + \underbrace{\Delta p_{fri}}_{\text{friction}} \tag{10}$$

The **momentum pressure loss** due to a change of momentum of the gaseous and liquid phase is calculated according to Eq.11.

$$\Delta p_{mom} = \left[\dot{m}_A^2 \cdot \left(\frac{(1-\dot{x})^2}{\rho_l \cdot (1-\varepsilon)} + \frac{\dot{x}^2}{\rho_g \cdot \varepsilon} \right) \right]_{\dot{x}_{inlet}}^{\dot{x}_{outlet}} \tag{11}$$

Δp_{mom}	as momentum pressure loss [Pa],
ε	as void fraction $[-]$,
ρ	as density of a phase $[^{kg}/_{m^3}]$,
$\dot{m}_A$	as total mass flux $[^{kg}/_{m^2 \cdot s}]$,
$\dot{x}$	as vapour fraction $[-]$.

The momentum pressure loss is calculated by the difference of using Eq.11 with a vapour fraction at the outlet and the inlet of the horizontal straight pipe. The void fraction can be described either by the homogeneous or heterogeneous approach (see Sec.4.3).

The **frictional two phase pressure loss** of a horizontal straight pipe is determined by the correlation of *Friedel* [11]. This correlation uses the heterogeneous approach for all two phase flow regimes. The two phase pressure loss Δp_{fri} results from the frictional pressure loss of the liquid phase $\Delta p_{fri,l}$ and a two phase multiplier R. For the calculation of the liquid frictional pressure loss all the mixture is assumed to flow as liquid.

$$\Delta p_{fri} = \Delta p_{fri,l} \cdot R \tag{12}$$

The two phase multiplier according to *Friedel* [11] is calculated in Eq.13.

$$\begin{aligned} R \;=\; & (1-\dot{x})^2 + \dot{x}^2 \cdot \frac{\zeta_g}{\zeta_l} \cdot \frac{\rho_l}{\rho_g} + 3.43 \cdot \dot{x}^{0.69} \\ \cdot\; & (1-\dot{x})^{0.24} \cdot \left(\frac{\rho_l}{\rho_g}\right)^{0.8} \cdot \left(\frac{\eta_g}{\eta_l}\right)^{0.22} \\ \cdot\; & \left(1 - \frac{\eta_g}{\eta_l}\right)^{0.89} \cdot Fr_l^{-0.047} \cdot We_l^{-0.033} \end{aligned} \tag{13}$$

$\eta_{g/l}$	as dynamic viscosity of a phase $[^{kg}/_{m \cdot s}]$,
$\rho_{g/l}$	as density of a phase $[^{kg}/_{m^3}]$,
$\zeta_{g/l}$	as Darcy friction factor of a phase $[-]$,
Fr_l	as liquid Froude number $[-]$,
R	as two phase multiplier $[-]$,
$Re_{g/l}$	as Reynolds number of a phase $[-]$,
We_l	as liquid Weber number $[-]$,
$\dot{x}$	as vapour fraction $[-]$.

The Darcy friction factor $\zeta_{g/l}$ for each phase is calculated in dependence of the corresponding Reynolds number in Eq.14.

$$\zeta_{g/l} = \begin{cases} \underbrace{\dfrac{64}{Re_{g/l}}}_{Re_{g/l} \leq 1055} \\ \underbrace{\left(0.87 \cdot \ln\left(\dfrac{Re_{g/l}}{1.96 \cdot \ln\left(Re_{g/l} - 3.82\right)}\right)\right)^{-2}}_{Re_{g/l} > 1055} \end{cases} \tag{14}$$

In Fig.13 the *Friedel* [11] correlation used for two phase pressure loss is validated against measurement results according to *SINTEF* [12]. As medium CO_2 at a saturation temperature of 10°C and a mass flux between 200 to 400 $^{kg}/_{m^2 \cdot s}$ has been used. Both momentum and frictional pressure loss at mean fluid properties are considered.

In a technical report of *SINTEF* [12] several different pressure loss correlations are validated against experimental pressure loss data for evaporation of CO_2. The correlation of *Friedel* [11] delivers the most accurate results with a mean deviation of 22%. The validation of the implemented two phase pressure loss shows good agreement according to *SINTEF* [12]. Nevertheless the pressure loss is underestimated at a low vapour fraction.

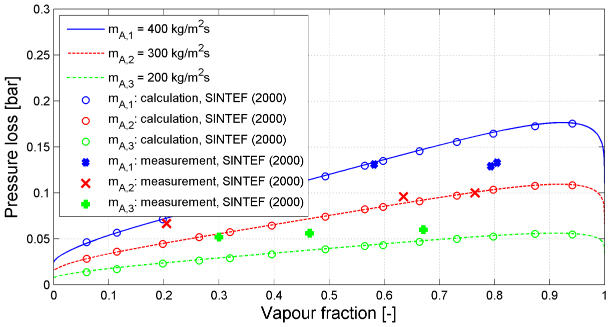


Figure 13: Validation of local pressure loss during flow boiling in a horizontal straight pipe. The correlation of *Friedel* [11] is validated against measurement results from *SINTEF* [12] with CO_2 as medium.

5 Industrial applications

This section gives examples for the usage of heat transfer and pressure loss correlations from FLUIDDISSIPATION in industrial system simulation. The dissipation correlations of the library are implemented into different thermo-hydraulic frameworks like MODELICA_FLUID.

5.1 Environmental control system

The evaluation of the dissipation correlations from FLUIDDISSIPATION is also carried out by Dassault Aviation as partner in the EuroSysLib project. In Fig.14 measurement data from a physical test bench representing an evaluation version of an aircraft have been compared to a corresponding simulation model using FLUIDDISSIPATION. There is a good agreement for first evaluation results of subsystems from the environmental control system in Fig.14.

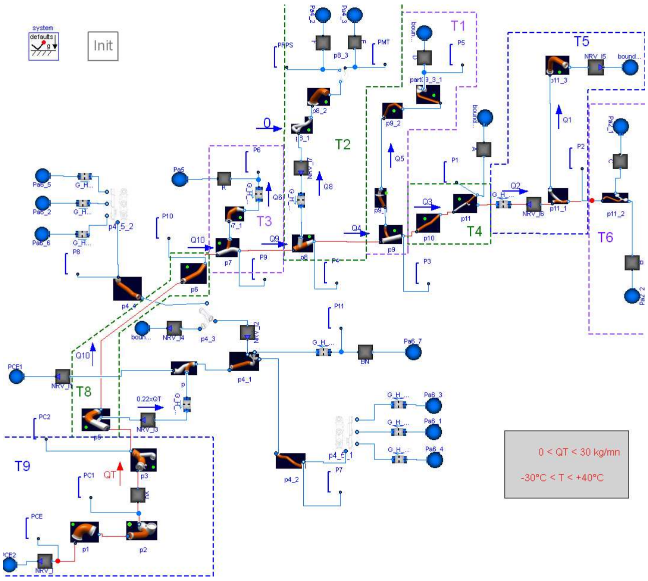


Figure 14: Evaluation of an environmental control system using FLUIDDISSIPATION carried out by Dassault Aviation.

5.2 Supplemental cooling system

A simulation model of an indirect cooling cycle for an air distribution circuit used for electrical equipment cooling is shown in Fig.15. Here dissipation correlations are integrated into a thermo-hydraulic library called HYDRONICS. Liquid water is used as cooling medium. The aim of this system is to cool different air flows in two heat exchanger. The optimal distribution of cold water is achieved by two control valves.

5.3 Aircraft engine feeding system

A simulation model of an aircraft engine feeding system is shown in Fig.16. Here FLUIDDISSIPATION is implemented into MODELICA_FLUID as thermo-hydraulic framework. An incompressible medium is used for the engine fuel (Jet A-1). The aim of this system is to control distribution of fuel for aircraft engines out of pressurised tanks. The simulation model analyses the system behaviour after the opening of the intercommunication valve in the emergency case. In the nominal case an intercommunication valve and the dump valve is closed. In this case the left hand pump feeds the left engine as the right hand pump does for the right engine. Both intercommunication valve and dump valve are opened if fuel has to be dumped from the aircraft in an emergency case.

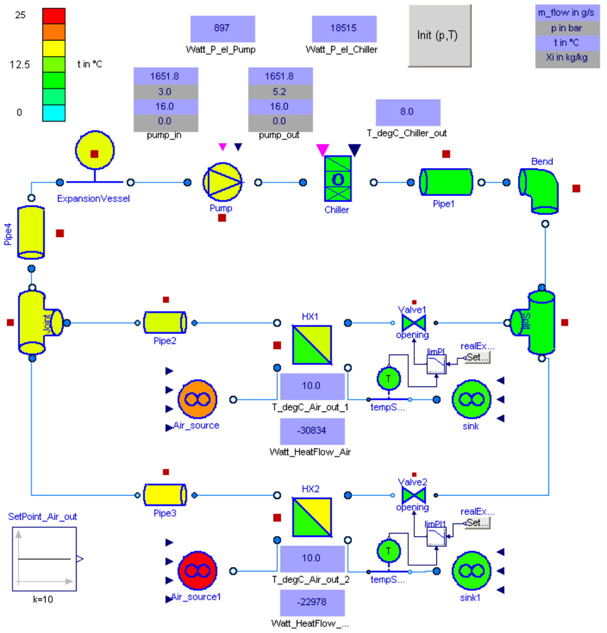


Figure 15: Simulation model of indirect cooling cycle for an air distribution circuit used for electrical equipment cooling. FLUIDDISSIPATION correlations are implemented in HYDRONICS as library.

6 Summary

The FLUIDDISSIPATION library offers a broad range of verified and validated convective heat transfer and pressure loss correlations for energy system. The library has been numerically optimised for efficient transient simulations. All correlations are documentation in detail describing its scope of usage.

MODELICA_FLUID models are provided as examples for the application into a thermo-hydraulic framework.

Scope, implementation concept, numerical challenges, verification and validation of the FLUIDDISSI-

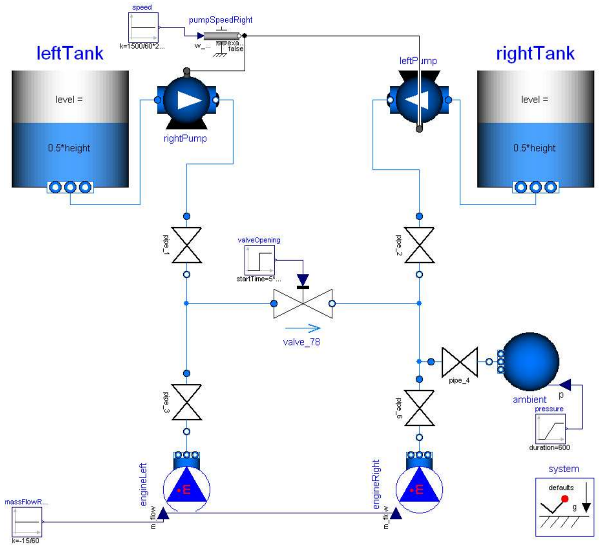


Figure 16: Simulation model of an aircraft engine feeding system using flow models with FLUIDDISSIPATION correlations and MODELICA_FLUID as thermo-hydraulic framework.

PATION library has been described as well as industrial applications of thermo-hydraulic energy systems.

FLUIDDISSIPATION is free for commercial and non-commercial applications and it can be used under the terms of the Modelica Licence 2.

Acknowledgement

The authors like to thank Eric Thomas for his evaluation results and Nina Peci for her contribution of two phase flow correlations.

The FluidDissipation library is developed within the European research project EuroSysLib-D funded by German Federal Ministry of Education and Research (promotional reference 01IS07022B). The project is started in October 2007 and will end in March 2010. The authors bear the sole responsibility for the content of this publication.

SPONSORED BY THE

Federal Ministry
of Education
and Research

References

[1] T. Vahlenkamp and S. Wischhusen. Fluiddissipation - a centralised library for modelling of heat transfer and pressure loss. In B. Bachmann, editor, *Proceedings of the 6th International Modelica Conference*, volume 1, pages 173–178, Bielefeld, Germany, March 2008.

[2] Casella, Francesco et al. The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks. In *Proceedings of the 5th International Modelica Conference*, pages 631–640, Linköping, Sweden, 2006. The Modelica Association.

[3] S. Wischhusen. *Dynamische Simulation zur wirtschaftlichen Bewertung von komplexen Energiesystemen*. PhD thesis, Technische Universität Hamburg-Harburg, 2005.

[4] A Bejan and A.D. Kraus. *Heat Transfer handbook*. John Wiley & Sons, 2nd edition, 2003.

[5] VDI. *VDI - Wärmeatlas: Berechnungsblätter für den Wärmeübergang*. Springer Verlag, 9th edition, 2002.

[6] I. E. Idelchik. *Handbook of hydraulic resistance*. Jaico Publishing House, Mumbai, 3rd edition, 2006.

[7] M.M. Shah. A general correlation for heat transfer during film condensation inside pipes. *Int. J. Heat Mass Transfer*, 22:547–556, 1979.

[8] M.K. Dobson and J.C. Chato. Condensation in smooth horizontal tubes. *Journal of Heat Transfer*, 120:193–213, 1998.

[9] K.E. Gungor and R.H.S. Winterton. A general correlation for flow boiling in tubes and annuli. *Int. J. Heat Mass Transfer*, 29:351–358, 1986.

[10] N. Kattan and J.R. Thome. Flow boiling in horizontal pipes: Part 2 - new heat transfer data for five refrigerants. *Journal of Heat Transfer*, 120:148–155, 1998.

[11] L. Friedel. Improved friction pressure drop correlations for horizontal and vertical two phase pipe flow. *3R International*, 18:485–491, 1979.

[12] R. Pettersen, J.; Rieberer and S.T. Munkejord. Heat transfer and pressure drop characteristics of evaporating carbon dioxide in microchannel tubes. Technical report, SINTEF Energy Research, 2000.

[13] D.S. Miller. *Internal flow systems*, volume 5th of *BHRA Fluid Engineering Series*. BHRA Fluid Engineering, 1984.

Modeling Chemical Reactions in Modelica By Use of Chemo-bonds

François E. Cellier
ETH Zürich
Switzerland
FCellier@Inf.ETHZ.CH

Jürgen Greifeneder
ABB AG, Corporate Research Center
Germany
Juergen.Greifeneder@DE.ABB.Com

Abstract

This paper describes a new methodology for modeling and simulating chemical reaction systems using vectors of chemo-bonds, called multi-chemo-bonds. Chemical reactions are usually described through mass flows alone. Yet in reality, they are convective flows, as the reactants carry their volume and heat with them in the reactions. Each combined mass/volume/heat flow can be described by a chemo-bond. The combination of all such flows can be described by a vector of chemo-bonds, i.e., a multi-chemo-bond.

Keywords: object-oriented modeling of chemical reactions; chemo-bonds; thermo-bonds; multi-chemo-bonds; convective flows

1 Introduction

Traditionally, chemical reaction systems are described as pure mass flow systems. There is no need to consider the energy flows as well, as long as the thermo-dynamical properties of the reaction system can be ignored. As chemical reactions are characterized by capacitive storage only, i.e., they don't feature inductive storage, the mass balance and energy balance equations are decoupled from each other [2].

A first attempt of describing the thermodynamics of chemical reaction systems in a systematic way was reported in [5]. Unfortunately, the primary author of that paper, Aharon Katzir-Katchalsky, died prematurely during an attack by Palestinians on the airport of Tel-Aviv. After his death, this line of research stopped for several years.

His research was continued 12 years later by a student of one of the authors, Michael Amrhein, who described for the first time a chemical reaction system by means of bond graphs [1].

In the remainder of this paper, we start out with modeling the molar flows (Section 2) and put them together in a multi-bond representation (Section 3). In Section 4, the corresponding volume and entropy flows will be added to the molar flows resulting in a chemo-bond model. Section 5 is similar to Section 3 and introduces multi-chemo-bonds. Section 6 finally offers some conclusions.

2 The basic model

In Amrhein's work [1], the mass flow variable,

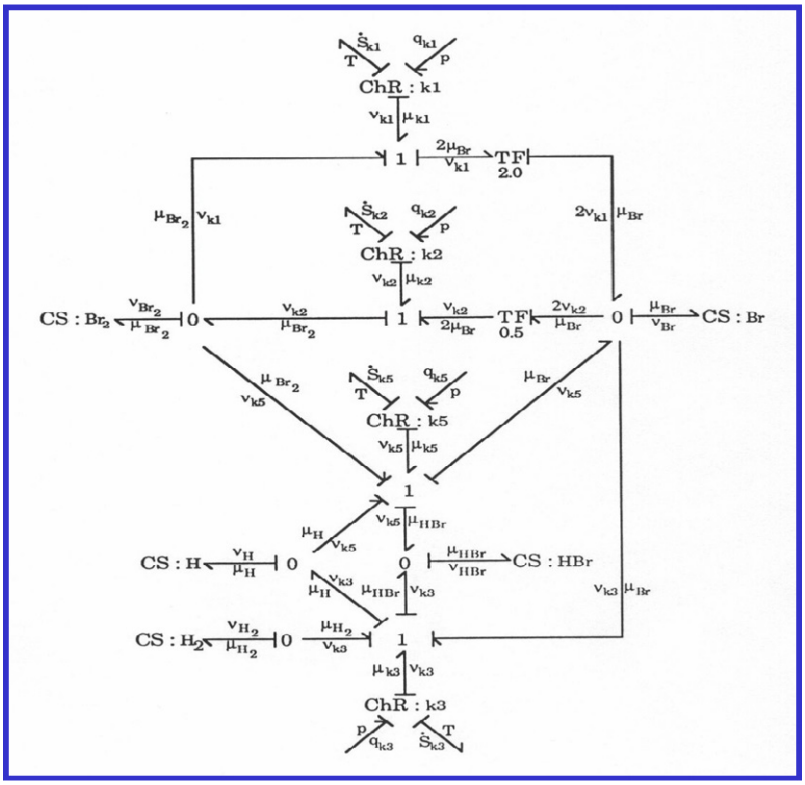


Fig.1: Chemical reaction bond graph

i.e., the *molar flow* of a reactant in a reaction, is accompanied by a potential variable, the *chemical potential*, such that the product of the two variables denotes energy flow. Amrhein's bond graph simulated simultaneously the mass flow and the mass-energy flow through the reaction system. He modeled a hydrogen-bromine reaction to be simulated under different operating conditions. The simplest experimental setup freezes the temperature and the pressure, i.e., the chemical reaction is simulated under isothermal and isobaric operating conditions. Amrhein's model is shown in Fig.1.

Amrhein developed his new modeling methodology years before Modelica was created, and he coded his models in the old Dymola language before a graphical interface had become available for Dymola. Hence the bond graph of Fig.1 is not a code, but only a picture that was drawn manually to make the code better understandable.

The CS-elements represent the (capacitive) storage of the five reactants, whereas the ChR-elements represent the five individual step reactions. They are connected by a network of bonds, junctions, and transformers representing the chemical reaction network.

Yet, the graph shown in Fig.1 is far from complete. First, it only shows four ChR-elements, although the hydrogen-bromine reaction exhibits five individual step reactions, namely:

$$\begin{array}{ccccccc} & & Br_2 & \underset{k_2}{\overset{k_1}{\rightleftharpoons}} & 2\ Br^\bullet & & \\ Br^\bullet & + & H_2 & \underset{k_4}{\overset{k_3}{\rightleftharpoons}} & HBr & + & H^\bullet \\ Br_2 & + & H^\bullet & \overset{k_5}{\rightarrow} & HBr & + & Br^\bullet \end{array}$$

Reaction #4, the least important of the five step reactions, was generously left out of the graph to keep the graph planar. Second, the chemical reactors, i.e., the ChR-elements, require state information that is being computed by the CS-elements. Hence there need to be added signal paths between the CS-elements and the ChR-elements that were also left out for enhanced readability.

A completed graph of Amrhein's model, now coded in a current version of Dymola using BondLib [4] is shown in Fig.2. The grey lines are used to connect bonds with junctions and vice-versa, i.e., they represent the same energy flows as the bonds connected to them. The blue lines represent information flows. 0 and 1 represent junctions, and the half-arrows represent bonds.

The model of Fig.2 can be simulated in Dymola without any problems, and for a temperature of T = 800 K and a pressure of p = 102 kPa, we obtain the trajectories shown in Fig.3.

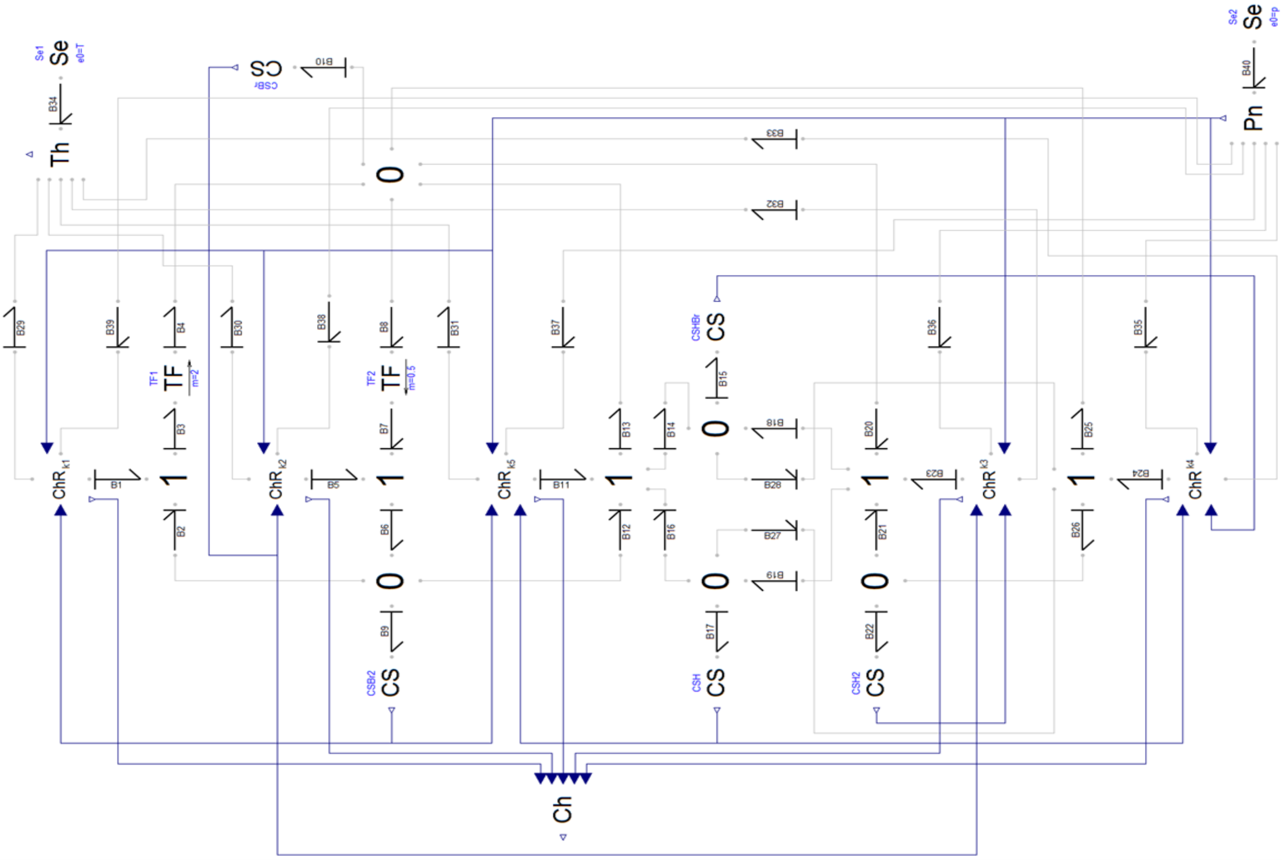


Fig.2: Completed chemical reaction bond graph

Initially, we start out with 0.0075 mol of H_2 and Br_2 each, create and consume the temporary atoms $Br^{\bullet}$ and $H^{\bullet}$, and after roughly 0.07 s, we end up with 0.015 mol of HBr.

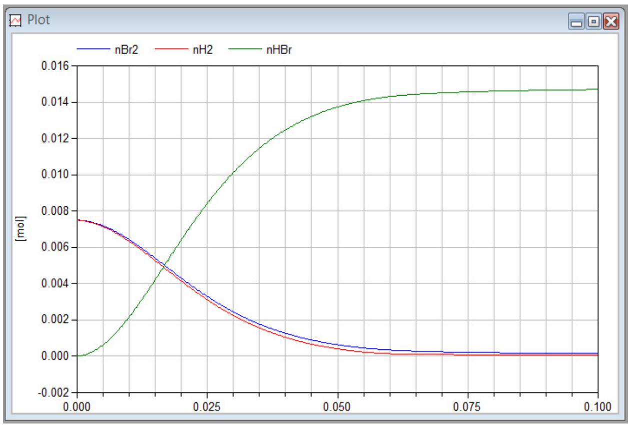


Fig.3: Chemical reaction trajectories

The same results could have been obtained more easily by writing down the reaction rate equations directly, but the presented bond graph model is superior, because it represents not only the dynamics of the mass flows through the reactions, but also the corresponding energy flows. Each chemical bond contains two variables, an effort μ, representing a chemical potential, and a molar flow rate ν, representing the mass flow, such that:

$$P = \mu \cdot \nu$$

is the power flowing through the bond.

Hence the bond graph can also be used to compute the power flows through the reaction network, and we notice that the reaction is mildly exothermic, as documented in Fig.4, where the entropy gain is being shown. Roughly ΔS = 0.06 J/K of entropy are being generated during the reaction.

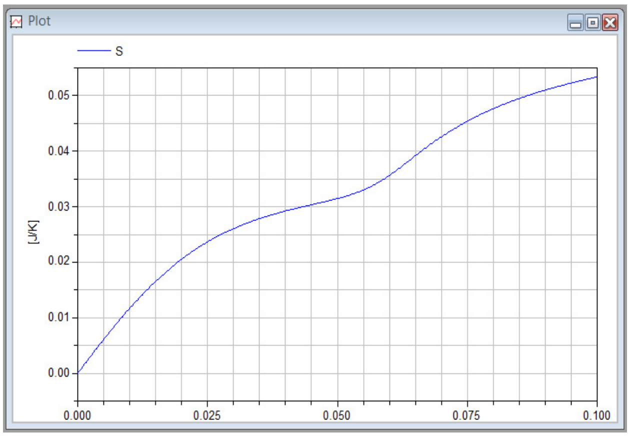


Fig.4: Entropy gain during chemical reaction

Unfortunately, the bond graph of Fig.2 is a mess, and few researchers will be willing to go through the agony of describing chemical reactions in this fashion. Therefore in the remainder of this paper, we shall improve on the model step by step, and create ever better models that can be used more easily.

3 The Chemical Reaction Network

The chemical potentials, μ_i, of the reactants are computed by the CS-elements. These are then transferred across the chemical reaction network, which computes the chemical potentials of the individual step reactions, μ_{ki}. The ChR-elements use that information to determine the molar flow rates of the step reactions, ν_{ki}, which are then transferred back across the chemical reaction network, which computes the molar flow rates of the reactants, ν_i. These are then integrated inside the CS-elements into the states, i.e., the number of moles, n_i, which in turn are needed by the ChR-elements to compute the flow rates. The state vector is transferred from the CS-elements back to the ChR-elements through separate signal paths, i.e., outside the bond graph.

As already demonstrated in [2], the chemical reaction network, relating reactants and step reactions to each other, can be interpreted as a multi-port transformer. Consequently, we can write:

$$\begin{pmatrix} \nu_{\mathrm{Br}_2} \\ \nu_{\mathrm{Br}\bullet} \\ \nu_{\mathrm{H}_2} \\ \nu_{\mathrm{H}\bullet} \\ \nu_{\mathrm{HBr}} \end{pmatrix} = \begin{pmatrix} -1 & 1 & 0 & 0 & -1 \\ 2 & -2 & -1 & 1 & 1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \nu_{k_1} \\ \nu_{k_2} \\ \nu_{k_3} \\ \nu_{k_4} \\ \nu_{k_5} \end{pmatrix}$$

$$\begin{pmatrix} \mu_{k_1} \\ \mu_{k_2} \\ \mu_{k_3} \\ \mu_{k_4} \\ \mu_{k_5} \end{pmatrix} = \begin{pmatrix} -1 & 2 & 0 & 0 & 0 \\ 1 & -2 & 0 & 0 & 0 \\ 0 & -1 & -1 & 1 & 1 \\ 0 & 1 & 1 & -1 & -1 \\ -1 & 1 & 0 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \mu_{\mathrm{Br}_2} \\ \mu_{\mathrm{Br}\bullet} \\ \mu_{\mathrm{H}_2} \\ \mu_{\mathrm{H}\bullet} \\ \mu_{\mathrm{HBr}} \end{pmatrix}$$

This can also be written as:

$$\vec{\nu}_s = \mathbf{N} \cdot \vec{\nu}_r \quad , \quad \vec{\mu}_r = \mathbf{M} \cdot \vec{\mu}_s$$

where **N** is the transfer matrix for the flow rates, and $\mathbf{M} = \mathbf{N}^{\mathbf{T}}$ is the transfer matrix for the chemical potentials.

As we can describe the chemical reaction network using a multi-port transformer, it makes sense to extract the five CS-elements into a single CF-element, a capacitive field. As shown in Fig.5, this CF-element has on its left side, the five state variables, grouped into a state vector, whereas on the right side, there is now a vector of bond connectors, each representing one reactant.

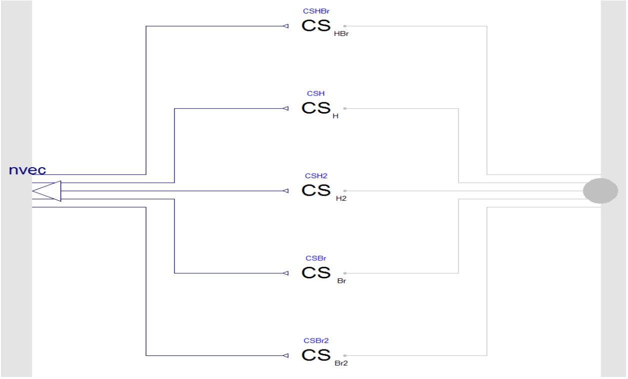


Fig.5: Capacitive field vector bond graph

For further processing, the vector of (grey) regular bond connectors needs to be converted to a single (blue) multi-bond connector, as shown in Fig.6. As the conversion element has been implemented as a bond, we need to add a 0-junction also, in order to comply with the rule of all of our bond graph libraries that all macro elements must end in junctions, rather than in bonds.

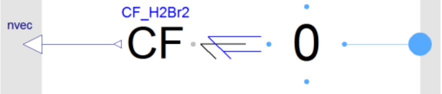


Fig.6: Conversion of bond vector to multi-bond

In the same way as with the five CS-Elements, we can also stack the five ChR-elements into a single RF-element, a resistive field, as shown in Fig.7.

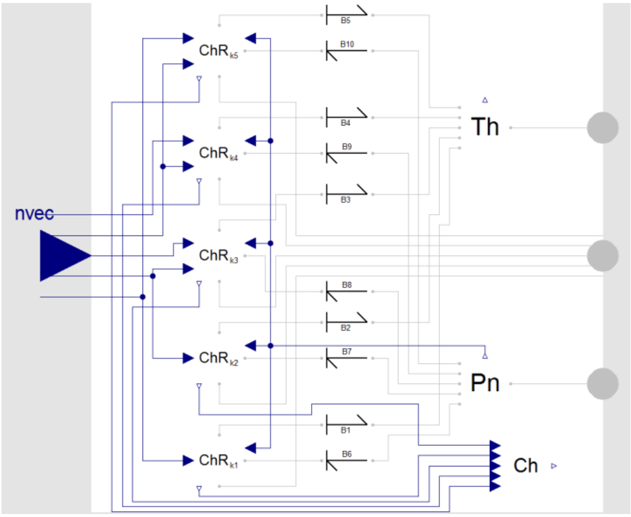


Fig.7: Resistive field vector bond graph

The attentive reader might already have discovered that the resistive field has three bond graph connectors: The one at the center on the right side is a vector connector representing the five mass flows as discussed before, whereas the bond graph connectors on top and below the mass flow connector are single connectors, one representing the heat flow and the other representing the volumetric flow. Under isothermal and isobaric operating conditions, these two connectors are used to impose from the outside the desired temperature of $T = 800$ K and the desired pressure of $p = 102$ kPa.

Finally, the vector connector of the RF-element also needs to be converted to a multi-bond connector, as shown in Fig.8.

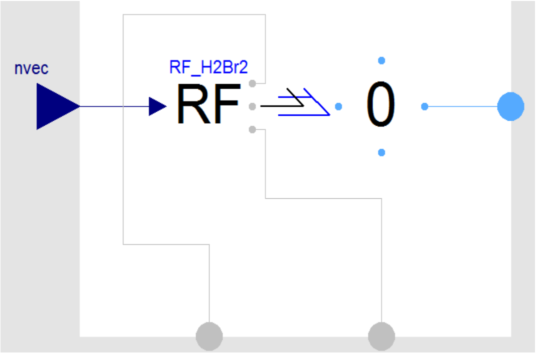


Fig.8: Conversion of bond vector to multi-bond

We are now ready to model the entire chemical reaction using MultiBondLib [6], a second bond graph library designed particularly for modeling mechanical systems in two and three space dimensions. Yet, the library can also be used for any other model that requires vectors of bonds.

The resulting model is shown in Fig.9. The chemical reaction network has not been modeled graphically in this model, but rather, was represented using a multi-port transformer (TF_H2Br2). The unconnected MBG_defaults model shown at the bottom right corner of Fig.9 represents the multi-bond world model that sets the default dimension of all multi-bonds to a value of five.

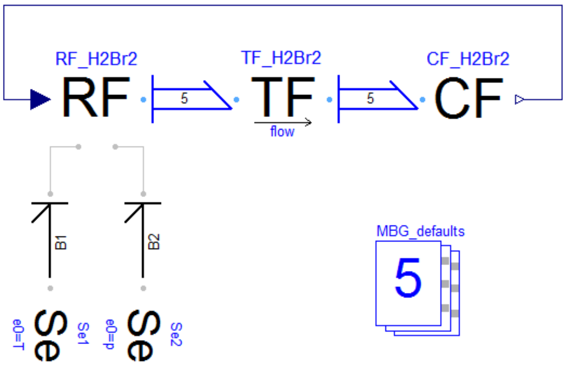


Fig.9: Chemical reaction modeled using multi-bonds

The simulation results obtained with this code are exactly the same as with the model of Fig.2, as these are truly identical models.

4 Thermo-bonds and Chemo-bonds

As shown in Fig.10, an excerpt of Fig.1, each ChR-element has three bond graph connectors, one representing mass flow (variables μ and ν, i.e., chemical potential and molar flow rate), a second representing volumetric flow (variables p and q, i.e., pressure and volumetric flow rate), and a third representing heat flow (variables T and *Sdot*, i.e., temperature and entropy flow rate). Each of the three pairs, when multiplied, represents a power flow measured in Watts.

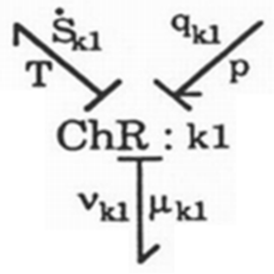


Fig.10: ChR-element

Although the three bonds represent different physical phenomena, they are mathematically identical. For this reason, it is important that bond variables are declared as ‘Real’ in Modelica, rather than be associated with particular measurement units. These variables may inherit measurement units through the higher layers of the model architecture, but by themselves, bond graph variables are neutral.

It is not possible to have mass flows without accompanying volume flows and heat flows, because the masses always carry their own volume and heat with them. The internal energy of matter of a mass can be written as:

$$U = T \cdot S - p \cdot V + \mu \cdot n$$

with the corresponding power flow:

$$Udot = T \cdot Sdot - p \cdot q + \mu \cdot \nu$$

Thus, each mass flow can be interpreted as a parallel connection of three individual power flows, one representing the mass flow itself, a second representing the accompanying volumetric flow, and a third representing the heat flow.

As these three flows belong together, we have created a third bond graph library, called ThermoBondLib [3] that has been designed specifically for modeling convective flows.

Each thermo-bond represents a parallel connection of three regular bonds, one for each of the three types of power flow. This is shown in Fig.11.

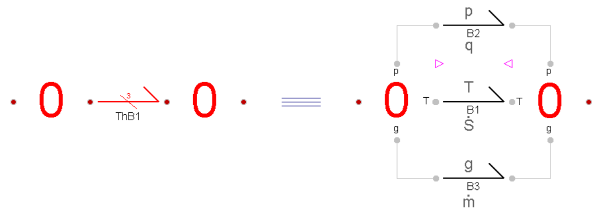


Fig.11: Composition of a thermo-bond

Since the thermo-bonds represent a specific physical phenomenon, it made sense to associate thermo-bonds with measurement units, i.e., whereas the regular (black) bonds of BondLib [4] and the (blue) multi-bonds of MultiBondLib [6] are neutral, the (red) thermo-bonds of ThermoBondLib [3] have been associated with measurement units explicitly.

For convective flows, it was convenient to measure the mass flow, *mdot*, in kg/s. Consequently, the corresponding effort variable, g, must be measured in J/kg, such that their product is once again a power flow measured in Watts. The effort variable of mass flow, g, is the Gibbs potential, which sometimes is also called specific Gibbs energy.

Notice that g has the same units as h, the specific enthalpy, but it is not the same quantity:

$$h = g + T \cdot s$$

where s denotes the specific entropy, i.e., the entropy per unit mass.

Note: The Gibbs potential is still currently missing in the SIunits library of Modelica.

Since mass flows cannot occur without accompanying volume and heat flows, it makes sense to interpret chemical reactions as convective flows, and replace the (black) regular bonds of Fig.2 by (red) thermo-bonds. In this way, there will be no need any longer to treat the thermal and volumetric flows separately from the mass flows, and the ChR-elements will now only have one thermo-bond connector.

Unfortunately, it is inconvenient to represent mass flows in chemical reactions as absolute mass flows, measured in kg/s, because chemical reactions occur in relation to the number of molecules involved, and not in relation to the weight of the reactants. Correspondingly, chemical mass flows are given as molar flow rates, ν, measured in mol/s, and correspondingly, the associated effort variable is the chemical potential, μ, measured in J/mol.

It is easy to convert between the two types of mass flows. This is simply a transformation, as shown in Fig.12. It would have been possible to deal with molar flow rates using the ‘redeclare’ feature of Modelica, but it was simpler to create yet another bond graph library, ChemBondLib, this time featuring green chemo-bonds, that are identical to the red thermo-bonds except for the way in which the mass flows are being represented.

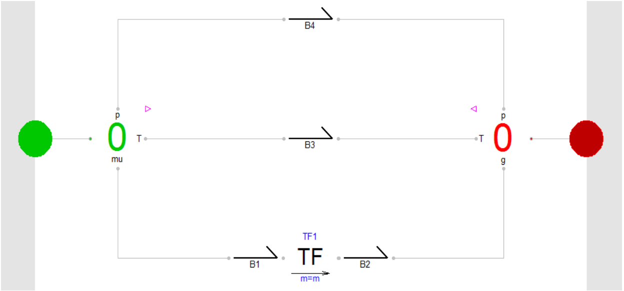


Fig.12: Conversion from absolute mass flow rates to molar flow rates

We are now ready to formulate the third version of the chemical reaction model. This new model is shown in Fig.13.

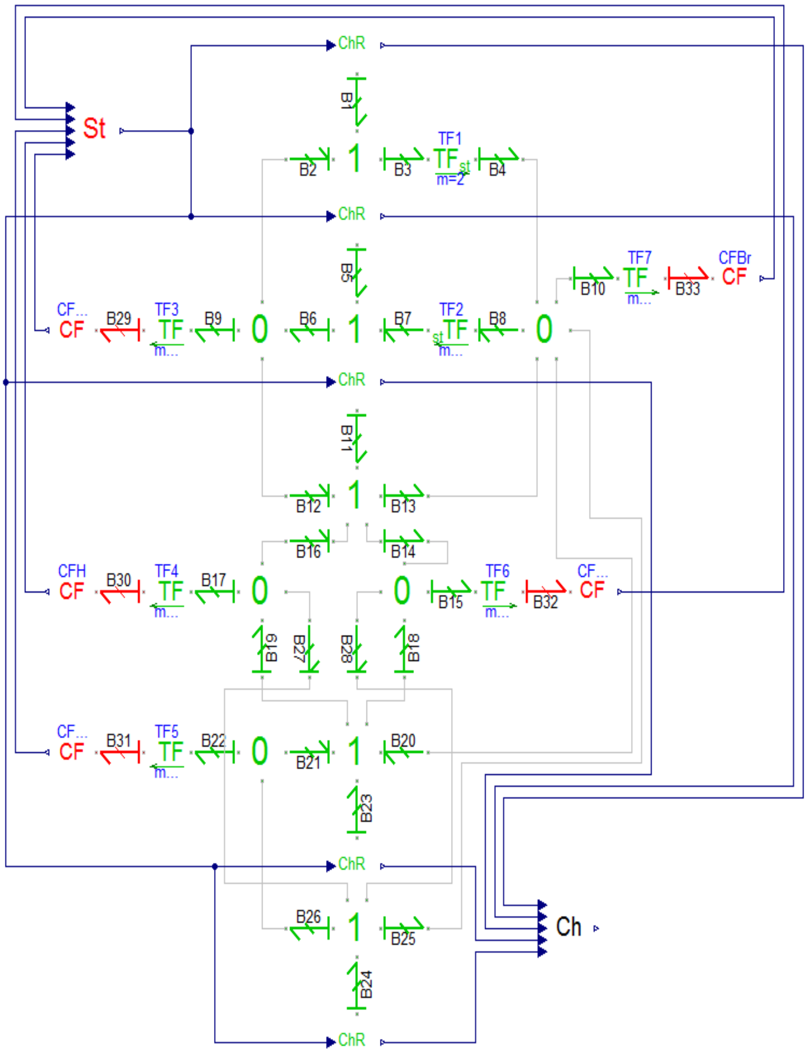


Fig.13: Chemical reaction modeled by chemo-bonds

The entire chemical reaction network is now shown in green. Each of the green bonds models three parallel energy flows. As the substances, represented through the (red) CF-elements, are modeled using absolute masses (they are the very same CF-elements that we had introduced in [3] before), one of the transformers shown in Fig.12 is placed between the (green) chemical reaction network and each of the (red) CF-elements.

The model is much simpler than that of Fig.2, because there is no longer any need to deal with the heat and volumetric flows separately. These flows are now transferred across the reaction network together with the mass flows.

The CF-elements now compute the three potential (effort) variables, T, p, and μ. These variables are then transferred across the chemical reaction network to the side of the step reactions. The ChR-elements compute the three flow variables, *Sdot*, q, and ν. These are transferred back across the reaction network to the side of the reactants.

Each CF-element computes a partial state vector including the three variables S, V, and n. The partial state vectors are concatenated to a complete state vector in the St-model that transfers the state information back to the ChR-elements, as the reactor models require the state information to compute the flows.

As expected, the simulation results obtained by this third version of the model are identical to those received earlier.

5 Multi-chemo-bonds or Chemo-multi-bonds ?

We can now once again eliminate the graphical representation of the chemical reaction network and replace it by a multi-port transformer.

To this end, we again stack the five CF-elements, as shown in Fig.14.

On the right side, we now have a vector of (green) chemo-bond connectors, each representing three flows, a heat flow, a volumetric flow, and a molar mass flow. Thus, the vector chemo-bond connector can be interpreted as a multi-chemo-bond connector.

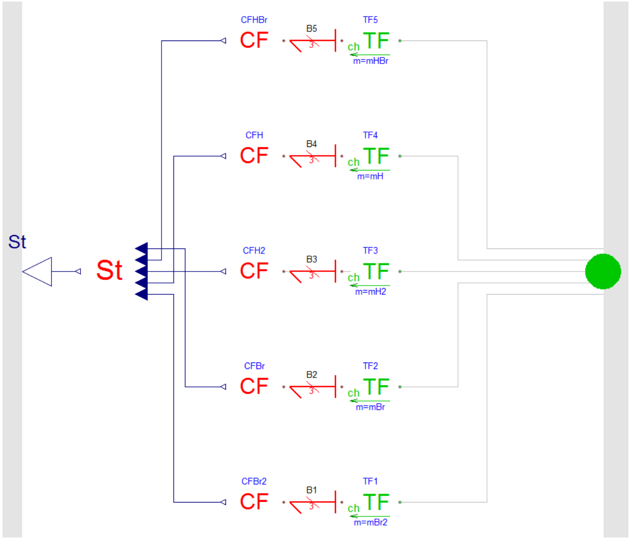


Fig.14: Capacitive reactant field

Fig.15 shows, how the vector of individual chemo-bonds gets converted to a single multi-bond. Here, the multi-bond connector represents a vector of length 15.

However for reasons that will become clear very soon, it was more convenient to rearrange the sequence of bonds within the multi-bond in such a way that the multi-bond contains first the five heat flows, followed by the five volumetric flows, followed by the five molar mass flows. Thus, the multi-bond can be interpreted as a chemo-multi-bond connector.

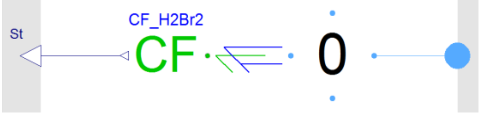


Fig.15: Conversion of connectors

In the same fashion, we can also stack the five ChR-elements, as shown in Fig.16, with the corresponding conversion of connectors shown in Fig.17.

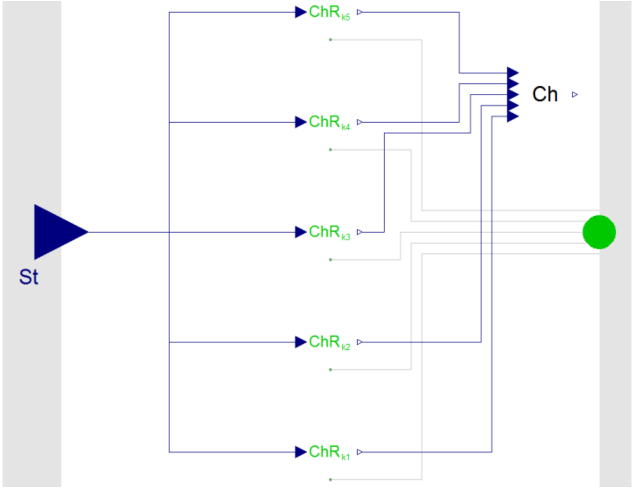


Fig.16: Resistive reaction field

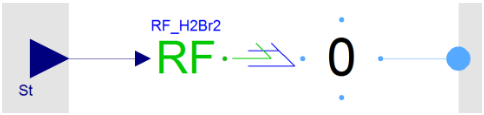


Fig.17: Conversion of connectors

As expected, the stacked reactor model of Fig.16 is much simpler than the corresponding stacked reactor model of Fig.7, because the thermal and pneumatic flows are not handled separately any longer.

The complete model (fourth version) is shown in Fig.18.

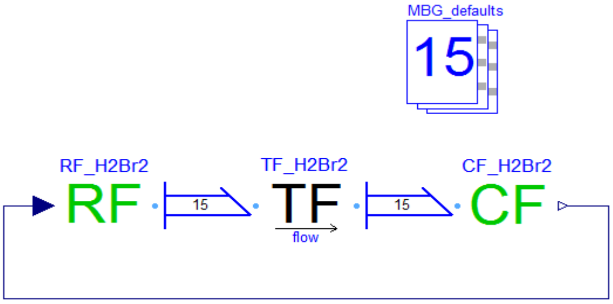


Fig.18: Chemical reaction modeled by multi-chemo-bonds

This time around, the transformation matrix of the multi-port transformer is of size 15x15. Yet, it can be composed easily from the previously introduced **N**-matrix, assuming that we operate on multi-chemo-bonds rather than chemo-multi-bonds:

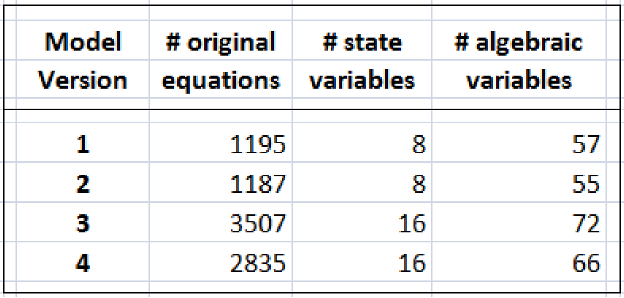
$$\mathrm{Z} = zeros(5,5)$$

$$\mathrm{M} = \begin{pmatrix} \mathrm{N} & \mathrm{Z} & \mathrm{Z} \\ \mathrm{Z} & \mathrm{N} & \mathrm{Z} \\ \mathrm{Z} & \mathrm{Z} & \mathrm{N} \end{pmatrix}$$

The transformation matrix is a block-diagonal matrix containing the same **N**-matrix three times along the diagonal, once used to transform the heat flows, once used to transform the volumetric flows, and once used to transform the molar mass flows.

A comparison of the computational efficiencies of the four models is given in Table 1.

Table 1: Comparison of computational efficiency



Model Version	# original equations	# state variables	# algebraic variables
1	1195	8	57
2	1187	8	55
3	3507	16	72
4	2835	16	66

The first two models are indistinguishable concerning their computational efficiencies. The third and fourth model are a bit less efficient computationally. The reason is the following. In the first two models, we recognized that we could add up the partial entropy flows and the partial volume flows and integrate them together, whereas in models three and four, each CF-element contains three separate states, one describing the partial entropy, a second describing the partial volume, and a third describing the partial mass of each of the five reactants.

6 Conclusions

In this paper, we have shown how different types of bond graphs can be used to describe chemical reaction systems.

Only a single reaction system was used for demonstration, namely a hydrogen-bromine reaction simulated under isothermal and isobaric operating conditions.

Although we have been able to demonstrate that the models become simpler as we treat chemical reactions as convective flows, there is a yet much more important reason for doing this, a reason that does not become obvious from reading this paper alone.

Amrhein simulated the same reaction system under a series of different operating conditions [1,2]. He replaced the isothermal condition by an adiabatic condition, for example, and he replaced the isobaric condition by an isochoric condition.

It becomes clear from reading his documents that each of his models looks slightly different, i.e., the bond graph had to be adjusted a bit from one set of operating conditions to another.

The reason for this inconvenience is that Amrhein did not compute all quantities where they naturally belong. Consequently, his models violated some of the premises of object-oriented modeling. All of the potential (effort) variables and all of the state variables should be computed at the side of the reactants, whereas all of the flow variables should be evaluated at the side of the step reactions.

Using the convective flow approach to modeling chemical reaction systems, it won’t be necessary any longer to modify the bond graph from one set of operating conditions to another, because all variables are indeed being computed where they belong.

The implementation of the (red) CF-elements used in the simulations of this paper are not yet fully general. As we were only interested in isothermal and isobaric operating conditions, it sufficed to set the temperature, T, and the pressure, p, to their desired values inside the CF-elements. Since both the temperature and the pressure were assumed constant, also the chemical potentials, μ_i, are constant, and consequently, also those values could be entered as parameters.

In the general case, T, p, and μ_i will not be constant, and they will have to be computed from state information, as already demonstrated in the general CF-elements for air and water presented in [3].

Temperature and pressure equilibration between different reactants can be modeled using heat exchange (HE) elements and pressure/volume exchange (PVE) elements, placed between the different CF-elements, as demonstrated in [3].

In the special case of an isothermal operating condition, a controlled entropy flow out of the storages (CF-elements) could then be added to the model that equals the entropy flow generated by the reaction system, such that the total entropy, and thereby also the temperature remain constant. Alternatively, an HE-element could be added between the reaction system and the environment that guarantees that the temperature of the reaction system remains at the ambient temperature.

In the special case of an isobaric operating condition, either a controlled volume flow would need to be imposed from the outside, or alternatively, a PVE-element could be added between the reaction system and the environment that guarantees that the pressure of the reaction system remains at the ambient pressure.

Thus, general CF-elements can be used independently of the operating conditions, which then are imposed on the model by external control flows in the same way as a chemical engineer would set up his or her experiment in the lab.

Unfortunately, we did not have either the time or the space in this paper to demonstrate how such a setup would need to be modeled using the ChemBondLib and ThermoBondLib Modelica libraries, and therefore, the proposed generalization will have to be postponed to another time and another publication.

The research presented in this paper represents only a very first step in an ongoing research effort. Reactions among ideal gases are particularly easy to model and simulate, and isothermal and isobaric operating conditions are among the most convenient operating conditions that may be assumed.

Much more research is needed before we can claim that we have created a universal approach to modeling and simulating all kinds of chemical reaction systems in a truly object-oriented physically inspired manner.

References

[1] Amrhein, M.: *Bond Graph Modeling of Chemical Reaction Dynamics*. MS Thesis, Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ, 1990

[2] Cellier, F.E.: *Continuous System Modeling*. Springer-Verlag, New York, 1991

[3] Cellier, F.E. and Greifeneder, J.: ThermoBondLib – A New Modelica Library for Modeling Convective Flows, In: *Proc. 6th International Modelica Conference*, Bielefeld, Germany (2008) Vol.1, 163-172

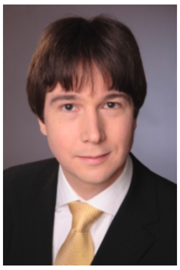
[4] Cellier, F.E. and Nebot, A.: The Modelica Bond Graph Library, In: *Proc. 4th International Modelica Conference*, Hamburg, Germany (2005) Vol.1, 57-65

[5] Oster, G.F., Perelson, A.S., and Katzir-Katchalsky A.: Network Thermodynamics: Dynamic Modelling of Biophysical Systems, In: *Quarterly Reviews of Biophysics* (1973) Vol. 6(1), 1-134

[6] Zimmer, D. and Cellier, F.E.: The Modelica Multi-bond Graph Library, In: *Proc. 5th International Modelica Conference*, Vienna, Austria (2006) Vol.2, 559-568

Author Biographies

François E. Cellier received his BS degree in electrical engineering in 1972, his MS degree in automatic control in 1973, and his PhD degree in technical sciences in 1979, all from the Swiss Federal Institute of Technology (ETH) Zurich. Dr. Cellier worked at the University of Arizona as professor of Electrical and Computer Engineering from 1984 until 2005. He then returned to his home country of Switzerland. Dr. Cellier's main scientific interests concern modeling and simulation methodologies, and the design of advanced software systems for simulation, computer aided modeling, and computer-aided design. Dr. Cellier has authored or co-authored more than 200 technical publications, and he has edited several books. He published a textbook on Continuous System Modeling in 1991 and a second textbook on Continuous System Simulation in 2006, both with Springer-Verlag, New York.



Jürgen Greifeneder received a diploma degree in Engineering Cybernetics from the University of Stuttgart, Germany in 2002. He then switched to the University of Kaiserslautern, where he received a Ph.D. in Electrical and Computer Engineering in 2007 with a dissertation on the formal analysis of temporal behavior of Networked Automation Systems (NAS) by use of probabilistic model checking. Scientific stays at the University of Arizona (USA), at the Ecole Normale Supérieure de Cachan (F), and at the Universidade de Brasília (BR) completed his education. Since 2008, Dr. Greifeneder is with the ABB Corporate Research Center, Ladenburg, Germany, working in the area of automation engineering with a focus on simulation-based engineering. Dr. Greifeneder's primary research interests concern modeling and simulation methodologies.

Modelica Library for Improved Spacecraft Resource Budgeting

Niccolo Cymbalist Marc-Andre Lauriault Chahé Adourian (supervisor)
Space Technologies, Canadian Space Agency
John H. Chapman Space Centre 6767 Route de l'Aéroport Saint-Hubert, QC, Canada

Abstract

The *SpacecraftLib* library has been developed in Modelica for use in the domain of Systems Engineering for space systems with a special emphasis on modularity, usability and ease of modification and expansion. It is a multidisciplinary tool which combines all the relevant subsystems. Power, command and data handling, and mechanical models are integrated into a single Modelica *device* in order to model as completely as possible the behaviour of a physical onboard device. We will describe the tool, examine a case study and briefly analyze the results of a simulation.

Keywords: spacecraft simulation; resource budgeting; systems engineering; command network

1. Introduction

Modeling and simulation tools in Systems Engineering for spacecraft have the potential to improve the efficiency of the design process. One task in which simulation may be effectively utilized is to assist in the generation of requirements through improved resource budgeting. Budgeting in this context is defined as the process of characterizing the components which affect an overall system parameter while focusing on system level requirements and trade-offs [1].

A simulation tool for use in Systems Engineering as a whole, and specifically the resource budgeting exercise, must meet the following requirements:

- **Simple**. The model must have the capability of being rapidly and easily assembled and modified.
- **Multidisciplinary**. The behaviour and interactions of power, command and data handling, mechanical and thermal systems and link behaviour must be modeled together.
- **Appropriate level of detail**. The model must be accurate enough to define requirements.
- **Easily Customizable and Expandable**. The tool must be able to be extended for use in more advanced stages of the design process.

Currently available spacecraft simulation tools are generally either simple spreadsheet based models, mission design tools (*STK*)[2], complex custom built simulators or tools targeted at a specific subsystem, such as Attitude and Orbit Control Systems (AOCS)[3]. While these tools are all useful in their respective domains, they do not meet the needs of a resource budgeting tool for spacecraft Systems Engineering.

This paper introduces a Modelica library specifically designed for resource budgeting, capable of expanding to a full design tool, called *SpacecraftLib.* It can be used to build a ground station and multiple spacecraft models, each including power, payload, command and data handling subsystems and link models. The spacecraft models receive and react to time-tagged commands during the simulation and interact with commercial spacecraft modeling software for advanced functions.

The use of this tool will allow the user to optimize the level of complexity of the simulator at each stage of the design process, especially in the early stages of the design, by increasing or decreasing the complexity of the models built using *SpacecraftLib*. This will lead to a more effective and efficient design process.

This library is implemented in Modelica because we find it well suited to hybrid, multidisciplinary modeling due to its modularity and ease of use.

DOI: 10.3384/ecp09430037

2. The *SpacecraftLib* library

The *SpacecraftLib* library is divided into 4 main sections, each containing components used to model a different subsystem.

These components are easily assembled into *devices*, which behave as physical onboard devices would. They have mass and inertia, consume power, generate data and interact with the user or onboard computer via the command network.

- The *DataBudget* package includes components for processor, data flow, command network and communications equipment modeling.
- The *PowerBudget* package includes components used for power generation and distribution modeling. It also includes components used to model the power consumption characteristics of onboard devices.
- The *Mechanical* package includes a version of the Multibody library expanded to account for advanced gravity and magnetic fields modeling. It includes a modified *world* model, as well as reaction wheel, extendable solar array and thruster assemblies. It also includes controllers for each of the assemblies.
- The *OrbitalMechanics* package includes an ephemerides generator for the nine planets and the Moon based on external ephemeris tables.

We provide the user with an inheritable *device* template with components common to all devices: a command port and command decoder, power consumption characteristics and a state box. To model the behavior of a specific device, the user may *extend* the *device* template to a new class and add the desired components from the available subsystem packages (see Figure 1).

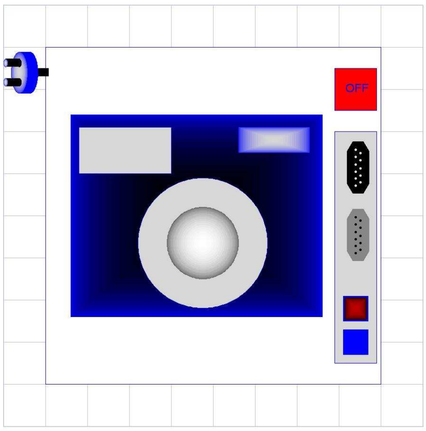


Figure 1. Camera device

The devices are in turn assembled into a complete spacecraft model which is initialized into an orbit. The model exchanges data with the ground station, generates and consumes power and responds to user commands.

SpacecraftLib is complemented by ephemeris tables and solar flux data from *STK* and a commercial spacecraft modeling library called *Spacecraft Control Toolbox*[4]. The latter was converted from MATLAB to C to interface with Modelica.

2.1 *DataBudget* section

SpacercraftLib provides components to model the processor unit(s), the command network, the flow of data between different devices and the ground station, and the communication equipment on the ground and onboard the spacecraft(s). The results of the simulation allow the user to accurately define the requirements of the command and data handling system onboard the spacecraft and on the ground.

2.1.1 Data flow modeling

In *SpacecraftLib* data is treated as a flow, behaving such that it may be generated, deleted, or compressed. Data modeling components include various data links, data storage units, data sources and sinks, and data processing units. These components are purely conceptual, designed to model the behavior of a data handling system instead of modeling the actual physical components of the system.

Data ports are *flow* connectors together with a control line. Data ports may be active or passive.

The active ports define the bit rates (in or out) while the passive ports simply accept this flow.

The passive port may send back information regarding the state of the receiving device via the control line. This allows the active device to stop removing (feeding) data when the receiving device is not able to give (accept) data. Link components are either variable bit rate links with a maximum bit rate set by the user or links that compress or expand the volume of data. The latter are used to model data compression and EDAC (error detection and correction) operations. Switch components include on/off switches and splitters.

2.1.2 Command network modeling

The effort to maximize the usability of the tool led to the creation of a flexible layered command bus with 'plug and play' behavior. It allows for rapid assembly or modification of a model by dropping in a new device, naming it, and connecting it to the bus. The command network model is portable and has potential applications outside the field of spacecraft engineering.

The command network contains two complimentary levels of hierarchy. One level is name based and one level is based on numerical indices. The name based level roughly corresponds to the physical location of the device to which the command is intended for, and is set by the user. The second level is based on numerical indices, and is automatically generated and hidden from the user. This level is used internally to map the network of devices so that each 'parent' device knows the location of all its 'child' devices and is able to forward the command to the appropriate device.

This combination of name and numeric based addressing separates the user from the numerical address system, allowing the user to specify the command destination using only the physical location of the device, as demonstrated in Figure 2.

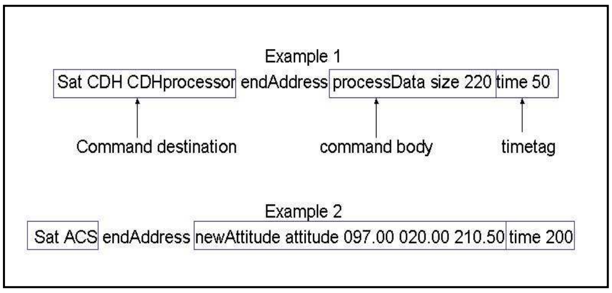
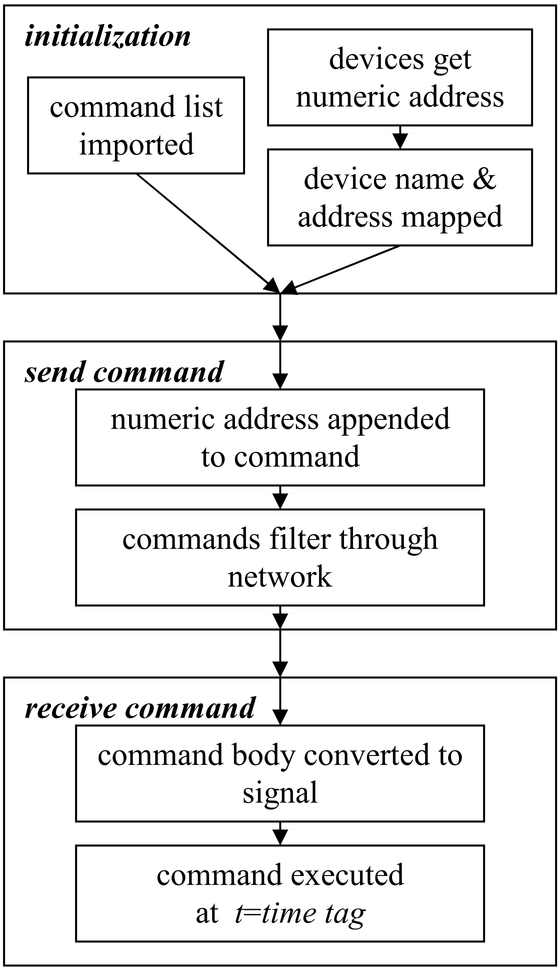


Figure 2. Command syntax examples

In order to achieve 'plug and play ' level usability, the Modelica code is complemented by a custom built C library. The Modelica model (Modelica side) and C library (C side) run concurrently throughout the simulation. Flowchart 1 illustrates the sequence of events in the command network model.


initialization
command list imported
devices get numeric address
device name & address mapped
send command
numeric address appended to command
commands filter through network
receive command
command body converted to signal
command executed at t=time tag

Flowchart 1. Sequence of events in command network model

The command network is composed of four components on the Modelica side (see Figure 3), including the *commandInput, networkNode, device* and *satControls* components.

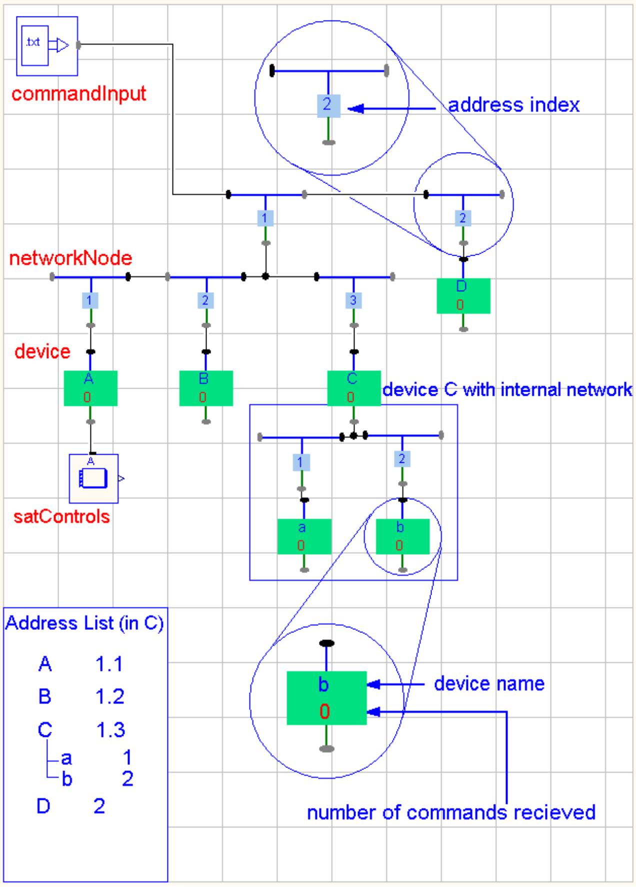


Figure 3. Command network example

1. The *commandInput* component imports the textual commands from the user, parses and translates them and stores them in a buffer. It retrieves the appropriate numerical address for each command from the automatically generated device list on the C side and appends it to the command (see *networkNode* description, next paragraph, for details on the address generation mechanism). *CommandInput* then sends the commands through the network in packets. The size and frequency of the packets are defined by the user.

2. The *networkNode* component is a node in the network between different levels. It contains a user assigned address index which is used to generate the addresses of devices under it (at initialization) and to filter the commands by their address indices (when the commands are sent during the simulation). The Modelica code automatically generates an address at initialization by propagating an empty array from the top to the lower levels of the network. At each node in the network, the empty array gets progressively populated, recording its path through the network from the command source (which may be either the *commandInput* component or a parent *device*) to each device.

3. The *device* component is at the receiving end of the command network. The user assigns a name to the *device* instance as a parameter. Upon initialization, the *device* receives a unique identifier (UID) from the C side and registers itself to the list of devices by passing its name, UID, address and parent device UID (if applicable) back to the C side. When a *device* receives a command or packet of commands, it either passes them on to a sub-device through its internal network or feeds the commands to the *satControls* component.

4. The *satControls* component receives the commands, extracts the time tag, translates them to the appropriate signals and places the commands in a queue. When simulation time matches the time tag the command is executed.

On the C side, there is a unique identifier generator and the address list which contains the device UID, address, name and parent/child relationship with other devices. The C code and Modelica model interface via three *external C* functions.

1. The *registerUID* function is called by the *device* component to obtain a unique identifier from the C side.

2. The *registerDevice* function is called by the *device* component to pass its name, address, UID and parent UID to the C side to be added to the device list (mapped).

3. *getDeviceAddress* is called by the *commandInput* and *device* components, to retrieve the device and sub device addresses, respectively.

2.1.3 Processor modeling

The processor model is used to test sequences of instructions in a process and to evaluate the processing load on the CPU. The main components are the processor model template and the individual process blocks (see Figure 4).

The processor model template is an extendable version of the *device* model, with the addition of data ports and additional parameters to characterize the processing power and internal bitrates. The user may select from the available process blocks to build a new processor model, or implement custom process blocks. Standard processor options are also available.

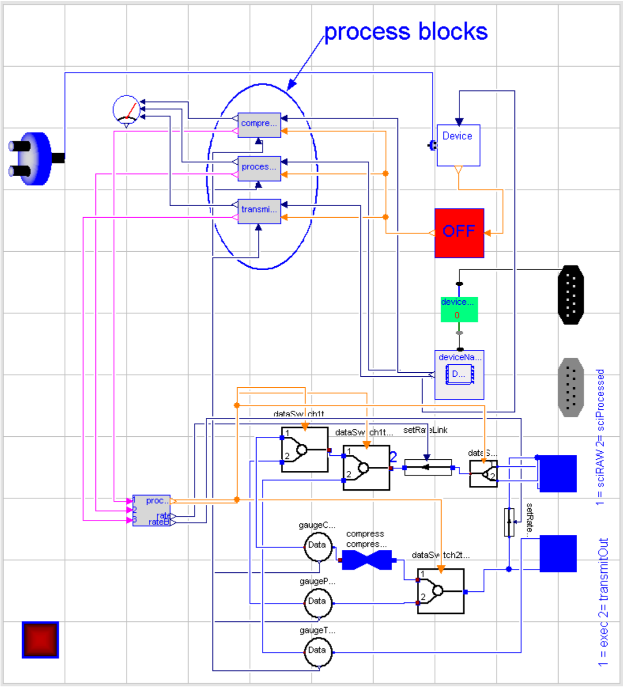


Figure 4. An example of a processor model

Each process block contains a set of instructions which are executed in sequence when the process block is activated. The processor block executes the instruction by sending signals to the 'dumb' components in the processor, such as the data link, compress and switch components.

2.1.4 Communications modeling

The communications modeling components include:

- Onboard communication equipment consisting of customizable TX and RX antennas. User defined parameters include antenna gain, transmission power and frequency.
- Ground station antenna, characterized by its location in latitude and longitude, horizon angle, gain and by the parameters affecting attenuation.
- The link analysis block uses Princeton Satellite functions to calculate the theoretical maximum uplink and downlink bitrates (Shannon limit) based on the ground station and onboard antenna parameters and the location of the spacecraft and ground station.

2.2 *PowerBudget* section

The power budget section contains the components related to the generation, storage, distribution and consumption of power which would be found in a typical photovoltaic cell based power subsystem. Customization is achieved through parameterizing the components and through redeclaration. For instance, the user may use one large solar array at one average temperature or multiple smaller solar arrays of different temperatures to account for temperature differences across a physical solar array.

- The solar array model is parameterized by the number and area of solar cells, maximum power current, maximum power voltage, array temperature and temperature coefficients. It generates power according to the angle of incidence of the panel and the intensity of the solar flux.
- The battery model takes into account the cell capacity, maximum depth of discharge, charge/discharge ratio, and maximum charge and discharge rate.
- The power distribution model interacts with the power consumption block in the *device* model to provide power according to the state of device: on, off or idle. The *device* has parameters to specify the power consumption at each state.

2.3 *Mechanics* section

In order to model the physical characteristics of each device as well as the spacecraft Attitude and Orbit Control System, the standard Modelica Multibody *world* model and *body* model were modified to incorporate more complex gravity and magnetic fields, as well as gravity gradient effects [5]. Various actuators including reaction wheels, thrusters, torque rods and their respective controllers are included in the *Mechanics* section.

2.3.1 *world* model

The extended *world* model includes the option to use a spherical harmonics gravity model instead of point gravity. This component models the Earths magnetic field using the International Geomagnetic Reference Field (IGRF) and incorporates a default spherical earth visualization where the radius is set to the average Earth radius. The user may customize the precision of the gravity and magnetic field models by specifying the number of coefficients used for the gravity and magnetic field polynomials. The *world* model detects when the spacecraft comes in contact with the earth surface.

2.3.2 *complexBody* model

This model is derived from the standard *body* model, modified to allow the user to account for gravity gradient torque, if desired. The gravity gradient calculation function is in the *world* model.

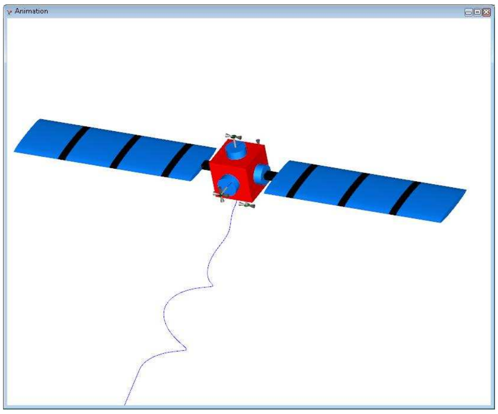


Figure 5. *Multibody* spacecraft visualization

Available actuators include reaction wheels, thrusters and magnetic torque rods. All are assembled from the standard Multibody library components and the modified *complexBody* model.

- Reaction wheels are generally mounted on each of the 3 axes. One spare wheel is mounted obliquely and ready to take over in case one of the 3 primary wheels fails. In order to accurately model any unique reaction wheel assembly, the user can specify the orientation of each wheel, shape, size and density of the rotor.

- Thrusters are modeled by exerting a frame force on the thruster nozzle. The user specifies the mass of the thruster assembly and the maximum force exerted by the thruster, as well as the position and orientation of the thruster nozzle.

- Magnetic torque rods are modeled by exerting a torque on the torque rod body. The component *magneticFieldSensor* is used to measure the magnitude and orientation of the magnetic field in order to apply the appropriate torque.

There are controller blocks for performing operations including detumbling (stabilizing the spacecraft into a certain attitude after separation from the launcher, see Figure 5) and momentum unloading (using the torque rods to decrease the momentum which has accumulated in the reaction wheels), as well as routine changes in attitude and orbit.

2.4 Orbital mechanics

The orbital mechanics section includes a precise ephemeredes model based on tables generated by *STK*. It generates the position in heliocentric equatorial coordinates for the nine planets and the Moon by interpolating the ephemeris tables using a Lagrange polynomial expansion.

3. Case study

To verify the behavior of *SpacecraftLib* , we built a generic spacecraft (called *Sat,* see Figure 6) with all the major subsystems. These include command and data handling (C&DH), power control and distribution unit (PCDU), Tracking, Telemetry and Control (TTC), Attitude Control System (ACS) and a payload consisting of 2 sensors. We orbited the spacecraft for 2 orbits, and uploaded a list of commands to be performed over the duration of its short ‘mission’.

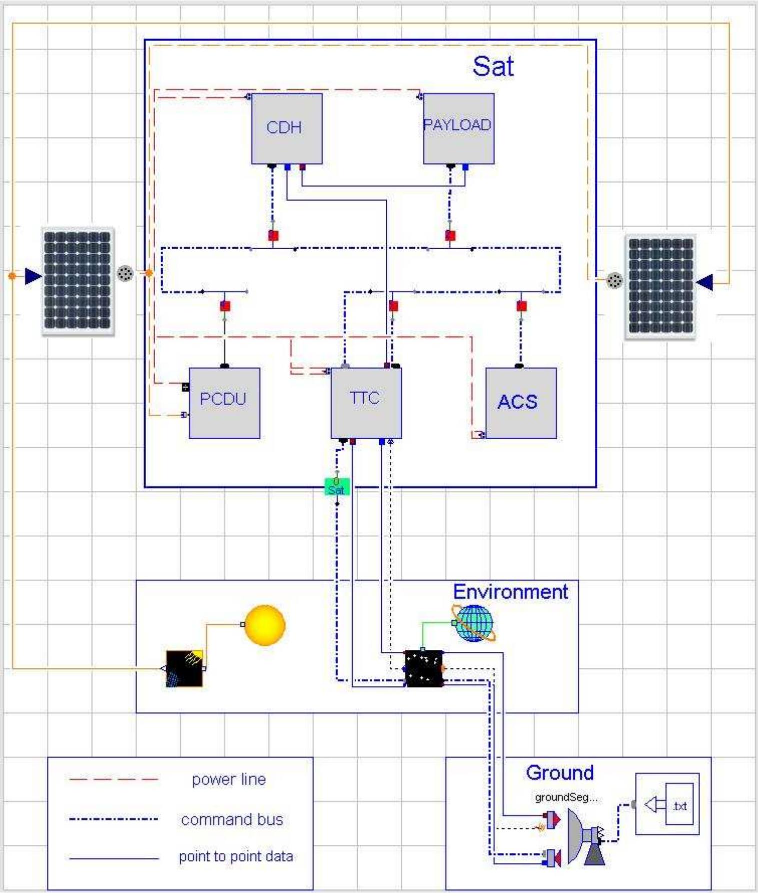


Figure 6. Model of complete spacecraft *'Sat'* and ground station

- C&DH. This subsystem is composed of 1 processor with a maximum processing power of 100 KIPS and internal bitrate of 10 Mbps, a science data memory unit, and an execution memory unit.
- PCDU. This subsystem includes the power controller and a battery, and is connected to 2 solar arrays.
- TTC. This subsystem includes a processor, buffer memory, a transmitting antenna and a receiving antenna.
- ACS. For this example, the attitude control system is a dummy system to model power consumption. Multibody components are not used and the orbital parameters for the duration of the simulation are imported from STK.

The ground station is located in St Hubert, Quebec, at the location of the Canadian Space Agency while the spacecraft orbits at an altitude of approximately 820 km in a near polar, circular orbit, with an inclination of 98.7 degrees.

We entered the commands into a .txt file before compiling the model and ran the simulation for 12000 seconds (about 2 orbits).

The commands included image capture and compression operations, device state changes (on, off and idle), attitude changes and data transmission operations. The model reacted to these commands at the appropriate times, exhibiting behaviour which would allow us to optimize the design.

3.1 Results and Analysis

We will examine the behaviour of the C&DH subsystem as well as the link behaviour.

In Figure 7, the upper plot shows the amount of data in the memory space allocated to unprocessed science data. Each spike corresponds to an unprocessed image entering the raw data memory, and immediately being removed, processed and placed in the memory allocated to processed data (not plotted).

The lower plot in Figure 7 shows the status of the ‘shutter’ on the camera sensors. At each pulse, the ‘camera’ takes an image and generates a certain amount of data, which is passed to the raw data memory.

In Figure 8, the upper plot shows the maximum theoretical bitrate possible according to the Shannon-Hartley theorem[1] and the actual transmission bit rate. The lower plot shows the amount of data in the TTC memory (buffer). As soon as there is a link, the data in the TTC memory is transmitted.

Certain facts which have an impact on the design are immediately evident. From the upper plot in Figure 7, we can conclude that for this operating profile, there must be at least 350 to 400 Mb of storage allocated to the raw data from the sensors.

From Figure 8 we see that the communications system is not optimized for this operating profile. At a downlink bitrate of 10 Mbps it takes only a fraction of the available link duration to download all the data. This is an indication that communication equipment capable of 10 Mbps downlink speeds is not necessary for this imaging rate, which will have secondary and tertiary effects on the power subsystem specifications, solar array size, and eventually on the total mass of the spacecraft. Moreover, the closer we get to the Shannon limit of the channel the more we must invest in terms of hardware and power for advanced coding techniques.

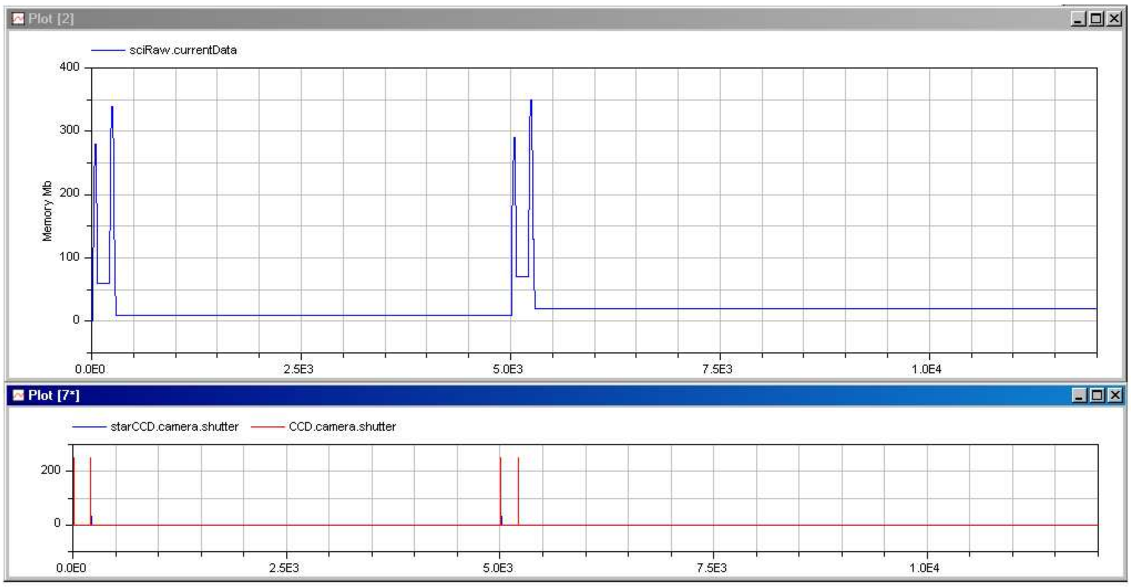


Figure 7. Science memory level (upper), camera state (lower)

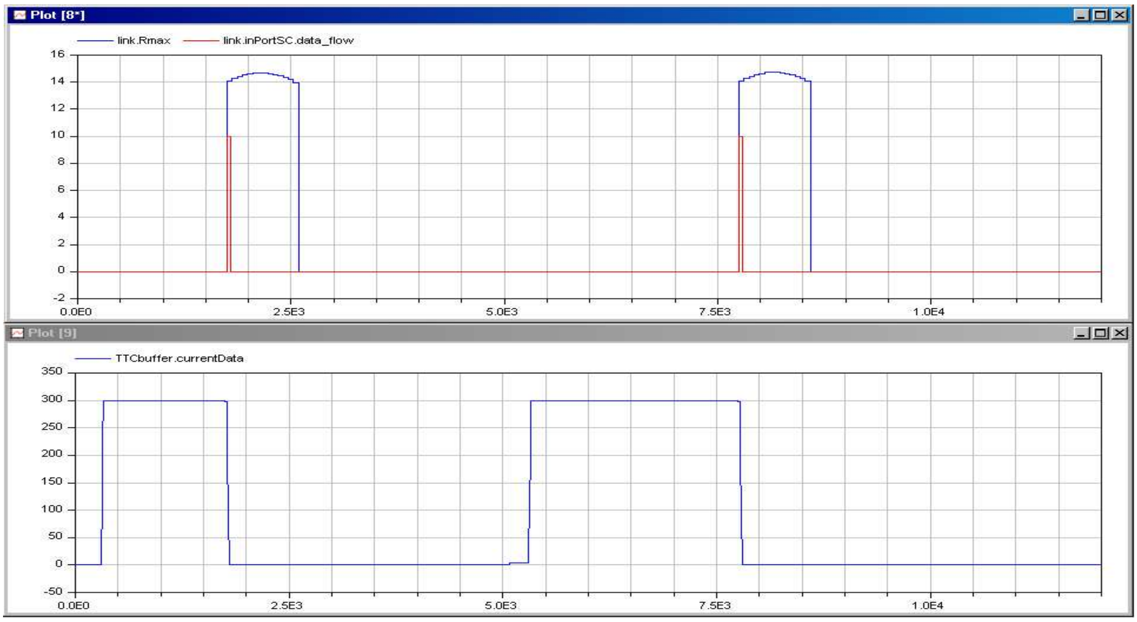


Figure 8. Maximum theoretical bitrate and actual bitrate (upper), TTC buffer level (lower)

4. Conclusion

Modelica is proving to be well suited to hybrid modeling of power, data and command systems as applied to a generic spacecraft modeling tool for Systems Engineering, when complemented by our C library. It provides a flexible graphical modeling environment that allows for fast and accurate estimates for use in resource budgeting. The precision of the estimates depends on the fidelity of the subsystem models, so further improvement of the subsystem models will allow the tool to progress to a full design tool. Various features of the Spacecraft Budgeting Library, notably the command network model, are portable and may be reused for other applications.

References:

[1] W.J. Larson, and J.R.Wertz, editors. *Space mission analysis and design*. Microcosm Press/Kluwer Academic Publishers, 1999.

[2] *STK 8*, Analytical Graphics, Inc.

[3] T. Pulecchi, F. Casella, M. Lovera. A Modelica Library for Space Flight Dynamics. In *Proceedings of the 5th Modelica Conference, Vienna, Austria,* volume 1, page 107.

[4] *Spacecraft Control Toolbox,* Princeton Satellite Systems, Inc.

[5] Built by Dyna Benchergui and Andre-Claude Gendron

Modelling and simulation of a fault-tolerant electrical motor for aerospace servovalves with Modelica

Gianpietro Di Rito Roberto Galatolo
Dipartimento di Ingegneria Aerospaziale - Università di Pisa (Italy)
Via Caruso, 8 – 56122 Pisa (Italy)
g.dirito@ing.unipi.it r.galatolo@ing.unipi.it

Abstract

The paper deals with the design and the simulation of electro-magnetic actuators via object-oriented modelling. The study is carried out in the Modelica-Dymola environment, and focuses on the quadruple-coil direct-drive motor of a modern fly-by-wire servovalve. Starting from basic information about material properties and from a schematic representation of the system geometry, the motor model is created mainly using the components of the *Modelica_Magnetic* library. The motor performances are then characterised with reference to both the normal operating condition (four active coils) and the worst-case fault-tolerant condition (only two active coils), in terms of current-to-force and current-to-displacement curves. The Modelica model is finally validated by comparing the simulation results with experimental data obtained during previous research activities. The Modelica results are also compared with those provided by a Matlab-Simulink model of the motor, pointing out the advantages of the object-oriented approach for the study of complex electro-magnetic systems. The easy modelling of magnetic circuit networks and the inherent simulation of magnetic material properties allow to achieve accurate results very efficiently, taking into account physical phenomena that are often disregarded during preliminary design phases, such as magnetic saturation or magnetic flux dispersions.

Keywords: fault-tolerant aircraft systems; electro-magnetic actuators; object-oriented modelling.

1 Introduction

The use of direct-drive servovalves, essentially based on the use of a rare-earth magnet electrical motor for controlling the valve spool motion, demonstrated to be strategic for the enhancement of the performance and the reliability of modern fly-by-wire hydraulic actuators [1]. Actually, as a result of the separation between the electrical and the hydraulic section of the servovalve, fault-tolerant actuators can be obtained with simpler architectures, and the valve spool position control can be designed with great flexibility [2].

A fly-by-wire actuator with direct-drive servovalve is a complex multi-physical system, where electrical, magnetic, electronic, hydraulic and mechanical phenomena are strongly connected with each other, so the modelling and simulation can be problematic if the designer/analyst does not have technical expertise in all domains. At the same time, an actuator model is required to be accurate in the early phases of a project, well before the item is actually constructed, especially if safety-critical and high-performance application is concerned (e.g. primary flight control actuators). In this context, the object-oriented approach provided by Modelica [3, 4] can represent a very convenient solution for the rapid prototyping, the analysis, and the performance characterisation of such systems.

In the work, the model of a quadruple-coil direct-drive motor for fly-by-wire servovalves is developed in the Modelica-Dymola environment, and validated with experimental data in terms of static performances. The paper is organised into three sections. The first one reports a brief description of the system principle of work, together with a simple analytical model of the electro-magnetic section of the motor. The second part is focused on the main features of the Modelica model, mostly developed by using the components of the *Modelica_Magnetic* library [5], while the third section is dedicated to the model validation. In particular, the Modelica results are compared with both experimental data (obtained during previous research activities) and simulation

DOI: 10.3384/ecp09430091

results of a Matlab-Simulink model of the motor [6], in order to point the advantages of the object-oriented modelling for the study of complex electro-magnetic systems.

2 System description

2.1 Motor layout

The internal architecture of the direct-drive motor is schematically depicted in Fig. 1. The valve spool movement is obtained by a rare-earth magnet electrical motor with four coils operating in a flux-summed configuration. When no current circulates in the coils, the armature (rigidly linked to the valve spool) is centred with respect to its endstrokes, as in this condition the centring spring is unloaded and the magnetic fluxes induced by the permanent magnets provide equal and opposite forces. When the currents are not zero, the magnetic flux induced by the coils causes an unbalanced armature polarisation. A magnetic force is therefore generated, and the resulting spool movement allows to control the actuator hydraulic power.

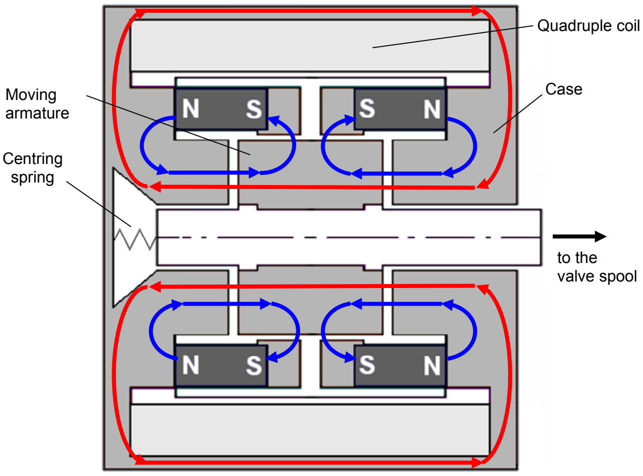


Figure 1 – Motor layout and basic magnetic network.

Modelling the dynamics of a direct-drive motor is a complex topic, since several nonlinear phenomena are involved. First of all, the magnetic force is a nonlinear function of both the current and the spool position, since the polarisation efficiency is considerably increased if the armature approaches the endstroke. Moreover, the effect of unavoidable system nonlinearities such as magnetic saturation or sliding friction cannot be neglected if the simulation is referred to the complete range of operating conditions.

2.2 Principle of work

In order to clarify the basic working principle of the motor, a simple analytical model of the electro-magnetic section of the system is here provided. The magnetic field of the motor can be basically represented by three magnetic fluxes [7]: the one that links the coils with the variable air gaps (φ_c), and the two ones linking the left and right magnets with the left and right variable air gaps respectively (φ_{ml} and φ_{mr}).

The resulting magnetic circuit network is shown in Fig. 2.

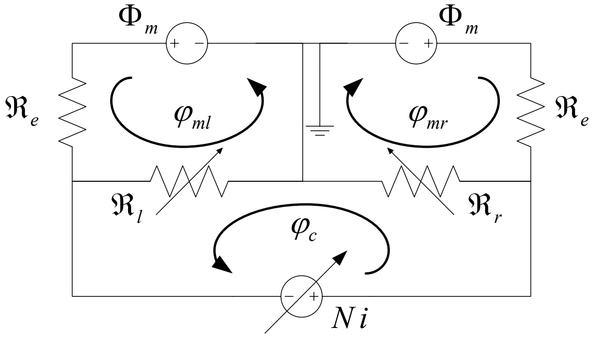


Figure 2 – Simplified magnetic network of the motor.

The network is characterised by three reluctances: two ones related to the variable air gaps ($\Re_l$ and $\Re_r$), and an overall equivalent reluctance that takes into account all the soft ferrite parts, the radial clearance, and the reluctance of the permanent magnets ($\Re_e$).

Assuming that all magnetic flux tubes have the same cross-section area (A), the three reluctances are given by Eqs. (1-3), where μ is the air magnetic permeability, g is the motor air gap, and x is the armature displacement from the centred position.

$$\Re_e = l_e / \mu A \tag{1}$$

$$\Re_l = (g - x) / \mu A \tag{2}$$

$$\Re_r = (g + x) / \mu A \tag{3}$$

By solving the circuit equations (Eq. (4)), the magnetic fluxes can be obtained as functions of the armature position and the coil current (Eqs. (5-7).

$$\begin{bmatrix} \Re_e + \Re_l & 0 & \Re_l \\ 0 & \Re_e + \Re_r & -\Re_r \\ \Re_l & -\Re_r & \Re_l + \Re_r \end{bmatrix} \begin{bmatrix} \varphi_{ml} \\ \varphi_{mr} \\ \varphi_c \end{bmatrix} = \begin{bmatrix} \Phi_m \\ \Phi_m \\ N i \end{bmatrix} \tag{4}$$

$$\varphi_{ml} = \frac{\mu A}{l_e}\left[\frac{1}{1+g/l_e - x^2/g\,l_e}\Phi_m + \ldots \right. \qquad (5)$$
$$\left. \ldots - \frac{1+g/l_e - x/g - x^2/g\,l_e}{2\,(1+g/l_e - x^2/g\,l_e)} N\,i\right]$$

$$\varphi_{mr} = \frac{\mu A}{l_e}\left[\frac{1}{1+g/l_e - x^2/g\,l_e}\Phi_m + \ldots \right. \qquad (6)$$
$$\left. \ldots + \frac{1+g/l_e + x/g - x^2/g\,l_e}{2\,(1+g/l_e - x^2/g\,l_e)} N\,i\right]$$

$$\varphi_c = \frac{\mu A}{l_e}\left[\frac{x/g}{1+g/l_e - x^2/g\,l_e}\Phi_m + \ldots \right. \qquad (7)$$
$$\left. \ldots + \frac{2+l_e/g + g/l_e - x^2/g\,l_e}{2\,(1+g/l_e - x^2/g\,l_e)} N\,i\right]$$

The force provided by the motor can be then obtained by applying the virtual work principle, by differentiating the system magnetic co-energy with respect to the armature position, Eqq.(8-9).

$$F_m = \frac{\partial E_m}{\partial x} = \frac{\Phi_m}{2}\frac{\partial(\varphi_{ml}+\varphi_{mr})}{\partial x} + \frac{N\,i}{2}\frac{\partial \varphi_c}{\partial x} \qquad (8)$$

$$F_m = \frac{\mu A}{l_e\, g}\left[\frac{2\Phi_m{}^2\, x/l_e}{(1+g/l_e - x^2/g\,l_e)^2} + \ldots \right. \qquad (9)$$
$$\left. \ldots + \frac{\Phi_m N i\,(1+g/l_e + x^2/g\,l_e)}{(1+g/l_e - x^2/g\,l_e)^2} + \frac{N^2 i^2 (1+g/l_e)\,x/g}{2\,(1+g/l_e - x^2/g\,l_e)^2}\right]$$

As shown by Eq. (9), the magnetic force provided by the direct-drive motor is composed of three basic terms: the first one is exclusively due to the permanent magnets, it depends on the armature position and its effect can be viewed as a diminution of the centring spring stiffness; the second term is the most important force contribution, it is linear with the coil currents and it is produced by the interaction between the magnetic fluxes induced by the currents and the ones induced by the permanent magnets; the third term is exclusively due to the coils, and it tends to be important when the motor works with high currents [7].

2.3 Main requirements for application on aerospace servovalves

From the working principle point of view, there are no significant differences between a single-coil direct-drive motor and a quadruple-coil one, but specific design requirements must be addressed for the application on fault-tolerant aerospace servovalves:

- Linearity: a linear relationship between the command current and the valve displacement must be provided by the motor in all the operating conditions. Nonlinear phenomena, such as magnetic saturation, magnetic hysteresis or sliding friction, must be kept at negligible levels, especially in the vicinity of the “null region” of the valve (centred motor armature);
- Functionality in case of electrical failures: the direct-drive motor must maintain its functionality even after two electrical failures, i.e. the working valve stroke must be completely covered also in the worst-case failure condition;
- Functionality in case of contaminated fluid: even in the worst-case failure condition[1], the motor must have sufficient force capability to shear a metallic chip blocked in one of the servovalve port (chip shear force). The cross-area of the blocked chip is generally referred to the maximum valve opening.

Undoubtedly, the coverage of each of the above-mentioned requirements must be assessed through experiments, but predictions and analyses are necessary during the whole design development, especially because the tests on the functionality in case of failures are complex and expensive. For these reasons, modelling and simulation activities become essential, and the need of accurate system models is evident.

3 Modelica model development

The model of the direct-drive motor has been divided into two sections: the one related to the permanent magnets, and the other to the quadruple coil.

3.1 Permanent magnet section

Previous works of the authors highlighted the role that the magnetic flux dispersion has in this type of system [6, 7]. For this reason, the permanent magnet section of the motor has been modelled with the magnetic circuit network shown in Fig. 3. The magnetic field of this motor section is composed of six fluxes: two ones linking the permanent magnets and the variable air gaps,

[1] Depending on the application, this aspect of the requirement could be relaxed, since extremely small probability of occurrence are related to multiple events.

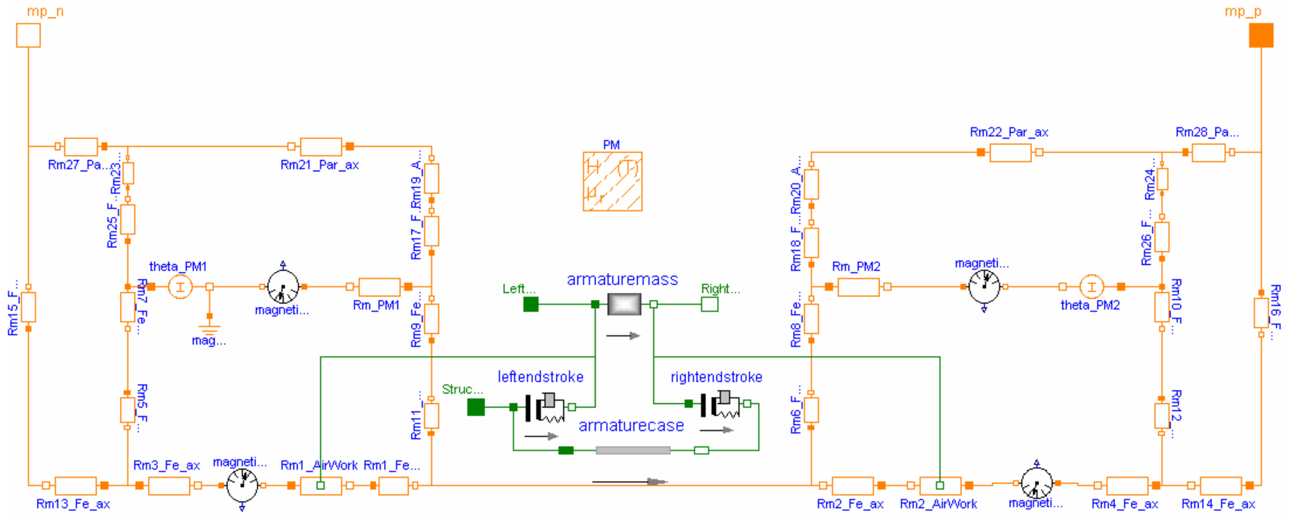


Figure 3 – Permanent magnet section of the motor model.

and four flux leakages that take into account the secondary paths of the magnetic fluxes.

The mechanical part of the model simply represents the armature mass and the containing case, with two *ElastoGap* components of the *Modelica.Mechanics.Translational* library, which simulate the left and right endstroke of the motor.

The interfaces of the resulting object-model are two magnetic ports for the integration with the magnetic network of the command coils, and two mechanical flanges, to be connected with the fixed structure of the hydraulic actuator and to the valve spool respectively.

3.2 Quadruple-coil section

The quadruple-coil section has been modelled taking into account of the secondary magnetic paths related to each coil (Fig. 4), so that the magnetic field contains four magnetic dispersion loops.

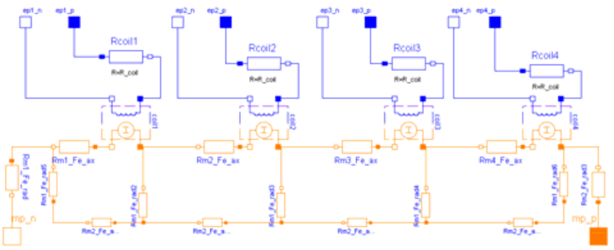


Figure 4 – Quadruple-coil section of the motor model.

The external interfaces of the resulting component are eight electrical pins (two ones for each of the four command electronics) and two magnetic ports for the integration with the magnetic network of the permanent magnet section.

3.3 Complete model

The complete model of the direct-drive motor (Fig. 5) is finally obtained by magnetically linking the permanent magnets with the coils (reproducing the actual physical relationship between the two motor sections), and by adding a *Spring* and a *Stop* object-models coming from the *Modelica.Mechanics.Translational* library for simulating the centring spring and the effects of sliding friction respectively.

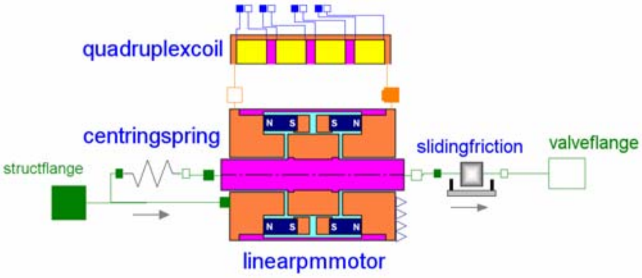
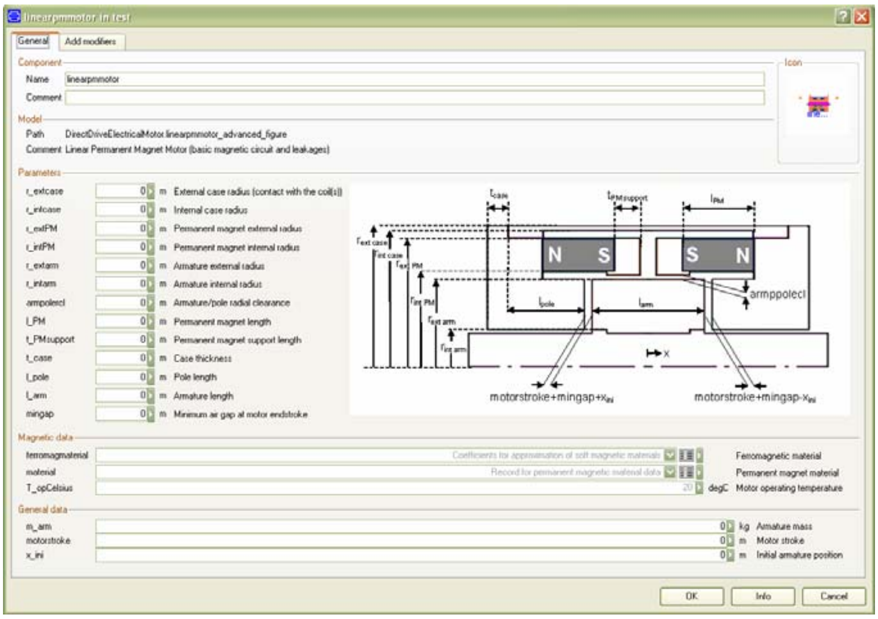


Figure 5 – Model of the fault-tolerant direct-drive motor.

3.4 Graphical user interface

The modelling activity led to the creation of three Modelica object-models: *linearpmmotor*, *quadruplexcoil* and *directdrivemotor*. All the object-models have been created providing the user with the possibility of tuning and selecting the system parameters via dialog box (e.g. geometrical dimensions, coil windings, coil



Component

Name linearpmmotor

Comment

Model

Path DirectDriveElectricalMotor.linearpmmotor_advanced_figure

Comment Linear Permanent Magnet Motor (basic magnetic circuit and leakages)

Parameters

r_extcase	0	m	External case radius (contact with the coil(s))
r_intcase	0	m	Internal case radius
r_extPM	0	m	Permanent magnet external radius
r_intPM	0	m	Permanent magnet internal radius
r_extarm	0	m	Armature external radius
r_intarm	0	m	Armature internal radius
armpolecl	0	m	Armature/pole radial clearance
l_PM	0	m	Permanent magnet length
t_PMsupport	0	m	Permanent magnet support length
t_case	0	m	Case thickness
l_pole	0	m	Pole length
l_arm	0	m	Armature length
mingap	0	m	Minimum air gap at motor endstroke

Magnetic data

ferromagmaterial	Coefficients for approximation of soft magnetic materials		Ferromagnetic material
material	Record for permanent magnetic material data		Permanent magnet material
T_opCelsius	20	degC	Motor operating temperature

General data

m_arm	0	kg	Armature mass
motorstroke	0	m	Motor stroke
x_ini	0	m	Initial armature position

Figure 6 – Dialog box for the model of the permanent magnet section.

resistance, material properties, etc.).

An example is given in Fig. 6, where the graphical user interface of the *linearppmmotor* object-model is reported.

4 Simulation and validation

The Modelica model of the motor has been developed with reference to the fault-tolerant direct-drive motor of the servovalve of a modern fly-by-wire actuator, which has been experimentally characterised by the authors during previous research activities [6]. The model validation has been achieved by comparing the Modelica results with the experimental data in terms of current-to-force and current-to-displacement curves, with reference to both the normal operating condition (four active coils) and the worst-case fault-tolerant condition (only two active coils).

In this section, the results of a Matlab-Simulink simulation of the motor [6] are also reported, in order to highlight the advantages of the object-oriented modelling with respect to a more classical approach.

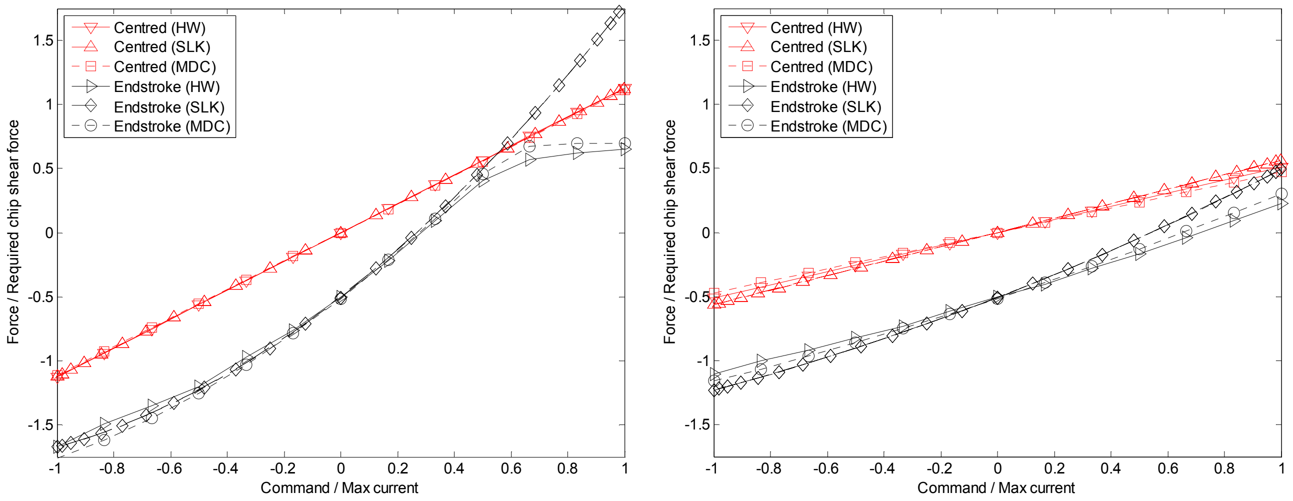
4.1 Current-to-force characteristics

The current-to-force characteristics of the motor are reported in Fig. 7, where the net force (i.e. including the centring spring effect) is plotted as a function of the coil current for different armature positions (centred and endstroke). The plot shown in Fig. 7.a is referred to the normal operating condition, while the Fig. 7.b refers to the case of only two active coils. The net force values have been normalized with respect to the required chip shear force of the valve.

The most interesting considerations can be done with reference to Fig. 7.a. The results show that both the Modelica (MDC) and the Simulink (SLK) models provide satisfactory results if a working condition with centred armature is concerned: both the direct-drive motor models exhibits a linear behaviour, with negligible errors with respect to experimental data. Furthermore, both the models provide good predictions in terms of cheap shear force capability[2]. On the other hand, the MDC model is more accurate when the armature is placed at the endstroke and high positive currents are applied. This is because the magnetic saturation effects are not taken into account in the SLK model, and the magnetic force capability of the motor is overestimated.

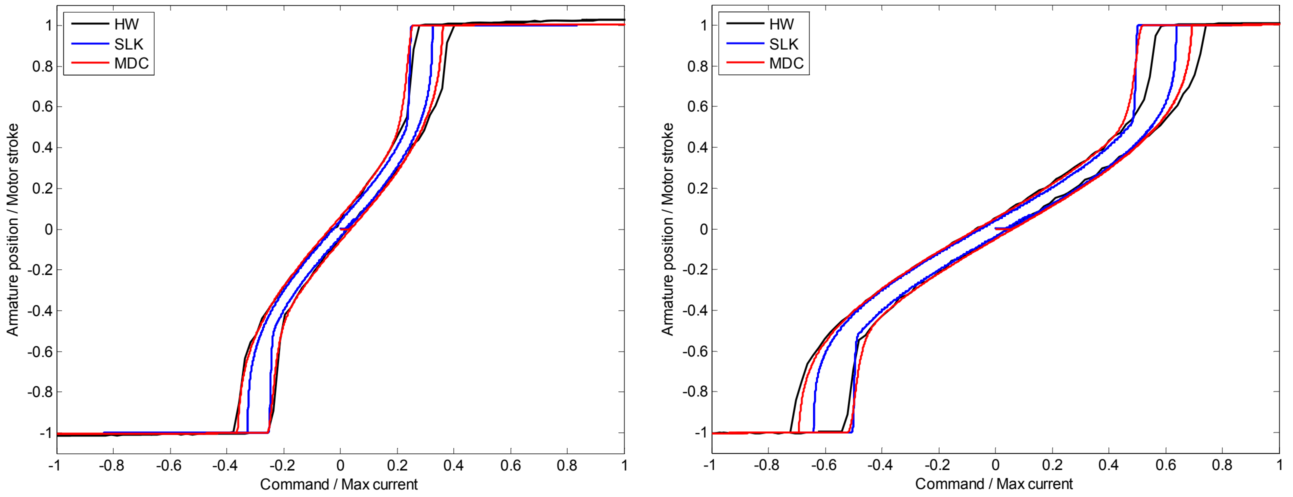
The analytical model proposed in section 2.2, though simplified, can be useful for understanding the phenomena, since the magnetic saturation of soft ferrite parts is not taken into account as well. Equation 7 points out that the magnetic flux

[2] The chip shear force capability can be obtained from Fig. 6 by measuring the net force provided at endstroke (i.e. maximum cross-area of the blocked chip) when the maximum negative current is applied.



(a) (b)

Figure 7 –Motor force characteristics with four active coils (a) and with two active coils (b).



(a) (b)

Figure 8 — Current-to-displacement curve with four active coils (a) and with two active coils (b).

linked to the coils at endstroke (x=±g) reaches very high values if the magnetomotive force related to the coils acts in the same direction of that of the permanent magnets (i.e. maximum positive current at x=g, and maximum negative current at x=−g), implying the possibility of magnetically saturating the soft ferrite parts. Without taking into account the magnetic saturation, the motor force at endstroke is parabolic with respect to coil current (Eq. (9)), and this behaviour is clearly exhibited by the SLK model (Fig. 7.a). On the other hand, the MDC model significantly deviates from the parabolic behaviour for currents higher than 0.4 A, satisfactorily reproducing the hardware response. This is because the MDC model inherently takes into account the magnetic properties of the motor parts. Each reluctance of the magnetic network (Fig. 3) is defined as a physical object, characterised by specific geometrical and magnetic properties (material properties can be directly defined by the user or selected on a pre-defined database [5]).

Concerning Fig. 7.b, where the fault-tolerant working condition is concerned, the differences between the models demonstrate to be minor (magnetic saturation is not present, since the magnetomotive force of the coils is halved), even if the prediction errors are reduced with the MDC model.

4.2 Current-to-displacement characteristics

Figure 8 shows the comparison between the simulation and the experimental data in terms of current-to-displacement curves. The results are obtained by measuring the armature displacement while a sinusoidal low-frequency command current (±i_{max} at 0.02Hz) is applied to the coils.

It can be noted that the both the models match very well the experimental data in both the operative conditions, even if the prediction errors tend to increase when the armature approaches the endstrokes.

4.3 Effects of the soft ferrite magnetic properties on motor performances

During the development of the motor models, no specific information about the properties of the soft ferrite parts was available. This aspect is not critical for the most applications, since electro-magnetic actuators are designed to avoid magnetic saturation and the reluctances related to the soft ferrite parts are negligible with respect to air gaps or rare-earth magnets. As shown in the proposed study, this approach is not suitable for aerospace application, since the design is basically driven by low-weight, high-performance, safety-critical requirements, and system nonlinearities can become important.

Actually, the MDC results shown in Figs. 7-8 have been obtained after a specific sensitivity analysis carried out by varying the material of the soft ferrite parts of the motor. The activity has been initially performed by selecting the materials provided by the pre-defined database of the *Modelica_Magnetic* library (Vacofer S2, Permenorm 3601 K3, Hyperm0, Vacoflux50). The effect of magnetic properties demonstrated to be minor if the soft ferrite parts do not saturate, while significant discrepancies from experiments have been observed in the "saturation region" with all the selected materials. A specific study has thus been done to define an appropriate B-H curve of the soft ferrite parts of the motor for obtaining a good experimental matching.

The results of this activity are reported in Figs. 9-10. Figure 9 shows the "tuned" B-H curve compared with two materials of the *Modelica_Magnetic* database, while Fig. 10 reports the current-to-force curves at endstroke (four coils active) for hardware, SLK model and two MDC models, the one using the Vacoflux50 material and the other using the "tuned" material. The model accuracy in the saturation region strongly depends on the transition to magnetic saturation, which is assumed to be more abrupt for the "tuned" material (Fig. 9).

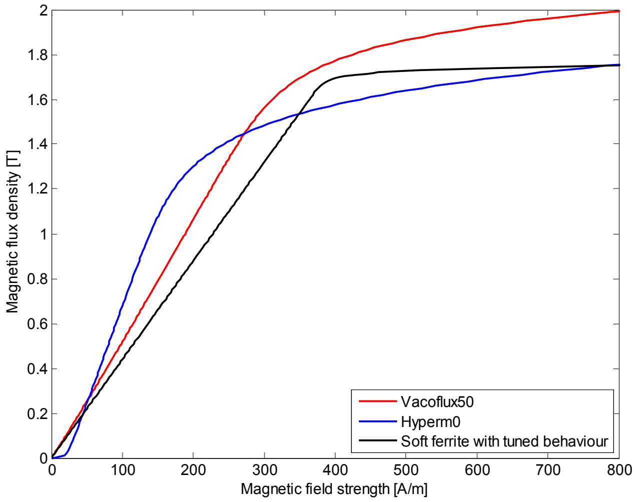


Figure 9 – B-H curves of tested soft ferrite material.

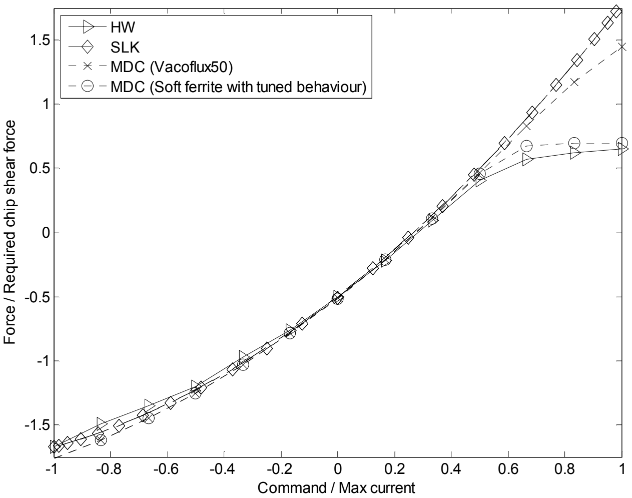


Figure 10 – Effects of soft magnetic material on motor force characteristics at endstroke (four active coils).

5 Conclusions

The Modelica model of a quadruple-coil direct-drive motor for aerospace servovalves is developed and validated with experimental data. The model predictions demonstrated to satisfactorily match the hardware response in normal operating condition (four active coils) as well as in the worst-case fault-tolerant condition (only two active coils). The inherent simulation of the magnetic saturation provided by the Modelica components allows to achieve accurate results in the whole range of operating conditions. The results of the Modelica model are also compared with those provided by a Simulink model of the motor, pointing out the advantages of the object-oriented modelling for the study of complex electro-magnetic systems.

References

[1] Pratt R. W., "Flight control systems: practical issues in design and implementation", Institution of Engineering and Technology, Stevenage, 2000.

[2] Miller F. G., "Direct drive control valves and their applications", Proceedings of the IMechE International Conference on Aerospace Hydraulics and Systems, London (UK), 1993, pp. 1–16.

[3] Elmqvist H., Mattsson S. E., and Otter M., "Modelica - the new object-oriented modeling language", 12th European Simulation Multiconference (ESM'98), Manchester (UK), pp. 1-5, 1998.

[4] Otter M., and Elmqvist H., "Modelica: language, libraries, tools, workshop and EU-project RealSim", Simulation News Europe, pp. 3-8, 2000.

[5] Bodrich T., Roschke T., "A magnetic library for Modelica", Proceedings of the 4th International Modelica Conference, Hamburg (Germany), Vol. 2, pp. 559-565, 2005.

[6] Di Rito G., and Galatolo R., "Experimental and theoretical study of the electrical failures in a fault-tolerant direct-drive servovalve for primary flight actuators", Proceedings of the Institution of Mechanical Engineers, Part I, Journal of Systems and Control Engineering, v. 222, no. I8, 2008, pp. 757-769.

[7] Di Rito G., "Experimental validation of theoretical and numerical models of a DDV linear force motor", Proceedings of the 3rd FPNI-PhD Symposium, 2004, pp. 105–114.

Preliminary design of electromechanical actuators with Modelica

Marc Budinger (*), Jonathan Liscouet (*)

Université de Toulouse, INSA/UPS, Laboratoire de Génie Mécanique de Toulouse
Toulouse, 31077, France
marc.budinger@insa-toulouse.fr

Yvan Lefevre(**), Julien Fontchastagner(**) and Abdenour Abdelli(**)

Université de Toulouse, INPT/CNRS, Laboratoire Plasma et Conversion d'Energie, Toulouse, 31071, France
Yvan.Lefevre@laplace.univ-tlse.fr

Loig Allain(***)

LMS-Imagine
La Cité Internationnale
84 quai Charles de Gaulle
69006 LYON
loig.allain@lmsintl.com

Abstract

This article deals with a methodology for a computer-aided design of electromechanical actuators from the preliminary design of components to the detail design of the electrical motor. The developed library of components for the simulation takes advantage of the non-causal and object oriented characteristics of the Modelica language. The capabilities of the Modelica language and the LMS.Imagine.Lab AMESim or Dymola Platforms are strongly used in order to build a fully integrated process to design and size the different component of the final actuator. The proposed approach is illustrated with the sizing of a flight control actuator.

Keywords: preliminary design, inverse simulation, scaling laws, electromechanical actuator, brushless motor

1 Introduction

Thanks to the development of powers electronics and permanent magnets, electromechanical actuators are very promising with respect to, e.g. automatic operating mode, power management, reliability, maintainability. For this reason, it can be very interesting to replace current actuators based on another technology less promising in these fields (e.g. hydraulic) with electromechanical actuators. A good illustration of this tendency is the research effort towards the "more electric aircraft" in aeronautics [1]. An electromechanical actuation system is very complex to design and to optimize, especially because of its multidisciplinary characteristic [2]. This paper presents a new methodology to help the engineer from the preliminary to the detailed design of electromechanical actuators. Modelica coded libraries used here are especially to encompass 2 steps in the V design cycle (Figure 1) :

- The power sizing (part 2) which aims at sizing and specify the various components of the operating system in order to meet the specification requirements (on nominal points or on mission profile) in terms of effort and speed (and therefore power).
- The detailed design of components (part 4), brushless motor here, which allows the designers to obtain fine sizing of components in order to enable the fabrication and more accurate simulations.

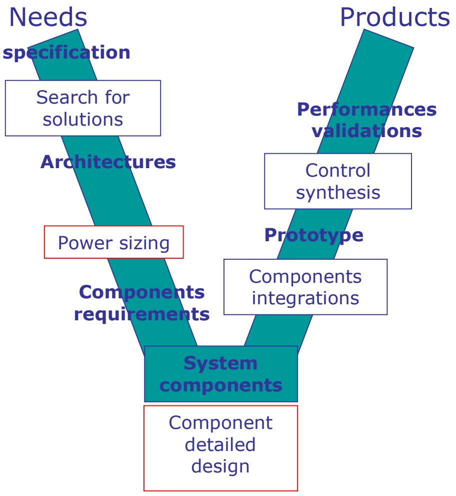
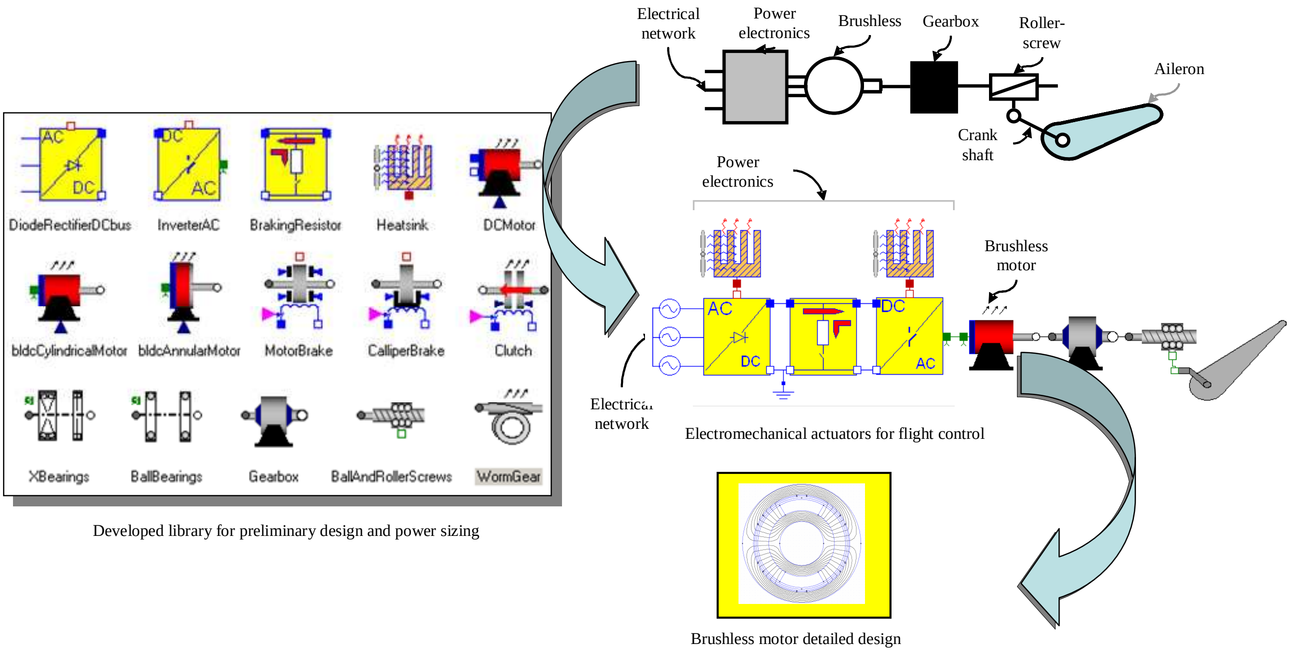


Figure 1: V design cycle

The libraries presented in this paper are illustrated in part 3 and 5 by the design a flight control actuator (see Figure 2) from global specifications to the fine sizing of the brushless motor.

DOI: 10.3384/ecp09430099



Developed library for preliminary design and power sizing

Electromechanical actuators for flight control

Brushless motor detailed design

Figure 2: Rapid design of an electromechanical actuator for a primary flight control surface of an aircraft

2 Preliminary design library for components specifications

2.1 – Sizing wave

As indicated in the previous part, the power sizing aims at determining the correct size of the devices in order to comply with the mission profile of effort and speed. Then the general methodology can be expressed as follows (see Figure 3):

- Based on the load and associated mission profile, effort and speed of each component are calculated following the entire actuation chain.
- Every component is sized such that it covers the mission profile curve established between effort and speed and requirements on other sizing parameters are satisfied, e.g. the RMS torque for an electric motor.

The simulations require many parameters that are known as "**simulation parameters**" (eg: Inertia, stiffness, thermal time constant, etc). They feed directly the equations that are solved by the equation solver.

Designers want to scan a large range of solutions quickly without searching for these very numerous "**simulation parameters**". It is preferable for them to work directly on a small number of "**definition parameters**" that characterize the components they use in a more technological way (e.g., Torque, speed, speed reducing ratio, etc.).

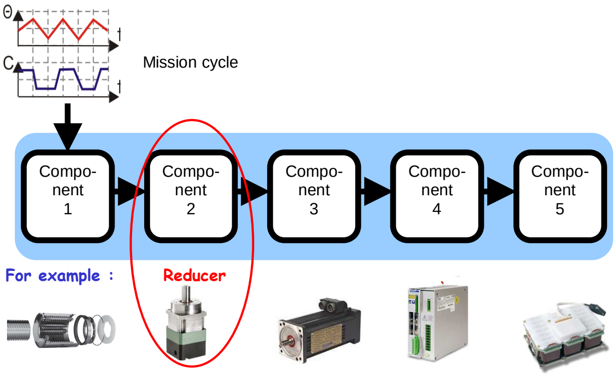


Figure 3: Power sizing

The simulations aim at confirming the selection of the components and carrying out a comparison analysis between different architectures. The simulation should allow the assessment of "**sizing parameters**" (eg: RMS torque for electric motors, RMC torque for speed reducers, etc) and of "**comparison parameters**" (e.g., mass of the component, etc).

Traditionally, the designers look into catalogues of manufacturing companies to get simulation parameters and make comparison between solutions. It requires repeated revisions of the design and of course, a lot of work. To avoid this time consuming iterations, some simulation software are equipped with large databases of full ranges of components [3-5]. For our preliminary design library, it has been decided to develop models of the components on the basis of scaling laws in order to avoid the huge tasks of databases creation and maintenance. These transparent scaling laws provide users with the relationships between the **definition parameters** with **simulation, sizing** and **comparison parameters**.

For example, these quantities are for mechanical speed reducers (epicyclic gearing for example):

- **Definition parameters**: torque output, gear ratio, desired service life;
- **Simulation parameters**: equivalent inertia, efficiency, parameters necessary for evaluation of service life;
- **Comparison parameters**: mass, volume;
- **Sizing parameters**: maximum torque and speed during a mission cycle, equivalent torque in fatigue.

2.2 – Inverse simulation with Modelica

The methodology described here requires the evaluation of the power variables of all the components for an imposed mission profile [6]. Traditional system simulation software such as Simulink does not achieve this kind of simulation using standard models and by making simple assembly of components. The simulation language Modelica [7, 8] is non-causal and does not impose a direction to the variables. In this way, a variable can be either the input or the output depending on the engineer needs.

2.3 – Scaling laws

The scaling laws, also called similarity laws, allow the study of the effect of varying representative parameters of a given system. They are used in different domains as microsystems [9], mechanics [10], hydraulics, fluid mechanics to compare different actuator technologies [11], to adapt the dimensions of a mock-up in fluid dynamics, to size mechanic, hydraulic or electric systems, to develop and rationalize product families or to evaluate costs . This article uses the notation proposed by M. Jufer in [12] for scaling laws calculation. The scaling ratio of a given parameter is calculated as

$$l^* = l'/l \tag{1}$$

With *l*, the parameter of the component taken as reference and *l'*, the parameter of the studied component.

A homothetic scaling of all the geometrical dimensions leads to link them all to their reference value by a single ratio l^*. Models are developed only for components with a geometrical similitude. In this way, the evolution of a volume *V* of a cylinder in case of an identical evolution for all geometrical dimensions is

$$V^* = l^{*3} \tag{2}$$

This last result remains valid for any other geometry. In the same way, it is possible to calculate the evolution of the mass *M* and rotating inertia *J* as function of the dimension *l*:

$$M^* = l^{*3} \tag{3}$$

$$J^* = l^{*5} \tag{4}$$

During scale change of components (e.g. motor, reducer, mechanism) some constraints must remain constant. These sizing constraints ensure an adequate use and life time for the components. For mechanical components, the constraints in the materials are limited by the elastic or fatigue limits. The use of scaling laws allows the direct determination of the simulation parameters (such as inertia) and comparison (e.g. with respect to mass) from the definition parameters (e.g., the torque). Thus, instead of using heavy databases that are difficult to build, only one reference component for each type of technology is required.

2.4 – Operating areas and sizing laws

During the simulation, it is necessary to verify the behavior of different components along the mission cycle to prevent their degradation. Generally, two types of operational limits should be distinguished.

The first limit is due to a rapid deterioration of the component: this limit corresponds to the surface operating areas which are expressed in term of energy quantities such as effort and speed. For speed reducers, this corresponds to the material's elastic limits, followed by the absolute torque that the reducer can withstand and that is limited by the maximum mechanical constraint that its weakest point can withstand. It is possible to develop models to determine torque and speed limits with the help of scaling laws.

The second type of operational limits corresponds to the gradual deterioration caused by damage accumulations which limit a component's service life or reliability. For speed reducers, this corresponds to the material's fatigue limits. In practice, the fatigue torque and the mean speed of a reducer are calculated to ensure the desired service life and reliability.

2.5 –Software implementation

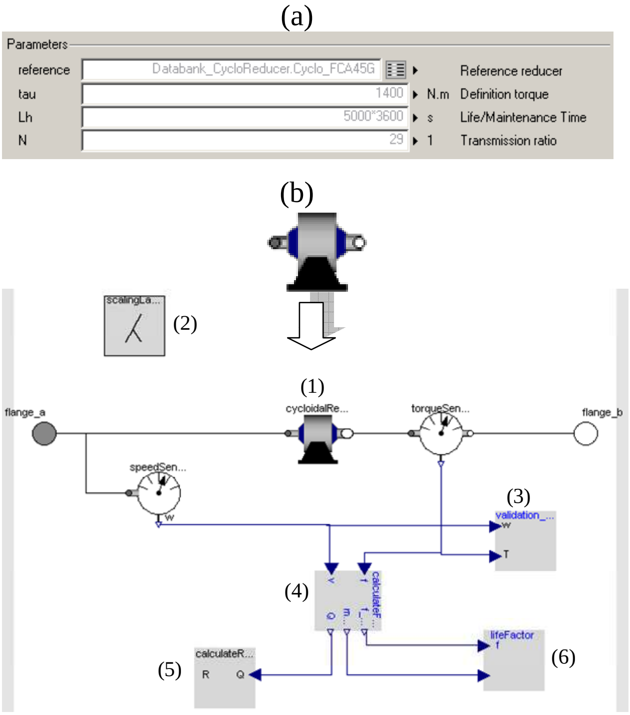
The joint use of inverse simulation, scaling laws and calculation of sizing quantities has been implemented by the authors in a Modelica library for preliminary design of electromechanical actuators. Figure 1 illustrates the nature of the components that are frequently used in electromechanical actuation systems. The assembly of these components can help to model a large number of architectures. The next part of the article illustrates their use.

These Modelica blocks or components were coded in a similar way to that illustrated by the example of the reducer in Figure 4. The parameters setting interface, Figure 4 (a), presents parameters to be set by the user. The aim of their minimal number is to facilitate the manual use, exploration or optimization during the preliminary design phases. The example of the reducer requires:

- A reference of an industrial component that is a feature of a product range or of a technology which contains in a record all reference parameters.
- The definition parameters, here the torque and the gear ratio of the reducer: all the simulation parameters and the comparison parameters are calculated from theses quantities and from parameters of the reference component.
- The service life of the component: the component's reliability is calculated assuming that the mission profile of the mission to be simulated is repeated during the service life.

Figure 4 (b) shows the internal structure of the component, which is representative of the notions developed previously in the article and includes:

1. A physical model, which allows inverse simulation, making it possible to inversely determine for each component the effort and speed of the mission profile. It allows as well the calculation of the characteristic dynamic quantities of degradation or of physical limitations.
2. Scaling laws, which calculate all the parameters necessary for simulation on the basis of parameters provided by the user.
3. Validation of the use of the component in the authorized functioning area. It verifies that the definition parameters can meet the mission profile.
4. Calculation of the cumulated damages. It takes into account the characteristic of fatigue.
5. Reliability calculation for the duration (number of hours) specified in the user interface. It allows the user to size the component accordingly to a given reliability.
6. Calculation of continuous quantities that are equivalent to the mission profile for a typical reliability of 90%. It helps the user to specify the sizing quantities adapted to the mission profile.


(a)

Parameters

reference	Databank_CycloReducer.Cyclo_FCA45G		Reference reducer
tau	1400	N.m	Definition torque
Lh	5000*3600	s	Life/Maintenance Time
N	29	1	Transmission ratio

Figure 4: Parameters setting interface and inside structure of a component

3 Preliminary design of an aileron actuator

3.1 –Case study presentation

For this case study, an electromechanical actuator equivalent to the current hydraulic one is developed. In this way, the kinematics of the aileron and its actuator remains identical (i.e., use of a crank shaft and pivots and actuator stroke of 4 cm). In the current configuration, two actuators are connected in parallel to the load in an active-damping mode, where one acts on the aileron and the other is damping. However, for simplification purposes the de-clutching and damping aspects are not addressed in this study.

In order to give a frame to the design, the choice of the reduction ratios of the roller-screw and speed reducer is driven by an imposed electrical motor speed. First, the pitch of the roller-screw is minimized (5 mm/rev), thus the mass of the speed reducer is minimized. The observation of manufacturer catalogues [13] shows that 10 000 rpm corresponds to the maximum speed of the motors characterized by the max power required by the application (~110 W). Accordingly, the speed reducer ratio is adjusted to match a motor speed of 10 000 rpm, in order to minimize the motor output torque and thus its mass. The model of the load used for this study consists of an equivalent aileron moment of inertia of 1 kgm2, with a crankshaft of 45 mm. The effort from the antagonist hydraulic actuator is caused by the relatively low windage friction (about 107 N/(m/s)²) and inertia of the hydraulic cylinder. It does not impact the power sizing notably and is therefore neglected. In the same way, the low frictions in the pivots of the load kinematics are not taken into account. Finally, the aerodynamic efforts are given along with the mission profile as a function of time.

On the one hand, the mechanical components are sized with respect to the maximum effort and speed, as well as the fatigue cumulated over their specified lifetime. On the other hand, the motor is sized with respect to the maximum effort and speed, as well as thermal constraints (e.g., temperature, RMS torque). As a consequence, two mission profiles are used for sizing: One representative of maximum effort and speed and another representative of the thermal constraints. Figure 5 illustrates these two mission profiles propagated at the actuator output (pivot between the actuator and crank shaft) in the force-speed power plan.

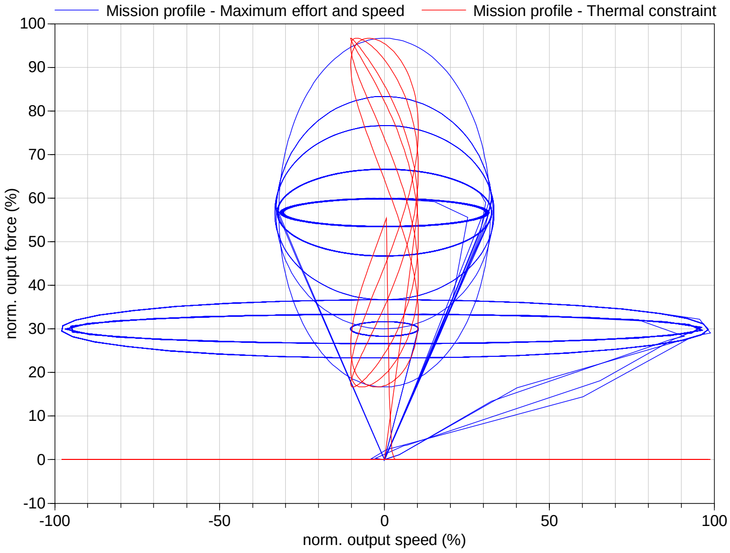


Figure 5: Mechanical (blue) and thermal (red) aileron mission profiles at the actuator output.

3.2 –Results

The results obtained by following the sizing methodology described previously are collected in the table of Figure 6. From these results, it is possible to carry out a mass and integration analysis efficiently. In this way the total mass of the actuator is the sum of the component masses returned by the different components models. Figure 6 also includes the component references used by the scaling laws, as well as off-the-shelf components the closest to the scaled ones. It appears clearly that despite the references are often far away from the scaled components (e.g., the nominal torque of the speed reducer reference is more than twenty times that of the scaled one) the scaling laws lead to existing off-the-shelf products accurately.

The comparison between the scaled and off-the-shelf actuators illustrates the accuracy of the developed approach. One can notice that the off-the-shelf actuator is a little heavier than the scaled one (-23%). This difference of mass is mainly due to the fact that when there is no off-the-shelf component matching exactly the scaled one, then the next bigger component is selected. In the same way, the chosen reducer has a maximum reduction ratio lower than for the scaled one. As a consequence, the electric motor has a more important RMS torque and is bigger. From the dimensions listed, the actuator geometry can be represented within the wing profile to verify its integrability as shown in Figure 6. The maximum length of the actuator is given by the distance between the pivots 1 and 2. Figure 6 shows that mounting all the components in-line does not allow the actuator to fit within these two pivots. A solution consists in mounting the brushless motor and the reducer alongside the roller screw thanks to spur-gears implemented between the reducer output and the roller-screw input.

In order to take advantage of the developed actuator model, further design explorations could be carried out : effect of varying the length of the crank shaft, evaluating the influence of the actuator lifetime, assessing the interest of an active/active configuration to reduce the mission duty ...

	Brushless motor	Speed reducer (epicyclical)	Roller screw (stroke = 40 mm)	

Reference	MAXON EC-60-167131,
	RMS torque = 0.83 Nm.
	Diameter = 6 cm,
	length = 12.9 cm,
	mass = 2.45 kg.
		REDEX-ANDANTEX SRP1,
		nominal torque = 370 Nm,
		reduction ratio = 7.
		Diameter = 17 cm,
		length = 18 cm,
		mass = 13.8 kg.
			SKF TRK 44 (roller-screw),
			SKF BLRU 4 (end-bearing),
			nominal forces = 86.9 kN.
			External diameter = 8.6 cm,
			total length = 30 cm,
			total mass = 11.4 kg.	

Scaled	RMS torque* = 0.23 Nm.
	Diameter = 4 cm,
	length = 9 cm,
	mass = 0.8 kg.
		Nominal torque* = 17 Nm,
		reduction ratio = 71.
		Diameter = 6.4 cm,
		length = 7.8 cm,
		mass = 0.8 kg.
			Nominal force** = 26 kN,
			pitch = 5 mm/rev.
			ext diameter = 4.7 cm,
			total length = 14.2 cm,
			total mass = 1.5 kg.
				Actuator mass
				= 3.1 kg.

Off-the-shelf	MAXON EC-45-136212,
	RMS torque = 0.28 Nm
	Diameter = 4.5 cm,
	length = 10.1 cm,
	mass = 1.1 kg.
		NEUGART PLE 60,
		nominal torque = 18 Nm,
		reduction ratio = 64.
		Diameter = 6.3 cm,
		length = 11.8 cm,
		mass = 1.1 kg.
			SKF TRK 21 (roller-screw),
			rated force = 27.85 kN,
			pitch = 5 mm/rev.
			SKF BLRU 2 (end-bearing),
			rated force = 27.9 kN.
			ext diameter = 4.9 cm,
			total length = 16.2 cm,
			total mass = 1.8 kg.
				Actuator mass
				= 4 kg.

* sized with respect to the maximum effort to transmit. ** sized with respect to lifetime (fatigue or thermal constraint).

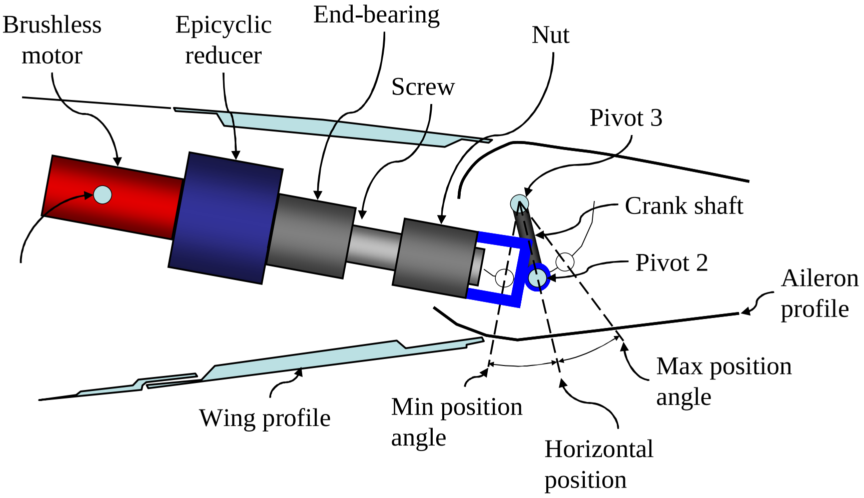


Figure 6: Actuator sizing for a crank shaft length of 45 mm, a component lifetime of 48 000 hours and a active/damping configuration (baseline).

4 Electrical motor design

4.1 Electrical actuator design

The design of an electrical actuator can be undertaken as a step-by-step procedure. It starts with the application requirements that are determine by the previous preliminary design. Then the type of the electrical actuator, that can meet the requirements, has to be chosen. Knowing the type of actuator, its sizes must be determined. This can be achieved by means of a sizing model.

Symbol (Unit)	Name of quantities
Tn (N.m)	Nominal torque
Ωn (rad.s^{-1})	Nominal speed
Vn (V)	Supply voltage rms value

Table I: The main requirements

To illustrate this design procedure, a tool to help engineers to size a permanent magnet brushless dc motor has been developed. This tool is based on analytical models of permanent magnet motor as those proposed in [14][15][16][17]. Analysis of these models show that the sizes of the motor can be obtained from only three main requirements among all those obtained from the preliminary design: the nominal torque, the nominal speed and the rms value of the electrical network supplying voltage (Table I). A man of the art approach is then applied to help the users to make different choices. The first choices that the user has to make concern the design choices such as the motor form factor, which is the ratio of the bore diameter to its length, or the mean value of the magnetic field in the gap of the motor (Table II).

Symbol (Unit)	Name
λ	Form factor: the ratio of the bore diameter D (m) to the length of the motor L (m)
AJ ($A^2.m^{-3}$)	Product of the electric loading A to current density J
BA ($N.m^{-2}$)	Airgap shear stress: product of the airgap flux density B (T) to the electric loading A
J ($A.m^{-2}$)	Current density

Table II: General design choice

Expert helps are given to users to make these choices straight forward for him. For instance, if the user wants a rather long motor, he chooses a form factor less than one.

Name	Typical values
Form factor	Long motor (<1), squared motor (=1), flat motor (>1)
Product AJ	Cooled by natural convection about 10^{10} A^2m^{-3}, Cooled by air forced convection about 10^{11} A^2m^{-3} Cooled by liquid forced convection > 10^{12} A^2m^{-3}
Airgap shear-stress (Nm^{-2})	Cooled by natural convection and totally enclosed motor: 3500 – 7000; High performance motor magnetized by rare earth or NdFeB magnets: 7000 – 21000; Liquid cooled motor: 70000 – 105000
Current density (Am^{-2})	Totally enclosed motor and cooled by natural convection: < 5 10^6; Cooled by air forced convection: 5 - 10 10^6; Liquid cooled motor: 10 – 50 10^6

Table III: Example of extpert help for the design choice

From the torque requirement (Table I) and the choice of the quantities in Table II, the bore diameter and the length of the motor can be calculated:

$$D = 2.\sqrt[3]{\frac{\lambda Tn}{4\pi BA}} \qquad L = \frac{D}{\lambda} \qquad (5)$$

Six other choices must be made by the users. Among them are the choices of the permanent magnet and of the magnetic material [18] [19]. The choice of magnetic material for instance is determined by the desired mean values of the magnetic flux density in the magnetic circuit of the motor: the teeth and the yokes of the motor[15][16].

This man of the art approach is done in Modelica by the use of Blocks, modeling the different choices of the users, and connected to each other. In each Block, the equation section allows by reversing the direct analytical sizing model to determine the appropriate geometrical parameter of the motor. For instance, the expression of the torque in the direct model is reverse in the ‘Block Design_Choice’ to compute the diameter bore and the length of the motor (5). So, these Blocks, modeling the requirements and the choices made by the user, determine the geometrical, physical and structural parameters of the motor. The table IV gives the list of the parameters that can be computed at this stage:

Symbol (Unit)	Names of the calculated quantities
D (m)	Bore diameter
L(m)	Stack length
Egap (m)	Airgap thickness
Eai (m)	Magnet thickness
Ecs (m)	Stator yoke thickness
Ecr (m)	Rotor yoke thickness
Ns	Turn number of each phase
Ws (m)	Slot with
Ds (m)	Slot depth
Nenc	Number of slots

Table IV: The main quantities calculated

In a second part, these blocks are connected to two more Blocks. The first calculates the masses, the volumes and the inertia of the rotor and the second the different losses in the motor such as the winding losses [18] [16].

4.2 Electrical actuator model library

In a last Block the electrical parameters of the motor such as the no-load flux, the cyclic inductance and the resistance of each phase. This ‘Block Electrical_Parameters' can be connected to a Modelica user model that simulates the dynamic behavior of the motor. In order to help the user to make this kind of Modelica model a library based on lumped parameters model of electrical motors has been developed [20]. In this model the motor can have non-sinusoidal wave forms. With this library a user can simulate a variable speed motor fed by a PWM controlled voltage. Examples of simulation results are shown on figure 6 and figure 7. These results are obtained from a Modelica program that runs in an AMESIM environment.

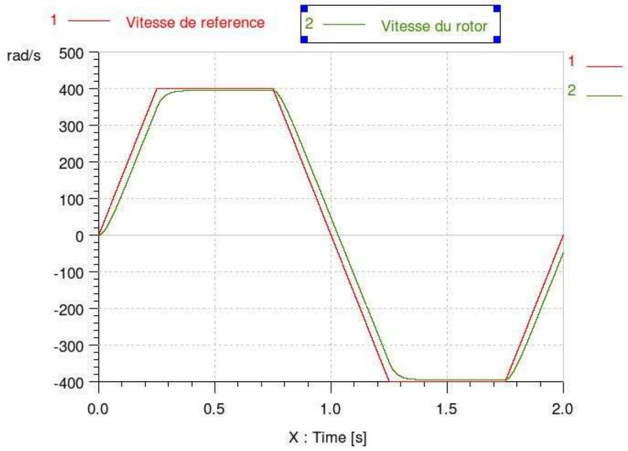


Figure 7: Simulation of trapezoidal speed cycle operation of a motor under design: reference speed and rotor speed

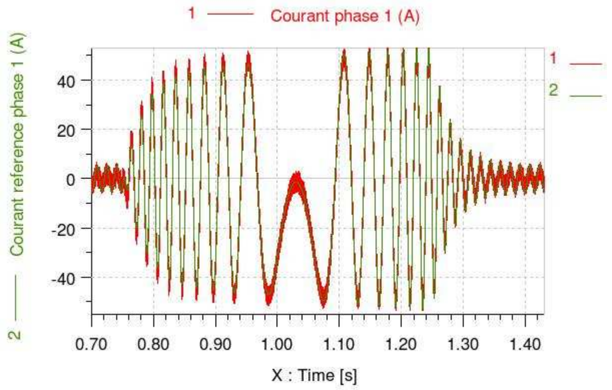


Figure 8: Simulation of trapezoidal speed cycle operation of a motor under design: reference current and current in a phase of the motor.

5 Detailed design of an aileron brushless motor

In this last section, the detailed sizing of a permanent magnet that meets of the requirements of the scaled motor presented in figure 6 is undertaken. The requirements of the projected motor are the following: nominal torque (Tn=0.23 N.m), nominal speed (Nn=10000 rev/mn), DC supply (U=48 V). The solution is inspired by the Maxon motor (EC45-136-212). The following general design choices have been made: product AJ of 2.10^{11} A^2m^{-3}, form factor l of : 0.4;

The chosen AJ product is very high. This value has been determined after many tries with the step by step procedure described in section 4. We are trying to find a set of dimensions that suits the structure of a slotless motor as suggested by the Maxon motor solution (Figure 6). This AJ product has been chosen with a very low airgap flux density (0.1 T) in order to be able to put the winding between the stator yoke and the magnet. Besides these choice, a shear stress of one thousand $N.m^{-2}$ is adopted which is very low according to the experts typical value of Table III. Again these values are obtained after running many times our sizing procedure. With the sizing procedure proposed, we find a motor whose main parameters are reported on table V.

Value of some chosen and calculated parameters	Name	Status (chosen or calculated)
p=1	Number of pole pairs	chosen
q=3	Number of phases	chosen
β =0.667	Rate of pole arc	chosen
D = 0.019 m	Bore diameter	calculated
L = 0.097 m	Stack length	calculated
g =0.97 mm	Airgap length	calculated
Ns = 38	Number of turns	calculated
I = 6. 49 A	Current	calculated
D_2 = 0.045 m	Diameter	calculated
P_j = 71 W	Joule loss	calculated
R_{LL} = 1.68 W	Resistance phase to phase	calculated
L_{LL} = 0.131 H	Inductance phase to phase	calculated
η = 0.77	Efficiency	calculated
M = 0.41 kg	Mass of active materials (iron, magnet, copper)	calculated

Table V: Main parameters of the sized motor

The main drawback of this motor is that the stack length is already greater than the required length on figure 6. The mass is half of the required mass, but it includes only active materials. The resistance is very high and the efficiency is relatively low. Except the length, all the parameters fit the requirement and are in accordance of the motor proposed by Maxon.

Table V shows that the scaled motor defined in Figure 6 can be found. This last section show what can be obtained from the detailed sizing procedure. One of the use of this procedure is the training of future motor designers. Such a procedure will give them the ‘physical sense’ that helps to design a motor.

6 Conclusions

The approach presented in this paper aims at improving the preliminary design and detailed design on the basis of an efficient sizing and rapid virtual prototyping.

For the preliminary sizing the keystone of this proposed approach is a uniform modelling of the components based on scaling laws that allows having a limited number of input parameters. Using scaling laws developed from the main physical constraints of the components makes the sizing representative of the state-of-the-art of technology. Moreover, it does not require a database or a large amount of data that are cumbersome to build and to maintain, but single component references only. The example has shown how the use of the developed library allows a fast modelling which can be useful for an exploration of different design configurations (active/damping, active/active) and design parameters (crank shaft length and lifetime) and thus supports well taking technical decisions early in the preliminary design by providing a rich insight into the design problematic in an efficient way.

For the detailed design of brushless motors, the man of the art approach presented in this paper to design an electromechanical actuator can be very useful to design quickly a motor ‘by hand’. The user controls at each step the effect of his choices on the performances of the motor. The example treated shows that it can be used to find non obvious solution with the help of a preliminary design method based of scaled laws. The results of this preliminary design forced to attempt non obvious choices different from those given by experts guide.

References

[1] T. Ford, “More-electric aircraft,” Emerald, vol. 77, 2005.

[2] F. Roos, “Towards a methodology for inte-grated design of mechatronic servo systems,” Text, KTH, Machine Design, 2007.

[3] Linear Motioneering, Danaher Motion, .

[4] ServoSoft, ControlEng, .

[5] Cymex - Alpha, WITTENSTEIN formerly alpha gear drives, .

[6] Jardin, Audrey, Marquis-Favre, Wilfrid, Thomasset, Daniel, Guillemard, Franck, et Lorenz, Francis, “ Study of a Sizing Methodology and a Modelica Code Generator for the Bond Graph Tool MS1,” University of Applied Sciences, Bielefeld, Germany: 2008, pp. 125-134.

[7] P. Fritzson et V. Engelson, “Modelica — A unified object-oriented language for system model-ing and simulation,” ECOOP’98 — Object-Oriented Programming, 1998, pp. 67-90.

[8] H. Elmqvist, D. Ab, S.E. Mattsson, et M. Ot-ter, “Modelica: The new object-oriented modeling language,” presented at The 12th European Simu-lation Multiconference,” In Proceedings of The 12th European Simulation Multiconference, 1998, pp. 127--131.

[9] P. Minotti et A. Ferreira, Les micromachines , Paris: Hermès, 1998.

[10] G. Spinnler, Conception des machines : principes et applications. 3, Dimensionnement , Lausanne: Presses polytechniques et universitaires romandes, Paris, 2005.

[11] B. Multon, H. Ben Ahmed, M. Ruellan, et G. Robin, “Comparaison du couple massique de di-verses architectures de machines tournantes syn-chrones à aimants,” Société de l'Electricité, de l'Electronique et des Technologies de l'Information et de la Communication (SEE), Paris, FRANCE (1995) (Revue), 2006, pp. 85-93.

[12] M. Jufer, Traite d'électricité vol9 :transducteurs, Presses Polytechniques et Universi-taires Romandes (PPUR), 1998.

[13] "Brushless DC Motors," On line ed, Maxon, Ed., 2009, http://www.maxonmotor.com/.

[14] Gordon R. Slemon and Xian Liu, “Modeling and Design Optimization of Permanent Magnet Motors”, Electric Machines and Power Systems, 20:71-92, 1992.

[15] J. R. Hendershot Jr and T. J. E. Miller, “De-sign of Brushless Permanent Magnet Motors”, Monographs in Electrical Engineering No. 37, Magma Physics Publishing and Clarendon Press, Oxford, 1994.

[16] Jacek F. Gieras and Mitchell Wing, “Permanent Magnet Motor Technology, Design and Applications”, Second Edition, Revised and Expanded, Marcel Dekker Inc., 2002.

[17] E. Fitan, F. Messine, and B. Nogarede, “A general analytical model of electrical permanent magnet machine dedicated to optimal design”, COMPEL, 22(4) :1037–1050, 2003.

[18] Gordon R. Slemon and Xian Liu, “Core Losses In Permanent Magnet Motors”, IEEE Transaction on Magnetics, vol. 26, No. 5, 1653-1655, September 1990.

[19] T. J. E. Miller, “Brushless Permanent-Magnet and Reluctance Motor Drives”, Monographs in Electrical Engineering No. 21, Clarendon Press, Oxford, 1989.

[20] T. Sebastian and G. R. Slemon. Transient modelling and performance of variable speed permanent magnet motors, IEEE Transaction on Magnetics, IA-25(1) :101–107, 1989.

HIL Simulation of Aircraft Thrust Reverser Hydraulic System in Modelica

Zhao Jianjun1 Li Ziqiang1 Ding Jianwan1 Chen Liping1 Wang Qifu1
Lu Qing2 WangHongxin2 Wu Shuang2
1: CAD Centre, Mechanical School, Huazhong Univ. Sci.& Tech. Wuhan, Hubei, China, 430074
2: Shanghai Aircraft Design and Research Institute, Commercial Aircraft Corp. of China Ltd.,
Shanghai, 200436
{jjzhao168, willhave, jwdingwh, chenliping.ty}@gmail.com wangqf@hust.edu.cn
lq70300@126.com whongxin@sina.com wushuanga@sohu.com

Abstract

This article describes a solution to create a hardware-in-the-loop (HIL) simulation system of civil aircraft thrust reverser with Modelica-based simulation platform -- MWorks in Windows system. The HIL system uses simulation platform “MWorks” to model and simulate the thrust reverser hydraulic system, and takes hardware -- PLC’s output signals as the inputs of the simulation. Modeling module, communication module, solving module, animation module and HIL control module are included in the simulation platform, whose key technology and implementation details are specified. The HIL system has been successfully applied to the simulation of ARJ21 aircraft thrust reverser hydraulic system. It can simulate the hydraulic system in normal status, fault status as well as other working conditions to verify control logic and evaluate key performance of the system, thereby helping to reduce the cost of experiments and to optimize the design of the system.

Keywords: Aircraft thrust reverser hydraulic system, real-time simulation, HIL, Modelica

1 Introduction

Thrust reverser [1] as a part of aircraft engine, is aircraft landing deceleration device, which can effectively shorten the distance of taxiing. Thrust reverser is a typical complex physical system, involving mechanical, electronic, hydraulic, control and other domains. In order to verify thrust reverser’s control logic, we could carry out ground experiment and flight experiment with real pieces of the thrust reverser, but this approach has high cost and poor security, and it is limited to different natural conditions. Moreover, with this approach, the test for extreme condition is very difficult.

Modelica-based HIL simulation system can resolve above-mentioned problems. Firstly, Modelica [2, 3] is a freely available, object-oriented language for modeling of large, complex, and heterogeneous physical systems. It is suited for multi-domain modeling. Models in Modelica are mathematically described by differential, algebraic and discrete equations. In Modelica we can model the entire thrust reverser, which involves mechanical, electronic, hydraulic and control domains. Secondly, HIL system uses both real logic control components and thrust reverser model to implement the simulation. This HIL system can verify the control logic in a variety of working conditions, and its cost is very low. Moreover, with this system, there is no need to consider the security.

This article introduces a solution to create an HIL simulation system of thrust reverser with Modelica-based simulation platform – MWorks [4] in common computer with Windows operating system. It use as an example the aircraft thrust reverser of Advanced Regional Jet for the 21st Century (ARJ21) which is designed and manufactured by Commercial Aircraft Corp. of China, Ltd. (COMAC). At first, it introduces the overall frame of the HIL simulation system, and then specifies several key modules of the simulation platform, which are modules of modeling, solving, communication, animation and HIL control, and finally demonstrates a successful application of this system in ARJ21 thrust reverser simulation.

2 System Overview

Generally, HIL simulation system is composed of host PC running on Windows operating system and target machine running on real-time operating sys-

DOI: 10.3384/ecp09430040

tem. This kind of system has high real-time capability, but is very expensive.

ARJ21 aircraft thrust reverser is driven by a hydraulic system, which is mainly controlled by six electromagnetic hydraulic valves, whose states all depend on the thrust reverser control switch. In the simulation, PLC as the thrust reverser controller generates 6 hydraulic valve control signals according to the state of the thrust reverser control switch and feedback signal from simulation platform. And the feedback signal will be only used for fault trigger. Therefore， the simulation does not need very high real-time capability.

The HIL simulation system, discussed in this article, does not need expensive "true" real-time system. It can run on general computer with Windows operating system and the sampling frequency can achieve 50Hz, which is enough for the requirements of the thrust reverser simulation.

In Figure 1 the system overview is shown. The HIL simulation system is implemented based on PLC and simulation platform “MWorks”, which consists of five software modules -- modeling module, solving module, communication module, animation module and HIL control module.

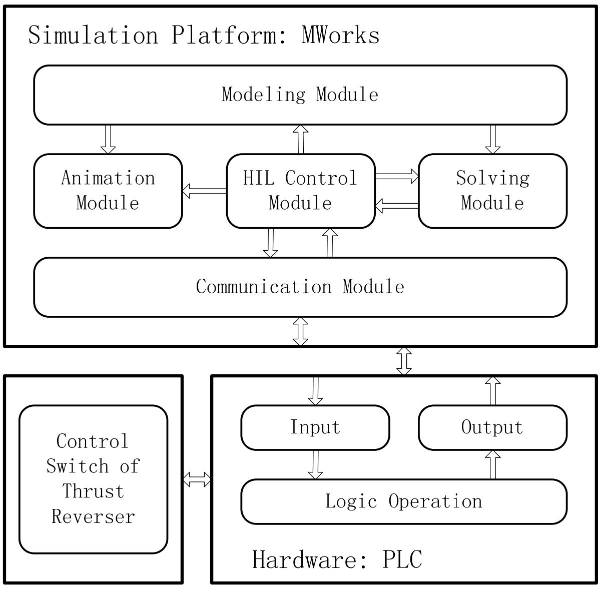


Figure 1: System overview

The PLC, used as the hardware part in the HIL system, receives electrical signal of control switch as well as simulation feedback signal, and sends control signal to the simulation platform after logic operation.

MWorks, a Modelica-based integrated development environment, is used as modeling and simulation platform for the HIL simulation system. The thrust reverser is the simulated object, which is modeled in Modelica. According to the model, the solving module generates the solver, which is responsible for real-time calculation. The communication module is responsible for real-time data exchange between simulation platform and the PLC. The animation module receives the result data from the solving module and drives 3D animation. The HIL control module, whose panel is shown in Figure 2, is responsible for starting and terminating the simulation, setting simulation parameters, displaying key data as well as communicating with other modules.

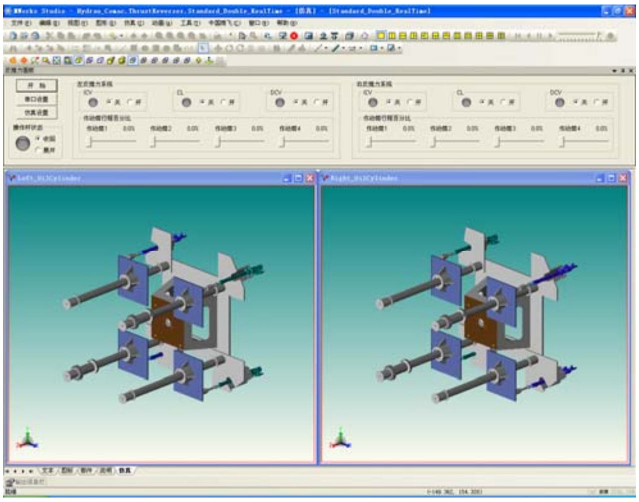


Figure 2: HIL Simulation System

The simulation process is as follows:

1) After analyzing the thrust reverser system, component models and system models are created in Modelica.

2) After setting simulation parameters with the panel of the HIL control module, the simulation begins: the HIL control module translates the model, and then the solving module generates a solver, which will be called in a new process.

3) The communication module is called by the HIL control module to receive control signals from PLC. After translating, these signals will be displayed on the panel, and sent to the solver process.

4) The solver process receives control signal and calculates in every cycle. When the calculation finishes, the solver sends the results to the HIL control module, and wait until the next cycle.

5) The HIL control module receives the results from the solver process and displays them on the panel of the HIL control module, and delivers them to the animation module to drive real-time animation. At the same time, the HIL control module calls the communication module to send the results as feedback signal to PLC.

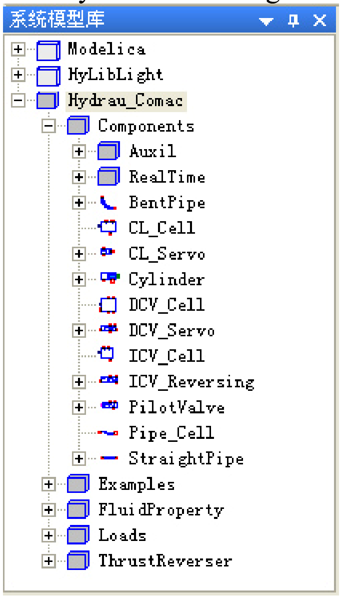
6) PLC uses the feedback signals and the state of control switch as input, and after logic operation, sends the control signal to the simulation platform.

7) Repeat the cycle from Step 3 until the termination of the simulation.

3 Key Technologies

3.1 Modeling

After analyzing ARJ21 aircraft thrust reverser hydraulic systems, we developed an exclusive hydraulic library: Hydrau_Comac, which is based on HyLibLight hydraulic library. Hydrau_Comac library provides ARJ21 thrust reverser hydraulic components and auxiliary library, such as Isolation Control Valve (ICV), Cowl Lock (CL), Directional Control Valve (DCV), hydraulic actuator, pipe, loads， and characteristics of fluid. These models are constructed according to their physical equations with their parameters calibrated by test results if necessary. To satisfy the requirements of the real-time capability, Hydrau_Comac library also provides simplified real-time component models. The structure of Hydrau_Comac library is shown in Figure 3.



- Modelica
- HyLibLight
- Hydrau_Comac
  - Components
    - Auxil
    - RealTime
    - BentPipe
    - CL_Cell
    - CL_Servo
    - Cylinder
    - DCV_Cell
    - DCV_Servo
    - ICV_Cell
    - ICV_Reversing
    - PilotValve
    - Pipe_Cell
    - StraightPipe
  - Examples
  - FluidProperty
  - Loads
  - ThrustReverser

Figure 3: Structure of Hydrau_Comac library

Based on HyLibLight library and Hydrau_Comac library, we modeled ARJ21 thrust reverser hydraulic system, provided simplified system model (Figure 4) for real-time HIL simulation, as well as detailed system model (Figure 5) for off-line simulation.

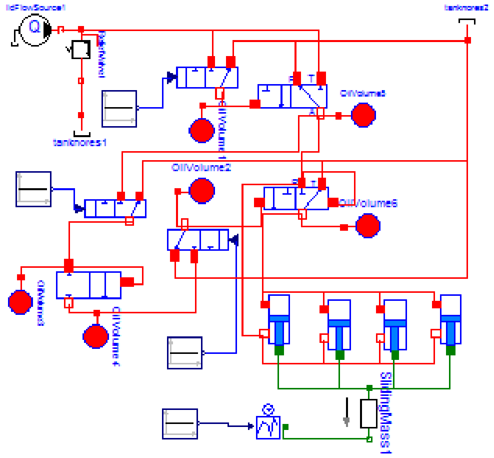


Figure 4: Real-Time System Model for Thrust Reverser

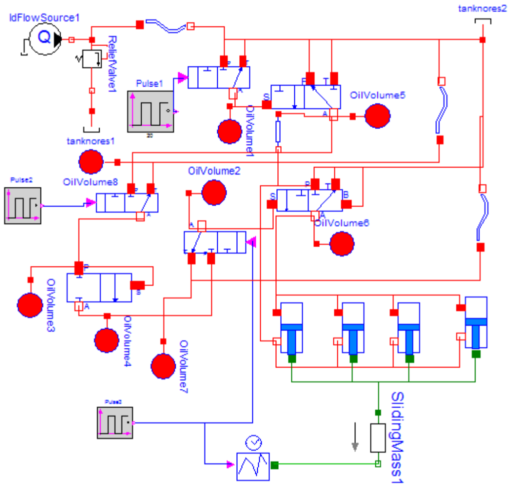


Figure 5: Off-line System Model with Pipes

3.2 Solving

Model solving in HIL simulation is different from in off-line simulation. The solving in HIL simulation needs to not only exchange data with external hardware, but also guarantee the synchronicity between physical time in real world and logic time in simulation.

In order to identify input and output data, we used “input” prefix and “output” prefix to modify input variables and output variables, thus we can ensure the order of the calculation -- from the input variables to output variables. Besides, according to Modelica specification, input variables and output variables are not only used for external communication, therefore external exchange data needs to be recorded in configuration file.

According to the records in configuration file, the solving module associates input/output variables with shared memory. The solver module reads input data from shared memory, and writes output data into there. The HIL control module writes input data coming from PLC into sharing memory, and reads output data from there.

The flow chart of real-time solving is shown in figure 6. In every sampling cycle, the solving module gets the input variables from sharing memory, and checks if their value changes, if changes, it means that there is changes in the outside world, which results in an event, so that the solving module need to do event iteration. Then the solving module calculates, and writes required output data into shared memory.

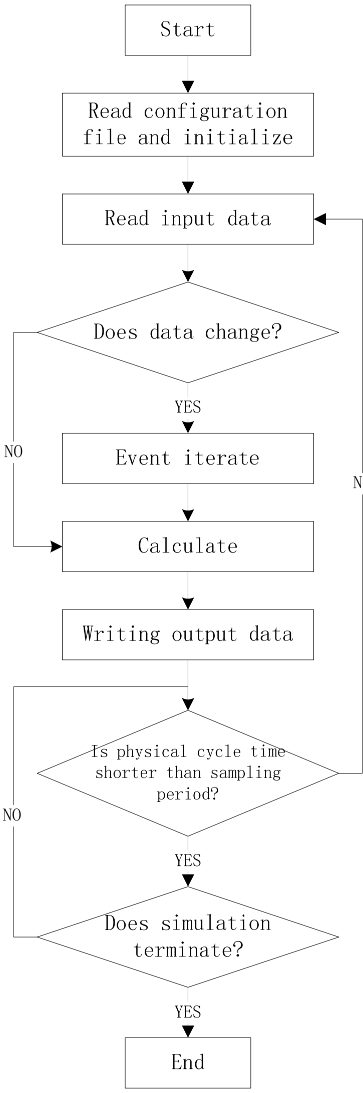

Start
Read configuration file and initialize
Read input data
Does data change?
YES
NO
Event iterate
NO
Calculate
Writing output data
Is physical cycle time shorter than sampling period?
NO
YES
Does simulation terminate?
YES
End

Figure 6: Flow Chart of Real-time Solving

We use timer to implement the synchronicity. By calling QueryPerformanceFrequency() function, we can obtain machine internal timer’s clock frequency, and by calling QueryPerformanceCounter() function at two time points, we can get a count. With the frequency and the count, we can know the precise time between that two time points. With this method, we can know the time spent in one cycle, and the time is called physical cycle time, which is a variable. The next cycle begins when the physical cycle time is longer than sampling period. The timing error of this method is less than 1ms.

In every cycle, the solving module checks whether the time spent on calculating is longer than the sampling period. If the calculation overruns the sampling period, but not more than the acceptable time, the module will report a warning. And if the calculation overruns the acceptable time, the module will report an error and quit. Therefore, in order to achieve high real-time capability, the simulation system needs to run on high-performance computer to ensure the speed of solving.

3.3 Communication

In HIL simulation, how to communicate between simulation platform and PLC and how to guarantee the precise communication frequency are key factor to real-time capability.

By using the communication module, simulation platform communicates with PLC through RS232 serial port . Communication parameters are as follows: 57.6kbps transmission rate, 8-bit data bit, 1-bit stop bit, no parity, and fixed word length data frame. The data transmitted from simulation platform to PLC will be converted to standard data frame according to the protocol. After receiving, the PLC will translate those data frames to retrieve the content.

The communication module calls Windows API function to carry out serial port communication: calling CreateFile() function to open the serial port, WriteFile() function to write data to the serial port, ReadFile() function to read data from serial port.

PLC uses high-speed serial port communication module CP341 to implement communication. FB7 function block of CP341 are responsible for receiving data from simulation platform, and FB8 function block of CP341 are responsible for sending data to simulation platform.

By using timer, the frequency of serial port communication can be controlled. Serial port communication frequency is the same as the sampling frequency. PLC uses its internal timer, whose minimum timing interval can be 10ms. Since the PLC is circuit working, so the precision of timing depends on the operational cycle of PLC control program. Under normal circumstances, the operational cycle of PLC control program can be less than 1ms, and the precision can achieve 1ms. The communication module, based on Windows operating system, uses multimedia timer "timeSetEvent()" for timing control, and implements serial port reading and writing operation in callback function, the precision can also achieve 1ms.

3.4 Animation

Generally, the implementation of Modelica multibody animation has 3 steps: firstly, the solver calculates the model to generate result data, which then will be used to form animation data; secondly, geometric models are created; thirdly, the geometric models are driven by the animation data and displayed on the screen.

For the real-time simulation, we need to fresh the animation data in every cycle, but it takes so long to fresh the data that the animation cannot satisfy real-time requirements. Fortunately, the thrust reverser has only one motion freedom, that is, the actuation can move back and forth. Therefore, we can create off-line animation at first, and then use the variable of actuator deployed length to control the display of that off-line animation, thus the synchronicity of the animation can be guaranteed.

Specific process is as follows: Firstly, establish the multi-body kinematic model of the thrust reverser, and execute off-line simulation to generate simulation results document; secondly, read the simulation results document to create 3D animation; thirdly, establish one to one mapping relationship between the variable of actuator deployed length and the off-line animation frames; finally, carry out the real-time simulation, obtain the value of that variable, and use it to drive the animation.

4 Application

This HIL simulation system has been successfully applied to the simulation of ARJ21 aircraft thrust reverser hydraulic system. The simulation platform UI is shown in Figure 7. Logic control hardware part is implemented with Siemens S7-300 series PLC, which includes power supply module, CPU module, discrete input module, discrete output module, analog input module, analog output module, serial port communication module and touch panel. PLC control program is developed with STEP7, and touch screen interface (Figure 7) is developed with Flexcible2005. PLC takes the thrust reverser control switch or the data from the touch screen as input signal, after some logic operation, it sends the output data as control signal to simulation platform.

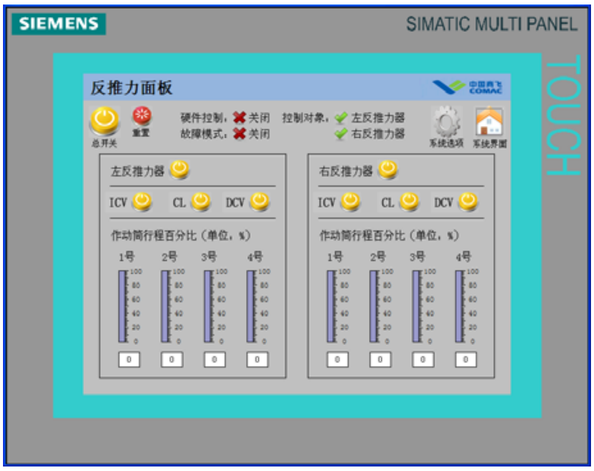


Figure 7: Touch Panel of PLC

MWorks runs on general computer with Windows operating system. our computer with simulation platform MWorks is a Dell desktop with Intel Core2 2.8G CPU, 2G RAM, ATI 3450HD graphics card and 19-inch liquid crystal display. In this configuration, the real-time simulation cycle of ARJ21 thrust reverser hydraulic system can achieve 20ms.

The result data and curves generated by this HIL simulation system are basically in agreement with the tests, the difference is acceptable. (Table 1, Figure 8, Figure 9).

Table 1: Deploying Time and Stowing Time of Actuator

	Deploying Time (s)	Stowing Time(s)
Experiment	1.08	2.68
Simulation	1.04	2.66
Error	3.7%	0.7%

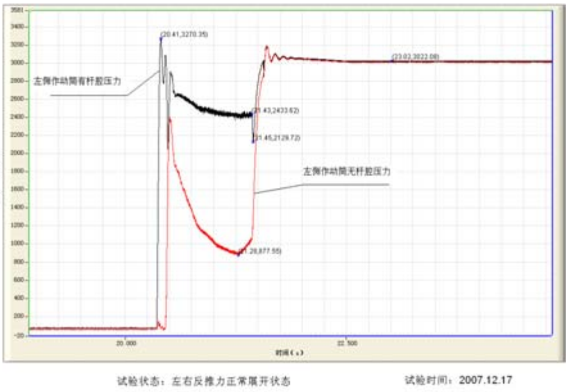


Figure 8: Experimental Curves of Pressure of The Actuator

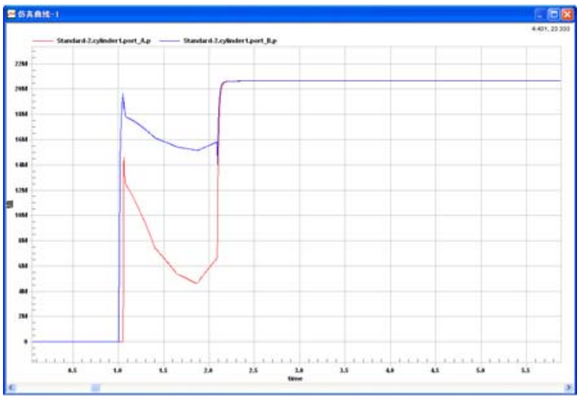


Figure 9: Simulation Curves of Pressure of The Actuator

This HIL simulation system has simple structure and low cost. Through the simulation of ARJ21 aircraft thrust reverser hydraulic system, we can verify the control logic in various working conditions, evaluate key performance of the system, so that the number and cost of the tests can be reduced, and the optimization of the design of ARJ21 aircraft hydraulic system and tests can be provided with basis.

5 Conclusions

This article demonstrates a Modelica-based HIL simulation solution exclusively developed for aircraft thrust reverser hydraulic system. The HIL simulation system, running on general computer with Windows operating system, communicates with external hardware through serial port. The cost of this HIL simulation system is very low, and its sampling period can be up to 20ms, so it's especially useful for those situations where very high real-time capability is not required.

The prototype application of the simulation of ARJ21 thrust reverser shows that this HIL simulation system, which uses Modelica language to model aircraft thrust reverser hydraulic system and connects with PLC control system, can greatly increase the efficiency of tests, and reduce the number and the cost of tests.

The future work is to enhance the real-time capability of the simulation with general Windows computer, as well as to use MWorks to generate target code, which can be used in real-time system.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No.60704019 and Grant No.60874064).

Special thanks to Medelon Corporation for authorized use of HyLibLight library.

Acronyms

ARJ21: Advanced Regional Jet for the 21st Century
COMAC: Commercial Aircraft Corp. of China, Ltd.
CL: Cowl Lock
ICV: Isolation Control Valve
DCV: Directional Control Valve
PLC: Programmable Logic Controller
HIL: Hardware-in-the-Loop

References

[1] Robert A Jones, Thrust reverser. US4373328, 1983,2.

[2] Peter Fritzson, Engelson Vadim. Modelica a unified object oriented language for system modeling and simulation[A]. Proceedings of the 12th European Conference on Object oriented Programming[C]. 1998, 67 - 90.

[3] Peter Fritzson, Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Piscataway, NJ: IEEE Press, 2004.

[4] FAN-LI Zhou, LI-PING Chen, YI-ZHONG Wu, JIAN-WAN Ding, JIAN-JUN Zhao, YUN-QING Zhang, MWorks: a Modern IDE for Modeling and Simulation of Multidomain Physical Systems Based on Modelica, Proceedings of the 5th International Modelica Conference, Volume 2, 725-732, 2006.

An OpenModelica Java External Function Interface Supporting MetaProgramming

Martin Sjölund, Peter Fritzson
PELAB Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
{marsj, petfr}@ida.liu.se

Abstract

A complete Java interface to OpenModelica has been created, supporting both standard Modelica and the metamodeling extensions in MetaModelica. It is bidirectional, and capable of passing both standard Modelica data types, as well as abstract syntax trees and list structures to and from Java and process them in either Java or the OpenModelica Compiler. It currently uses the existing CORBA interface as well as JNI for standard Modelica. It is also capable of automatically generating the Java classes corresponding to MetaModelica code. This interface opens up increased possibilities for tool integration between OpenModelica and Java-based tools, since for example models or model fragments can be extracted from OpenModelica, processed in a Java tool, and put back into the main model representation in OpenModelica.

Keywords: Java, OpenModelica, MetaModelica, external function, abstract syntax

1 Introduction

The main goal of this work is to create a Java interface that can be used to call Modelica functions and evaluate Modelica expressions as described in [4] and [2]. More importantly, it should be possible to use the interface to analyze the abstract syntax tree of OpenModelica from a Java application and create a Java mapping of the code loaded in OpenModelica. To make this possible, the OpenModelica compiler is currently being extended to support uniontypes needed for abstract syntax tree representation. At the time of this writing, most aspects of the OpenModelica abstract syntax tree support are operational. An additional goal is calling Java methods as external Modelica functions, analogous to external C Modelica functions. An external Java interface has not previously been available for OpenModelica. Compared to previous work in Modelica-Java interfaces [6] [5], based on Dymola, this interface also supports the MetaModelica [3] extensions, giving increased possibilities for model manipulation and tool integration.

2 Motivation

External Java functions can be used either as regular Modelica functions, calculating values for models. They could also be used for displaying graphs when simulating models. External Java functions could also be used for MetaProgramming (using MetaModelica types). The external Java interface could be used internally in the OpenModelica Compiler to write functions that can walk any given AST (e.g. to create a `String` representation of any uniontype tree). It is also possible to use the AST together with tools like StringTemplate[9] to transform the AST to e.g. C code or XML. External functions could also perform operations on the AST that are slow in MetaModelica (such as appending to a list or modifying elements without copying parts of the list). However, do note that converting an AST to Java and back is a costly operation and that using external C functions or using a better algorithm might be preferred in this case. An advantage compared to external C MetaModelica functions is that the Java functions are less prone to changes due to changes in Modelica code. If the order of records in a uniontype changes, the constants used to create the uniontype record also changes. OMC does not produce header files containing these constants. External C functions need these constants, while Java functions do not. It is also easier to access the data that MetaModelica structures contain using external Java functions than external C functions because standard Java classes are used as opposed to data pointers.

For communication from Java to OpenModelica, the scenarios are different. Either you have some Meta-

DOI: 10.3384/ecp09430121

Modelica code that does a transformation, but you have the data accessible in Java and you want to manipulate the result in Java. The interface allows you to either construct the corresponding Modelica data structure using Java classes, or to simply send a String. In both cases you receive a Java class corresponding to the Modelica data. The Java interface is also an Interactive Modelica session. This means you can do more manipulations and even call the OpenModelica API. As such you can access the Absyn AST for the currently loaded Modelica files, and manipulate them in Java. The existing API sends strings back and forth. With new API calls, it would be possible to send the actual AST and access it as an AST in the Java-based code.

By analyzing the code loaded in OpenModelica and creating a Java mapping of the loaded datatypes, you bring the programmer of a Java-based Modelica tool the ability to do some more extensive type checking in Java code. Instead of accessing fields by string and explicit type casting, `(ModelicaInteger)record.get("fieldA"))`, it is possible to use `record.get_fieldA()` instead.

3 Mapping of Datatypes

In order to introduce some compatibility between the Dymola and OpenModelica implementations of external Java functions, it makes sense to declare them in the same way (`'Package.Class.StaticMethod'`). The mappings between datatypes will not be the same because we'll use the same mapping when Java is the calling language as opposed to the Dymola version. By doing it this way you get a consistent interface that can also be naturally extended for MetaModelica types (`ModelicaTuple`, `ModelicaOption`). Because the full MetaModelica mapping (Table 1) uses Modelica-specific classes for all datatypes, it can't be used to call e.g. the Java method `Integer.parseInt` since it uses Java `String` and `int`. By annotating your external Java function declaration using `annotation( JavaMapping = "simple" )`, an alternative mapping (Table 2) will be used. This mapping only supports the most basic Modelica types and only one output value, but it can be used to call standard Java functions. This is a subset of the functionality that Dymola has, which also supports arrays, records and output variables that are not the return value of an external function call.

Table 1: OMC Mapping of Java Datatypes

Modelica	External Java
`Real`	`ModelicaReal`
`Integer`	`ModelicaInteger`
`Boolean`	`ModelicaBoolean`
`String`	`ModelicaString`
`Record`	`ModelicaRecord`
`Uniontype`	`IModelicaRecord`
`List<T>`	`ModelicaArray<T>`
`Tuple<T1,T2>`	`ModelicaTuple`
`Option<T>`	`ModelicaOption<T>`
`T[:]`	`ModelicaArray<T>`

Table 2: OMC Simple Mapping of Java Datatypes

Modelica	External Java
`Real`	`double`
`Integer`	`int`
`Boolean`	`bool`
`String`	`String`

If the `ModelicaRecord` datatype is represented by a `java.util.Map` from `String` to `ModelicaObject`, it follows that it can contain any datatype we use in OMC[1]. By using a `LinkedHashMap` the field keys are in the same order as they are in Modelica[2]. One advantage of this solution is that the Java mapping of a record does not depend on creating a Java class before the program is executed. The disadvantage is that you need to check that you received the correct record type, and then get the fields using the method `ModelicaObject get(String key)`. This is equivalent to performing type checking during runtime. For those who want functions to perform said typecasting, see Section 5.3 for a method that creates Java class definitions from Modelica code.

4 Calling Java External Functions from Modelica

When using external C functions, OMC translates a Modelica file (Listing 1) to a C file (Listing 2). First of all, OpenModelica copies all input variables before the external call is made since arrays (as well as variables

[1] The interface `ModelicaObject` includes MetaModelica constructs. Naming it (Meta)ModelicaObject would be more appropriate, but it isn't a valid identifier in Java.

[2] The Modelica standard enforces a strict field ordering because it is relevant for example in external C functions.

for Fortran functions) are passed by reference. Then the external call is performed and the output is copied into the return `struct` (since Modelica supports multiple output values).

Listing 1: exampleC.mo

```
function logC
  input Real x;
  output Real y;
external "C" y = log(x);
end logC;
```

Listing 2: logC.c

```
logC_rettype _logC(modelica_real x)
{
  logC_rettype out;
  double x_ext;
  double y_ext;
  x_ext = (double)x;
  y_ext = log(x_ext);
  out.targ1 = (modelica_real)y_ext;
  return out;
}
```

When using external Java functions, OMC should generate a C file that is similar to the ones generated by external C functions. External Java calls translated the variables to Java objects, and fetch the correct method from the JVM through the Java Native Interface (JNI). The flow of data in Figure 2 is explained in detail below. Before the call, each argument is translated to a JNI `jobject` (i.e. a C pointer to a Java class) and then after copying the result back to the respective C variable. This ensures that the code works in the same way as external C (and thus the “correct” Modelica behaviour). Compare the C file for external C (Listing 2) to the one for external Java (Listing 4, generated by the Modelica code in Listing 3). The Java code is essentially the same with the difference being that instead of one line of code for an external call, it is 17 lines of code to set up the Java call properly.

Listing 3: exampleJava.mo

```
function logJava
  input Real x;
  output Real y;
external "Java"
  y = 'java.lang.Math.log'(x)
  annotation(
    JavaMapping="simple"
  );
end logJava;
```

Listing 4: logJava.c

```
logJava_rettype _logJava(modelica_real
    x)
{
  logJava_rettype out;
  double x_ext;
  double y_ext;
  JNIEnv* __env = NULL;
  jclass __cls = NULL;
  jmethodID __mid = NULL;
  jdouble x_ext_java;
  jdouble y_ext_java;
  x_ext = (double)x;
  __env = getJavaEnv();
  x_ext_java = x_ext;
  __cls = (*__env)->FindClass(__env, "
      java/lang/Math");
  CHECK_FOR_JAVA_EXCEPTION(__env);
  __mid = (*__env)->GetStaticMethodID(
      __env, __cls, "log","(D)D");
  CHECK_FOR_JAVA_EXCEPTION(__env);
  y_ext_java = (*__env)->
      CallStaticDoubleMethod(__env,
      __cls, __mid, x_ext_java);
  CHECK_FOR_JAVA_EXCEPTION(__env);
  y_ext = y_ext_java;
  (*__env)->DeleteLocalRef(__env, __cls
      );
  out.targ1 = (modelica_real)y_ext;
  return out;
}
```

5 Calling Modelica Functions from Java

OpenModelica communicates with other tools through sockets or CORBA using its Interactive module. The Java interface can do the same, just as the Eclipse plugin (MDT) does. Figure 1 shows the existing Java-OpenModelica communication using CORBA. The OMCProxy class does not only communicate with OMC using CORBA. It also starts OMC in server mode if it can’t find a server to communicate with.

Figure 1: CORBA Communication

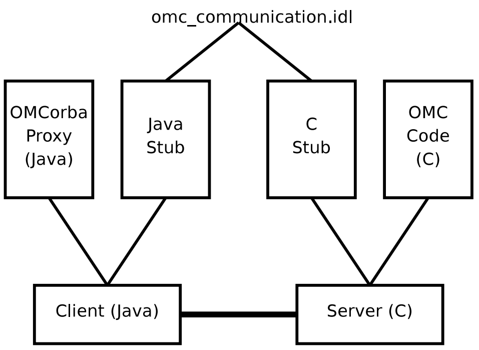


Figure 2: External Java Call (Data Flow)

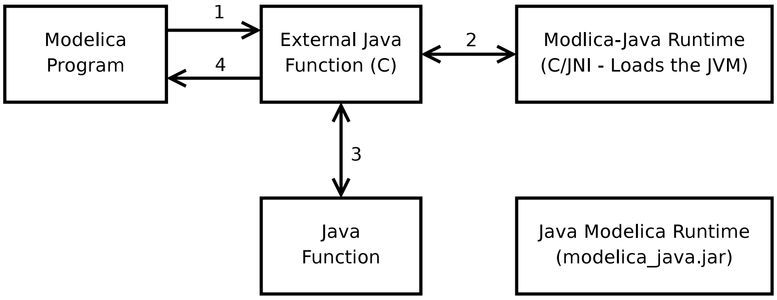
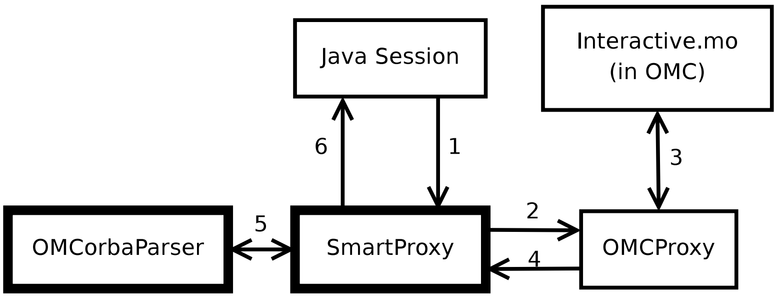

Modelica Program
External Java Function (C)
Modlica-Java Runtime (C/JNI - Loads the JVM)
Java Function
Java Modelica Runtime (modelica_java.jar)
1
2
3
4

Figure 3: Interactive Java Session (data flow)



The marked nodes in Figure 3 are what have been added on top of OMCProxy. SmartProxy only glues OMCProxy and OMCorbaParser together, so the user doesn't need to be aware that those classes exist. The CORBA interface is an untyped string-to-string function which means you can send `{1,2.0,3}` even though the Modelica standard disallows mixed types in arrays [4] [7].

Listing 5 contains an example of an interactive OpenModelica session. The user tells OMC to add a record definition to the AST, and then calls the record constructor. The result is a record.

Listing 5: Interactive OMC Session

```
>> record ABC Integer a;Integer b;
   Integer c; end ABC;
{ABC}
>> ABC(1,2,3)
record ABC
    a = 1,
    b = 2,
    c = 3
end ABC;
```

5.1 Mapping Textual Representations of MetaModelica Constructs to Java

All Modelica objects implement the dummy Java interface `ModelicaObject`, which helps tagging any Modelica data. Table 1 contained the mappings from Modelica types to Java types. The problem with the CORBA interface is that the textual representations are ambiguous. `{1,2,3}` can represent either a MetaModelica list or a Modelica array. `(1,2,3)` can represent either a MetaModelica tuple or multiple function output values. This implementation will treat both cases in the same way. `{1,2,3}` is represented by `ModelicaArray` while `(1,2,3)` is represented by `ModelicaTuple`. Both of these classes extend `java.util.Vector` (which supports both random access and implements the `List` interface).

5.2 Parsing CORBA Output

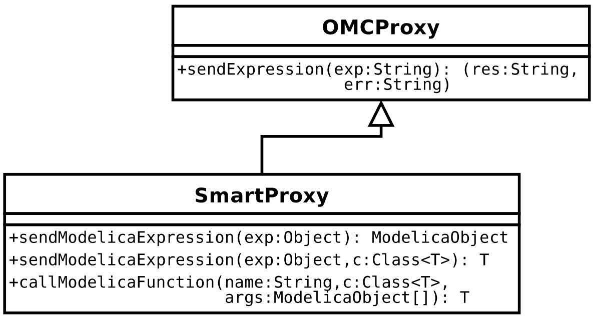
In order to create a reasonably efficient and maintainable parser ANTLRv3 [8] is used to parse the results from the Interactive interface. ANTLRv2 has been used in other parts of OpenModelica with good results, so the choice of parser was quite easy.

What you end up with at this point is an interface that can call Modelica functions, pass Modelica structures and cast the results to the expected type.

This parser translates strings parsed over the OpenModelica interactive interface to the basic Java classes. For example, records are translated to a "generic" record that uses the map interface instead of accessing fields more or less directly). This means you have to write wrapper classes if you want to access these fields without typing lots of code.

When sending an expression from Java to OpenModelica you get back a Java `ModelicaObject`. But if you already know the return type, you don't want to create a lot of code just to cast that object to the expected class. For this reason the Java call `sendModelicaExpression` (and related functions, see Figure 4) can take a Java `Class<ModelicaObject>` and after it has parsed the returned data, the function will attempt to cast the object to the expected class. Should the cast fail, it will also try to construct a new object of the return type using the object as the argument. Thus, all classes implementing `ModelicaObject` have a constructor taking a single `ModelicaObject` (where it will determine if the object is indeed a supertype of the expected type). This is because any record is parsed as a generic `ModelicaRecord` rather than e.g. `ExpressionRecord`. The `ExpressionRecord` constructor should analyze the `ModelicaObject` and determine if it is indeed a `ModelicaRecord` with the correct record name, field names and data types in the fields. The process of creating this class can be done automatically, see Section 5.3 for details on the implementation.

Figure 4: CORBA Communication Proxies


OMCProxy
+sendExpression(exp:String): (res:String, err:String)
SmartProxy
+sendModelicaExpression(exp:Object): ModelicaObject
+sendModelicaExpression(exp:Object,c:Class<T>): T
+callModelicaFunction(name:String,c:Class<T>, args:ModelicaObject[]): T

5.3 Translating MetaModelica Definitions to Java Classes

Since it would be nice to translate MetaModelica AST definitions to Java AST definitions, in the form of a Java JAR file, a second parser was created. This parser is to be used prior to the application development since it tells OMC to load a number of Modelica files and return an AST containing type definitions, functions, uniontypes and records of the files. Extracting the AST is done by a new API call, `getDefinitions`, in the OpenModelica compiler Interactive module, `Interactive.mo`. The output of the call is a tree in textual prefix notation, similar to LISP syntax. It contains a partial extraction of the syntax tree from the `Absyn` module. Note that the OpenModelica Interactive module uses the `Absyn.Program` AST and not the lowered intermediate tree `SCode.Program` or `DAE` ASTs. Because the AST may contain errors (type checking, syntax, etc), you may get some cryptic error messages in programs containing errors in for example unused functions since RML only compiles referenced functions. The textual extraction format is as follows (Modelica code to textual format to Java code):

5.3.1 Packages

Modelica packages are used to place its parts in its corresponding Java packages.

Modelica: `package myPackage; ... end myPackage;`

Intermediate: `(package myPackage ...)`

5.3.2 Type aliasing

In the example below, all occurrences of `myInt` will eventually be replaced by `ModelicaInteger`. The reason is that Java does not support type aliasing.

Modelica: `type myInt = Integer`

Intermediate: `(type myInt Integer)`

Java: `ModelicaInteger`

5.3.3 Records

Records are transformed into Java classes extending ModelicaRecord. The class has set and get functions for each field in the record. Fields of any extended records are looked up. The Java class will not inherit from a base record class because multiple inheritance is disallowed.

Listing 6: Modelica Record

```
record abc
```

```
  extends ab;
  Integer c;
end abc;
```

Intermediate: `(record abc (extends ab) (Integer c))`

Java: `class abc extends ModelicaRecord ...`

5.3.4 Replaceable Types

Replaceable types are handled using Java generics.

Modelica: `replaceable type T subtypeof Any`

Intermediate: `(replaceable type T)`

Java: `<T extends ModelicaObject>`

5.3.5 Uniontypes

Uniontypes are tagged using interfaces.

Listing 7: MetaModelica Uniontype

```
uniontype ut
  record ab
    Integer a; Integer b;
  end ab;
  record bc
    Integer b; Integer c;
  end bc;
end ut;
```

Intermediate: `(uniontype ut) (metarecord ab 0 ut (Integer a) (Integer b)) (metarecord bc 1 ut (Integer b) (Integer c))`

Listing 8: MetaModelica Uniontype (Java)

```
interface ut extends IModelicaRecord {
}
class ab extends ModelicaRecord
    implements ut {
    ...
}
class bc extends ModelicaRecord
    implements ut {
    ...
}
```

5.3.6 Functions

Functions are translated to classes extending `ModelicaFunction`. The method `call` performs the actual function call over the CORBA interface. Functions with multiple return values have two call methods, one that returns a `ModelicaTuple` and one that performs a call-by-reference.

Listing 9: Modelica Function

```
function add
  input Integer lhs;
  input Integer rhs;
  output Integer out;
algorithm
  out := lhs+rhs;
end add;
```

Intermediate: `(function abc (input Integer lhs) (input Integer rhs) (output Integer out))`

Listing 10: Modelica Function (Java)

```
class  add extends ModelicaFunction {
   ...
   ModelicaInteger call(ModelicaInteger
        lhs, ModelicaInteger rhs) {
     ...
   }
}
```

5.3.7 Partial Functions

Partial functions are undefined function pointers (can also be seen as as types). The Java implementation is essentially an identifier (it discards the in/output).

Listing 11: MetaModelica Partial Function

```
partial function addFn
  input Integer lhs;
  input Integer rhs;
  output Integer out;
end addFn;
```

Intermediate: `(partial function addFn)`

Java: `new ModelicaFunctionReference("addFn")`

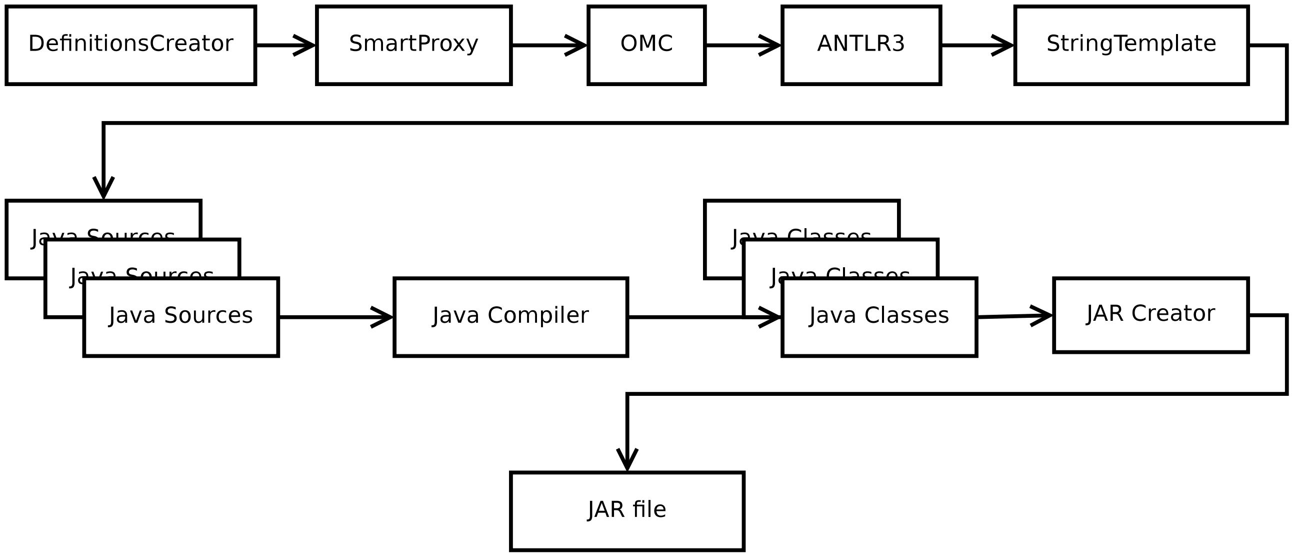
5.4 Translating Two Modelica Functions to Java Classes

The number of steps required to translate a Modelica file into a JAR-file containing all of the definitions is quite large. Figure 5 shows the flow of data and the steps are explained through a simple example. The Modelica code in Listing 12 will be used as the example for the translation from Modelica code to Java classes.

Listing 12: Modelica source to be translated to Java

```
package Simple
```

Figure 5: DefinitionsCreator data flow



```
function AddOne
  input Integer i;
  output Real out;
  Integer one = 1;
algorithm
  out := i+one;
end AddOne;

function AddTwo
  input Integer i;
  output Integer out1;
  output Integer out2;
algorithm
  out1 := i+1;
  out2 := i+2;
end AddTwo;
end Simple;
```

The process starts when you invoke `DefinitionsCreator`. Listing 13 shows how to create `~/examples/simple.jar` (with package prefix `org.openmodelica.example`) from `~/examples/Simple.mo`. The inner workings of the class are described below.

Listing 13: Invoking DefinitionsCreator

```
$ java -classpath $OPENMODELICAHOME/
    share/java/antlr-3.1.3:
    $OPENMODELICAHOME/share/java/
    modelica_java.jar org.openmodelica.
    corba.parser.DefinitionsCreator ~/
    examples/simple.jar org.openmodelica
    .example ~/examples Simple.mo
```

The string representation of the definitions in the AST returned by OMC is:

Listing 14: getDefinitions String corresponding to the Modelica functions

```
(package Simple
(function AddOne
  (input Integer i)
  (output Real out))
(function AddTwo
  (input Integer i)
  (output Integer out1)
  (output Integer out2))
)
```

By using the OMCorbaDefinitions ANTLRv3 grammar [8] and StringTemplate templates [9], Java source files (Listings 15 and 16) corresponding to the definitions are created.

Listing 15: Corresponding Java source for AddOne

```
public class AddOne extends
    ModelicaFunction {
  public AddOne (SmartProxy proxy) {
    super("AddOne", proxy);
  }
  public  ModelicaReal call (
      ModelicaInteger i) throws
      ParseException,ConnectException
  {
    return proxy.callModelicaFunction("
        Simple.AddOne", ModelicaReal.
        class, i);
  }
}
```

Listing 16: Corresponding Java source for AddTwo

```
public class AddTwo extends
   ModelicaFunction {
  public AddTwo (SmartProxy proxy) {
    super("AddTwo", proxy);
  }
  public  ModelicaTuple call (
      ModelicaInteger i) throws
      ParseException,ConnectException
  {
    return proxy.callModelicaFunction("
        Simple.AddTwo", ModelicaTuple.
        class, i);
  }
  public  void call (ModelicaInteger i,
       ModelicaInteger out1,
      ModelicaInteger out2) throws
      ParseException,ConnectException
  {
    ModelicaTuple __tuple = proxy.
        callModelicaFunction("Simple.
        AddTwo", ModelicaTuple.class, i)
        ;
    java.util.Iterator<ModelicaObject>
        __i = __tuple.iterator();
    if (out1 != null) out1.setObject(
        __i.next()); else __i.next();
    if (out2 != null) out2.setObject(
        __i.next()); else __i.next();
  }
}
```

The Java files are compiled using `javac`, the Java Compiler. They are then archived using the `java.util.jar` class. Because StringTemplate is used, the code could potentially be re-targeted in order to create for example C# definitions, but the Java compilation and JAR steps would need to be replaced with functions that could handle C#.

6 Limitations

The implementation requires access to the OpenModelica CORBA interface or external functions generated by OpenModelica. As such, OpenModelica needs to be fully bootstrapped before the interface can be used internally in OpenModelica. At the moment, it can be used for simple Modelica/MetaModelica programs.

Java is quite limited when it comes to generics. Generics in Java is just something that helps the programmer do static type checking. In running code, Java has no concept of generic types and is totally unchecked. This is one of the reasons why `ModelicaTuple` is untyped in Java.

7 Related Work

Dymola has the capability to call Modelica functions and the Dymola API from external Java functions [5]. Their approach was to use a single entry-point (com.dynasim.dymola.interpretMainStatic). This is probably a bit faster than passing and parsing strings, and would have been possible to accomplish in OpenModelica as well. The OpenModelica CORBA interface is more akin to the Dymola external interface described in [6]. It also uses strings to communicate with applications, but can construct some native types for example when communicating from Modelica to Matlab.

8 Future

The OpenModelica compiler is currently being extended to support the datatypes introduced in MetaModelica needed to represent and communicate abstract syntax trees. Another planned extension is to replace the current text-based CORBA interface with a directly linked version, giving higher performance.

As work progresses, support for new datatypes needs to be added in the Interactive module since the CORBA interface depends on this module being updated. Most of the work so far has been limited to compiling code using these datatypes (e.g. the uniontype implementation [1]).

9 Conclusions

A complete bidirectional Java interface to OpenModelica including support of the MetaModelica language extensions has been created. It is capable of passing basic and structured data types including syntax trees to and from Java and process them in either Java or the OpenModelica Compiler. It uses the existing CORBA interface and is capable of automatically generating the Java classes corresponding to MetaModelica code. This new interface opens up new possibilities for tool integration and model manipulation.

References

[1] Björklén S. Extending Modelica with High-Level Data Structures: Design and Implementation in OpenModelica. Linköping, Sweden: Master's thesis, Department of Computer and Information Science, Linköping University, 2008.

[2] Fritzson P. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, 940 pages. Wiley-IEEE Press, 2004.

[3] Fritzson P. MetaModelica Programming Guide, June 2007 draft. http://openmodelica.org/.

[4] Modelica Association. The Modelica Language Specification Version 3.0, September 2007. http://www.modelica.org/.

[5] López J.D., Olsson H. Dymola interface to Java - A Case Study: Distributed Simulations. In: Proceedings of the 5th International Modelica Conference, Vienna, Austria, 4-5 September 2006.

[6] Olsson H. External Interface to Modelica in Dymola. Proceedings of the 4th International Modelica Conference, Hamburg, Germany, 7-8 March 2005.

[7] OpenModelica. OpenModelica System Documentation, January 2009. http://openmodelica.org/.

[8] Parr T. ANTLR Parser Generator. http://antlr.org/.

[9] Parr T. StringTemplate Template Engine. http://stringtemplate.org/.

Towards a Text Generation Template Language for Modelica

Peter Fritzson[*], Pavol Privitzer[+], Martin Sjölund[*], Adrian Pop[*]
[+]Institute of Pathological Physiology, First Faculty of Medicine, University in Prague
[*]PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
pavol.privitzer@if1.cuni.cz, {petfr, marsj,adrpo}@ida.liu.se

Abstract

The uses, needs, and requirements of a text generation template language for Modelica are discussed. A template language may allow more concise and readable programming of the generation of textual models, program code, or documents, from a structured model representation such as abstract syntax trees (AST). Applications can be found in generating simulation code in other programming languages from models, generation of specialized models for various applications, generation of documentation, web pages, etc. We present several template language designs and some usage examples, both C code generation and Modelica model generation. Implementation is done in the OpenModelica environment. Two designs are currently operational.

Keywords: template language, unparsing, pretty printing, code generation, Modelica.

1 Introduction

Traditionally, models in a modeling language such as Modelica are primarily used for simulation. However, the modeling community needs not only tools for simulation but also languages and tools to create, query, manipulate, and compose equation-based models. Examples are parallelization of models, optimization of models, checking and configuration of models, generation of program code, documentation and web pages from models.

If all this functionality is added to the model compiler, it tends to become large and complex.

An alternative idea that already to some extent has been explored in MetaModelica [9][21] is to add extensibility features to the modeling language. For example, a model package could contain model analysis and translation features that therefore are not needed in the model compiler. An example is a PDEs discretization scheme that could be expressed in the modeling language itself as part of a PDE package instead of being added internally to the model compiler.

Such transformation and analysis operations typically operate on abstract syntax tree (AST) representations of the model. Therefore the model needs to be converted to tree form by *parsing* before transformation, and later be converted back into text by the process of *unparsing*, also called *pretty printing*.

The MetaModelica work is primarily focused on mechanisms for mapping/transforming models as structured data (AST) into structured data (AST), which is needed in advanced symbolic transformations and compilers.

However, there is an important *subclass* of problems mapping structured data (AST) representations of models into text. Unparsing is one example. Generation of simulation code in C or some other language from a flattened model representation is another example. Yet another use case is model or document generation based on text templates where only (small) parts of the target text needs to be replaced.

We believe that providing a template language for Modelica may fulfill a need for an easier-to-use approach to a class of applications in model transformation based on conversion of structure into text. Particularly, we want to develop an operational template language that enables to retarget OpenModelica compiler simply by specifying a package of templates for the new target language.

1.1 Structure of the Paper

Section 2 tries to define the notion of template language, whereas Section 3 gives more detailed language design requirements, uses, motivation, and design principles. Section 4 shows an example of a very concise template language, its uses, and lessons learned. Section 5 presents model-view-controller separation which has important implications for the design. Section 6 presents a small interpreted template language prototype.

Section 8 briefly discusses applications in code generation from the OpenModelica compiler, whereas Sec-

DOI: 10.3384/ecp09430124

tion 9 presents related work, followed by conclusions in Section 10.

2 What is a Template Language?

In this section we try to be more precise regarding what is meant by the notion of template language.

2.1 Template Language

Definition 1. Template Language. A template language is a language for specifying the transformation of structured data into a textual target data representation, by the use of a parameterized object "the template" and constructs for specifying the template and the passing of actual parameters into the template.

One could generalize the notion of template language to cover target language representations that are not textual. However, in the following we only concern ourselves with textual template languages.

Definition 2. Template. A template is a function from a set of attributes/parameters to a textual data structure.

A template can also be viewed as a text string with holes in it. The holes are filled by evaluating expressions that are converted to text when evaluating the template body. More formally, we can use the definition from [17] (slightly adapted):

A template is a function that maps a set of attributes to a textual data structure. It can be specified via an alternating list of text strings, t_i, and expressions, e_i, that are functions of attributes a_i:

$$F(a_1, a_2, ..., a_m) ::= t_0\, e_0 ... t_i\, e_i\, t_{i+1} ... t_n\, e_n\, t_{n+1}$$

where t_i may be the empty string and e_i is restricted computationally and syntactically to enforce strict model-view separation, see Section 5 and [18]. The e_i are distinguished from the surrounding text strings by bracket symbols. Some design alternatives are angle brackets <...>, dollar sign $...$, combined <$...$>. Evaluating a template involves traversing and concatenating all t_i and e_i expression results.

Definition 3. Textual Data Structure. A textual data structure has text data such as strings of characters as leaf elements. Examples of textual data are: a string, a list (or nested list structure) of strings, an array of strings, or a text file containing a single (large) string. A textual data structure should efficiently be able to convert (flattened) into a string or text file.

2.2 Unparser Specification Language

Definition 4. Unparser Specification Language. A special case of template language which is tailored to specifying unparsers, i.e., programs that transform an abstract syntax (AST) program/model representation into nicely indented program/model text.

Example: The unparser specification language in the DICE system [3] was used to specify unparsers for the Pascal and Ada programming languages. The unparser specification was integrated with the abstract syntax tree specification, to which it referred. See also the example in Section 4.

3 Requirements and Motivation

What are our requirements on a template language for Modelica? Why don't use an existing template language, e.g. one of those mentioned in Section 9. In fact, do we need a template language extension at all? Why not just program this presumable rather "simple" task of converting structure into text by hand in an ordinary programming language? In the following we briefly discuss these issues.

- *Need for a template language*? Conversion of structure into text has of course been programmed many times by hand in a multitude of programming languages. For example, the unparser and the C code generator in the current OpenModelica compiler are hand implemented in MetaModelica. An advantage is usually good performance.

  However, the disadvantages include the lack of extensibility and modeling capability mentioned in Section 1. Another problem is that the code easily gets cluttered by a mix of (conditional) print statements and program logic. A third problem is reuse. For example, when generating target code in similar languages C, C#, or Java, large parts of the output is almost the same. It would be nice to re-use the common core of the code, instead of (as now) need to develop three versions with slight differences
- *Performance needs*. There are different performance needs depending on application. A template language that is mainly used for generation of html pages may need more flexibility in the order of text generation (lazy evaluation), whereas a language used to specify a code generation from AST needs higher performance. Compilation should not take too long even when you compile a hundred thousand lines of models represented as a million AST nodes.
- *Intended users*. Are the intended users just a few compiler specialists, or a larger group including modeling language users who wants easy-to-use tool extensibility?
- *Re-implement/re-use an existing template language*? Why not re-implement (or re-use) an existing tem-

plate language such as for example ST [17] for StringTemplate? This choice depends on the character of the existing language and its implementation, efficiency, and complexity of tool integration.

3.1 Language Design Principles

The following are language design principles [12]:

- *Conceptual clarity*. The language concepts are well defined.
- *Orthogonality*. The language constructs are "independent" and can be combined without restrictions.
- *Readability*. Programs in the language are "easy" to read for most developers.
- *Conciseness*. The resulting program is very short.
- *Expressive Powe*r. The language has powerful programming constructs.
- *Simplicity*. Few and easily understood constructs.
- *Generality*. Few general constructs instead of many special purpose constructs.

Some of these principles are in conflict. Conciseness makes it quick to write but often harder to read, not as easy to use, sometimes less general. Expressive power often conflicts with simplicity.

3.2 Language Embedding or Domain Specific Language?

Should the template language be a completely new language or should it be embedded into an existing language as a small extension to that language?

A language that addresses a specific problem domain is called *domain specific language* (DSL). DSLs can be categorized as *internal* or *external* [4][5].

Internal DSLs are particular ways of using a host language in a domain-specific way. This approach is used, e.g., for the pretty printer library in Haskell where document layouts are described using a set of operators/functions in a language-like way [23].

External DSLs have their own custom syntax and a separate parser is needed to process them. As an example, StringTemplate [18][17] is an external DSL and is provided for three different host languages: Java, C# and Python.

If you only need the template language for simple tasks, or tasks that do not require high performance and tight communication with the host language, a separate language might be the right choice. A small language may be quicker learn and focused on a specific task.

On the other hand, embedding into the host language makes it possible to re-use many facilities such as: efficient compilation, inheritance and specialization of templates, reuse of common programming constructs, existing development environment, etc., which otherwise need to be (partly) re-developed. A disadvantage is that the host language grows if the extension cannot be well separated from the host language.

Proliferation of DSLs might also be a problem. For example, consider a large application with extensive usage of, say, twenty different DSLs that may have incompatible and different semantics for language constructs with similar syntax. This might lead to a maintenance nightmare.

Also, what is exactly domain specific in a text template language? The answer is probably only the handling of the template text string with holes in it, switching between text mode and attribute expressions, and implicit concatenation of elements. All the rest, e.g., expression evaluation, function call, function definition, control structures, etc., can be essentially the same as in a general purpose language.

The design trade-offs in this matter are not easy and the authors of this paper do not (yet) completely agree on all choices. Therefore, in this paper we partly explore several design choices for a template language for Modelica.

4 A Concise Template Language

To make the basic ideas of a template language more concrete, we first present a very concise template language [4] which is primarily an unparser specification language. It has been used to specify unparsers for Pascal, Ada, and Modelica. Specifications are very compact. Implementation is simple and efficient.

We will use the following simple Modelica code example to illustrate this template language:

```
while x<20 loop
  x := x+y*2;
end while;
```

This code needs the abstract syntax tree nodes for its internal representation, specified as follows including small template language unparsing strings.

There are two statements nodes types: `ASSIGN` and `WHILE`. `ASSIGN` has two children,. `lhs` of type `PVAR` and `rhs` of type `EXPR`.

A typical assignment looks like `"variable := expression"`. The unparsing specification `"@1 := @2"` means: `@` signals a command that the next character has special interpretation. `@1` means: unparse the first child node. The following characters in the string `" := "` are just output as they are. The next command: `@2` means: unparse the second child of the `ASSIGN` node.

```
// Statement nodes STM
ASSIGN : (lhs: PVAR;
          rhs: EXPR) : "@1 := @2";
```

```
WHILE   : (condition: EXPR;
          statements: STM_LIST) :  "while
@1 loop @+@n @2;@n@q@-@nend while;@n"
```

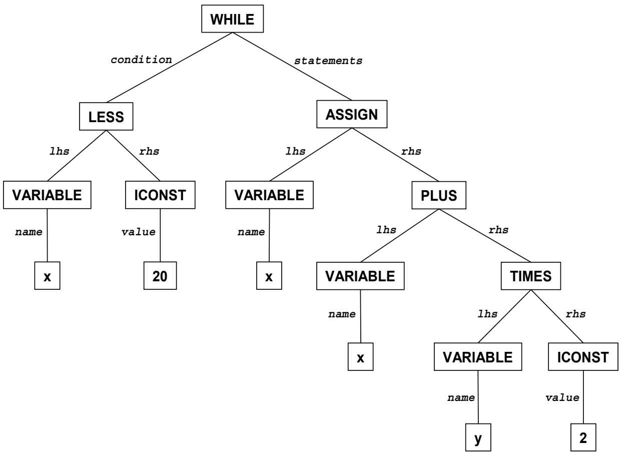


Figure 1. Abstract syntax tree of the while loop.

The template string for `while` has `statements` as a statement list. The semicolon `;` and new line `@n` between `@2` and `@q` (for quit) are emitted between each list item. `@+` and `@-` increase/decrease indentation level.

```
// Expression nodes EXPR
PLUS    : (lhs:EXPR; rhs: EXPR) :
                        "@1+@2" LPRIO 4;
TIMES   : (lhs:EXPR; rhs: EXPR) :
                        "@1*@2" LPRIO 5;
LESS    : (lhs:EXPR; rhs: EXPR) :
                        "@1<@2" BPRIO 3;
VARIABLE : (name: STRING)   : "@1";
ICONST   : (value: INTEGER): "@1";
```

The expression nodes also specify associativity and priority. The latter controls whether parentheses should be emitted. `LPRIO 4` means left associative, priority 4.

4.1 Usage Experience

The full abstract syntax and unparsing specification for Pascal is only 4 pages, and not that hard to write. The full Ada specification is 9 pages, still quite reasonable for a big language. Fifteen years later, such a specification was also developed for Modelica 1.2.

This became more complicated than the one for Ada. Also, maintenance became an issue, especially for other people than the original specification developer. People found the extremely concise unparsing template strings very hard to read and debug. Eventually we decided to rewrite the unparser into normal programming language code (mix of print statements and standard code). Not as elegant, but easier to maintain. Thus, conciseness made specifications short to write, but too hard to read and use/maintain. Another option could have been to redesign the language, e.g. introducing names instead of positions, but there was no time.

5 Model View Controller Separation

A strong design principle argued to especially relevant for template languages is model-view-controller separation [16]. First we define these terms in the context of a template language:

- Model – the data structure, e.g. an AST, to be converted to text according to the view.
- Controller – the piece of software that controls the application of the view to the model, e.g. a tree traversal algorithm applying the templates to the tree nodes.
- View – the mapping from attributes to text, i.e., the actual templates in a template language.

The value of this principle is strongly argued in [16], according to experience with the ST functional template language [17] in the StringTemplate system. Such separation gives more flexibility (multiple views), easier maintainability, better reuse, more ease-of-use, etc.

It is argued that the template language should be kept simple, program computation logic should not be too much intertwined with emitting text. If complex computation needs to be done, it should instead be done on the model (in our case the AST).

Our template language design has been strongly influenced by this principle.

6 A First Template Language for Modelica

A template language maps model items to text attributes (sometimes through intermediate stages). The attributes are referred to by named references in the templates. During template evaluation, the named references are replaced by the text values of these attributes. Thus, a template usually contains two items: a text with named placeholders, and a mapping from attribute names to text values, i.e. a dictionary.

In an advanced implementation (Section 7) the dictionary part can be left out if the template compiler is able to automatically map variable names to string values without an intermediary dictionary data structure.

In the rest of this section we present a first design of a simple template language based on the language embedding idea, together with some examples.

6.1 Text Output with a String Function

As previously mentioned in Section 2.1, a template is a function from structured data, e.g. record structures or abstract syntax trees, to a textual data structure, where the text can be returned as a string or output to a file.

Starting with a small code example:

```
while x < 20 loop ... end while;
```

This can be represented as an abstract syntax tree according to Section 7.3 Section 6.4, from which we have extracted two definitions:

```
uniontype Statement  "Algorithmic stmts"
 record WHILE  "While statement"
   Exp condition;
   list<Statement> statements;
 end WHILE;
end Statement;

uniontype Exp  "Expressions"
 record BINARY  "Binary operator"
   Exp lhs;
   Operator op;
   Exp rhs;
 end BINARY;
end Exp;

type AST = Statement; "Current AST type"
```

We would like to produce the following output from the example abstract syntax tree (AST):

```
The expression loops while x < 20.
```

Below we show three variants of Modelica functions producing this output, where the third one is based on the Modelica template language. Here we assume that an intermediary dictionary is not needed.

6.1.1 Function Returning a String

This function converts the AST example into a string by concatenating string pieces and using the built-in Modelica 3.1 `String` function to convert any record to a string. A locally defined `String` function can be defined within each record type definition (not shown here)

```
function mkString
  input AST whileStm;
  output String out :=
   "The expression loops while " + String(
     whileStm.condition.lhs.name) +
    " < " +  String(
    whileStm.condition.rhs.value) + ".";
end mkString;
```

6.1.2 Function with File Output

If we instead would like to output to a file without first concatenating strings, it might appear as follows:

```
function emitString
  input AST whileStm;
  input FILE file;
algorithm
  print(file,
    "The expression loops while ");
  print(file, String(
    whileStm.condition.lhs.name));
  print(file, " < ");
  print(file,
     String(whileStm.condition.rhs.value));
  print(file, ".");
end emitString;
```

6.1.3 Function Based on a Template

The following function uses the Modelica template language syntax defined in Section 6.3. The idea is to automatically generate the string function in Section 6.1.1 or the file output function in Section 6.1.2.

The escape-code << on a single line signals the start of the template section, and >> on a single line ends it. Text (excluding the first and last single lines) is just used verbatim. Pieces of text are automatically concatenated or output to a file. The escape-code <$ signals the beginning of some piece of Modelica code that should be automatically converted to a string, and $> ends it.

```
function templString
  input AST whileStm;
<<
The expression loops while
<$whileStm.condition.lhs.name$> <
<$whileStm.condition.rhs.value$>.
>>
end templString;
```

One can also let all template functions inherit common characteristics from a common base function, e.g.:

```
function templString
  extends TemplateFunction;
<<
...
>>
end templString;
```

6.1.4 Benefits of Template Functions

The main benefit of the text template approach is that the string conversion, concatenation, and file output code can be generated automatically instead of hand implemented, which increases readability and model-view-controller separation.

Another benefit supported by some template engines (e.g., StringTemplate [17]) is lazy evaluation – all the data structure pieces need not be evaluated in the order they are referred to in the template; instead evaluation is automatically delayed if needed, until the final result is output.

6.2 The Simple Template Language Dictionary

The simple template language dictionary used for lookup in the following small examples is defined below via the `DictItemList` constant, with a simple mapping from key to object. The number of datatypes that the dictionary can hold is very limited compared to more advanced template engines. The idea is that everything in the model is a Boolean, a string, a collection of strings, or a nested dictionary (to allow recursive datatypes). First we define the dictionary data types needed:

```
uniontype Dict

  record ENABLED
  end ENABLED;

  record STRING LIST
   list<String> strings;
  end STRING LIST;

  record STRING
    String string;
  end STRING;

  record DICTIONARY
    DictItemList dict;
  end DICTIONARY;

  record DICTIONARY LIST
    list<DictItemList> dict;
  end DICTIONARY LIST;
end Dict;

record DictItem
  String key;
  Dict dict;
end DictItem;

type DictItemList = list<DictItem>;
```

Then we define a sample dictionary to be used in some of our examples:

```
constant DictItemList sampleDict = {
  DictItem("EnableText", ENABLED() ),

  DictItem("People", DICTIONARY_LIST( {
    DICTIONARY( {
     DictItem("Name", STRING("Adam")),
     DictItem("Fruits", STRING_LIST(
       {"Orange "} )
     } ),
    DICTIONARY( {
     DictItem("Name", STRING("Bertil")),
     DictItem("Fruits", STRING_LIST(
       {"Apple", "Banana", "Orange "} )
     } )
   }),

  DictItem("WHILE", ENABLED()),

  DictItem("condition",
   DICTIONARY( {
    DictItem("lhs", DICTIONARY({
     DictItem("VARIABLE", ENABLED()),
     DictItem("name",STRING("x"))
    })),
    DictItem("rhs", DICTIONARY({
     DictItem("ICONST", ENABLED()),
     DictItem("value",STRING("20"))
    }))
   }) )
};
```

6.3 Template Syntax

Below are the constructs used in the simple template language. Each construct contains the identifier used in the compiled template, as well as the character sequence used to construct it.

Note: This is a preliminary, rather cryptic syntax that was quick to implement by an interpreter. Below are also some examples of more readable Modelica syntax are shown for certain constructs.

A key is a string that does not contain any characters using $, or ", and does not start with #,!,=,^, or _. It is used for lookup of attributes from the dictionary environment. The dictionary environment is a simple linked environment where the current scope has the highest priority.

In the Modelica-syntax variant, `<$ $>` are used to contain Modelica code and/or attribute names.

`FOR_EACH` loops and `RECURSION` both change the dictionary environment. If the key contains dots, they are used for nested lookup.

Only items of the type `DICTIONARY` can be accessed recursively, but the last element can be of any type (e.g. `DICT1.DICT2.DICT3`.key).

6.3.1 Lookup of a Key Value

If `lookup(dict,key)` returns a string, this becomes the output.

Template syntax:
```
$key$
```

Modelica-like template syntax:
```
<$key$>
```

or a variant with explicit Modelica lookup syntax that can be used inside Modelica code context:
```
keyValue(dict,"key")
```

Example template:
```
The expression loops while
$condition.lhs.name$ <
$condition.rhs.value$.
```

Modelica-like example template:
```
The expression loops while
<$condition.lhs.name$> <
<$condition.rhs.value$>.
```

Example output:
```
The expression loops while x < 20.
```

6.3.2 Checking non-empty Attribute Values

If `lookup(dict,key)` returns any non-empty value (empty strings and lists are empty values), run body. The general syntax also includes elseif and else clauses.

Template syntax::
```
$=key$body$/=
```

Modelica-like template syntax (where `[]` means 0 or 1 times, { } means 0 or >= 1 times):

```
<$if key then$>body{<$elseif$>body}
[<$else$>body] <$end if$>
```

Abstract syntax:

```
COND(cond_bodies={(key,true,body)},else_bo
dy={})
```

Example template:

```
$=WHILE$This is a while expression.$/=
```

Modelica-like example template:

```
<$if WHILE then$>This is a while
expression.<$end if$>
```

Example output:

```
This is a while expression.
```

6.3.3 Checking for Empty Attribute Value

Checking for empty attribute values. The opposite of checking nonempty values.

Template syntax::

```
$!key$body$/!
```

Modelica-like template syntax (where `[]` means 0 or 1 times, { } means 0 or >= 1 times):

```
<$if not key$>
body {<$elseif$> body} [<$else$> body]
```

Abstract syntax:

```
COND(cond_bodies={(key,false,body)},else_b
ody={})
```

Example template:

```
$!ASSIGN$This is not an assignment.$/!
```

Modelica-like example template:

```
<$if not ASSIGN then$>This is not an
assignment.<$end if$>
```

Example output:

```
This is not an assignment.
```

6.3.4 For Each Iteration

Use `lookup(dict,key)` to fetch a `STRING_LIST`, `DICTIONARY` or `DICTIONARY_LIST` value, then iterate over the elements in the fetched item. Iterating over `DICTIONARY` and `DICTIONARY_LIST` modifies the dictionary environment (it adds the dictionary to the top-most dictionary in use). The (optional) separator is inserted verbatim between the result of each iteration.

In the Modelica syntax case, an ordinary array iterator { } is used to collect the results of the iterations, and the `insertSep` function to insert separator strings between the items.

Template syntax:

```
$#key[#sep]$body$/#
```

Modelica-like template syntax without separators:

```
<${$>body<$for this in <$key$>}$>
```

Modelica-like template syntax with separators:

```
<$insertSep( {$>body<$ for this in
<$key$>}, sep="...")$>
```

Abstract syntax:

```
FOR_EACH(...)
```

There is an example in the next section.

6.3.5 Current Item Value in Iterations

Only valid when looping over a `STRING_LIST` value. Outputs the current value item string.

Template syntax:

```
$this$
```

Modelica-like template syntax:

```
<$this$>
```

Abstract syntax:

```
CURRENT_VALUE(...)
```

Example template with nested `for each` (first key is `People`, retrieving a dictionary list where each person dictionary has a key `Name` with string value and another key `Fruits` with string list value:

```
$#People$$Name$ has the following
fruits:\n
$#Fruits#, $$this$$/#\n
$/#
```

Modelica-like example template:

```
<${$><$Name$> has the following fruits:\n
<$insertSep($><$Fruits$><$, sep=", ")$>
<$for person in People}$>
```

Modelica-like example template with explicit `keyValue` calls:

```
<${keyValue(person,"Name")$> has the
following fruits:\n
<$ insertSep(keyValue(person,"Fruits"),
sep=", ") for person in People}$>
```

Output:

```
Adam has the following fruits:
Orange
Bertil has the following fruits:
Apple, Banana, Orange
```

6.3.6 Recursion

Use `lookup(dict,key)` to fetch a `DICTIONARY` or `DICTIONARY LIST` value. It will then use the current scope (from `FOR EACH` or the global scope) to iterate over the elements from the `DICTIONARY LIST` as the new top of the dictionary environment. The current auto-indentation depth is concatenated to the indent.

Note: the special construct for recursion on the current template is unnecessary in the Modelica syntax case, since you can just call the template with the same name. Calling templates is shown in Section 6.3.8.

Template syntax:

```
$^key[#indent]$body$/^
```

Modelica-like template syntax, where each subtemplate to be called would need to be explicitly named:

```
<$subtemplate()$>
```

Abstract syntax:

```
RECURSION(...)
```

6.3.7 Increasing Indentation

Opens up a new scope and adds indent to the indentation level.

Template syntax:

```
$_indent$body$/_
```

Abstract syntax:

```
ADD_INDENTATION(...)
```

Example template, where we use * instead of space to be more visible as indentation whitespace:

```
$_***$$=EnableText$\n
Listing all the people:\n
$^People#......$
$/=
$!EnableText$$Name$\n
$/!$/_
```

Output:

```
***Listing all the people:
***Adam
***......Bertil
***......
```

6.3.8 Calling a Pre-Compiled Template

When compiling a template, you also send the engine a list of keys mapped to pre-compiled templates. Calling a template opens up a new scope.

Template syntax:

```
$:subtemplate$:
```

Modelica-like template syntax:

```
<$subtemplate()$>
```

Abstract syntax:

```
INCLUDE(...)
```

Example template:

```
$:AddIndentationExample$$:CurrentValueExam
ple$
```

Modelica like example template:

```
<$AddIndentationExample()$>
<$CurrentValueExample()$>
```

Output:

```
    Listing all the people:
    Adam
    ......Bertil
    ......Adam has the following fruits:
Orange
Bertil has the following fruits:
Apple, Banana, Orange
```

6.4 Generating C Code from a While Loop

We return to the while loop example shown previously in Section 4, to be represented as an AST:

```
while x<20 loop
  x := x+y*2;
end while;
```

The abstract syntax types can be found in Section 7.3.

6.4.1 Small Template Language Example

Templates for emitting C code from the AST of a while loop:

```
$=WHILE$\n
while ($#condition$$:Exp$$/#) {
$^statements#  $\n
}
$/=
$=ASSIGN$
\n$lhs.name$ = $#rhs$$:Exp$$/#;
$/=

$=BINARY$
($^lhs$ $#op$$:op$$/# $^rhs$)
$/=
$=ICONST$    $=PLUS$  $=TIMES$   $=LESS$
$value$      +        *          <
$/=          $/=      $/=        $/=
$=VARIABLE$
$name$
$/=
```

7 Susan – A Compiled Template Language for Modelica

The template language shown in Section 6 (the concise cryptic syntax variant) was implemented as an interpreted external DSL that has both advantages and disadvantages. First the advantages:

- Strictly adheres to the model-view-controller separation as in [16].
- The language is small, and does not perform computation on the model, as advocated in [17].
- Simple to implement and modular.

There are also disadvantages:

- The non-Modelica syntax is cryptic, hard to read.
- Interpretation does not give enough performance.

As the next step we have developed an improved template language design and implementation called *Susan*, with the following main advantages:

- Presumable increased readability
- Compiled to gain maximum performance
- MVC separation is enforced in a more suitable way in context of MetaModelica as the host language

- The language is mature enough to provide a complete vehicle for target code generator specifications in the OpenModelica compiler (OMC) environment.
- The syntax and semantics complies with the MetaModelica type system for textual templates

To summarize, this is a functional, strongly typed, expression oriented template language.

7.1 MVC and Control

Susan's design is strongly influenced by the StringTemplate's (ST) [17] language, briefly described in Section 9.3, and below.

ST's control logic, i.e., conditional inclusion of template parts, is restricted to querying attributes only for their presence/absence or true/false values. This is designed to strictly prevent entanglement of Model and View (MVC). It is primarily obeying the rules *"the view cannot make data type assumptions"* and *"the view cannot compare dependent data values"* [16].

Before an ST template can be rendered to text the attribute values must be transferred to it completely. It is then the work of the Controller to bridge the gap from the Model to the template, e.g. extract data from a database, call some business logic on the Model or walk over an AST, and then transfer the proper values as template attributes.

Susan also transfers data from the Model to the template View, but integrates more control into the View in terms the match construct (Section 7.9).

7.2 Strongly Typed Templates

MetaModelica extends the Modelica type system with union types to facilitate construction of tree-like data structures, in particular Abstract Syntax Trees (ASTs) for efficient modeling of languages.

In our early interpreted template language design we have been using a simple template dictionary (Section 6.2) as an analogy to ST's object model. While general and simple the creation and dynamic lookup implies a certain performance loss.

In order to increase efficiency, we need to avoid the dictionary. As a consequence, templates should be able to directly access MetaModelica data structures. This lead us to strongly typed templates with read-only semantics, with some more control included.

Making templates strongly typed has advantages like generating more efficient code, and avoiding errors that otherwise might occur in applications if only dynamic typing would be used.

7.3 Template Package Type Views

Templates in the Susan language are grouped in *packages*. Each template package can import one or more *type views*, i.e., sets of AST type definitions. Each type view uses MetaModelica syntax and resides in a separate file. Here we will use a type view that can model the while loop example from Section 4:

```
package OriginalPackageName

uniontype Statement  "Algorithmic stmts"
  record ASSIGN  "An assignment stmt"
    Exp lhs; Exp rhs;
  end ASSIGN;

  record WHILE  "A while statement"
    Exp condition;
    list<Statement> statements;
  end WHILE;
end Statement;

uniontype Exp  "Expression nodes"
  record ICONST  "Integer constant value"
    Integer value;
  end ICONST;

  record VARIABLE "Variable reference"
    String name;
  end VARIABLE;

  record BINARY  "Binary ops"
    Exp lhs; Operator op;  Exp rhs;
  end BINARY;
end Exp;

uniontype Operator
  record PLUS end PLUS;
  record TIMES end TIMES;
  record LESS end LESS;
end Operator;

end OriginalPackageName;
```

The `OriginalPackageName` is the name of the original MetaModelica package where types included in the type view are fully defined. A type view can use types from several packages. It usually specifies a subset of the original types defined in several packages and from these types suitable parts can be selected. For example, there can be additional union tags in the `Statement` type, but only those two specified can be used by templates that use this view. Similarly, more record fields can be originally defined in the `ASSIGN` record but only `lhs` and `rhs` can be read inside the template package with the view imported.

AST type view files can be shared across different target languages as a kind of type interface to the compiler generated output ASTs (e.g., simulation code ASTs). It is also an essential feature to support scenarios where users are not allowed to see all original types (e.g., a commercial Modelica compiler) but still can see and use the intended subset to extend the code generator.

In addition to type views, templates automatically understand all MetaModelica built-in types: `String`, `Boolean`, `Integer`, `Real`, `list`, `Option`, `tuple`, and `Array` types.

7.4 Template Definition

A *template definition* in Susan has a C-like function signature with a name and formal typed arguments, instead of a Modelica-like signature as in the design of Section 6 The body is a single template expression without explicit delimiters:

```
templ-name(Type1 n1, Type2 n2, ...) ::=
                          template-expression
```

A template's textual output is the result of the template expression evaluated with the actual parameter values in its scope. All parameters are input and read-only; in general, all values bound to names are read-only inside template expressions.

Unlike ST, which uses *dynamic scoping* of *attributes*, this language uses *lexical scoping*. After the symbol `::=`, a new lexical scope is created for template parameters that are only accessible by their names inside the scope. Nested lexical scopes can also be created by other constructs, e.g. in map expressions.

ST uses the concept of an implicitly available *default attribute*, named `it`, to decrease the verbosity in some common expression forms. This concept has been adapted for Susan as an implicitly available variable.

In the following sections we provide short descriptions of the five kinds of Susan's expressions:

Textual template expressions, named value references, template calls, match and *conditional expressions, and map expressions.*

7.5 Textual Template Expressions

A fundamental concept used for *textual template expressions* is a ”text with holes in it”. An example is

```
'Dear Mr. <name>.'
```

When the expression is rendered to text, the value of the `name` parameter is filled into `<...>` angle-bracketed marked hole and the brackets are discarded.

We have chosen single quotes, unlike ST, because we wanted double quotes to be reserved for string constants, thus

```
 "Dear Mrs. <nice>"
```

is a constant textual template expression precisely following Modelica string syntax without any holes, and it respects ordinary escape characters like `"\n"` or `"\t"` for new line and tab characters.

To support readability (or verbatimness) of templates to the maximum extent, the `<<...>>` delimiting pair can be also used for longer templates with holes as follows, where there is a rule that a new line right after the opening delimiter and a new line right before the closing delimiter are ignored:

```
<<
Hi '<name>',
today is <dayName>.
>>
```

There is an equivalent to `<<...>>` for longer constant texts, the `%X...X%` verbatim string delimiting pair, where the `X` can be an arbitrary character where pairs of `()` `[]` `{}` are respected like

```
%(
\\ (Really) '<verbatim>' "text\n"
)%
```

or like

```
%*Some shining <*> is over there!*%
```

Everything inside the `%X...X%` is taken verbatim with complete lack of escapes.

We have provided the basis for the text part of the language, e.g. used in this complete template example:

```
hello(String person) ::= <<
Hello <person>!
>>
```

7.6 Named Value References

In the previous section, *named value references* were already used in the examples. A value can be referred by name when it is in the scope of the expression.

Automatic to-string conversion applies for all primitive MetaModelica types (`String, Integer, Real, Boolean`) and for all generic types of primitive types except of `tuple` types, i.e., `list`, `Option` and `Array`. Examples of automatic to-string conversion:

```
templ1(Integer i, Real r, Boolean b)::=
  'Is <b> that <i> = <r>?'

templ2(list<String> names,
       Option<Integer> optId) ::=
  'allNames<optId> = "<names>";'

templ3(String hello) ::= hello
```

`Option` typed values are output conditionally when they hold a value (the value of `SOME`). List types are output in sequence, i.e., effectively the concatenation of the string equivalents of their elements. These to-string conversion rules are elaborated recursively, that is, also a value of type `list<Option<Integer>>` is automatically to-string convertible.

For `list` and `Array` typed values a separator option can be specified right after the value name, like:

```
nameList(list<String> names) ::=
                  'Names are: <names ", ">.'
```

There are more possible options for multi-valued expressions tailored for structuring the output text properly, see Section 7.11.

7.7 Template Calls

Templates can be *called* from other templates. Recursive calling of templates is allowed, too. The syntax is:

templ-name(arg_1, arg_2, …, arg_n)

where `templ-name` is the name of the called template, *arg*$_i$ are actual parameter expressions, and *n* can be 0 or more. Parameters are strongly typed with automatic to-string conversion when applicable. Usually actual parameters are named value references or other template expressions, but literal constants of `Integer`, `Real` and `Boolean` types can also be used (it is a sort of restriction to be able to create only non-structured constant values). Some examples:

```
sayN(String msg, Integer n) ::=
  'Say "<msg>" <n> times.'

say3(String msg) ::= sayN(msg,3)

whatToSay(String word) ::= <<
What to say?
<say3('Susan is <word>!')>
>>
```

7.8 Iterative Map Template Expressions

The *map template expression* is used to iterate over lists (or a scalar). It is conceptually similar to map functions heavily used in functional languages instead of imperative constructs like for-loops.

There are several possible design choices of syntax for this construct. The current choice (inspired by ST) is to use the colon (`:`) as a map operator:

value-expr **of** *elem-pattern* : *templ-expr*

However, `:` can be a bit cryptic and hard to see embedded in code. Other possibilities could be:

```
map(templ-expr, value-expr, elem-pattern)
```

or the Modelica iterative expression (without pattern):

```
templ-expr(x) for x in value-expr
```

The above means: "Map element(s) of the *value-expr* that matches *elem-pattern* using *templ-expr*; Concatenate results if they are multiple."

The redesigned part compared to ST is the **`of`** keyword that is a shortcut of meaning close to "*consists* ***of*** *element(s) like*". The colon "`:`" then creates a new nested scope for template invocation in an element-wise manner. If the *value-expr* is a scalar value it is treated as a single element.

Value-expr is usually a named value reference, but can be an external or intrinsic function call (see Section **Error! Reference source not found.**).

Elem-pattern is most often a single name value binding or a tuple pattern matching expression, but the same syntax and semantics applies here as for the pattern matching case rules in match-expressions. This, it can work as a filter for elements to be mapped, see the next section for more about patterns.

Templ-expr can be any valid template expression. For example,

```
gentlemen(list<String> names) ::= <<
Hello<names of name: ', Mr. <name>'>!
>>

pairList(
  list<tuple<String,Integer>> pairs
) ::= <<
Pairs:<pairs of (s,i):'(<i>,<s>)'", ">.
>>
```

where `name` binds each element value of `names` list to be used in the provided textual template after the "**:**" and the `pairList` template binds the two values of the `pairs` input parameter to map them with the textual template. The `", "` is the optional separator string that is used as a delimiter when concatenating the mapping results.

Map expressions can be used also for scalar typed values, most useful for `tuple` types, like

```
firstSI(tuple<String,Integer> pair) ::=
  pair of (s,_) : s
```

The implicit variable `it` is always implied after the "**:**", semantically as the "**`of`** `...`" clause is always rewritten to "**`of`** `it` **`as`** `elem-pattern`". The "**`of`** `...`" clause is then optional with the meaning "**`of`** `it`". Combining this with implicit referencing of `it` when omitting the parameter on a single parameter template call, the intention of the map expression is most succinct, for example:

```
intDecl(String varName) ::=
  'int <varName>;'

intDecls(list<String> varNames) ::= <<
/* integer local variables */
<varNames : intDef() \n>
>>
```

However, when the mapping template has more parameters, all of them must be explicit; while the implicit value can still be referred by the name '**`it`**'.

And again, we have specified an optional separator to new line in the form of unquoted escaped string `\n`. There are more options that are useful in various formatting scenarios, see Section 7.11 for their special syntax and semantics.

7.9 Match-Expressions

For example, consider the union type `Statement` from the type definition in Section 7.3. To read record values for an input value of the type in MetaModelica we might use a match-expression with positional pattern matching case rules like these (only fragments):

```
function statement
  input Statement inStatement;
...
match inStatement
  local
   Exp lhs, rhs;
   list<Statement> stmts;
case ASSIGN(lhs,rhs)
//lhs and rhs bound to respective values
   then ...;
case WHILE(stmts) equation
//stmts has value of statements here
...
```

Templates are supposed only to have *read* access to data structure (e.g. AST) attributes, making the usual local variable definitions unnecessary

The *match-expression* in the Susan language has the syntax:

```
[match value-expr]opt
  case pattern-expr  then template-expr
  case pattern-expr2 then template-expr2
  ...
```

`Value-expr` is usually a named value reference, but can also be an external or intrinsic function call.

The **`match`**`...` clause is optional, assumed to have the form **`match`** `it` when omitted. Each **`case`** opens a scope after **`then`**, with the record field names of the matched record node visible, e.g. `lhs` and `rhs` in the `ASSIGN` node. The `statement` function as a template:

```
statement(Statement stmt) ::=
 match stmt
  case ASSIGN  then
//lhs and rhs visible in the immediate scope
  …
  case w as WHILE then
  //w.statements visible while w not hidden
...
```

7.10 Conditional Expressions

Conditional expressions (or *if-expressions*) can be considered as syntactic variants of match-expressions. The general syntax is:

```
if cond-expr then template-expr
[else template-expr2]opt
```

where **`if`** *`cond-expr`* can be only have two forms:

```
if [not]opt value-expr …
if value-expr is [not]opt pattern-expr …
```

The first form is intended to query values for their zero-like values, enumerated by type:

```
Boolean              false/true
Integer and Real     0/non-0,
String, list and Array    empty/non-empty
Option               NONE/SOME.
```

The second form uses pattern matching and is, for the case without **`not`**, semantically equivalent to:

```
match value-expr
 case pattern-expr  then template-expr
 case _ /*the rest*/then template-expr2
```

For the case with **`not`**, the expressions after **`then`** are switched (unlike the patterns).

For all forms, when the **`else`** branch is not specified it is assumed to be the empty string.

7.11 Automatic Indentation and Options

Well indented documents and code are much easier to read than non-indented. Indentation levels are automatically and recursively tracked. For example,

```
lines2(list<String> lines) ::= <<
  <lines \n>
>>

lines4(list<String> lines) ::= <<
  <lines2(lines)>
>>
```

Giving a list of strings to the `lines2` template, all the strings are concatenated using new line as delimiter and indented by 2 spaces. Giving the same list to `lines4` template, the indentation becomes 4 spaces.

There is a set of *(template) expression options* that can be specified with following syntax:

```
<templ-expr sep; opt1=val1; opt2; ...>
```

We have already used the *separator* option in its short form. A separator option is applicable for all multi-result expressions (e.g., map expressions). It has also a *named option* equivalent (a fragment):

```
<lines; separator=\n>
```

Expression options can be specified only in the direct lexical context of **`<...>`** or (…). The latter is intended for expressions that occur in the top-most or a nested lexical context (e.g., after the **`then`** keyword), for example (fragments),

```
... ::= (lines \n)
... then (exps : exp(); separator=";\n")
```

In the above examples, the indentation is also applied after any new line embedded in the strings. Sometimes such behavior is not desirable.

There are four indentation controlling options: *anchor*, *absIndent*, *relIndent* and *indent*. They set integer values defaulting to 0 when unspecified. While active, their semantics says: ”apply my behavior when outputting the first non-space character *after* a new line*e*”. Specifically, *anchor* means ”indent relative to where I start*ed*“, *absIndent* means ”indent absolutely“, *relIndent* means ”indent relative to actual inden*t*“ and *indent* means ”break the rule, put my indent immediately and behave like relIndent”.

There are even more options, in addition to *separator* , where the most notable are *wrap* and *align*.

Combining indentation controlling options with wrapping/aligning options, most formatting scenarios can be addressed.

7.12 The While Example Using Susan

We have now prepared the ground for the complete while-loop example. Given these templates

```
statement(Statement stmt) ::=
 match stmt
  case ASSIGN then <<
<exp(lhs)> = <exp(rhs)>;
  >>
  case WHILE  then <<
while(<exp(condition)>) {
  <statements : statement() \n>
}
  >>

exp(Exp e1) ::=
 match e1
  case ICONST   then value
  case VARIABLE then name
  case BINARY   then
   '(<exp(lhs)> <oper(op)> <exp(rhs)>)'

oper(Operator) ::=
  case PLUS then "+"
  case TIMES then "*"
  case LESS then "<"
```

The `oper()` template uses the short form of the match. Being fed this ASTvalue of type `Statement`:

```
WHILE(
 BINARY( VARIABLE("x"),LESS(),ICONST(20)),
{ASSIGN( VARIABLE("x"),
   BINARY( VARIABLE("x"),
   PLUS(), BINARY( VARIABLE("y"),
           TIMES(),ICONST(2))))})
```

the `statement()` template will generate this text

```
while((x < 20)) {
  x = (x + (y * 2));
}
```

7.13 The Susan Compiler

The Susan compiler translates source code in the Susan language into the MetaModelica language. The first prototype of the compiler was fully implemented in MetaModelica. Then, its own code generator was re-implemented using the Susan language.

8 Applications in Code Generation

The current code generation in OpenModelica 1.4.5 is hand implemented and transforms the DAELow AST into a list of strings which later is concatenated into the generated code. The only target language is C.

The new template-based code generation brings several advantages:

- Separation of concerns – developing a new code generator is much simpler.
- New target languages (e.g., generating Java code) can be added more easily to the code generator.
- Also end-users (modelers) can develop code generators, specified by template-based models, that can be dynamically linked into the compiler.

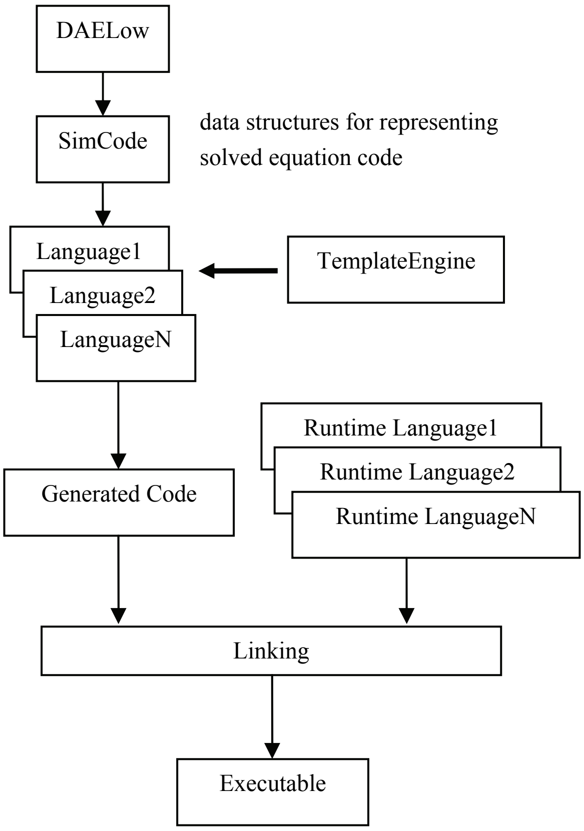


Figure 2. Usage of template-based code generators for producing target code in different languages.

9 Related Work

Template engines and languages can be used to generate code, documentation or web pages. Most of them claim to use a Model-View-Controller concept (MVC), even though many violate some of the MVC principles. Many tools are based on Java and thus need to be fed XML data or Java classes.

9.1 Ctemplate from Google

Ctemplate [11] is a C++-based template engine that is less complex than most of the Java-based alternatives. The input is a basic dictionary structure. An example of a ctemplate template:

```
Hello {{NAME}},
You have just won ${{VALUE}} !
{{#IN_CA}}${{TAXED_VALUE}} after taxes.{{/
IN_CA}}
```

The code to use this template is rather complex [22].

9.2 Apache Velocity

Velocity [2] is a Java-based tool that generates output using templates. It is mainly used to serve webpages, SQL and PostScript but can also be used for code generation].

The data consists of Java classes that are fed to the engine. Velocity applies the classes to the template using directives like if-else, foreach (for iterable classes like lists) and can set/get its own variables inside the template. An example Velocity template:

```
class Structure [
#foreach( $var in $list )
  public $var.type.name $var.name ;
#end
}
```

9.3 StringTemplate

StringTemplate [18] with the ST language [17] is a template engine tightly integrated with ANTLR [1], including language bindings for Java, C++, and Python. It has been designed [16] to strictly enforce the MVC concept, and is mostly used for generation of web pages.

According to the main author, Terrence Parr [17] only four basic template constructs are needed:

- Attribute reference, `$name$ or <name>`.
- Conditional template inclusion based on presence/absence of an attribute, $if(flag)$text$endif$.
- Recursive template references.
- Template application to a multi-valued attribute (e.g. `names`) similar to lambda functions and LISPs map operator, `$names: templToApply()$`.

The template language, called ST, is actually a functional language. A template example follows:

```
("Hello, $name$\n" +
   "While you were gone $names;
      separator=\", \"$
      called you.",
    DefaultTemplateLexer.class);
```

Use of the template:

```
import org.antlr.stringtemplate.*;
import org.antlr.stringtemplate.language.*;

class sttest {
public static void main (String [] args) {
  StringTemplate hello= new StringTemplate
  ("Hello, $name$\n" +
   "While you were gone $names;
      separator=\", \"$
      called you.",
    DefaultTemplateLexer.class);
  hello.setAttribute("name","General");
  String [] names = {"Alpha", "Bravo",
                     "Charlie" };
  hello.setAttribute("names", names);
  System.out.println(hello.toString() );
} }
```

Output:

```
Hello, General
While you were gone Alpha, Bravo, Charlie
called you.
```

9.4 Structured Representation Approaches

Invasive software composition [3] is somewhat related to template languages. Programs are decorated with hooks that can be replaced during composition. Operations are typically on abstract syntax instead of strings.

10 Conclusions

The uses, needs, and requirements of text generation template language for Modelica have been discussed.

Several template language designs and some usage examples and experience have been presented, both C code generation and Modelica model generation. There are difficult tradeoffs between different language design options regarding properties like generality, conciseness, consistency, efficiency, etc.

Three Modelica-related designs have been created. The first presented design is embedded in MetaModelica has not yet been implemented due to lack of resources. The second is a simple interpreted template language (as an external DSL) which was implemented and tried early on. The third (Susan) is a recently implemented compiled template language. It is efficient since it is compiled to MetaModelica. The language has several nice features and has already been used for its compilation to MetaModelica. However, some design remains and there is still discussion among the authors regarding the right syntax and semantics in some cases. The language looks very promising as a powerful tool for specifying code generation and similar tasks.

10.1 Future Work

The next mile-stone is to re-implement the code generator of the OpenModelica compiler using the Susan language, for at least two target languages (C/C++, C# and perhaps Java). This will further refine the design and implementation. Moreover, good tooling is important also for template languages. As a start, keyword coloring will soon be available in the OpenModelica MDT (Modelica Development Tooling) environment.

11 Acknowledgements

This work has been supported by Vinnova in the ITEA2 OPENPROD project, by the Swedish Research Council (VR), by the Czech National Research Programme, project No.2C06031, "e-Golem", and by Creative Connections s.r.o., Czech Republic. The Open Source Modelica Consortium supports the OpenModelica work. Peter Aronsson from MathCore Engineering AB gave useful feedback during the design.

References

[1] ANTLR. http://www.antlr.org. Access Nov 2007.

[2] Apache Software Foundation. Velocity Users Guide, 2008.: http://velocity.apache.org/engine/releases/velocity-1.6.1/user-guide.html. Jan 2009.

[3] Uwe Assmann. *Invasive Software Composition*. ISBN 3540443851, 9783540443858, 334 pages. Springer Verlag, 2003.

[4] Martin Fowler: *Domain Specific Language* http://www.martinfowler.com/bliki/ DomainSpecificLanguage.html.

[5] Martin Fowler. *Domain Specific Languages* http://martinfowler.com/dslwip/

[6] Peter Fritzson. *Towards a Distributed Programming Environment based on Incremental Compilation*. PhD thesis no 109, Linköping University, April 13, 1984.

[7] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. *Simulation News Europe*, 44/45, Dec 2005. http://www.openmodelica.org

[8] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, 940 pages, Wiley-IEEE Press, 2004.

[9] Peter Fritzson, Adrian Pop, and Peter Aronsson. Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica. In *Proc. of the 4th International Modelica Conference*, Hamburg, Germany, March 7-8, 2005.

[10] Peter Fritzson, Adrian Pop, Kristoffer Norling, and Mikael Blom. Comment- and Indentation Preserving Refactoring and Unparsing for Modelica. In *Proc. 6th Int. Modelica Conf. (Modelica'2008)*, Bielefeld, Germany, March.3-4, 2008.

[11] Google. ctemplate, 2008. http://code.google.com /p/google-ctemplate/. Accessed 2009.

[12] Kenneth C. Louden. *Programming Languages, Principles and Practice*. ISBN 0-534-95341-7, Thomson Brooks/Cole, 2003.

[13] Modelica Association. *The Modelica Language Specification Version 3.0*, September 2007. http://www.modelica.org.

[14] Martin Mikelsons. Prettyprinting in an interactive programming environment. In *Proc. of ACM SIGPLAN SIGOA symposium on Text manipulatio*n. Portland, Oregon, 1981.

[15] Eclipse website. http://www.eclipse.org. Referenced Nov 2007.

[16] Terence Parr. Enforcing Strict Model-View Separation in Template Engines. http://www. stringtemplate .org,. May 2004. Accessed May 2009.

[17] Terence Parr. [DRAFT] A Functional Language For Generating Structured Text. http://www.stringtemplate.org. May 2006. Accessed May 2009.

[18] Terence Parr. StringTemplate documentation. http://www.stringtemplate.org. Access May 2009.

[19] Peter Fritzson, Adrian Pop, and Peter Aronsson. Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica. In *Proceedings of the 4th International Modelica Conference*, Hamburg, , March 7-8, 2005.

[20] Adrian Pop, Peter Fritzson, Andreas Remar, Elmir Jagudin, and David Akhvlediani. OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging. In *Proc 5th International Modelica Conf. (Modelica'2006)*, Vienna, Austria, Sept. 4-5, 2006.

[21] Adrian Pop. *Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages*. www.ep.liu.se. PhD Thesis No. 1183, June 5, 2008.

[22] Martin Sjölund. *Bidirectional External Function Interface Between Modelica/MetaModelica and Java*. Master Thesis. Linköping Univ, Aug. 2009.

[23] Philip Wadler. A Prettier Printer. *Journal of Functional Programming*,1998, pp 223-244. Draft version :homepages.inf.ed.ac.uk/wadler/papers /prettier/prettier.pdf Implementation PPrint by Daan Leijen

Higher-Order Non-Causal Modelling and Simulation of Structurally Dynamic Systems

George Giorgidze Henrik Nilsson

Functional Programming Laboratory
School of Computer Science
University of Nottingham
United Kingdom

{ggg,nhn}@cs.nott.ac.uk

Abstract

This paper explores a novel approach to the implementation of non-causal modelling and simulation languages supporting highly structurally dynamic systems. One reason the support for structural dynamics is limited in present mainstream non-causal modelling and simulation languages is that they are designed and implemented on the assumption that symbolic processing of models and ultimately compilation of simulation code takes place prior to simulation. We seek to lift that restriction, without sacrificing efficiency, by exploiting just-in-time (JIT) compilation to allow new simulation code, reflecting structural changes, to be generated as the simulation progresses. Our work is carried out in a framework called Functional Hybrid Modelling that supports *higher-order modelling*, as higher-order modelling lends itself naturally to expressing structural dynamism. However, the central ideas of the paper should be of general interest in the area of structural dynamism. The paper provides an in-depth description of the implementation techniques we have developed as well as a performance evaluation.

Keywords: Non-causal Modelling and Simulation, Structurally Dynamic Systems, Functional Programming, Just-In-Time Compilation, Symbolic/Numerical Methods

1 Introduction

When developing dynamic models of physical systems, it is often desirable to model major changes in system behaviour by changing the *differential algebraic equations* (DAEs) that describe the dynamics of the system. These major changes can be due to the modelled system itself exhibiting structural changes, due to a need to change to simplified models of parts of a system for periods of time, and so on [12]. Models whose equational description change over time are called *structurally dynamic*, and each structural configuration is known as a *mode* of operation. Structurally dynamic systems are an example of the more general notion of *hybrid* systems, systems that exhibit both continuous and discrete behaviour.

Unfortunately, the support offered by current modelling languages for expressing structurally dynamic systems (as well as hybrid systems in general) is somewhat limited [13, 20, 22]. This is true in particular for *non-causal* modelling languages, which is the class of modelling languages with which we are concerned in this paper. There are a number of reasons for this limited support, many of them related to the technical difficulties of simulating structurally dynamic models, such as identifying suitable state variables for different modes and proper transfer of the state between modes [12, 13].

However, there is also one less fundamental reason, namely the common assumption that most or all processing to put a model into a form suitable for simulation will take place *prior* to simulation [18, 21]. By enforcing this assumption in the *design* of a modelling language, its *implementation* can be simplified as there is no need for simulation-time support for handling structural changes. For instance, a compiler can typically generate static simulation code (often just sequences of assignment statements) with little or no need for dynamic memory management. This results in good performance. But the limitations are also obvious: for example, the number of modes must be modest as, in general, separate code must be generated for each mode. This rules out supporting *highly* structurally dynamic systems: systems where the number of modes is too large to make explicit enumeration feasible, or even a priori unbounded.

There are a number of efforts to design and implement modelling and simulation languages with improved support for structural dynamics. Examples include HYBRSIM [14], MOSILAB [19], and Sol [22]. Of these, Sol is likely the most flexible. However, thus far, implementations have either been interpreted (HYBRSIM and Sol), or the language has been restricted so as to limit the number of modes to make it feasible to compile code for all modes prior to simulation (MOSILAB).

DOI: 10.3384/ecp09430137

This paper contributes towards the design and implementation of modelling and simulation languages by demonstrating support *both* for modelling of highly structurally dynamic systems *and* for compilation of simulation code for efficiency. We present a prototype implementation of a non-causal language allowing arbitrary structural changes during simulation. Central to this capability, and the focus of this paper, is that the equations that describe the current operating mode are compiled into simulation code at each structural change using a code-generation framework supporting just-in-time (JIT) compilation: the Low Level Virtual Machine (LLVM) [7]. We describe the compilation process as well as the necessary supporting run-time machinery, and we provide small but detailed benchmarks that demonstrate that the generated simulation code is fairly efficient and that the overhead of the processing of structural changes is not unreasonable, particularly not for an early prototype. As far as we know, this is the first time JIT compilation has been used for dynamic compilation of simulation code to enable efficient simulation of highly structurally dynamic models in the context of non-causal modelling. The implementation is available on-line[1] under the open source BSD license.

This work has been carried out in the context of our research on Functional Hybrid Modelling (FHM) [17, 18], a novel approach to purely declarative languages for non-causal, hybrid modelling and simulation. A central aspect of FHM is that models are *first-class entities*. This means they can be manipulated programmatically (past as arguments to functions, returned as results of functions, etc.), just like any other type of value such as integers or Booleans. This is called *higher-order* modelling [3].[2]

Higher-order modelling, with just a minimum of additional language constructs, lends itself very well to *expressing* highly structurally dynamic systems: all that is needed is the means to allow new model fragments to be computed not only before simulation starts, but also *during* simulation, at events, and to be integrated into the simulated system at those points. This is the approach taken by FHM. Indeed, the ease by which higher-order modelling can express structural dynamics was partly what motivated our research into FHM in the first place [17].

However, at its core, this paper is concerned with techniques for *implementing* languages supporting modelling of highly structurally dynamic systems, and we would thus like to emphasise that the ideas and results presented in this paper are not limited to the setting of FHM, but are, on the whole, applicable to modelling and simulation languages supporting structurally dynamic systems in general. We would also like to reiterate that the focus of this paper is squarely on the mechanics of integrating dynamic code generation into the implementation of a non-causal modelling language: many of the other technical problems briefly mentioned above remain to be solved.

The rest of this paper is organised as follows. Section 2 provides background on FHM and LLVM. FHM is introduced by means of an example that is also used in the remainder of the paper, so we recommend that all readers, even if already familiar with FHM, take at least a quick look at this section. Section 3 explains how our language is implemented, with a particular emphasis on the methods we use to support highly structurally dynamic systems. The performance of our prototype implementation is evaluated in Section 4. Related work is discussed in Section 5. Finally, Section 6 considers future work and conclusions are given in Section 7.

2 Background

2.1 Functional Hybrid Modelling

In the following, we give a brief overview of Functional Hybrid Modelling (FHM) to explain the notation used in the rest of the paper and to provide some general background. In particular, we introduce Hydra, the FHM language we are currently working on. For details, see earlier papers on FHM [17, 18]. However, we again remind the reader that in the present context, FHM and Hydra should mostly be seen as a particular syntax that is convenient for expressing structurally dynamic systems; neither is central to the contributions of this paper.

With FHM, we seek to develop a small but expressive, purely declarative, non-causal and hybrid modelling language. A key motivation is to develop simple and clear semantical foundations for this class of modelling languages, with a view to paving the way for improvements such as more flexible support for hybrid modelling and type systems exploiting domain knowledge in new ways [15].

Our hypothesis is that the aims of FHM can be realised by identifying the core semantical concepts of non-causal and hybrid modelling and embedding these as first-class entities in a declarative host language. This achieves a separation of concerns that we believe is both sound and expedient, as it highlights similarities and differences with other classes of languages and allows us to focus our research on what is specific to non-causal, hybrid modelling languages.

2.2 Signal Relations

FHM was inspired by Functional Reactive Programming (FRP) [4] and then in particular Yampa [16]. FRP is an approach to reactive programming that in many ways can be seen as *causal* modelling. It is realised by enriching a functional language with a first-class notion of functions operating on signals, *signal functions*, where a signal is a time-varying value. In other words, signal functions are very much like "blocks" in a causal modelling language like Simulink, except having first-class status, which means that ordinary functions can operate on signal functions achieving what in many modelling languages would be considered meta-modelling capabilities. In particular, new signal functions can be computed as a system is running and

[1] `http://cs.nott.ac.uk/~ggg/`

[2] As it is reminiscent of *higher-order functions*. A function is higher-order if some of its arguments or result is function-valued. Not to be confused with other meanings of *higher-order*.

then "switched in" to become part of that system, allowing highly structurally dynamic systems to be modelled [16].

What distinguishes non-causal from causal modelling languages is that models are described in terms of undirected equations over time-varying entities. In other words, *relations* on signals as opposed to functions on signals. The essence of FHM is thus the addition of first-class relations on signals, *signal relations*, to a functional language.

There are consequently *two levels* to FHM: the *functional level*, concerned with defining ordinary functions operating on time-invariant values, and the *signal level*, concerned with the definition of relations between signals (time-varying values), and, *indirectly*, the definition of the signals themselves as solutions satisfying the constraints imposed by the signal relations. The definitions at the signal level may freely refer to entities defined at the functional level, which is crucial in the following. Also, defined signal relations are ordinary time-invariant values at the functional level (first-class entities). Signals, on the other hand, are *not* first-class entities at the functional level. However, as discussed in the following, *instantaneous values* of signals can be propagated back to the functional level allowing e.g. the future system structure to depend on signal values at discrete points in time.

For those familiar with languages like Modelica, note that a signal relation has many similarities to a class: fundamentally a signal relation is just an encapsulated set of equations that constrain a number of signal variables. Also like a class, signal relations can be *instantiated*, creating copies of the encapsulated equations imposing their constraints on some variables in the current context. This is called signal relation *application*. Unlike Modelica, there is no class hierarchy and thus no inheritance, so the *only* way to reuse equations is by signal relation application. On the other hand, signal relations are first class entities, giving a lot of additional expressive power, whereas classes are not.

2.3 Hydra by Example: The Breaking Pendulum

Let us illustrate the key aspects of FHM and Hydra through an example. As this paper is concerned with structural dynamism, we chose a variation of a breaking pendulum [11, pp. 31–33] as it is representative yet small. The pendulum is modelled as a point mass m at the end of a rigid, massless rod, subject to gravity $m\vec{g}$; see Figure 1. Additionally, the rod could break at some point in time, causing the mass to fall freely. The breaking of the pendulum causes a significant change in the structure of the system, so much that languages like Modelica do not support non-causal simulation of a pendulum that breaks during simulation [18].

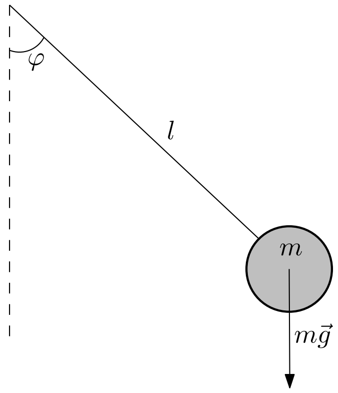


Figure 1: A pendulum subject to gravity.

We start by modelling a free-falling body in Hydra. Let us begin by defining some type abbreviations and constants for convenience. Hydra is implemented as an *embedding* in Haskell [5], the *host language*. The functional level discussed above is thus provided by Haskell, saving us the work of implementing a functional language from scratch. Type abbreviations and constants are ordinary time-invariant definitions and defined at the functional level; i.e., directly in Haskell. We define the position and velocity to be pairs of reals, and the state of a body in motion to be given by its position and velocity. An entity of type *Body* is thus an aggregate of four real-valued fields. We also define the gravitational acceleration g to a suitable value:

$$\begin{aligned}
&\mathbf{type}\ \mathit{Position} = (\mathit{Double}, \mathit{Double})\\
&\mathbf{type}\ \mathit{Velocity}\ = (\mathit{Double}, \mathit{Double})\\
&\mathbf{type}\ \mathit{Body}\quad\ \ = (\mathit{Position}, \mathit{Velocity})\\
&g :: \mathit{Double}\\
&g = 9.81
\end{aligned}$$

A model of a free-falling body is a signal relation parametrised on the initial state of the body. In Hydra, exploiting that signal-relations are first-class entities, a parametrised signal relation is just a function that *computes* the appropriate signal relation from the parameters and returns this computed signal relation as its result. This is thus an example of higher-order modelling. In our case, we obtain a function that computes a signal relation constraining a variable of type *Body* (or rather, its four real-valued fields) given an initial value of the state, also of type *Body* (→ is the type former for functions in Haskell):

$$\mathit{freeFall} :: \mathit{Body} \rightarrow \mathit{SR}\ \mathit{Body}$$

The function *freeFall* is defined by *pattern matching* on its one argument of type *Body*, binding (functional-level) variables $x0$, $y0$, $vx0$, and $vy0$ to the initial position and velocity:

$$\mathit{freeFall}\ ((x0, y0), (vx0, vy0)) = \ldots$$

The bound variables scope over the body of the function, here indicated by an ellipsis.

Note that the Haskell syntax for function application is simply juxtapositioning; e.g., f 1 denotes the application of the function f to the argument 1, g 2 3 the application of the function g to the two arguments 2 and 3, and so on. Parentheses are not part of the syntax of function application as such, although they are used for grouping and, as here, to denote tuples. The syntax of function definition mirrors the syntax of function application. The function *freeFall* is thus syntactically a function of a single argument: a pair of pairs of reals.

The body of *freeFall* defines the parametrised signal relation. To do this, we need to work at the signal level. The Hydra embedding is achieved through *quasiquoting*, a Haskell extension provided by the Glasgow Haskell Compiler (GHC) [10]. Quasiquoting allows custom syntax to be realised by arranging for source text delimited by quasiquotes to be passed to a custom-written function that parses the text and translates it into abstract syntax of the *host* language. In GHC, this is done prior to type checking. Quasiquoting is thus rather similar to what can be achieved through a pre-processor, except more tightly integrated with the compiler and less work to implement. The Hydra opening quasiquote is $[\$hydra|$, and the closing quote is $|]$. Between them, we have signal-level definitions expressed in our custom syntax. The complete definition of *freeFall* is as follows:

```
freeFall ((x0, y0), (vx0, vy0)) = [$hydra|
   sigrel ((x, y), (vx, vy)) where
      init (x, y) = ($x0$, $y0$)
      init (vx, vy) = ($vx0$, $vy0$)
      (der x, der y) = (vx, vy)
      (der vx, der vy) = (0, − $ g$)
|]
```

The keyword **sigrel** starts the definition of a signal relation. It is followed by a pattern that introduces *signal variables* giving local names to the signals that are going to be constrained by the signal relation. This pattern thus specifies the *interface* of a signal relation, similarly to how function arguments specify the interface of a function. After the keyword **where** follow the equations that define the relation. These equations may introduce additional signal variables as needed. Equations marked by the keyword **init** are initialisation equations used to specify initial conditions.

To refer to *functional-level* entities at the signal level, inside the quasiquotes, such references need to be *antiquoted* by enclosing them between \$-signs. Arbitrary Haskell expressions, using any *functional-level* variable in scope outside the quasiquoted block, are allowed between the antiquotes. However, none of the *signal-level* variables are in scope in antiquoted code. The abstract syntax for each antiquoted Haskell expressions is then spliced in at the point of the antiquote. For example, note how the functional-level parameters defining the initial position and velocity ($x0$, $y0$, $vx0$, $vy0$) are referenced in the initialisation equations. A signal relation may thus depend on functional-level values, thus making it parametrised. As we will see, these values can be *any* kind of functional-level values, including signal relations, thus achieving higher-order modelling without further ado.

Of course, the quasiquotes and antiquotes are just an artifact of our specific approach to implementing Hydra. A less "noisy" syntax would certainly be possible (with some implementation effort). However, the explicit quoting makes the distinction between the functional level and the signal level manifest, which is helpful at least for explanatory purposes.

Readers who are familiar with Modelica might find it illuminating to compare the Hydra model above with a Modelica version. To facilitate comparison, we have kept the Modelica version as close as possible to the Hydra version, rather than trying to provide the "most natural" Modelica model:

```
model FreeFall
  parameter Real x0, y0, vx0, vy0;
  Real x(start=x0), y(start=y0);
  Real vx(start=vx0), vy(start=vy0);
equation
  der(x) = vx;
  der(y) = vy;
  der(vx) = 0;
  der(vy) = -g;
end FreeFall;
```

Here, g is assumed to be a constant defined elsewhere. Note how Modelica parameters, that denote entities that remain constant during continuous integration, correspond to functional-level variables in Hydra, while Modelica variables that change during continuous integration correspond to signal-level Hydra variables. A difference, though, is that Modelica parameters can only change at the very start of a simulation, while Hydra functional-level variables can change at every event occurrence.

In a completely analogous way to the free-falling mass, we define a parametrised signal relation that models the pendulum in its unbroken mode. The parameters are the length of the rod l and the initial angle of deviation $phi0$:

```
pendulum :: Double → Double → SR Body
pendulum l phi0 = [$hydra|
   sigrel ((x, y), (vx, vy)) where
      init phi = $phi0 $
      init der phi = 0
      init vx = 0
      init vy = 0
      (x, y) = ($l $ ∗sin phi, − $ l $ ∗cos phi)
      (vx, vy) = (der x, der y)
      der (der phi) + ($g $ / $ l$) ∗ sin phi = 0
|]
```

Again, for comparison, we give a Modelica version:

```
model Pendulum
  parameter Real l, phi0;
  Real x, y, vx, vy;
  Real phi(start=phi0), phid;
equation
  x = l * sin(phi);
  y = -l * cos(phi);
  vx = der(x);
  vy = der(y);
  phid = der(phi);
  der(phid) + (g/l) * sin(phi) = 0;
end Pendulum;
```

We proceed to extend the above definition into a signal relation that also provides an *event signal* defining when the pendulum is to break. An event signal is a signal that is only defined at discrete points in time, *events*. In this case, an event is generated at a specified point in time, and the value of the event signal is the state (position and velocity) of the pendulum at that point. Note how the new signal relation is defined by extending the previous one. The parametrised signal relation $\mathit{pendulum}$ is applied to the length of the pendulum and the initial angle of deviation at the *functional level* (within antiquotes), thus computing a signal relation. This relation is then applied at the signal level, through *signal relation application* (the operator $\diamond$), instantiating the equations of $\mathit{pendulum}$ in the context of $\mathit{breakingPendulum}$ and thus effectively extending the definition of $\mathit{pendulum}$:

$$\begin{array}{l}
\mathit{breakingPendulum} :: \mathit{Double} \to \mathit{Double} \to \mathit{Double} \\
\qquad\qquad\qquad\qquad\quad \to \mathit{SR}\ (\mathit{Body}, E\ \mathit{Body}) \\
\mathit{breakingPendulum}\ t\ l\ \mathit{phi0} = [\$\mathit{hydra}| \\
\quad \mathbf{sigrel}\ (((x, y), (vx, vy)), \\
\qquad\qquad \mathit{event}\ e@((\_, \_), (\_, \_)))\ \mathbf{where} \\
\qquad \$\,\mathit{pendulum}\ l\ \mathit{phi0}\ \$\ \diamond\ ((x, y), (vx, vy)) \\
\qquad \mathit{event}\ e = ((x, y), (vx, vy))\ \mathit{when}\ \mathit{time} = \$t\ \$ \\
|]
\end{array}$$

The process of unfolding signal relation applications is called *flattening* and is in many ways similar to the transformation of hierarchical models in languages like Modelica into a flat system of equations, a process usually also referred to as flattening. Unfolding signal relation application in Hydra is straightforward: the actual arguments (signal-valued expressions) to the right of the signal relation application operator $\diamond$ are simply substituted for the corresponding formal arguments (signal variables) in the body of the signal relation to the left of $\diamond$. The only real issue is that *name capture*, accidental clashes of variable names[3], must be avoided by an appropriate renaming strategy.

Finally, we simulate the actual breaking of the pendulum by switching from the pendulum equations to the free fall equations at the point where the event is generated. This is accomplished by the switch-combinator[4] with the following type signature:

$$\mathit{switch} :: \mathit{SR}\ (a, E\ b) \to (b \to \mathit{SR}\ a) \to \mathit{SR}\ a$$

The as and bs in the above type signature are *polymorphic type variables* that can be instantiated to *any* specific type. For example, a would be a pair of reals for a binary signal relation.

The switch-combinator is another example of higher-order modelling: it takes two signal relations as arguments (one plain signal relation and one parametrised signal relation; i.e. a function returning a signal relation) and combines them into a new signal relation that initially imposes constraints according to the first relation, and after the switch according to the second relation. Again, this is in many ways just syntax that happens to fit well in our FHM setting. One could envision alternative ways of expressing switching from one set of equations to another, such as conditionals where each branch give one set of equations.

In more detail, the switch-combinator expects an event signal output from its first signal relation argument. The first time this event signal is defined, the switch combinator *applies* its second argument (the parameterised signal relation) to the value of the event signal at this point, computing a new signal relation of the same type as the first signal relation argument, and then replacing the equations from the first signal relation with those of the newly computed signal relation:

$$\begin{array}{l}
\mathit{mainSR} :: \mathit{SR}\ \mathit{Body} \\
\mathit{mainSR} = \mathit{switch}\ (\mathit{breakingPendulum}\ 10\ 1\ (\mathit{pi}\ /\ 4)) \\
\qquad\qquad\qquad\qquad \mathit{freeFall}
\end{array}$$

Note that the switching event carries the state of the pendulum at the breaking point as a value of type Body. This value is passed to *freeFall*, resulting in a model of the free-falling mass that is initialised in a way that ensures that the position and velocity of the mass become continuous signals.

2.4 The Low Level Virtual Machine

The Low Level Virtual Machine (LLVM) [7] is a language-independent, portable, optimising, compiler back-end. As its name suggests, it provides a compiler with a virtual machine target and takes care of translating the LLVM code into code for any specific, concrete, architecture supported by the LLVM. There are a number of backends with capabilities similar to LLVM that we could have used instead. However, LLVM is rather typical, so the following discussion, while centred around LLVM, is mostly relevant also for other similar backends.

The LLVM has been very carefully engineered to on the one hand be sufficiently high-level to be truly portable across different architectures, shielding the the compiler from the low-level details of the ultimate target. In that sense, it is a principled alternative to using a language like C as a compiler target for portability. Also, this makes it possible to support generic, target-independent optimisations at the LLVM level, saving the compiler writer from the burden of implementing many standard optimisations from scratch. On the other hand, LLVM has also been engineered to be sufficiently low-level to not get in the way of generating high-performance code, and to not make any assumptions about the source language. Taken together, this means that the LLVM it is an ideal target for compilers for a wide range of different languages.

The LLVM is thus rather unlike what probably is the most common virtual machine, the Java Virtual Machine (JVM). The JVM is very Java-centric, making it a poor fit for any language which isn't similar to Java. Also the JVM is fundamentally a byte-code interpreter, which incurs performance

[3] For example, a local variable in a signal relation body having the same name as a variable in one of the actual arguments.

[4] A *combinator* is a function without free variables. Functions whose main purpose is to combine functions into new functions, are often referred to as combinators to emphasise this purpose.

penalties, even if just-in-time (JIT) compilation often is employed to speed up the interpretation. LLVM code, on the other hand, is designed to be compiled into code for the ultimate target architecture.

That said, the LLVM, like the JVM, does support adding in new code dynamically to running applications. The LLVM achieves this through JIT compilation. An application that needs to invoke code that is generated dynamically is linked with the LLVM JIT compiler. The code generator of the JIT compiler is the same as in LLVM static compiler [9]. The LLVM code generator has been shown to outperform the code generator of GNU Compilers Collection (GCC) in a number of benchmarks, both in speed of code generation and execution of generated code [8].

The JIT compiler can be invoked through the provided API whenever a piece of new code needs compiling. The JIT compiler will return a handle to the generated code. This can then be called, just like any statically compiled function, and equally efficient. It is also possible to dispose of dynamically generated code that is no longer needed. The LLVM JIT capabilities turned out to be a very good fit for supporting structurally dynamic simulation, as will be discussed in the following.

3 Simulation

In this section we describe how simulation is performed in Hydra. The process is illustrated in Figure 2 and is conceptually divided into four stages. In the first stage, a signal relation is flattened and subsequently transformed into a mathematical representation suitable for numerical simulation. In the second stage, this representation is just-in-time compiled into efficient machine code. In the third stage, the compiled code is passed to a numerical solver that simulates the system until the end of simulation or an event occurrence. In the fourth stage, in the case of an event occurrence, the event is analysed, a corresponding new signal relation is computed and the process is repeated from the first stage. In the following, each stage is described in more detail.

3.1 Symbolic Processing

As a first step, all signal variables are renamed to give them distinct names. This is to simplify the process of *flattening*, signal relation application unfolding (see section 2.3). All event variables are also given distinct names to allow the event handler to identify a corresponding event variable in the original unflattened signal relation at the moment of an event occurrence (see section 3.4). Having carried out this preparatory renaming step, all signal relation applications are unfolded until the signal relation is completely flattened.

Further symbolic processing is then performed to transform the flattened signal relation into a form that is suitable for numerical simulation. In particular, derivatives of compound signal expressions are computed symbolically. In the case of higher-order derivatives, extra variables and equations are introduced to ensure that all derivatives in the flattened system are first order. While the numerical solver used in the current implementation handles higher-index systems of equations, it is desirable to perform index reduction symbolically at this stage as well [1, 23]. Hydra does not yet do this, but we intend to implement symbolic index reduction in the future.

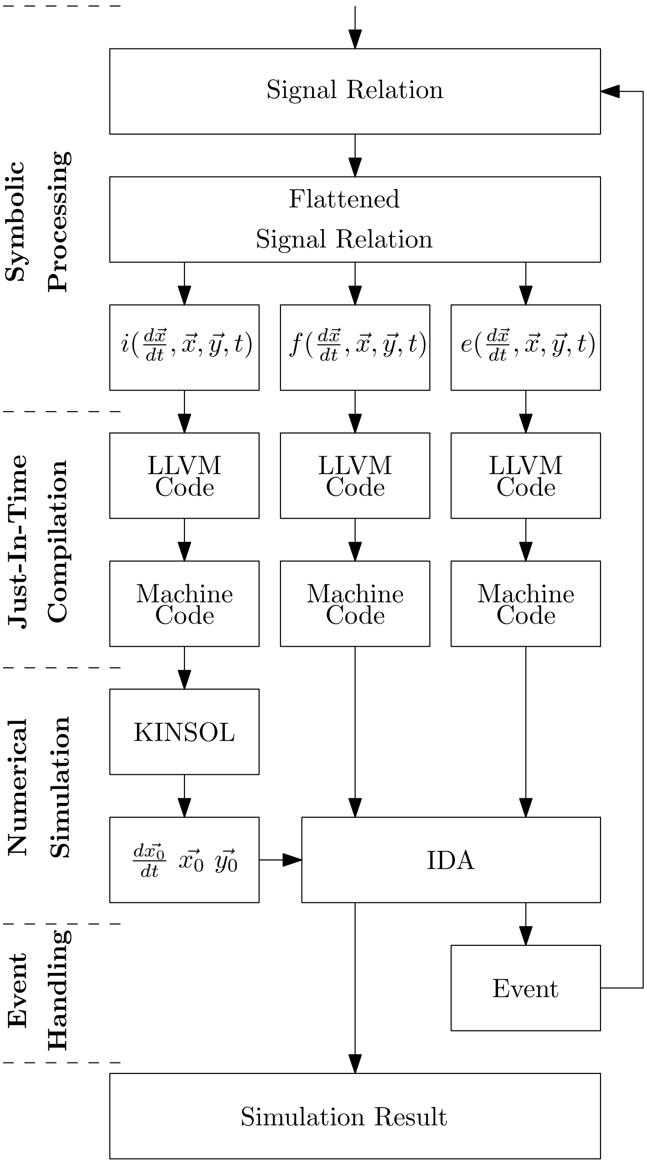

Symbolic Processing
Just-In-Time Compilation
Numerical Simulation
Event Handling
Signal Relation
Flattened Signal Relation
$i(\frac{d\vec{x}}{dt},\vec{x},\vec{y},t)$
$f(\frac{d\vec{x}}{dt},\vec{x},\vec{y},t)$
$e(\frac{d\vec{x}}{dt},\vec{x},\vec{y},t)$
LLVM Code
LLVM Code
LLVM Code
Machine Code
Machine Code
Machine Code
KINSOL
$\frac{d\vec{x_0}}{dt}$ $\vec{x_0}$ $\vec{y_0}$
IDA
Event
Simulation Result

Figure 2: Simulation run-time system of Hydra

Finally, the following equations are generated at the end of the stage of symbolic processing:

$$i(\frac{d\vec{x}}{dt},\vec{x},\vec{y},t) = 0, t = t_0 \tag{1}$$

$$f(\frac{d\vec{x}}{dt},\vec{x},\vec{y},t) = 0 \tag{2}$$

$$e(\frac{d\vec{x}}{dt},\vec{x},\vec{y},t) = 0 \tag{3}$$

Here, $\vec{x}$ is a vector of differential variables, $\vec{y}$ is a vector of algebraic variables, t is time, and t_0 is the starting time for the current set of equations. Equation 1 determines the initial conditions for Equation 2 (i.e., the values of $\frac{d\vec{x}}{dt}$,$\vec{x}$

and $\vec{y}$ at time t_0). Equation 2 is the main DAE of the system that needs to be integrated in time starting from the initial conditions. Equation 3 specifies the event conditions.

3.2 Just-in-time Compilation

The generated equations are implicitly formulated ones: the mathematical representation of non-causal signal relations. In general, it is not possible to transform these implicit equations into explicit ones; i.e., to completely causalise them [1]. Consequently, a system of implicit equations needs to be solved at the start of the simulation of each structural configuration mode and at every integration step. For example, a numerical solution of an implicitly formulated DAE (Equation 2) involves execution of the function f, a number of times (sometimes hundreds or more at each integration step), with varying arguments, until it converges to zero. The number of executions of f depends on various factors including the required precision, the initial guess, the degree of non-linearity of the DAE, etc.

To enable efficient simulation, it is important to compile the functions i, f, and e to a representation that can be executed efficiently. In addition, as Hydra allows the equations to change completely during simulation, it follows that compilation cannot be performed only before the simulation starts, but has to be performed during the simulation as well, whenever the equations change.

The simulation run-time system of Hydra supports JIT machine code generation using the compiler infrastructure provided by LLVM. The functions i, f and e are compiled into LLVM instructions that in turn are compiled by the LLVM JIT compiler into machine code for the processor architecture the simulation is running on. As noted above, the process of JIT compilation is triggered by the simulation run-time system at every discrete event that changes the equations of the system. The generated machine code is then passed to the numerical solver.

3.3 Numerical Simulation

The numerical suite used in the current implementation of Hydra is called SUNDIALS [6]. The following components of SUNDIALS are used:

- KINSOL: nonlinear algebraic equation systems solver
- IDA: differential algebraic equation systems solver

The code for the function i is passed to KINSOL that numerically solves the system and returns initial values (at time t_0) of $\frac{d\vec{x}}{dt}$,$\vec{x}$ and $\vec{y}$. These vectors together with the code for the functions f and e are passed to IDA that proceeds to solve the DAE by numerical integration. This continues until either the simulation is complete or until one of the events defined by the function e occurs. Precise event detection facilities are provided by IDA.

3.4 Event Handling

At the moment of an event occurrence, the numerical simulator terminates and presents the following information to an event handler:

- Name of the event signal variable for which an event occurrence has been detected
- Time t_e of the event occurrence
- Instantaneous values of the signal variables (i.e., values of $\frac{d\vec{x}}{dt}$, $\vec{x}$ and $\vec{y}$ at time t_e)

In the case of the breaking pendulum model, the name of the detected event signal variable is e. In addition, $t_e = 10$, $x \approx 0.21$, $y \approx -0.98$, $vx \approx 2.25$ and $vy \approx 0.49$. Here, x, y, vx and vy are the signal variables that are constrained by the $pendulum$ signal relation.

Next, the event handler traverses the original unflattened signal relation and finds the event value expression (i.e., a signal-level expression) that corresponds to the aforementioned event signal variable. In the case of the breaking pendulum model, the expression is $((x, y), (vx, vy))$. This expression is evaluated by substituting the instantaneous values of the corresponding signals for the variables. In the case of the breaking pendulum model, the computed value is $((0.21, -0.98), (2.25, 0.49))$. However, note that this now is a functional-level expression. This is the only place in Hydra where instantaneous values of signals are passed back to the functional level.

As a final step, the event handler applies the second argument of the switch combinator (i.e., the function to compute the new signal relation to switch into) to the functional-level event value. In the case of the breaking pendulum model, the function *freeFall* is applied to $((0.21, -0.98), (2.25, 0.49))$.

The result of this application is a new signal relation. The simulation process continues from the first stage of symbolic processing and onwards by discarding the old signal relation and simulating the new one. In the case of the breaking pendulum model, the $pendulum$ signal relation is discarded together with the machine code that was generated for it by the LLVM JIT compiler.

In the current implementation, the new signal relation is flattened and new equations generated without reusing old ones from previous modes. In other words, events are not treated locally. In addition, the state of the whole system needs to be transferred for global and explicit reinitialisation of the entire system at every event using a *top level* switch, like in the breaking pendulum example. We hope to address these issues in the future: see Section 6.

4 Performance

In this section we provide an initial performance evaluation of the current prototype implementation of Hydra. We are mainly concerned with the overheads of mode switching (computing new structural configurations at events, symbolic processing of the equations, and JIT compilation) and how this scales when the size of the models grow in order to establish the feasibility of our approach. The time spent on numerical simulation is of less interest at this point: as we are using standard numerical solvers, and as our model equations are compiled down to native code with efficiency on par with statically generated code (see section 2.4), this aspect of the overall performance should be roughly similar to what can be obtained from other compilation-based modelling and simulation language implementations. For this reason, and because other compilation-based, non-causal modelling and simulation language implementations do not carry out dynamic reconfiguration, we do not compare the performance to other simulation software. The results would not be very meaningful.

The evaluation setup is as follows. The numerical simulator integrates the system using variable-step, variable-order BDF (Backward Differentiation Formula) solver [1]. Absolute and relative tolerances for numerical solution are set to 10^{-6} and trajectories are printed out at every point where $t = 10^{-3} * k, k \epsilon \mathbb{N}$. For static compilation and JIT compilation we use GHC 6.10.4 and LLVM 2.5 respectively. Simulations are performed on a 2.0 GHz x86-64 Intel® Core$^{\text{TM}}$2 CPU. However, presently, we do not exploit any parallelism, running everything on a single core.

Let us first consider the model of the breaking pendulum from Section 2.3. We simulate it over the time interval $t \epsilon [0, 20]$, letting the pendulum break at $t = 10$. Table 1 shows the amount of time spent simulating each mode of the system, and within that how much time that is spent on each of the four conceptual simulation process stages (see Section 3). As can be seen, most time (80–90 %) is spent on numerical simulation, meaning the overheads of our dynamic code generation approach was small in this case. Also, in absolute terms, it can be seen that the amount of time spent on symbolic processing, JIT compilation, and event handling was small, just fractions of a second.

	Pendulum $t \epsilon [0, 10)$		Free Fall $t \epsilon [10, 20]$	
	CPU Time		CPU Time	
	s	%	s	%
Symbolic Processing	0.0001	0.2	0.0000	0.0
JIT Compilation	0.0110	18.0	0.0077	9.1
Numerical Simulation	0.0500	81.8	0.0767	90.9
Event Handling	0.0000	0.0	-	-
Total	0.0611	100.0	0.0844	100.0

Table 1: Time profile of the breaking pendulum simulation

However, the breaking pendulum example is obviously very small (just a handful of equations), and it only needs to be translated to simulation code twice: at simulation start and when the pendulum breaks. To get an idea of how the performance of the prototype implementation scales with an increasing number of equations, we constructed a hybrid model of an RLC circuit (i.e., a circuit consisting of resistors, inductors and capacitors) with dynamic structure. In the first mode the circuit contains 200 components, described by 1000 equations in total (5 equations for each component). Every time $t = 10 * k$, where $k \epsilon \mathbb{N}$, the number of circuit components is increased by 200 (and thus the number of equations by 1000) by switching the additional components into the circuit.

	200 Components 1000 Equations $t \epsilon [0, 10)$		400 Components 2000 Equations $t \epsilon [10, 20)$		600 Components 3000 Equations $t \epsilon [20, 30)$	
	CPU Time		CPU Time		CPU Time	
	s	%	s	%	s	%
Symbolic Processing	0.063	0.6	0.147	0.6	0.236	0.5
JIT Compilation	1.057	10.2	2.120	8.3	3.213	6.6
Numerical Simulation	9.273	89.2	23.228	91.1	45.140	92.9
Event Handling	0.004	0.0	0.006	0.0	0.008	0.0
Total	10.397	100.0	25.501	100.0	48.598	100.0

Table 2: Time profile of structurally dynamic RLC circuit simulation, part I

	800 Components 4000 Equations $t \epsilon [30, 40)$		1000 Components 5000 Equations $t \epsilon [40, 50)$		1200 Components 6000 Equations $t \epsilon [50, 60]$	
	CPU Time		CPU Time		CPU Time	
	s	%	s	%	s	%
Symbolic Processing	0.328	0.4	0.439	0.4	0.534	0.3
JIT Compilation	4.506	4.9	5.660	5.1	6.840	4.3
Numerical Simulation	86.471	94.7	105.066	94.5	152.250	95.4
Event Handling	0.011	0.0	0.015	0.0	-	-
Total	91.317	100.0	111.179	100.0	159.624	100.0

Table 3: Time profile of structurally dynamic RLC circuit simulation, part II

Tables 2 and 3 show the amount of time spent in each mode of the system and in each conceptual stage of simulation of the structurally dynamic RLC circuit. In absolute terms, it is evident that the extra time spent on the mode switches becomes significant as the system grows. However, in relative terms, the overheads of our dynamic code generation approach remains low at about 10 % or less of the overall simulation time.

While JIT compilation remains the dominating part of the time spent at mode switches, Figure 3 demonstrates that the performance of the JIT compiler scales well. In particular, compilation time increases roughly linearly in the number of equations. In addition, it should be noted that the time spent on symbolic processing and event handling remains encouragingly modest (both in relative and absolute terms) and grows slowly as model complexity increases. There are also many opportunities for further performance improvements: see Section 6 for some possibilities.

Our approach offers new functionality in that it allows non-causal modelling and simulation of structurally dynamic systems that simply cannot be handled by static approaches. Thus, when evaluating the feasibility of our approach, one should weigh the overheads against the limitation and inconvenience of not being able to model such systems non-causally.

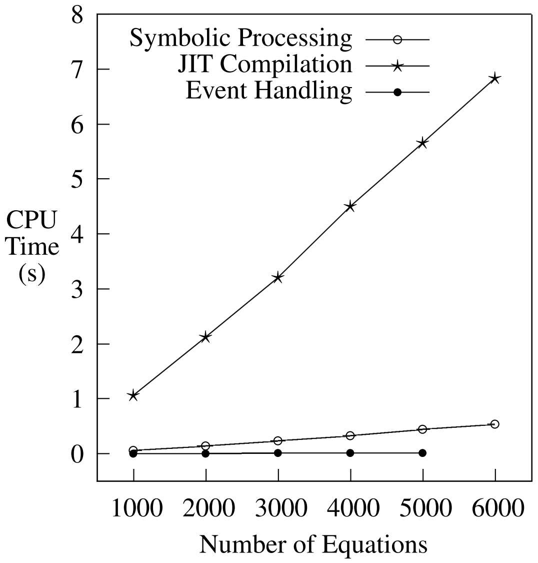


Figure 3: Plot demonstrating how CPU time spent on mode switches grows as number of equations increase in structurally dynamic RLC circuit simulation

5 Related Work

5.1 Sol

Sol is a Modelica-like language [21, 22]. It introduces language constructs that enable the description of systems where objects are dynamically created and deleted, thus aiming at supporting modelling of highly structurally dynamic systems. So far, the research emphasis has been on the design of the language itself along with support for incremental dynamic recausalisation and dynamic handling of structural singularities. An interpreter is used for simulation. The work on Sol is thus complementary to ours: techniques for dynamic compilation would be of interest in the context of Sol to enable it to target high-end simulation tasks; conversely, algorithms for incremental recausalisation is of interest to us to minimise the amount of work needed to regenerate simulation code after structural changes (see Section 6).

5.2 MOSILAB

MOSILAB is an extension of the Modelica language that supports the description of structural changes using object-oriented statecharts [19]. This enables modelling of structurally dynamic systems. It is a compiled implementation. However, the statechart approach implies that all structural modes must be explicitly specified in advance, meaning that MOSILAB does not support *highly* structurally dynamic systems. Even so, if the number of possible configurations is large (perhaps generated mechanically by meta-modelling), techniques like those we have investigated here might be of interest also in the implementation of MOSILAB.

5.3 Modelling Kernel Language

Broman [2, 3] is developing the Modelling Kernel Language (MKL) that is intended to be a core language for non-causal modelling languages such as Modelica. Broman takes a functional approach to non-causal modelling, similar to the FHM approach [17, 18]. One of the main goals of MKL is to provide a formal semantics of the core language. Currently, this semantics is based on an untyped, effectful λ-calculus.

Similarly to Hydra, MKL provides a λ-abstraction for defining functions and an abstraction similar to **sigrel** for defining non-causal models. Both functions and non-causal models are first-class entities in MKL, enabling higher-order, non-causal modelling. The similarity of the basic abstractions in Hydra and MKL leads to a similar style of modelling in both languages.

Thus far, the work on MKL has not specifically considered support for structural dynamics, meaning that its expressive power in that respect is similar to current mainstream, non-causal modelling and simulation languages like Modelica. However, given the similarities between MKL and FHM/Hydra, MKL should be a good setting for exploring support for structural dynamics, which ultimately could carry over to better support for structural dynamics for any higher-level language that has a semantics defined by translation into MKL. Again, the implementation techniques discussed in this paper should be of interest in such a setting.

6 Future Work

In the current implementation of Hydra, a new flat system of equations is generated at each mode switch without reusing the equations of the previous mode. It would be interesting to identify exactly what has *changed* at each mode switch, thus allowing reuse of the equations from the previous mode as much as possible. We hope to benefit from Zimmer's related work on incremental symbolic processing methods for structurally dynamic, non-causal simulation [22, 23]. In particular, information about the equations that remain unchanged during the mode switches provides opportunities for the JIT compiler to reuse the machine code from the previous mode, thus reducing the burden on the JIT compiler and consequently the compilation time during mode switches.

We are considering approaches for further performance improvements by taking advantage of multi-core hardware. In principle, the functions i, f and e (see Figure 2) could be JIT compiled in parallel, which should give a respectable speedup on a two-core system. The functions could also be broken down into smaller functions, each independently compilable, thus potentially keeping a fair number of cores busy simultaneously.

Another, or possibly complementary, idea, is a mixed interpreter and JIT compiler approach. Numerical simulation would start directly after the symbolic processing stage by *interpreting* the simulation code. At the same time, a JIT compilation process would be forked in the background. This seems particularly easy in the context of LLVM as it provides both an interpreter and JIT compiler for LLVM code. Numerical simulation would initially progress with an extra interpretive overhead. However once the JIT compilation process is finished, the interpreter would be substituted by the JIT compiled code. On multi-core hardware, this may lead to significant performance improvements by decreasing the delays during the mode switches and improving overall simulation time.

In the current version of Hydra, transfer of system state between modes and reinitialisation has to be coded explicitly at every switch for the entire system. This quickly becomes infeasible as models grow. Better language support for declaratively specifying implicit state transfer for unchanging parts of a system is thus something that we intend to look into.

7 Conclusions

This paper presents a novel approach to the implementation of non-causal modelling and simulation languages. It allows symbolic processing and code generation to be carried out as the model undergoes structural changes during the simulation, thus enabling non-causal modelling and simulation of highly structurally dynamic systems. Our approach provides an efficient alternative to interpreted implementations of structurally dynamic modelling languages and, at the same time, lifts the restrictions that are associated with pre-simulation compilation of non-causal modelling languages.

Our work is carried out in the framework of Functional Hybrid Modelling (FHM) because, by supporting higher-order modelling, this provides expressive language features for describing structurally dynamic systems. However, our implementation approach can be applied to other modelling languages that aim to support structural dynamism.

Acknowledgements. This work was supported by EPSRC grant EP/D064554/1. We would like to thank the anonymous reviewers for their thorough and constructive feedback that helped improve the paper.

References

[1] Kathryn Eleda Brenan, Stephen La Vern Campbell, and Linda Ruth Petzold. *Numerical solution of initial-value problems in differential-algebraic equations*. SIAM, Philadelphia, 1996.

[2] David Broman. Flow Lambda Calculus for declarative physical connection semantics. Technical Reports in Computer and Information Science 1, Linköping University Electronic Press, 2007.

[3] David Broman and Peter Fritzson. Higher-order acausal models. In Peter Fritzson, François Cellier, and David Broman, editors, *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)*, number 29 in Linköping Electronic Conference Proceedings, pages 59–69, Paphos, Cyprus, 2008. Linköping University Electronic Press.

[4] Conal Elliott and Paul Hudak. Functional reactive animation. In *Proceedings of ICFP'97: International Conference on Functional Programming*, pages 163–173, June 1997.

[5] George Giorgidze and Henrik Nilsson. Embedding a functional hybrid modelling language in Haskell. In *Refereed Proceedings of the 20th International Symposium on the Implementation and Application of Functional Languages (IFL '08)*, University of Hertfordshire, Hatfield, UK, September 2008. To Appear.

[6] Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, 31(3):363–396, 2005.

[7] Chris Lattner. LLVM: An Infrastructure for Multi-Stage Optimization. Master's thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL, Dec 2002. *See* `http://llvm.org`.

[8] Chris Lattner. Introduction to the llvm compiler system. In *Proceedings of International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, Erice, Sicily, Italy, 2008.

[9] Chris Lattner and Vikram Adve. The LLVM Compiler Framework and Infrastructure Tutorial. In *LCPC'04 Mini Workshop on Compiler Research Infrastructures*, West Lafayette, Indiana, Sep 2004.

[10] Geoffrey Mainland. Why it's nice to be quoted: quasiquoting for haskell. In *Haskell '07: Proceedings of the ACM SIGPLAN workshop on Haskell workshop*, pages 73–82, New York, NY, USA, 2007. ACM.

[11] The Modelica Association. *Modelica – A Unified Object-Oriented Language for Physical Systems Modeling: Tutorial version 1.4*, December 2000.

[12] Pieter J. Mosterman. *Hybrid Dynamic Systems: A Hybrid Bond Graph Modeling Paradigm and its Application in Diagnosis*. PhD thesis, Graduate School of Vanderbilt University, Nashville, Tennessee, May 1997.

[13] Pieter J. Mosterman. An overview of hybrid simulation phenomena and their support by simulation packages. In *HSCC '99: Proceedings of the Second International Workshop on Hybrid Systems*, pages 165–177, London, UK, 1999. Springer-Verlag.

[14] Pieter J. Mosterman, Gautam Biswas, and Martin Otter. Simulation of discontinuities in physical system models based on conservation principles. In *Proceedings of SCS Summer Conference 1998*, pages 320–325, July 1998.

[15] Henrik Nilsson. Type-based structural analysis for modular systems of equations. In Peter Fritzson, François Cellier, and David Broman, editors, *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools*, number 29 in Linköping Electronic Conference Proceedings, pages 71–81, Paphos, Cyprus, July 2008. Linköping University Electronic Press.

[16] Henrik Nilsson, Antony Courtney, and John Peterson. Functional reactive programming, continued. In *Proceedings of the 2002 ACM SIGPLAN Haskell Workshop (Haskell'02)*, pages 51–64, Pittsburgh, Pennsylvania, USA, October 2002. ACM Press.

[17] Henrik Nilsson, John Peterson, and Paul Hudak. Functional hybrid modeling. In *Proceedings of PADL'03: 5th International Workshop on Practical Aspects of Declarative Languages*, volume 2562 of *Lecture Notes in Computer Science*, pages 376–390, New Orleans, Lousiana, USA, January 2003. Springer-Verlag.

[18] Henrik Nilsson, John Peterson, and Paul Hudak. Functional hybrid modeling from an object-oriented perspective. In Peter Fritzson, François Cellier, and Christoph Nytsch-Geusen, editors, *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)*, number 24 in Linköping Electronic Conference Proceedings, pages 71–87, Berlin, Germany, 2007. Linköping University Electronic Press.

[19] Christoph Nytsch-Geusen, Thilo Ernst, André Nordwig, Peter Schwarz, Peter Schneider, Matthias Vetter, Christof Wittwer, Thierry Nouidui, Andreas Holm, Jürgen Leopold, Gerhard Schmidt, Alexander Mattes, and Ulrich Doll. MOSILAB: Development of a modelica based generic simulation tool supporting model structural dynamics. In *Proceedings of the 4th International Modelica Conference*, pages 527–535, Hamburg, Germany, 2005.

[20] Günther Zauner, Daniel Leitner, and Felix Breitenecker. Modelling structural-dynamics systems in Modelica/Dymola, Modelica/MOSILAB, and Any-Logic. In Peter Fritzson, François Cellier, and Christoph Nytsch-Geusen, editors, *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)*, number 24 in Linköping Electronic Conference Proceedings, pages 99–110, Berlin, Germany, 2007. Linköping University Electronic Press.

[21] Dirk Zimmer. Enhancing Modelica towards variable structure systems. In Peter Fritzson, François Cellier, and Christoph Nytsch-Geusen, editors, *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)*, number 24 in Linköping Electronic Conference Proceedings, pages 61–70, Berlin, Germany, 2007. Linköping University Electronic Press.

[22] Dirk Zimmer. Introducing Sol: A general methodology for equation-based modeling of variable-structure systems. In *Proceedings of the 6th International Modelica Conference*, pages 47–56, Bielefeld, Germany, 2008.

[23] Dirk Zimmer. An application of Sol on variable-structure systems with higher index. In *Proceedings of the 7th International Modelica Conference*, Como, Italy, 2009.

Operator Overloading in Modelica 3.1

Hans Olsson[1], Martin Otter[2], Hilding Elmqvist[1], Dag Brück[1],
[1]Dassault Systèmes, Lund, Sweden (Dynasim)
[2]German Aerospace Centre (DLR), Institute for Robotics and Mechatronics, Germany
Hans.Olsson@3ds.com, Martin.Otter@DLR.de,
Hilding.Elmqvist@3ds.com, Dag.Bruck@3ds.com

Abstract

The constructor and operator overloading introduced in Modelica 3.1 is discussed. The goal is that elementary operators like "+" or "*" can be overloaded for records. This makes it possible to define and use, in a convenient way, complex numbers, polynomials, transfer functions, state space systems, etc. The chosen approach is different to other languages: (a) Only scalar operations need to be overloaded. Array operations are then automatically available, so the growth of the number of overloaded functions is avoided. (b) Automatic type casts between different data types is performed using overloaded constructor functions. Again this reduces the number of overloaded functions. (c) The approach is conservative and only allows overloading if no ambiguity is present, in order to not introduce pitfalls into the language. This is reached by basing the overloading on disjoint sets of matching functions and not on a priority match.

Keywords: overloading, automatic overloading of arrays, overloading without ambiguities.

1 Introduction

Operator overloading is a well known concept in computer science and is available in languages such as Ada (ANSI 1983), C++ (ISO 1998), C#, Mathematica, Matlab and Python. In 2002-2005 the Modelica Association has worked on operator overloading for the Modelica language and several different versions have been designed by different people, especially to avoid some of the known problems of overloading from other languages. The work was then suspended for some years to concentrate on the improved safety in Modelica 3.0. Work has restarted in 2008: Based on a prototype implementation in Dymola and by applying this prototype to the Beta version of the Modelica_LinearSystems2 library *(Baur et. al. 2009)*, the 7th design version from 2005 was revised considerably and finally resulted in a version that has been included in Modelica 3.1 *(Modelica 2009).*

The overloading introduced in Modelica 3.1 is seen as a first step and more features might be introduced later, based on the gained experience. The design is conservative and restrictive in order to reduce the probability to introduce pitfalls in the language. For example, ambiguities are not allowed. This is opposed to other languages where ambiguities are often resolved by priorities in function matches. An important, new feature is that it usually suffices to overload scalar operations and that array operations are automatically mapped to the overloaded scalar operations. The benefit is that explosive growth of the number of overloaded functions to define all possible combinations of data types and number of array dimensions is avoided.

2 Example with Complex numbers

The basic properties of operator overloading in Modelica 3.1 shall first be demonstrated by an example to introduce a user-defined data type Complex. In section 3, the formal rules are defined and design considerations are explained.

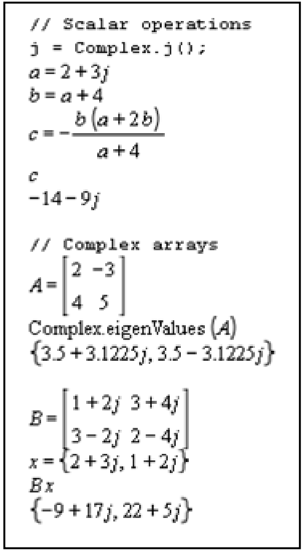
Assume a record "Complex" with overloaded scalar operators is available (see below). When using this definition in an interactive environment, e.g., in a Modelica script file that is executed by Dymola *(Dymola 2009)*, then in the command window of Dymola the output as shown in the right part of **Figure 1** appears.

DOI: 10.3384/ecp09430100

Script file

```
// Scalar operations
j = Complex.j();
a = 2 + 3*j
b = a + 4
c = -b*(a + 2*b)/(a+4)
c

// Complex arrays
A  = [2,-3; 4,5]
Complex.eigenValues(A)

B = [1+2*j, 3+4*j;
     3-2*j, 2-4*j]
x = {2+3*j, 1+2*j}
B*x
```

Output window of Dymola



// Scalar operations

j = Complex.j();

$a = 2 + 3j$

$b = a + 4$

$c = -\frac{b\left(a+2b\right)}{a+4}$

c

$-14 - 9j$

// Complex arrays

$A = \begin{bmatrix} 2 & -3 \\ 4 & 5 \end{bmatrix}$

Complex.eigenValues (A)

$\{3.5 + 3.1225j, 3.5 - 3.1225j\}$

$B = \begin{bmatrix} 1+2j & 3+4j \\ 3-2j & 2-4j \end{bmatrix}$

$x = \{2+3j, 1+2j\}$

Bx

$\{-9 + 17j, 22 + 5j\}$

Figure 1: Using the overloaded Complex data type in a script file (left) and the output in the command window of Dymola 7.3 (right).

From this example it can be seen that the user defined Complex type can hardly be distinguished from a built-in type like Real. In particular, standard array operations can be applied on Complex, although only the scalar operations are overloaded. Also type casts from Real or Integer to Complex are automatically performed, for example in "a = 2 + 3*j" where 2 is added to the Complex expression "3*j").

The "essential" difference to a built-in type is the name look-up: If a variable is declared as "Real a", then it is first determined whether "Real" is a built-in type before performing another lookup. If a variable is declared as "Complex c", then "Complex" is searched hierarchically from the current scope up to the global scope. For example, if a user introduces an own "Complex" type in the local scope, then this type is used and not the one from the global scope.

For the example above, the following definitions are needed:

```
record Complex
  Real re "Real part";
  Real im "Imaginary part";

  function j
    output Complex result;
  algorithm
    result := Complex(0,1);
  end j;

  operator 'constructor'
    function fromReal
      input  Real re;
      input  Real im=0;
      output Complex result;
    algorithm
      result = Complex(re=re,im=im);
    end fromReal;
  end 'constructor';

  operator '+'
    function add
      input  Complex c1;
      input  Complex c2;
      output Complex result;
    algorithm
      result := Complex(c1.re + c2.re,
                        c1.im + c2.im);
    end add;
  end '+';

  operator '-'
    function negate
      input  Complex c;
      output Complex result;
    algorithm
      result := Complex(- c2.re,
                        - c2.im);
    end negate;

    function subtract
      input  Complex c1;
      input  Complex c2;
      output Complex result;
    algorithm
      result := Complex(c1.re - c2.re,
                        c1.im - c2.im);
    end subtract;
  end '-';

  // also: '*', '/', '^', '==', '<>'

  operator 'String'
    function toString
      input  Complex c;
      input  String name="j";
      output String s;
    algorithm
      s := String(c.re);
      if c.im <> 0 then
         s := if c.im > 0 then
                 s + " + "
              else
                 s + " - ";
         s := s + String(abs(c.im))
                + name;
      end if;
    end toString;
  end 'String';
end Complex;

function eigenValues
   input  Real    A [:,:];
   output Complex ev[size(A, 1)];
   import Modelica.Math.Matrices;
protected
   Integer nx=size(A, 1);
   Real    evr[nx,2];
   Integer i;
algorithm
   evr := Matrices.eigenValues(A);
   for i in 1:nx loop
     ev[i] := Complex(evr[i, 1],
                      evr[i, 2]);
   end for;
end eigenValues;
```

As can be seen, operator overloading is defined for functions that are defined in a record. The record definition holds a data structure in the usual way (here: two Real variables). Operators are defined in a record with the new construct

```
operator <name>
  …
end <name>
```

where <name> is the operator to be overloaded enclosed in apostrophes. This has the advantage that a valid, unique Modelica name is used which is very close to the operator that shall be overloaded.

Inside an "operator", one or more Modelica functions are defined. There are no particular requirements for these functions with the exception that every function must have exactly one output argument and that the number of arguments without a default value must be identical to the number of arguments required from the respective operator (e.g., function "add" inside operator '+' must have exactly two arguments without a default value. If there are more arguments, all must have a default value.

The special operator 'constructor' serves two purposes: First it gives different record constructors to provide various ways to generate an instance of the record. Second it is used to define automatic type casts. Examples:

```
// Default record constructor:
c1 = Complex(1,2);  // c1 = 1+2*j;

// Overloaded constructor "fromReal":
c2 = Complex(3);  // c2 = 3+0*j;

// Automatic type cast due to "fromReal":
c3 = c1 + 5;  // c3 = 6+2*j;
```

No overloaded operator is defined to add a Complex to a Real. However, a constructor is defined to generate a Complex number from the literal "5" and then there is an overloaded operator to add two Complex numbers.

3 Rules for Overloading

In this section the rules for the operator overloading are stated and design decisions are discussed.

3.1 Overloaded operators

A Modelica **record** can define the behavior for operations such as constructing, adding, multiplying etc. This is done using the specialized class **operator** (a restricted class similar to **package**) comprised of functions implementing different variants of the operation for the record class in which the respective **operator** definition resides. The overloading is defined in such a way that ambiguities are not allowed and give an error. Furthermore, it is sufficient to define overloading for scalars. Overloaded array operations are automatically deduced from the overloaded scalar operations, if an appropriately overloaded function for arrays is not present. The **operator** keyword is followed by the name of the operation which can be one of:

'constructor', '+', '-' (includes both subtraction and negation), '*', '/', '^', '==', '<>', '>', '<', '>=', '<=', 'and', 'or', 'not', 'String'.

The functions defined in the operator-class in the record must take at least one argument of this record type as input, except for the constructor-functions which instead must return one component of the record type. All of the functions shall return exactly one output.

The record may also contain additional functions, packages of functions, and declarations of components of the record. To avoid problems with slicing, it is not legal to extend from a record with operators.

The precedence and associativity of the overloaded operators is identical to built-in operators (e.g. '*' has always higher precedence as '+'). Definition of new operator symbols is not allowed. These restrictions simplify specification and implementation, and improve translation speed.

Only overloading of the most important operators is defined. In the future, this list might be extended, but the goal is to first get experience with a minimum set of overloaded operators.

3.2 Matching Functions

All functions defined inside the **operator** class must return one output and may include functions with optional arguments, i.e. functions of the form

```
function f
   input  A_1 u_1;
   …
   input  A_m u_m = a_m;
   …
   input  A_n u_n;
   output B   y;
algorithm
   …
end f;
```

The vector P below indicates whether argument m of f has a default value (**true** for default value, **false** otherwise). A call $f(a_1, a2,\ldots, a_k, b_1 = w_1 ,\ldots, b_p= w_p)$ with distinct names b_j is a valid match for the function f, provided (treating Integer and Real as the same type)

- A_i = typeOf(a_i) for $1 \le i \le k$,
- the names $b_j = u_{Qj}$, $Qj > k$, A_{Qj} = typeOf(w_i) for $1 \le j \le p$, and
- if the union of {i: $1 \le i \le k$ }, {Qj: $1 \le j \le p$}, and {m: P_m **true** and $1 \le m \le n$ } is the set {i: $1 \le i \le n$}.

This corresponds to the normal treatment of a function call with named arguments, requiring that all inputs have some value given by a positional argument, named argument, or a default value (and that positional and named arguments do not overlap). Note that this only defines a valid call, but does not explicitly define the set of domains.

3.3 Overloaded constructors and operators

As defined in detail in the Modelica language specification *(Modelica 2009)*, using an operator (such as '+') goes through a number of steps where a set of functions is found, and if one of them is a matching function it is used; multiple matches are seen as an error.

Array operations are defined in terms of the scalar operation, for multiplication assuming that the scalar element form a non-commutative ring that does not necessarily have a multiplicative identity (since the definition in the specification implicitly assumes that addition is associative and commutative); the operations vector*vector and vector*matrix are explicitly excluded, since there are cases where this does not give the "natural" interpretation, e.g., for complex vectors. For the future it will be possible to extend operations with complex conjugate (allowing a clean definition of vector*vector) and zero (allowing e.g. matrix multiplication with zero inner dimensions); without invalidating existing models.

The precise rules for binary operations will be now presented to show the flavor of the definition:

Let *op* denote a binary operator like '+'and consider an expression *a op b* where *a* is of type *A* and *b* is of type *B*. An example is "2.0 + j", where "2.0" is of type Real and "j" is of type "Complex.

1. If *A* and *B* are basic types or arrays of such, then the corresponding built-in operation is performed (e.g., for "2 + 3", the built-in operation for two Integer numbers is performed).
2. Otherwise, if there exists exactly one function *f* in the union of *A.op* and *B.op* such that f(*a*,*b*) is a valid match for the function *f* , then *a op b* is evaluated using this function. It is an error, if multiple functions match. If A is not a record type, A.op is seen as the empty set, and similarly for B. Note, Having a union of the operators ensures that if A and B are the same, each function only appears once. In our example, "2.0 + j" has only a match in the Complex record after converting 2.0 to Complex: Complex.'+' and therefore a matching function was found.
3. Otherwise, consider the set given by *f* in *A.op* and a record type *C* (different from B) with a constructor, g, such that C.′constructor′.g(b) is a valid match, and f(a, C.′constructor′.g(b)) is a valid match; and another set given by *f* in *B.op* and a record type *D* (different from A) with a constructor, h, such that D.′constructor′.h(a) is a valid match and f(D.′constructor′.h(a), b) is a valid match. If the sum of the sizes of these sets is one this gives the unique match. If the sum of the sizes is larger than one there is an ambiguity which is an error.

   Informally, this means: If there is no direct match of "a op b", then it is tried to find a direct match by automatic type casts of "a" or "b", by converting either "a" or "b" to the needed type using an appropriate constructor function from one of the record types used as arguments of the overloaded "op" functions. Example using the Complex-definition from above:

   ```
   Real    a;
   Complex b;
   Complex c = a+b;
   // interpreted as:
   Complex.′+′(
    Complex.′constructor′.fromReal(a),b);
   ```
4. If A or B is an array type, then the expression is conceptually evaluated according to the rules for arrays (Modelica 2009, section 10.6). The resulting scalar operations are then treated with 1-3. Example:

   ```
   Complex A[2,2], x[2];
   Complex b[2] = A*x;
   // interpreted as:
   b[1] = A[1,1]*x[1] + A[1,2]*x[2];
   b[2] = A[2,1]*x[2] + A[2,2]*x[2];
   // The scalar operations can now be
   // treated with the rules for scalar
   // operations
   ```
5. Otherwise the expression is erroneous.

3.4 Syntactical simplification

In many cases there is only one function in the operator; either because only one makes sense or because another is not yet added. This is handled by stating that

```
operator function '*'
        …
end '*';
```

is treated in the same way as

```
operator '*'
  function multiply
    ...
  end multiply
end '*';
```

The advantage of the shorter form is that it reads nicer, and avoids introducing an arbitrary name of a function.

However, by stating that they are equivalent, no loss of functionality is introduced; and one can always later add additional overloaded variants in a safe way.

4 Design and future considerations

The overall design is intended as a first step, and intended to allow future extensions in a backward compatible way.

4.1 Operator as a "semi-package" in record

In the current design an operator defines a hierarchical level; grouping together the variants.

The alternative of having multiple overloaded functions with identical names and different signatures (as in C++) was considered; but rejected for several reasons including the fact that it would no longer be possible to uniquely reference a function by name. However, the syntactic simplification introduced avoids redundant levels.

Another alternative would be to have the operators defined in the enclosing scope of the record - similarly to Ada. This would have required a modification of the function call lookup to include some form of argument-dependent name lookup ("Koenig lookup") as in C++ (ISO 1998, Section 3.4.2). This would be complex to implement, and possibly influence existing function calls (note that in Modelica function calls normally use hierarchical names in contrast to many other languages). Furthermore it was found that it often leads to a two-step hierarchy where a record 'complex' was defined in a package 'complexPackage' merely containing the record and its operations (cf. "header files"); and this was not deemed attractive.

One of the drawbacks of this design is that new operations on existing types cannot be added without modification of classes, which may not be possible for protection or licensing reasons.

Stroustrup (1994, Chapter 11) describes several related design issues and tradeoffs for C++.

4.2 Symmetric

Binary operators are defined so that operations can either be found in left or right operands. This is needed in order to handle combinations with built-in types in a clean way.

4.3 Few priority levels

For function matching there are only a few levels defined; whereas, e.g. C++ has a much more detailed set of priorities between functions in order to handle type conversions and many arguments for general functions.

A number of such detailed rules were considered in the Modelica design group, but due to limited resources they could not be investigated. Thus such cases currently lead to ambiguities, these cases could in the future be disambiguated with more detailed rules – but the intent is that everything that is currently unambiguous will stay that way.

4.4 Fewer operators

It is common to define only a few operators and define others in terms of these. This is here done for array operations, but not for e.g. relational operators (usually everything is defined in terms of '<' and/or '=='). It was not clear how common overloaded relational operators will be in Modelica and for what purpose, and thus this was deemed as an issue that will be handled in the future.

An important consideration is whether relational operators will be used for general routines such as sorting as in the Standard Template Library of C++ (where '<' is more used as a sorting order than a mathematical total order); or for more general mathematical routines, e.g. computations for IEEE floating numbers including NaN where such rules do not hold.

4.5 Zero values and complex numbers

As indicated above matrix multiplication is currently undefined if the inner dimension is zero. A simple solution would be to introduce an operator 'zero' having no inputs and returning the additional identity of the class. An important consideration will be whether this operator should be required for matrix multiplication in general; and whether it should be used for other purposes.

Similarly vector*vector could be defined if there existed an operator 'conjugate' in the class.

4.6 Hierarchy of conversions

In the future it might be necessary to add another ‘constructor’-operator containing only explicit constructors – i.e. constructors that, as in C++, only will be called if the constructor is explicitly invoked and not for implicit conversions.

Without this care must be taken when designing multiple records such that conversions form an ordered hierarchy.

At one point in the design it was considered to have conversions in both directions and instead introduce additional operators to disambiguate calls; e.g. have Complex and ComplexPolar that both can be converted automatically from the other one and instead define operations such as addition to disambiguate the results:

```
record Complex
  …
  operator '+'
    function addComplex
      input Complex a;
      input Complex b;
      output Complex c;
    …
    function addPolar "Example only"
      input Complex a;
      input ComplexPolar b;
      output Complex c;
    …
end Complex;
```

The problem with this approach is that c+2 is ambiguous since it is not clear if 2 should be converted to polar or Cartesian form before being added. It would be possible to handle this by having an additional operation for addition with Real; but it was deemed that the resulting number of functions grew too much and a cleaner design was to remove addPolar.

5 Conclusion

Modelica 3.1 was released in May 2009. The operator overloading as introduced in this new version was discussed and examples are given to demonstrate the usage. The introduced operator overloading is seen as a first step, to gain experience with it in Modelica. Especially, it is clear that function overloading is missing and has to be introduced.

With respect to other languages, the design is restrictive, but has the advantage that it usually suffices to define overloaded scalar operations between the same types. Array operations and operations between different types can then be automatically deduced by a Modelica tool.

6 Acknowledgements

Partial financial support of DLR by BMBF (BMBF Förderkennzeichen: 01IS07022F) for this work within the ITEA project EUROSYSLIB (http://www.itea2.org/public/project_leaflets/EUROSYSLIB_profile_oct-07.pdf) is highly appreciated.

Furthermore, we would like to thank Marcus Baur (DLR) for fruitful discussions.

References

ANSI (1983): **Ada Language Reference Manual.** ANSI/MIL-STD 1815A.

Baur M., Otter M., and Thiele B. (2009): **Modelica Libraries for Analysis and Design of Linear Control Systems**. In F. Casella (editor): Proc. of the 7th Int. Modelica Conference, Como, Italy. www.modelica.org/events/modelica2009

Dymola (2009). **Dymola Version 7.3**. Dassault Systèmes, Lund, Sweden (Dynasim). Homepage: www.dymola.com.

ISO (1998): **International Standard, Programming Languages – C++.** ISO/IEC 14882:1998.

Modelica (2009). **Modelica Language Specification 3.1**. www.modelica.org/documents/ModelicaSpec31.pdf

Stroustrup B. (1994): **The Design and Evolution of C++.** Addison-Wesley.

An Application of Sol on Variable-Structure Systems with Higher Index

Dirk Zimmer
Department of Computer Science, ETH Zurich
CH-8092 Zurich, Switzerland
dzimmer@inf.ethz.ch

Abstract

This case study presents the model of an ideal trebuchet. Following the object-oriented modeling paradigm of Modelica, the trebuchet is composed out of ideal elements that belong to a planar mechanical library. The corresponding system of DAEs has index 3. During simulation, the model undergoes also various structural changes that manipulate the number of continuous-time state variables. Furthermore, elastic and inelastic collisions need to be modeled by force impulses. The model is provided in Sol, a derivative language of Modelica, specially designed for research in variables structure systems. *Keywords: Variable-Structure Systems; Index Reduction; Multi-Body Dynamics.*

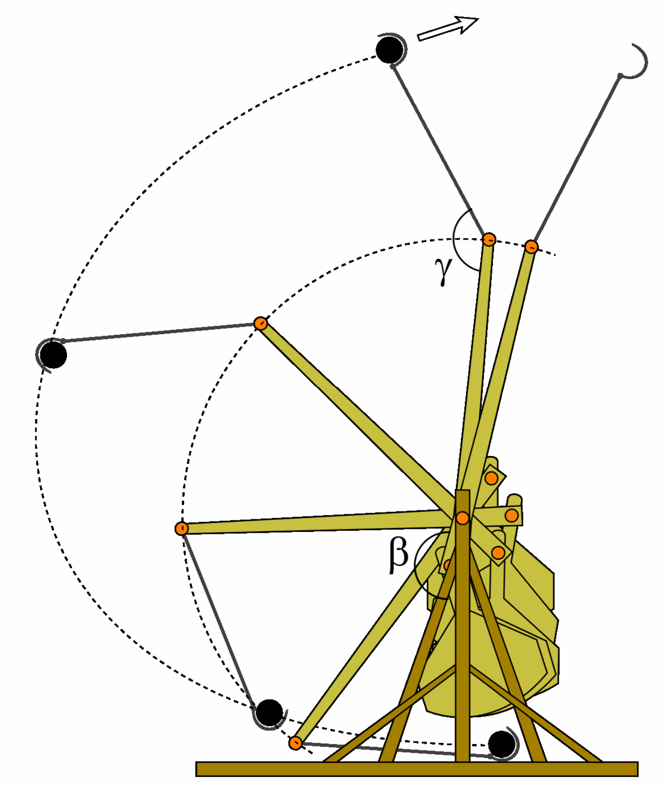
1 The Trebuchet

The Trebuchet is an old catapult weapon developed in the Middle Ages. It is known for its long range and its high precision. Figure 1 depicts a trebuchet and thereby presents its functionality. Technically, it is a double pendulum propelling a projectile in a sling. The rope of the sling is released on a predetermined angle γ when the projectile is about to overtake the lever arm.

Let us state a few assumptions for the model:

- All mechanics are planar. The positional states of any object are therefore restricted to x, y and the orientation angle φ.
- All elements are rigid.
- The sling's rope is ideal and weightless. It exhibits an inelastic impulse when being stretched to maximum length
- The revolute joint of the counterweight is limited to a certain angle β (in order to prevent too heavy back-swinging after the projectile's release). It also exhibits an inelastic impulse when reaching its limit.
- Air resistance or friction is neglected.

Whereas these idealizations simplify the parameterization of the model to a great extent, they pose serious difficulties for a general simulation environment. Such models, although being fairly simple, can neither be modeled nor simulated with Modelica yet. At least not in a truly object-oriented manner. Hence the trebuchet represents a suitable example for the framework of Sol that aims to enable the future handling of variable-structure systems within an object-oriented modeling paradigm.



source: wikimedia commons, modified by author

Mass of projectile: **30kg**	β:**200°**
Mass at lever arm: **100kg**	γ: **150°**
Counterweight: **10t**	

Figure 1: Functionality and specification of a trebuchet

2 Object-oriented Composition

Sol has been introduced at the Modelica Conference 2008 [9, 10]. It is a derivative language of Modelica, specially designed for research purposes in the field of variable-structure systems. Thus, Sol enables the creation and removal of equations or even complete objects anytime during the simulation. To this end, the modeler describes the system in a constructive way, where the structural changes are expressed by conditionalized declarations. These conditional parts can than get activated and deactivated of during run-time. The incentive for this project is to gain knowledge in language design and processing techniques that we think will be essential for Modelica's future development.

A simple planar mechanical library has been developed in Sol. It has been extended by equations for mechanical impulses in order to make discrete velocity changes possible. From this library we need the following components:

- 1x fixation	- 3x fixed translation
- 1x revolute joint	- 1x limited revolute joint
- 2x bodies with mass	- 1x ideal rope with mass

These components are connected as depicted in figure 2. Although the model diagram follows the iconographic of the MultiBody library [4], it serves illustration purposes only, since the modeling in Sol is still purely textual.

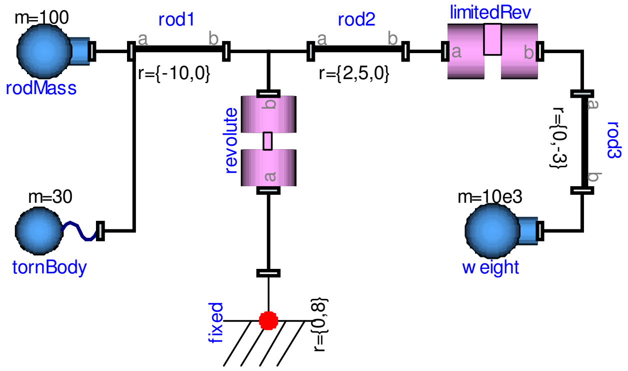


Figure 2: Model diagram of the trebuchet

The total model contains from 246 to 256 variables, depending on the current state of the model. The corresponding systems of DAE have the perturbation index 3. They need to be differentiated twice and there remain linear systems of equation to be solved.

The resulting object-oriented decomposition resembles typical examples from the Modelica domain but it is significantly more demanding since a structural change in any component may affect the total system.

```
model LimitedRevolute
  extends Interfaces.TwoFrames;
interface:
  parameter Real phi_start;
  parameter Real w_start;
  parameter Real l;
implementation:
  static Boolean contact;
  static Boolean fixated;
  static Boolean toFixate;
  static Boolean toRelease;
  static Real phi_a;
  static Real phi;
  static Real Wm;
  static Real We;

  if initial then
    fixated << false;
    toFixate << false;
    toRelease << false;
    phi_a << phi_start;
    We << w_start;
  end;

  when toFixate then
    toRelease << false;
    fixated << true;
  else when toRelease then
    toFixate << false;
    fixated << false;
  end;

  if fixated then
     phi = l;
     Wm = 0;
     contact << false;
     when fb.t < 0 then
        toRelease << true;
        phi_a << l;
     end;
  else then
    contact << (phi > l);
    static Real w;
    static Real Wa;
    w = der(x=phi, start << phi_a);
    when contact then
      w = 0;
      Wm = 0.5*Wa;
      We << w;
      toFixate << true;
    else then
       when fa.contactIn or fb.contactIn then
         w  = 2*Wm - Wa;
         We << w;
       else then
         static Real z;
         z  = der(x=w, start << We);
         Wa << w;
       end;
       fb.M = 0;
    end;
    fb.t = 0;
  end;

  fa.phi + phi = fb.phi;
  fa.t + fb.t = 0;
  fa.Wm + Wm = fb.Wm;
  fa.M + fb.M = 0;

  fa.x = fb.x;          fa.y = fb.y;
  fa.fx + fb.fx = 0;    fa.fy + fb.fy = 0;
  fa.Vmx = fb.Vmx;      fa.Vmy = fb.Vmy;
  fa.Px + fb.Px = 0;    fa.Py + fb.Py = 0;

  fa.contactOut << contact or fb.contactIn;
  fb.contactOut << contact or fa.contactIn;
end LimitedRevolute;
```

Figure 3: The model of a limited revolute joint.

3 Example component

Whereas the top-model can be neatly decomposed into general applicable components, the modeling of these components requires a skilled modeler. To attain a better understanding, let us take a look at the modeling code of one of the components that triggers a structural change: The limited revolute joint. The corresponding code is presented in figure 3.

Since Sol is very similar to Modelica the code shall be roughly understandable without further introduction. Let us go into the details.

An elbow is one possible representation of a limited revolute joint. The model has two major modes: *free* or *fixated*. The mode *free* is equivalent to a normal revolute joint whereas the model equals a fixed orientation in the *fixated* mode. Since the transition between these two states causes a discrete change in velocity, it involves an inelastic impulse on the rigidly connected components. Furthermore impulses from other components (as for instance the ideal rope) need to be handled as well in this component.

The different modes and their transitions are presented in the graph of figure 4, where the continuous-time modes are depicted as round boxes and the rectangular boxes denote discrete intermediate modes. The transitions are represented by arrows and their labels denote the event that triggers the transition. Those without a label are triggered immediately.

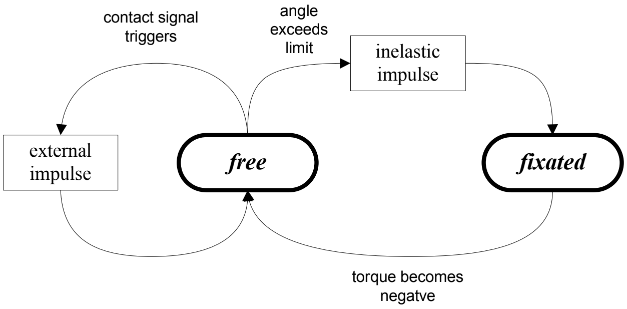


Figure 4: Mode-transition graph of the limited revolute

In the modeling code, the two continuous modes are expressed by the Boolean variable `fixated` and are modeled by an if-statement. We recognize that the first branch represents the fixated mode and does not contain any derivatives whereas the second branch (for the free mode) usually defines two derivatives. Hence, the free mode defines two potential state-variables: the position `phi` and the corresponding velocity `w`. A switch between the two modes is therefore expected to change the number of total state-variables.

The number of continuous-time state variables is also affected by the mechanical impulses. These impulse events are modeled by when-branches that react to a contact signal that may be emitted by other components. In order to understand how the model interacts with other components let us take a look at variables of the connecting interface:

Continuous potential variables:

`x y phi`: the positional states:

Continuous flow variables:

`fx fy t`: forces and torque

Discrete potential variables:

`Vmx Vmy Wm`: mean velocities during impulse.

Discrete flow variables

`Px Py M`: force impulses and angular momentum.

Control signals:

`contactIn`: ingoing contact signal

`contactOut`: outgoing contact signal

This connector design is very similar to the one that has already been applied in the Modelica MultiBondLib [11]. It owns a separate set of variables for the continuous and discrete domain. The Boolean control signals are used to trigger and synchronize the events.

Any component model will have to relate these interface variables. For the limited revolute, the equations that relate the variables of the translational domain are trivial and are placed at the end of the model's main section.

Nevertheless, the equations for the impulse event require further explanation. A force impulse `P`, or angular momentum `M` respectively, causes a discrete change in the corresponding velocity. This change is best described by the mean velocity during the impulse. Let `Wa` be the angular velocity before the impulse and `We` the velocity after the impulse, then `Wm` is defined as the mean `(Wa+We)/2`. Please note that the product of the corresponding interface variables (e.g. `M*Wm`) represents the amount of work that is transmitted during the impulse.

Using these variables, the impulse behavior can be properly described: For any mass element, the equation

```
M = 2*I*(Wm-Wa)
```

holds. An inelastic impulse can be modeled by stating:

```
Wm = 0.5*Wa
```

Mostly and also in this example, these and other impulse equations form a linear system of equations that is distributed over several components. Hence they need to be activated synchronously. To this end, the Boolean contact signals are required to synchronize the impulse events in different components.

This is illustrated by the event for an external impulse (see figure 5). Before the event the velocity is stored by the auxiliary variable `Wa`. At the event, the differential equation is removed since the velocity is now determined by the impulse equation `w = 2*Wm - Wa`. This new velocity is also stored in the auxiliary variable `We` that is needed after the event when the differential equation gets reestablished and `We` is suggested as (re-)start value for the time integration.

```
when fa.contactIn or fb.contactIn then
  w  = 2*Wm - Wa;
  We << w; [1]
else then
  static Real z;
  z  = der(x=w, start << We);
  Wa << w;
end;
```

Figure 5: Excerpt from figure 3.

In the example of the trebuchet, this event is synchronously triggered with corresponding events from all body components and the other revolute joint. During the contact event, the number of continuous-time states is reduced.

The transition from the free mode to the fixated mode by the inelastic impulse is modeled in a similar manner. The trigger for this transition is a contact signal that becomes true when the angle `phi` exceeds the parameterized limit `l`. The contact signal is transmitted to the connected components in order to synchronize the following event. At this event, the continuous-time equations are replaced by the equations for an inelastic impulse and the variable `toFixate` is set to true. This causes a subsequent event that changes the continuous-time mode.

The reverse transition is modeled in accordance, but here a force impulse is not required. The established fixation is released when the torque acts in the opposite direction: `t < 0`.

4 Further components

Let us put aside the model of the limited revolute. From the remaining 8 components of the trebuchet model, there are two more components that exhibit structural changes. These are the standard revolute joint and the torn body.

The standard revolute joint is significantly simpler than its limited counterpart. It does not own multiple modes for the continuous-time simulation. Just an intermediate mode is required for the impulse handling. This influences the number of continuous-time state variables during the impulse. Typically the angular velocity of the revolute joint represents a state variable but during the impulse it is discretely determined.

The component for the torn body is more interesting. It owns 3 continuous-time modes with different continuous-time state variables:

1. The body is at rest as long as the rope has not been stretched.

   State-Variables: { }

2. The body represents a pendulum as long as the release angle γ has not been reached.

   State-Variables: { φ, ω }

3. The body is free.

   State-Variables: {x, y, φ, v_x, v_y, ω}

Furthermore, the transition between mode 1 and 2 has to be modeled by an inelastic impulse acting in rope direction. Another intermediate mode is required for the handling of external impulses. Figure 6 represents the corresponding transition diagram.

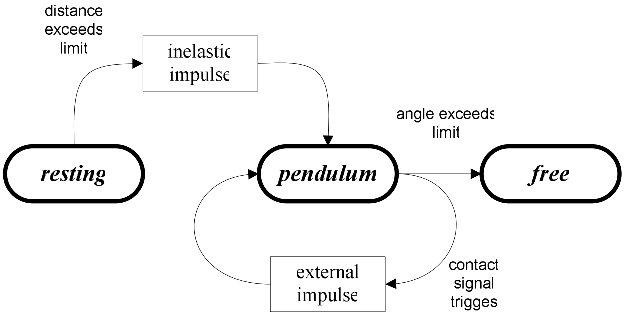


Figure 6: Mode-transitions graph of the torn body

In this way, the modeling of structural changes has been distributed on 3 of the 9 components. The object-oriented paradigm favors such a distribution. The modeling on the local level is not only easier to achieve than a complete description of the system, but also the resulting components represents meaningful entities by themselves and become usable in a generic fashion.

The modes of the total system, the trebuchet, result from the combination of its component's local modes during the simulation of the system. To get a better understanding, let us look at the simulation of the trebuchet.

[1] The symbol << represents a casual transmission - a statement that is similar to an assignment. Once applied, the variable on the left-hand side retains its value and remains determined until it gets re-determined by another causal transmission.

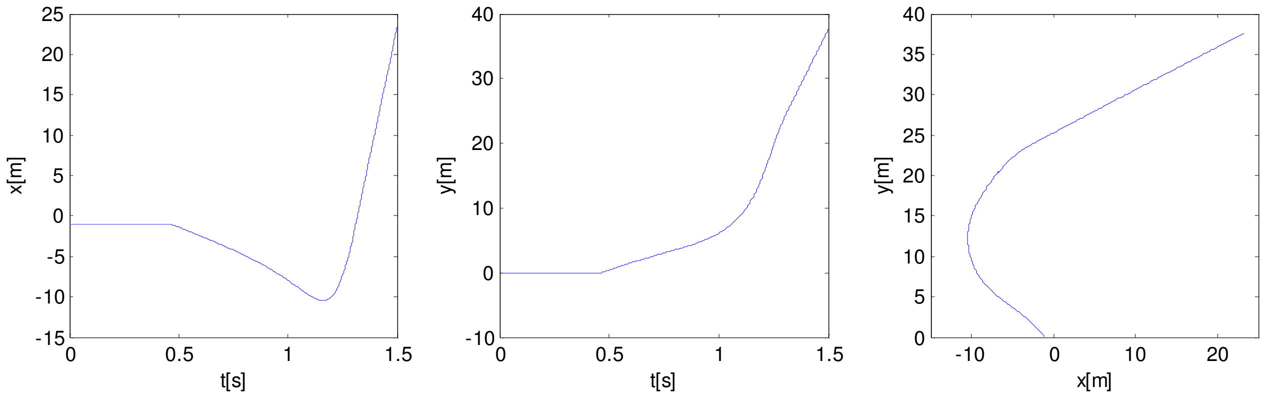


Figure 7: Trajectory of the projectile.

5 Simulation

Figure 7 presents the result of the simulation for the first 1.5 seconds. The model was simulated with Solsim, a console application that represents an interpreter of the Sol language. The main processing loop of the interpreter contains 3 stages:

- Instantiation and flattening
- Dynamic causalization
- Evaluation

In a classic Modelica translator these stages are executed once in sequential order. In Solsim, they form a loop (see also figure 9) and hence all these three stages can be repeated several times. Thus, the interpreter is able to handle almost arbitrary structural changes. All these stages are thereby programmed in way that they try to preserve the existing structure and prevent unnecessary perturbations. Further explanations can be found in [9] and in section 6.

In contrast to the elaborate processing techniques, the numerical algorithms that are included in Solsim are still on a very rudimentary level. Thus, an explicit Euler integration has been applied with a fixed step size of 1ms. In spite of this tiny step size and consequently the high number of iterations, Solsim was still able to parse, to setup and to simulate the whole system roughly within one second on common personal computer.

Let us take a look at the various structural changes that occur during the first two seconds of the simulation. Figure 8 presents an overview for the continuous-time modes and their corresponding state variables. The diagram presents the continuous-time modes for the torn body and the limited revolute with their corresponding state variables.

The transition between these modes may involve force impulses that require an intermediate mode. Such intermediate modes are depicted by a vertical line in the diagram.

The combination of modes of the components forms the modes of the complete system. In total there occur 5 modes where only 2 of them are equivalent. Furthermore, there are 2 intermediate modes for the inelastic impulses.

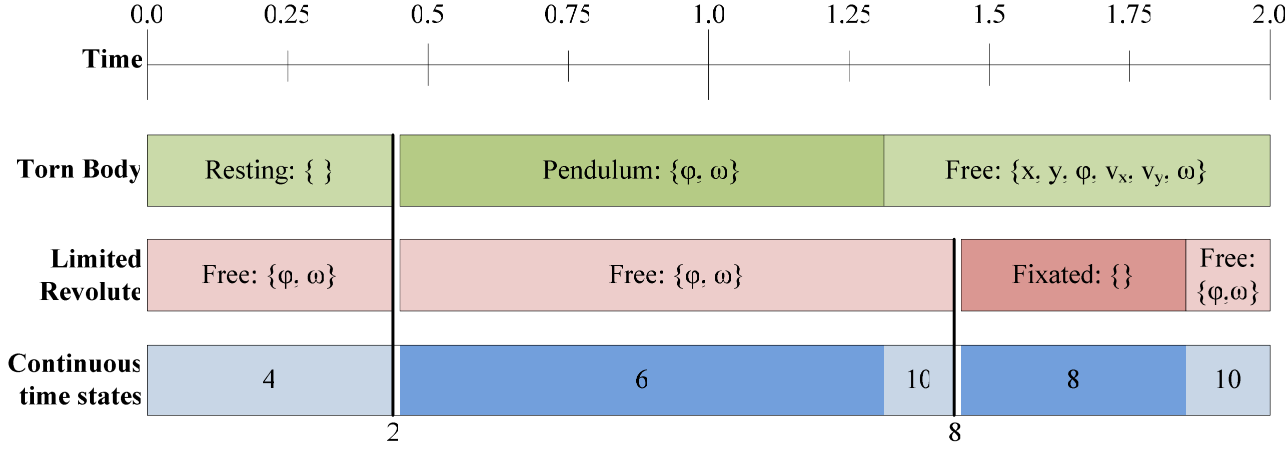


Figure 8: Timetable of the structural changes

In addition to the state variables that are listed in figure 8, there are two more state variables, namely the angle and angular velocity of the non-limited revolute joint. This holds with exception of the intermediate modes. Here, the velocities are disabled as state variables. Hence the number of continuous-time state variables in total varies from 2 to 10.

We recognize that the handling of these structural changes is a demanding problem. It contains a number of sub-tasks that need to be implemented by the simulation environment. Let us therefore review the principal processing steps and how they are affected by the variability in structure. We then continue with the integration of these tasks in the dynamic framework of Sol.

5.1 Event handling

Structural changes represent discrete events. The modeling of mechanical impulses requires that such events can be synchronized. On the other hand, the simulation environment must enable that several discrete events can be scheduled in a sequential order without any time advancement.

For this purpose, Solsim has implemented an event heap that is independent from the time integration. Time can only advance if the event heap is empty for the current time-frame. Preceding valuable contribution in this area are [2] and [5].

5.2 Sate Selection

The selection of feasible state variables is crucial for the time-integration of mechanical systems. Like Modelica, Sol also offers an option to prioritize potential state variables. The modeler can indicate preferred states and thereby support the simulation environment in its selection.

With respect to variable-structure systems, such a mechanism is especially important since a complete a priori analysis of the system might not be available or affordable in a dynamic framework.

5.3 Index reduction

In order to reduce the differential index of the DAE-system, symbolic differentiation has to be applied. Which parts that have to be differentiated depends on the current structure of the system. For instance, some equations of the torn body require differentiation while being in mode 2. After the transition to mode 3, no differentiations for this component are required anymore. Thus, Solsim keeps track of the required derivatives during the simulation.

The standard procedure for index reduction is known as Pantelides [6] algorithm. This algorithm presumes all potential state variables to be known and differentiates the occurring constraint equations.

This procedure is unfortunately inadequate for variable-structure systems. Therefore a different approach is implemented in Solsim: State variables are assumed a priori as unknown and the subsequent state selection is then integrated in the standard causalization procedure.

5.4 Tearing

For computational reasons, a transformation of the system into block-lower-triangular (BLT) form is aspired. The Dulmage-Mendelson permutation [7] is the most well known algorithm for this task, whose central part is the strong component analysis of the Tarjan algorithm [8]. This step identifies the blocks of the BLT. In a subsequent step, tearing variables may be chosen for the blocks that enable the application of iterative solvers.

Such a multi-step algorithm is not suited for a dynamic framework as Sol. Hence Solsim applies the tearing directly on the complete system and identifies the resulting blocks by the corresponding residuals. The block decomposition is therefore not necessarily optimal but mostly still adequate.

Simple heuristics are applied for the selection of tearing variables. Furthermore the modeler has the option to indicate suitable choices for tearing. For solving the corresponding equation system, Solsim applies a simple iterative solver.

6 Dynamic DAE Processing

Figure 9 presents the main processing scheme of the Solsim interpreter. Its centerpiece is the loop that consists in instantiation, dynamic causalization and evaluation. The evaluation of the system can be triggered by the algorithm for time integration or by the event handler. The evaluation of certain statements (e.g. an if-statement), may then involve the creation or removal of certain components and their corresponding equations. These changes need then to be dynamically handled by the processor for differential-algebraic equations.

Essentially, it is this dynamic DAE-processor (DDP) that defines Solsim's capabilities and enables the proper and efficient handling of even severe structural changes. The DDP takes the changes in the set of equations as input and generates a causality graph as output.

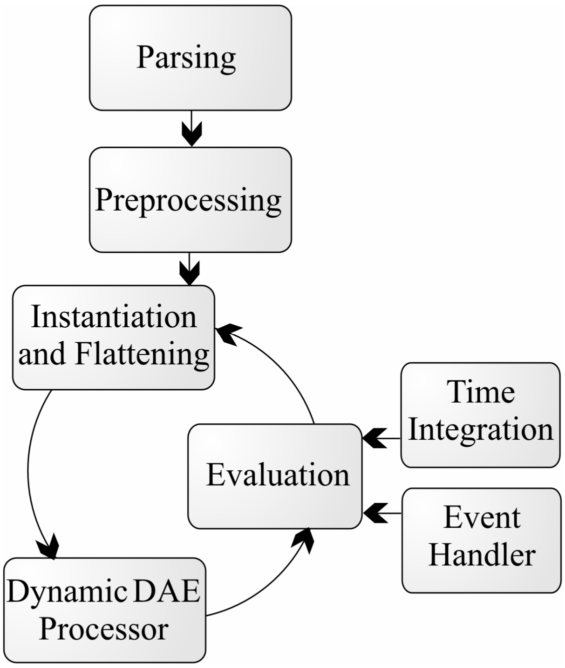


Figure 9: Processing scheme of Sol

The causality graph $G(E,V)$ is a directed acyclic graph where the vertices V correspond to the equations. The edges E are formed by those pairs of equations (s_1, s_2) where v is a variable of s_2 and determined by s_1.

Since the causality graph is an acyclic graph, it gives rise to a partial order on its vertices and can thus be used to schedule the set of causalized equations into an appropriate order for evaluation. The causality graph thereby enables the complete or partial update of a system and brings the system in a form that is suitable for numerical ODE solvers.

Any change in the set of equations will yield to an update of the causality-graph. The new equations need to be causalized and integrated into the graph. In the worst case, the exchange of a single equation will require the update of the whole system. Most changes, however, only affect parts of the system. In order to handle all these cases in an efficient manner, the dynamic DAE processor is strongly optimistic and tries to preserve the existing graph structure as much as possible.

The DDP essentially represents a set of update rules and graph-algorithms that trace each change in the set of equations and keep track of the current causalization. This has a profound impact on the handling of the tasks that have been outlined in section 5.2 to 5.4 (state selection, tearing, and differentiation).

Furthermore, the reverse counterparts of these tasks must be concerned too. The determination of a tearing variable can become obsolete and the tearing needs to be undone. The situation is similar for variables that have been selected as state variables. Also the time-derivative of a variable may not be required anymore if a change in set of equations occurs and shall therefore be eliminated.

In the DDP, the handling of all these tasks is not pursued by individual algorithms anymore. Instead, the corresponding processes are formulated as a closely interlinked set of update and downdate rules. This results in a rather complicated processing mechanism that is concerned with a good number of details. Unfortunately, this prevents any simple presentation of the DDP's functionality and hence it goes beyond the extent of this paper. For this reason, we aspire a journal publication in multiple parts and hope to publish it soon.

We can, however, outline the major principle of the DDP. In the first place, the DDP retains the causality graph as much as possible. To this end, equations remain potentially causalized, even if they lost their 'causal root'.

For any new equation, the DDP attempts its integration into the existing causality graph. This may lead to premature or speculative causalizations. In consequence, residuals may yield from overdetermined equations.

Whenever a residual is generated, their correspondent sources of overdetermination are examined. Based on this analysis, an appropriate action is taken in order to eliminate the overdetermination. This action is distinct from case to case. It can represent the undoing of former causalizations or state selections but also the extraction of an algebraic loop. In this way, the DDP enables the treatment of DAEs that result from variable-structure systems in an efficient manner.

7 Conclusions

The current framework of Sol represents a feasible solution for the modeling and simulation of variable-structure systems, although being rudimentary in many aspects. The example of the trebuchet demonstrates that the object-oriented modeling paradigm of Modelica can be successfully extended to variable-structure systems of higher index. The modeling of certain subparts can be quite demanding but the resulting components are fairly generic in their usage.

The Sol language by itself is even simpler than Modelica and hence major additions to the Modelica language would not be required (like state charts as in [3]). The power and expressiveness of Sol originates from the generalizations of successful Modelica concepts and not from the introduction of new paradigms.

These generalizations though, require new methods for the processing of such a model. This is a challenging task that demands new solutions for many major stages in the classic processing scheme. The simulator Solsim meets these requirements now to a sufficient extent.

Since Solsim is an interpreter it represents computational overkill for many specific applications and thus cannot be applied yet for computationally very demanding applications. Instead, it represents a truly general framework that can be applied to a broad range of models from various domains. We think that this approach is more promising in the long term, since specializations can still be implemented when necessary.

For instance, there is a sub-class of Sol models that is decomposable into a reasonably constrained number of modes. The trebuchet belongs to this sub-class. For such models, code corresponding to each mode can be compiled in advance and then be executed. There would be no principal problem in detecting members of this sub-class and alter the translation accordingly. For other cases, a just-in-time compilation may be desired. Corresponding solutions are meanwhile developed in the framework of Hydra [1].

Both the language Sol and the corresponding software Solsim need further extensions, refinement and optimization. But most of our future effort is planned for the completion of the whole framework. The Sol project shall be made openly accessible in a well-documented state. We thereby hope to establish a promising field for future research that lets us and other researchers elaborate new modeling and processing techniques.

Acknowledgments

I would like to thank Prof. Dr. François E. Cellier for his helpful advice and support. This research project is sponsored by the Swiss National Science Foundation (SNF Project No. 200021-117619/1).

References

[1] Giorgidze, G., H. Nilsson: Higher-Order Non-Causal Modelling and Simulation of Structurally Dynamic Systems. In: *Proc. 7th International Modelica Conference*, Como, Italy (2009)

[2] Nikoukhah, R., S. Furic: Synchronous and asynchronous events in Modelica: proposal for an improved hybrid model. In: *Proc. 6th International Modelica Conference* (2008) Bielefeld, Germany, Vol.2, 677-690.

[3] Nytsch-Geusen, C., et al.: Advanced modeling and simulation techniques in MOSILAB: A system development case study. In: *Proceedings of the Fifth International Modelica Conference*, Vienna, Austria (2006) Vol. 1, 63-71.

[4] Otter, M., H. Elmqvist and S.E. Mattsson: The New Modelica MultiBody Library. In: *Proc. 3rd International Modelica Conference*, Linköping, Sweden (2003), 311-330.

[5] Otter, M., H. Elmqvist and S.E. Mattsson: Hybrid Modeling in Modelica Based on the Synchronous Data Flow Principle. In: *Proc. IEEE International Symposium on Computer Aided Control System Design*, (1999) Hawaii, 151-157.

[6] Pantelides, C.: The Consistent Initialization of Differential-Algebraic Systems. In: *SIAM J. Sci. and Stat. Comput.* (1988) Vol 9, No. 2, 213-231.

[7] Pothen, A., Chin-Ju Fan: Computing the Block Triangular Form of a Sparse Matrix. In: *ACM Transactions on Mathematical Software* (1990) Vol 16, No. 4 303-324.

[8] Tarjan, R.: Depth-first search and linear graph algorithms. In: *SIAM Journal on Computing*. (1972) Bd. 1, No. 2, 146-160.

[9] Zimmer, D.: Introducing Sol: A General Methodology for Equation-Based Modeling of Variable-Structure Systems In: *Proc. 6th International Modelica Conference*, Bielefeld, Germany, (2008) Vol.1, 47-56

[10] Zimmer, D.: Enhancing Modelica towards variable structure systems. In: *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)*, Berlin, Germany (2007) 61-70

[11] Zimmer, D. and F.E. Cellier: The Modelica Multi-bond Graph Library, *Proc. 5th International Modelica Conference*, Vienna, Austria (2006) Vol.2, 559-568.

Biography

Dirk Zimmer received his MS degree in computer science from the Swiss Federal Institute of Technology (ETH) Zurich in 2006. He gained additional experience in Modelica and in the field of modeling mechanical systems during an internship at the German Aerospace Center DLR 2005. Dirk Zimmer is currently pursuing a PhD degree with a dissertation related to computer simulation and modeling under the guidance of Profs. François E. Cellier and Walter Gander. His current research interests focus on the simulation and modeling of physical systems with a dynamically changing structure.

Initial Value Calculation for DAE with Higher Index

Andreas Uhlig, Torsten Blochwitz, Uwe Schnabel, Tobias Nähring
ITI GmbH
Webergasse 1, 01067 Dresden, Germany
{Uhlig, Blochwitz, Schnabel, Naehring} @iti.de

Abstract

The solution of Differential-Algebraic Equations (DAEs) requires the calculation of initial values at the beginning of the simulation as well as after discontinuities. An approach is described that allows consistent initial value calculation (IVC) for higher index DAE with structural changes. To consider rigid impacts integral equations for the conservation of momentum are automatically generated. The method includes handling of fixed initial values and the observance of feasible regions.

Keywords: Initial Value Calculation, DAE, Index calculation, Impulsive Distribution

1 Introduction

A great variety of physical-technical systems can be expressed and modeled by Differential Algebraic Equations (DAEs). System simulation tools, especially Modelica simulators, have to solve DAEs of various structures. The general form of DAEs is written as

$$F(x,\dot{x},t)=0 \qquad (1)$$

with the vector of state variables $x \in \mathbf{R}^n$, the vector of the time derivatives $\dot{x}$ and the time t. The integration procedure requires dim (F) = n (but it system does not necessarily depend on all components of $\dot{x}$, see example 1). The first task for the time domain simulation is to find a *consistent initial value* at the start time t_0, i.e., a point x_0 for which a solution curve $x(t)$ of (1) with $x(t_0)=x_0$ exists. If the Jacobian matrix of partial derivatives $\partial F/\partial \dot{x}$ is regular everywhere then (1) defines an implicit system of Ordinary Differential Equations (ODE). With ODE any vector of initial values x_0 is consistent since (1) can locally be resolved for $\dot{x}$. The situation is different in the case of DAE since $\partial F/\partial \dot{x}$ can be singular. The system has *algebraic constraints* if $x \in \mathbf{R}^n$ exist for which there is no algebraic solution of (1) for the unknown $\dot{x}$. Additionally, the system has hidden constraints if there are algebraic solutions $x,\dot{x}$ of (1) for which x is not a consistent initial value, i.e., $\dot{x}$ is not the time-derivative of any solution curve.

Example 1: Consider the system

$$x_1=t, \quad x_2=\dot{x}_1 \qquad (2)$$

At $t=t_0$ a vector $x \in \mathbf{R}^2$ must fulfill the equation $x_1=t_0$ to correspond to an algebraic solution of (2). Therefore, the first equation imposes an algebraic constraint. Not all algebraic solutions are consistent initial values since regarding $\dot{x}_1$ as the time-derivative of x_1 further restricts x by $x_2=\dot{x}_1=\dot{t}=1$. Therefore, the second equation of (2) constitutes a hidden constraint.

In complex systems user often cannot determine consistent initial values for each of the variables because some of them must meet algebraic and hidden constraints. On the other hand he must insist on (i.e. fix) the initial values of certain variables. In this situation tools provide support by allowing *fixed* and *not fixed* values, the latter being used as guesses for the initial value calculation (IVC) executed by the software. The determination of initial values is treated in [1], [5], and [7]. However, the initial values have to be calculated not only at the beginning $t=t_0$ but also after discontinuities ([6]). Now the values at/before the discontinuity play the role of the initial values. Again one has to distinguish, which of these values are fixed and which may change. In this paper we consider an approach that determines consistent initial values from this input.

Figure 1 gives an overview about the main steps of simulation. Before a Modelica model is treated by a numerical solver symbolic simplifications and transformations are carried out (second block in Figure 1). During this process symbolic index reduction is applied in order to transform the system into

DOI: 10.3384/ecp09430122

an ODE or at least into a DAE with index 1 (see [1]). Here we refer to the *differentiation index* that describes how many times a DAE must be differentiated until - after some transformation - an ODE is obtained. The symbolic index reduction carries out these differentiations as far as possible. If it leads to a system without hidden constraints the remaining algebraic constraints can be solved within implicit blocks and so the system can be handled by standard ODE solvers. In this case the computation of consistent initial values is straightforward.

After the symbolic analysis of the model the numerical time-domain simulation starts (third block in Figure 1). It includes handling of discrete-time events through event iterations and time-continuous integration of the DAE-system (1). Time-discrete events are triggered when inequalities change their logical value depending on DAE-states and/or time. Discrete variables and equations describe the behaviour of the system at such events. A change of the time-discrete state may modify the DAE-system (1). As explained below, that calls for a numerical index reduction and a calculation of new consistent initial values. Time-discrete and time-continuous states influence each other. An event iteration takes place until the time-discrete state of the system is stabilized. After that the simulation continues with the time-integration of the DAE.

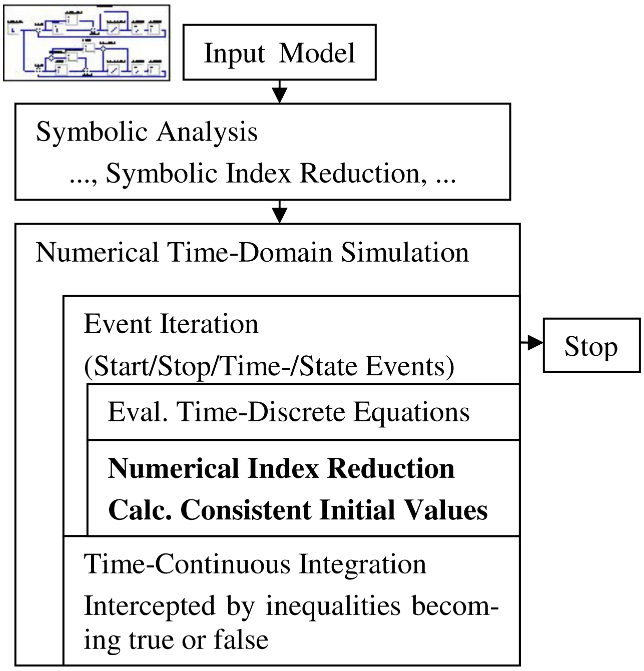

Input Model
Symbolic Analysis
..., Symbolic Index Reduction, ...
Numerical Time-Domain Simulation
Event Iteration
(Start/Stop/Time-/State Events)
Eval. Time-Discrete Equations
Numerical Index Reduction
Calc. Consistent Initial Values
Time-Continuous Integration
Intercepted by inequalities becoming true or false
Stop

Figure 1: Course of simulation flow

There are situations where the symbolic index reduction cannot differentiate certain equations analytically and some hidden constraints remain in the system for the numerical time-domain simulation. The most important cases are the following:

- *External Functions* do not provide symbolic expressions for equations or even their derivatives.
- If the system contains structural changes (that may happen through *conditional equations* in Modelica as explained in Example 2) the solver is faced with several branches having different indexes. Here symbolic index reduction does not solve the problem. To guarantee a low index all combinations of valid branches would have to be considered separately. In the worst case the analytical effort of regarding all branches increases exponentially. This is not practicable for large models.

Example 2: Consider the system with the single conditional equation

$$1 = \text{if } t < 0.5 \quad \text{then } x \text{ else } \dot{x}\,.$$

For t<0.5 the first branch is active and the differentiation index is 1. After that the second branch causes differentiation index 0. The symbolic index reduction does not distinguish between the two branches and wrongly deduces that the equation depends always on both variables x and $\dot{x}$ and the system is classified as to be of index 0. Therefore, the equation is not symbolically differentiated but the IVC cannot compute $\dot{x}$ at $t = 0$ without differentiation of the equation.

If the symbolic index reduction cannot free the system from hidden constraints it has to be supplemented by *numerical index reduction*. For this end it is necessary to calculate derivatives numerically.

In such cases ODE solvers cannot be applied directly for the simulation of the system.

Beyond this, there are more challenges. With impact events in mechanical models (see section 3) *conservation of momentum* is expected. Since the respective equations are normally not part of the model, they have to be deduced numerically from the equations of motion and considered during reinitialization after such events.

Even after index reduction some *algebraic equations* may remain. Furthermore, higher derivative variables often occur *nonlinear*. For both reasons the solution of a nonlinear system is required for the IVC and considered briefly in this paper.

At last, we discuss approaches to meet inequality constraints, that might be defined for some states x_i.

2 Numerical Index Reduction

For the description of the numerical index reduction we assume that (1) is the active set of DAE-equations after a step of the event iteration at time $t = t_d$. As explained in section 1 the system may still have hidden constraints. To transform these for the IVC into algebraic constraints the numerical index reduction supplements the original system with time-difference quotients of selected equations from (1). Thereby, the values of x and $\dot{x}$ after the event time t_d are expressed as the result of one or two steps of the Euler-forward method.

2.1 Index Calculation

The index calculation is the first part of the numerical index reduction. It determines which equations F_i and which states x_j have to be differentiated how many times. The algorithm can be divided into two steps:

1st: *assignment of equations to variables*
2nd: *determination of the number of necessary differentiation*s.

Assignment of equations to variables:
Roughly spoken the equations are assigned to variables (or their time derivatives) which 'can be calculated from their equations'. The *best assignment* is found with the help of the following discrete optimization procedure:
Every assignment of equations to variables can be described by a permutation σ of the numbers 1 to n. Here, $\sigma(j) = i$ means that F_i is assigned to x_j. We denote the set of all such permutations as $\mathrm{Perm}(n)$.
From the current numerical Jacobian matrices $\partial F / \partial x$ and $\partial F / \partial \dot{x}$ a cost matrix C is composed:

$$C_{ij} := \begin{cases} -n-2 & \text{if } \partial F_i / \partial \dot{x}_j \neq 0 \\ -n-1 & \text{if } \partial F_i / \partial x_j \neq 0 \wedge \partial F_i / \partial \dot{x}_j = 0 \\ 0 & \text{else} \end{cases}$$

An entry C_{ij} causes rather high costs if F_i does not depend on x_j. Otherwise, the caused costs are lower for F_i depending on the time derivative of x_j than only depending on x_j.

The σ that minimizes the overall costs $\sum_{j=1}^{n} C_{\sigma(j),j}$ is the chosen best assignment of equations to variables. For the minimization one can apply the algorithm from [3].

Determination of the number of necessary differentiations:

The number of necessary differentiations of equations is determined by an iterative procedure. How the equation F_i of the original system depends on the variables x_j is determined the numerical Jacobian and stored in the dependency matrix

$$D_{ij}^{0} := \begin{cases} 1 & \text{if } \partial F_i / \partial \dot{x}_j \neq 0 \\ 0 & \text{if } \partial F_i / \partial x_j \neq 0 \wedge \partial F_i / \partial \dot{x}_j = 0 \\ -\infty & \text{else} \end{cases}$$

for $i, j = 1,...,n$. Starting from D^0 a sequence of dependency matrices D^k for derived systems with differentiated equations is generated.

In the following we list the algorithm in 'quasi-Modelica' formulation:

```
$k := 0$; // Iteration index.
// $D^0$ already defined above.
while true loop
    for $j$ in $1:n$ loop
      // Determine the highest derivative of $x_j$ that
      // explicitly occurs within the current system:
      $d_j^k := \max\left\{ D_{i,j}^k \mid i=1,...,n \right\}$
    end for;
    for $j$ in $1:n$ loop
      if $D_{\sigma(j),j}^k > -\infty$ then // $x_j$ in eq. $\sigma(j)$?
         // Differentiate the $\sigma(j)$-th equation:
         for i in 1:n loop
            $D_{\sigma(j),i}^{k+1} := D_{\sigma(j),i}^k + d_j^k - D_{\sigma(j),j}^k$;
         end for;
      end if;
    end for;
   // Stop if no further differentiations were needed:
    if $D^{k+1} == D^k$ then break; end if;
    $k := k + 1$;
```

end while;
If $D^0_{\sigma(i),j} > -\infty$ then in each iteration the matrix entry $D^k_{\sigma(i),j} > -\infty$ stands for the order of the highest time-derivative $x_j^{\left(D^k_{\sigma(i),j}\right)}$ present in the $d := D^k_{\sigma(i),i} - D^0_{\sigma(i),i}$ times differentiated equation $F^{(d)}_{\sigma(i)}$. So, at the end of the algorithm one can read off the number of needed differentiations of equations and variables.
The algorithm ensures that the highest derivatives of the variables occur in the highest derivatives of their assigned equations if this is possible at all.

It determines the needed additional equations and variables for the re-formulated index-reduced DAE system.

The advantage of the above algorithm over the algorithm of Pantelides [1] is that it also handles singular systems.

2.2 Numerical Differentiation of Equations

The index calculation from section 2.1 tells us which equations of system (1) at $t = t_d$ have to be numerically differentiated for index reduction and if so, how many times. Only first and second order information is numerically generated. So, three is the highest differentiation index that can be handled by the solver itself.

If equation F_i is marked for differentiation the overall system is supplemented by the equation

$$0 = \frac{F_i(x + dt \cdot \dot{x}, \dot{x} + dt \cdot \ddot{x}, t_d + dt) - F_i(x, \dot{x}, t_d)}{dt}.$$

If second order information is needed the result of two Euler-forward steps is added, too:

$$0 = \Big[F_i\Big(x + 2dt \cdot \dot{x} + dt^2 \cdot \ddot{x},\\ \dot{x} + 2dt \cdot \ddot{x} + dt^2 \cdot \dddot{x}, t_d + 2dt\Big) -\\ - F_i(x, \dot{x}, t_d)\Big]/(2dt)$$

Here, the quantities in these equations have the following meaning:

x	…	value of the state variable at t_d
$\dot{x}$, $\ddot{x}$, $\dddot{x}$	…	1st, 2nd, and 3rd derivatives at t_d
dt	…	step size, automatically selected by the initial value solver

3 Conservation of Momentum

3.1 Motivation

To motivate the necessity of integral equations in simulation we shortly consider the equations of two centrally colliding soft elastic bodies which are only under the influence of the contact force F_C. The equations of motion of the two bodies are

$$\begin{aligned} m_1\dot{v}_1 &= F_C, & \dot{x}_1 &= v_1, \\ m_2\dot{v}_2 &= -F_C, & \dot{x}_2 &= v_2. \end{aligned} \tag{3}$$

And the behavioral description of the contact force is

$$F_C = \text{if } x_1 > x_2 \text{ then } k(x_2 - x_1) \text{ else } 0. \tag{4}$$

Here, we implicitly assume that body 1 with mass m_1 approaches with some start velocity v_{10} from the left hand side $(x_1 < x_2)$ while body 2 with mass m_2 from the right hand side with some start velocity $v_{20} < v_{10}$.

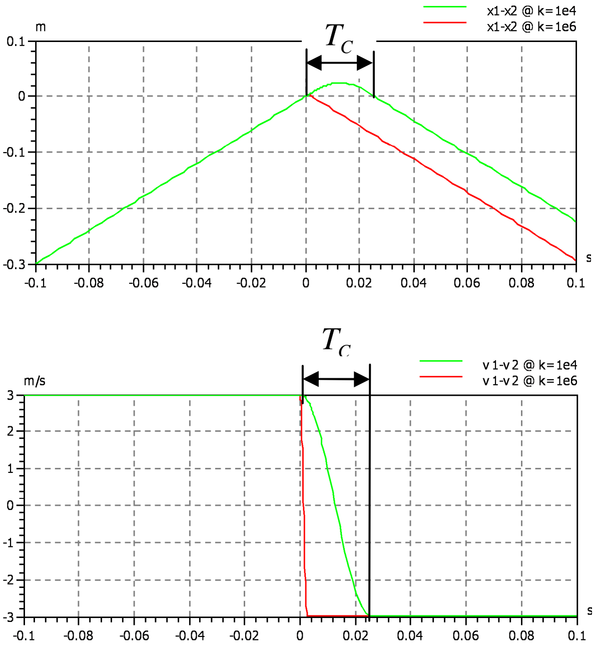


Figure 2: Displacement $x_1 - x_2$ and difference velocity $v_1 - v_2$ of the two masses for $k = 10^4$ N/m (green) and $k = 10^6$ N/m (red)

Figure 2 shows the displacement and the difference velocity for two different values of the contact stiffness. The higher the stiffness the shorter the impact time T_C and the maximal deformation becomes smaller. Furthermore, the difference velocity $v_1 - v_2$ changes sign within the contact phase.

In many practical situations the contact stiffness is very high such that the contact time span and the maximal deformation are relatively small compared to the time- and distance scale ,resp., of the interesting processes to be modeled. If, under these circumstances, the soft impact model (3), (4) is used, the solver is forced to reduce the computation time step size just for capturing the fast contact phase. This can be avoided by modeling the contact phase as one discrete impact event where the sign of the difference velocity changes, i.e., conceptually:

$$\text{when} \quad x_1 > x_2 \quad \text{at } t = t_d \quad \text{then}$$
$$v_1(t_d+0) - v_2(t_d+0) := -(v_1(t_d-0) - v_2(t_d-0));$$

With the jump in the difference velocity $v_1 - v_2$ also (at least one of) the velocities v_1 or v_2 must jump and they are no longer differentiable in the classical sense.

In those events the IVC algorithm reformulates the equations of motion (3) depending on $\dot{v}_1$ and $\dot{v}_2$ as integral equations:

$$\begin{aligned}0 &= \lim_{\varepsilon\to 0}\int_{t_d-\varepsilon}^{t_d+\varepsilon}(m_1\dot{v}_1(t_d) - F_C(t_d))dt\\ &= m_1(v_1(t_d+0) - v_1(t_d-0)) - \hat{F}_C\\ 0 &= \lim_{\varepsilon\to 0}\int_{t_d-\varepsilon}^{t_d+\varepsilon}(m_2\dot{v}_2(t) + F_C(t))dt\\ &= m_2(v_2(t_d+0) - v_2(t_d-0)) + \hat{F}_C\end{aligned}$$

Thereby, the new integral quantity $\hat{F}_C$ is automatically added by the solver.

One may even give the generated equations physical meaning. The new equations reflect the balance of momentum and $\hat{F}_C$ stands for the impulse exchanged by the colliding bodies at the time of impact.

In the context of impulsive distributions as introduced in [4] the force F_C is a linear combination

$$F_C(t) = \overline{F}_C(t) + \hat{F}_C \cdot \delta(t - t_d)$$

of a regular signal $\overline{F}_C$ and a Dirac-delta distribution shifted to the time of impact t_d and weighted by the integral quantity $\hat{F}_C$.

The introductory example is very simple. In practice, the simulator must be able to generate the impulse equations for much more complicated models. Special challenges are simultaneous state changes in multiple impact and end-stop elements, nonlinear transformations between masses and contact elements as well as rigid friction elements parallel to contact elements.

Figure 3 shows a SimulationX model combining all those aspects. It is a swinging pendulum (yellow) bumping against a vertically guided plate (blue) which initially lies on some end stop (gray). The nonlinear transformation of the angular position into the height of the nose of the pendulum and the friction of the vertical guide of the body are taken into account.

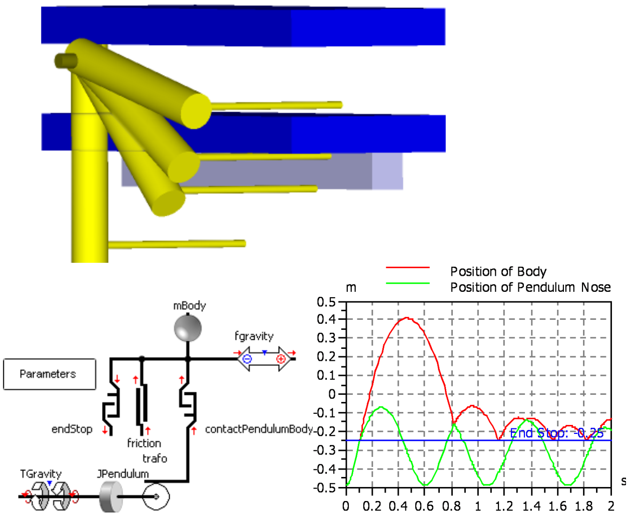


Figure 3: 3D-animation (top, multiple frames are shown), model structure (left), and results (right) of the swinging pendulum

3.2 Generation of Equations

In the transient simulation conservation of momentum is guaranteed by the equations of motion and the relation $\dot{v} = a$. In the IVC at an ideal impact this relationship is not directly used, therefore conservation of momentum must be ensured by additional conditions. For this purpose we integrate the equations of motion numerically and the Dirac-impulse in forces and accelerations resulting from the impact can be treated.

The solver needs model input to identify the impact source in the system (e.g., the contact force F_C in the example from section 3.1) and the jumping variables that determine indirectly the integral value of the impulsive variables for description of the impact (e.g., the jumping velocity difference $v_1 - v_2$ in the example). At present, this is done by a flag set by a function *SetImpact* and by an attribute *notFixed*, respectively. By evaluation of the dependencies of the system the IVC algorithm can determine the equations for which integrals are to be added:

Let $\overline{D}$ be the dependency matrix of the numerically index reduced order-one system in the current discrete state (see section 2.1). We define the dependency matrix D with rows (corresponding to equations) permuted into the order of the assigned variables: $D_{i,j} := \overline{D}_{\sigma(i),j}$. We define the vector b^0 of initial lower bounds for the impulse order. The indexes j that correspond to variables marked with *setImpact* get the lower bound $b_j := 0$. For the other variables the bounds are initially set to $b_j := -\infty$. The impulse order is found by iteratively adapting the lower bounds of variables x_i to the impulse order of the other variables and derivatives in the assigned equation $F_{\sigma(i)}$. This can be expressed as following iterative assignment

$$b^{k+1} := \text{rowmax}\left(\text{ones}(n,n)\cdot b^k + D\right) - diag(D)$$

to be run for $k = 0,1,...$ until the impulse order $I := b^k$ with $b^{k+1} == b^k$ is found.

The approach can handle even multiple impacts at the same time instance.

After the impulse order of all variables (and assigned equations) has been determined the integrals of equations $F_I(...) = 0$ with impulse order 0 are generated by the solver (currently, only equations up to impulse order 0 are handled). For the discussion we repartition the set of states and time-derivatives $x, \dot{x}$ into *regular variables* v_R and *impulsive variables* v_I. Locally, at $t = t_d$ the regular variables are piecewise continuous while the impulsive variables are linear combinations

$$v_I(t) = \overline{v}_I(t) + \hat{v}_I \delta(t - t_d)$$

of a regular part $\overline{v}_I$ and an impulsive part with the integral value $\hat{v}_I$. For time derivatives $\dot{x}_k$ that are components of v_I the integral value $\hat{\dot{x}}$ is determined by the jump height

$$\begin{aligned}\hat{\dot{x}}_k &= \int_{t_d-0}^{t_d+0}\left(\overline{\dot{x}}_k + \hat{\dot{x}}_k \delta(t - t_d)\right)dt \\ &= \left(x_k(t_d + 0) - x_k(t_d - 0)\right)\end{aligned}$$

of the corresponding state x_k at t_d. Here, the variable right limit $x_k(t_d + 0)$ is the unknown in the supplemented system of equations. For the other purely algebraic components of v_I the integral quantities $\hat{v}_I$ are additional unknowns.

One precondition for the system is that all equations depend at most quasi-linearly on impulsive variables, i.e., the partial system of these equations is representable as

$$F_I(...) \equiv A(v_R, t)\cdot v_I + F_R(v_R, t) = 0 \qquad (5)$$

with a matrix function A and a function F_R both only depending on the regular variables. It is assumed that A depends continuously on its arguments. For F_R only integrability and boundedness is required.

$$\begin{aligned}\lim_{\varepsilon\to 0}\int_{t_d-\varepsilon}^{t_d+\varepsilon} &\left(A(v_R, t)\cdot\left(\overline{v}_I(t) + \hat{v}_I\delta(t - t_d)\right) +\right. \\ &\left. + F_R(v_R, t)\right)\cdot dt = 0\end{aligned}$$

For brevity we do not note the time dependence of all variables explicitly.

Since the integrand of the partial integral

$$\lim_{\varepsilon\to 0}\int_{t_d-\varepsilon}^{t_d+\varepsilon}\left(A(v_R, t)\cdot\overline{v}_I(t) + F_R(v_R, t)\right)\cdot dt = 0$$

is bounded its integral vanishes identically for the limit $\varepsilon \to 0$ and only

$$\lim_{\varepsilon\to 0}\int_{t_d-\varepsilon}^{t_d+\varepsilon} A(v_R, t)\cdot\hat{v}_I\delta(t - t_d)dt = 0 \qquad (6)$$

remains to be considered. If v_R is continuous one directly obtains from the shifting-property of the Dirac-delta distribution the equation

$$A(v_R(t_d), t_d)\hat{v}_I = 0.$$

Things become more complicated when v_R is discontinuous at $t = t_d$. That happens with translational-rotational transmissions as in Figure 3.

This case is not directly covered by distribution theory for the following reason. Let $g : \mathbf{R} \to \mathbf{R}$ be jumping at 0 and otherwise continuous and let δ_ε be a family of L^1-approximations of the Dirac-delta distribution. Then the limit

$$\lim_{\varepsilon\to 0}\int_{-\infty}^{\infty} g(t)\,\delta_\varepsilon(t)dt$$

depends on the actual sequence of Dirac-approximations. Symmetric approximations $\delta_\varepsilon(t) = \delta_\varepsilon(-t)$ lead to

$$\lim_{\varepsilon\to 0}\int_{-\infty}^{\infty} g(t)\,\delta_\varepsilon(t)dt = (g_+ + g_-)/2.$$

with $g_\pm := \lim_{\varepsilon\to 0} g(\pm\varepsilon)$. The left- and right-hand approximations δ_ε^+ and δ_ε^- with $\delta_\varepsilon^\pm(t) = 0$ at $t \geq 0$ and $t \leq 0$, resp., lead to

$$\lim_{\varepsilon\to 0}\int_{-\infty}^{\infty} g(t)\,\delta_\varepsilon^\pm(t)dt = g_\pm.$$

As we show in the following, the symmetric approximation often matches the symbolic integration better.

Let g be sufficiently smooth. In constraint equations there are often additive terms of the form $g'(x_i(t))\dot{x}_i(t)$ with some state variable x_i. For smooth x_i the term has the anti-derivative $g(x_i(t))$. Therefore, one expects the 'symbolical integral' to be

$$I_S := \int_{t_d-0}^{t_d+0} g'(x_i(t))\dot{x}_i(t)dt = g(x_{i+}) - g(x_{i-})$$

even if the state x_i jumps at t_d. Expressing the integral with the help of the symmetric Dirac approximation one gets the 'numerical integral'

$$I_N := \int_{t_d-0}^{t_d+0} g'(x_i(t))(\bar{x}_i(t) + (x_{i+} - x_{i-})\delta(t - t_d))dt$$
$$= \tfrac{1}{2}(g'(x_{i+}) + g'(x_{i-})) \cdot (x_{i+} - x_{i-})$$

where $\bar{x}_i$ is the continuous part of x_i. Taylor series expansion of both integrals I_N and I_S for $\Delta x_i := x_{i+} - x_{i-}$ at $x_{i\mathrm{m}} := (x_{i+} + x_{i-})/2$ gives $g'(x_{i\mathrm{m}}) \cdot \Delta x_i + O\left(|\Delta x_i|^3\right)$. Therefore, both integrals are equal up to order two. For the corresponding limit of the one-sided Dirac approximation the numerical integrals match the symbolical integral only up to order one.

The foregoing remarks motivate that for the integral equation (6) with jumping signals v_R the result

$$\tfrac{1}{2}(A(v_{R-}, t_d) + A(v_{R+}, t_d))\hat{v}_I = 0 \qquad (7)$$

of the symmetric Dirac approximation should be used. Actually, we implemented

$$A\left(\tfrac{1}{2}(v_{R-} + v_{R+}), t_d\right)\hat{v}_I = 0 \qquad (8)$$

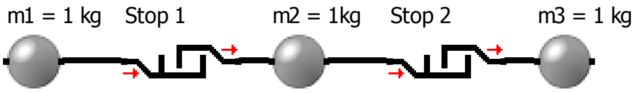
which is a second order approximation of (7) in the jump height $v_{R+} - v_{R-}$ at the midpoint $v_{R\mathrm{m}} := \frac{1}{2}(v_{R+} + v_{R-})$. As we have seen above the change is not crucial for the approximation and the implementation (8) leads to less function evaluations than (7).

DAE-systems with impacts result from structural changes. So the matrix A in the left-hand side of (5) cannot be easily identified through the symbolic analysis in advance. The integral equations (8) must be composed by evaluations of the original left-hand sides F_I at appropriate arguments. One way to represent the additional integral equations is:

$$F_I(v_{R\mathrm{m}}, \hat{v}_I, t_d) - F_I(v_{R\mathrm{m}}, 0, t_d) = 0 .$$

Hereby (as already mentioned above), the $v_{R\mathrm{m}}$ are the mean values $v_{R\mathrm{m}} := \frac{1}{2}(v_{R+} + v_{R-})$ of the left and the right limits of the regular variables at t_d. For continuous variables $v_{R,k}$ (e.g., non-jumping states with derivatives) $v_{R\mathrm{m},k} = v_{R+,k} = v_{R-,k}$. For the other variables $v_{R,k}$ the right limits $v_{R+,k}$ are unknowns in the extended system of equations (see section 4).

Example 3 (Conservation of Momentum):
Given are two end stops and three masses



The two end stops should have an impact at the same time with the impact coefficient ci_1 and ci_2. Then the DAE for the impact is

$$0 = \dot{x}_1 - \dot{x}_2 + ci_1(\dot{x}_1(t_d - 0) - \dot{x}_2(t_d - 0))$$
$$0 = \dot{x}_2 - \dot{x}_3 + ci_2(\dot{x}_2(t_d - 0) - \dot{x}_3(t_d - 0))$$
$$0 = m_1\ddot{x}_1 + F_1$$
$$0 = -F_1 + m_2\ddot{x}_2 + F_2$$
$$0 = -F_2 + m_3\ddot{x}_3$$

Here the velocities $\dot{x}_1$, $\dot{x}_2$, and $\dot{x}_3$ have a jump and the accelerations $\ddot{x}_1$, $\ddot{x}_2$, and $\ddot{x}_3$, and the forces of the end stops F_1 and F_2 are infinite. The numerical index reduction supplements the system with the difference quotients of the first two equations:

$$\begin{aligned}0 = ((\dot{x}_1 + dt \cdot \ddot{x}_1) - (\dot{x}_2 + dt \cdot \ddot{x}_2) + \\ + ci_1(\dot{x}_1(t_d - 0) - \dot{x}_2(t_d - 0)) - \\ - (\dot{x}_1 - \dot{x}_2 + ci_1(\dot{x}_1(t_d - 0) - \dot{x}_2(t_d - 0))))/dt \\ = \ddot{x}_1 - \ddot{x}_2\end{aligned}$$

and

$$\begin{aligned}0 = ((\dot{x}_2 + dt \cdot \ddot{x}_2) - (\dot{x}_3 + dt \cdot \ddot{x}_3) + \\ + ci_2(\dot{x}_2(t_d - 0) - \dot{x}_3(t_d - 0)) - \\ - (\dot{x}_2 - \dot{x}_3 + ci_2(\dot{x}_2(t_d - 0) - \dot{x}_3(t_d - 0))))/dt \, . \\ = \ddot{x}_2 - \ddot{x}_3\end{aligned}$$

Furthermore, the following integral equations are added to handle the velocity jumps in the end stops:

$$0 = m_1\dot{x}_1 + \hat{F}_1 - (m_1\dot{x}_1(t_d - 0) - F_1(t_d - 0)),$$
$$0 = -\hat{F}_1 + m_2\dot{x}_2 + \hat{F}_2 - (-F_1 + m_2\dot{x}_2 + F_2)(t_d - 0)$$

and

$$0 = -\hat{F}_2 + m_3\dot{x}_3 - (-F_2(t_d - 0) + m_3\dot{x}_3(t_d - 0))$$

From this we get the 10 unknowns: the velocities $\dot{x}_1$, $\dot{x}_2$, and $\dot{x}_3$, the accelerations $\ddot{x}_1$, $\ddot{x}_2$, and $\ddot{x}_3$, the forces of the end stops F_1 and F_2 and their integrals $\hat{F}_1 - F_1(t_d - 0)$ and $\hat{F}_2 - F_2(t_d - 0)$.

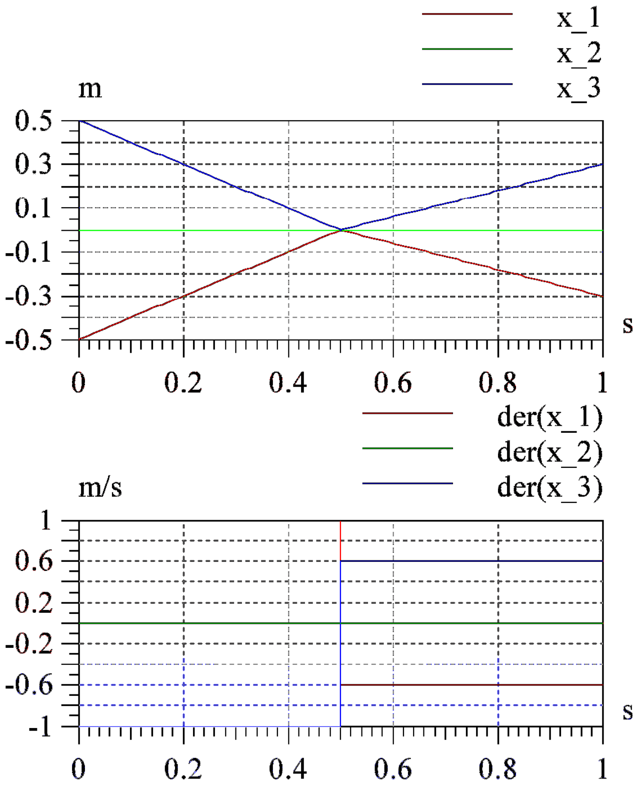


Figure 4: Simulation results for the displacements (top) and velocities (bottom) of the masses

Here it is $ci_1 = ci_2 = 0.6$. Other independent choices of ci_1 and ci_2 in the interval $[0;1]$ are possible. This shows that multiple impacts at the same time instant can be handled.

4 Fixed Initial Values

For $t = t_0$ it can be setup externally (Modelica attribute) which initial values of x and $\dot{x}$ are to be treated as *fixed*. Then they are not variables for the IVC. Moreover, there are rules for the solver to indicate and treat certain states as *fixed* or *not fixed*. Often a user would like to see the guesses for the *not fixed* variables as preferred results of the IVC at $t = t_0$. Therefore, in a first attempt the IVC algorithm can fix those ODE states which are not required for other fixations and initial equations. Only if this was not successful these experimentally fixed variables are released again.

The more interesting situation is after a discontinuity. Here a state x_j (same conditions hold for derivatives $\dot{x}_j$) is treated as *fixed* if

(i) it is specified as *fixed* externally or equation (1) depends on a higher derivative (which also can be a state or even dummy derivative; see [2]) and

(ii) the state is continuous and

(iii) the state is not externally defined as *not fixed.*

Clause (ii) is applied because variables with a discontinuity or infinite value (see section 3) cannot be treated as fixed.

If a higher derivative occurs in (1), that is not infinite, the state itself is continuous. Therefore, the second part of clause (i) fixes it. This also applies if the higher derivative is a dummy (see [2]), since in reality the relationship exists.

5 Solving the System of Equations

The final system of equations is solved by a Newton method. In the situations described in sections 2 and 3 additional equations and sometimes also variables are appended to (1). On the other hand, due to fixation of variables in the previous section such variables are omitted. Thus, as a rule the system of equation has rectangular shape.

Under-determined Systems: With an infinite number of solutions we can work with the minimum-norm solution, i.e.

$$\min\left\{\|s\|_2 : Jac * s = -res\right\}.$$

This does not need to be critical, e.g. when derivatives of certain states are not calculated in the course of IVC since they are used nowhere, as shown in the example below.

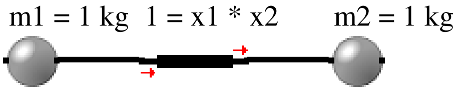
Example 4:

Let velocity v_i be a state. Its derivative $\dot{v}_i$ is the acceleration a_i. a_i is a state too, if it is a dummy-derivative, i.e. there is no equation $\dot{v}_i = a_i$. In the system of equation the derivative of v_i is always expressed by a_i. Thus $\dot{v}_i$ does not occur in the system of equation and hence cannot be calculated in the IVC.

Even over-determined systems, if correctly modeled can be solved after a discontinuity (Example 5).

Example 5:

Consider two masses linked by a mechanical constraint.



Equation before index reduction:

$$0 = m_1\ddot{x}_1 + m_2\ddot{x}_2$$

$$1 = x_1 x_2$$

Index reduction adds 1. and 2. derivatives:

$$0 = \dot{x}_1 x_2 + x_1\dot{x}_2$$

$$0 = \ddot{x}_1 x_2 + 2\dot{x}_1\dot{x}_2 + x_1\ddot{x}_2$$

If there is (caused by others parts of a model) a discontinuity at t_d then all of the states $x_1(t_d)$, $x_2(t_d)$, $\dot{x}_1(t_d)$, $\dot{x}_2(t_d)$ are fixed, i.e. keep the value of $(t_d - 0)$. $\ddot{x}_1(t_d)$ and $\ddot{x}_2(t_d)$ are variable. The system has 4 equations and 2 variables.

$$0 = m_1\ddot{x}_1(t_d) + m_2\ddot{x}_2(t_d)$$

$$1 = x_1(t_d - 0)x_2(t_d - 0)$$

$$0 = \dot{x}_1(t_d - 0)x_2(t_d - 0) + x_1(t_d - 0)\dot{x}_2(t_d - 0)$$

$$0 = \ddot{x}_1(t_d)x_2(t_d - 0) + 2\dot{x}_1(t_d - 0)\dot{x}_2(t_d - 0) + \\ + x_1(t_d - 0)\ddot{x}_2(t_d)$$

The second and the third equation are not changed at the discontinuity. So initial values for $\ddot{x}_1$ and $\ddot{x}_2$ can be determined from the first and fourth equation. Like in this example the surplus equations are often satisfied automatically. Thus, even a rectangular system can have a solution.

6 Ensuring the Feasible Region

Derived from real world conditions inequality constraints, especially lower and upper bounds, may be attached to the variables of a DAE:

$x_i > x_{i,Min}$ and $x_i < x_{i,Max}$

(If equality shall be included (e.g. $x_i \geq x_{i,Min}$) we use the relation $x_i > x_{i,Min} - absTol * ScaleFac$, with suitable constants.)

Our first choice – if possible - is to make use of the kernel of the system matrix. If the originally calculated correction s would lead to

$x_i^{new} \leq x_{i,Min}$ or $x_i^{new} \geq x_{i,Max}$ with $x_i^{new} = x_i^{old} + s_i$ and $x_{i,Min} < x_i^{old} < x_{i,Max}$

then we determine a Newton update s^N parallel to the border of the feasible region. We set $s_i^N = 0$ and calculate the rest of s^N from

*minimum norm solution + kernel vector * factor*

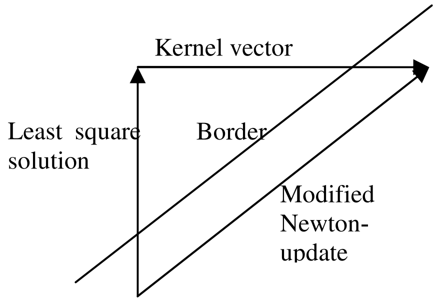


Figure 5: Newton update parallel to border of feasible region

If this approach is not applicable (no kernel) a damping factor α with $0 < \alpha < 1$ shall decrease the update vector $s^N = \alpha \cdot s$, such that for a given λ with $0 < \lambda < 1$

$$x_{i,Min} < (1-\lambda) * x_{i,Min} + \lambda * x_i^{old} \leq x_i^{old} + \alpha * s_i \\ \leq (1-\lambda) * x_{i,Max} + \lambda * x_i^{old} < x_{i,Max}.$$

7 Conclusions

The solution of DAEs requires the calculation of initial values at the beginning of the simulation as well as after discontinuities. In this paper some measures for special situations were described. First, we can, e.g. in the case of external functions and structural changes, execute the necessary differentiation of certain equations numerically. Second, in order to enable ideal components additional equations for initial values were developed. Hereby the conservation of momentum for impacts was ensured. This approach includes assumptions, which are fulfilled in many applications. In the current implementation, the model needs to be annotated with a *SetImpact* flag. The authors propose to include a suitable description of impulsive distributions (including impulsive jumps) in the Modelica specification.

Even if the resulting system of equations is not regular, the IVC can find a solution. At last we mentioned how fixed and non-fixed initial values and feasible region for the variables in the IVC are managed.

References

[1] Pantelides C.C.: The Consistent Initialization of Differential-Algebraic Systems. SIAM J. SCI. Stat. Comput., Vol. 9, No. 2, pp. 213-231, March 1988.

[2] Mattsson S.E. and Söderlind G.: Index reduction in differential-algebraic equations using dummy derivatives. SIAM Journal on Scientific Computing, Vol. 14, No. 3, pp. 677-692, May 1993.

[3] Jonker R. and Volgenant A.: A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems. Computing, Vol. 38, pp. 325-340, 1987.

[4] Kunkel P. and Mehrmann V.: Differential - Algebraic Equations. European Mathematical Society Publishing House, Zürich, 2006.

[5] Bauer I.: Numerische Verfahren zur Lösung von Anfangswertaufgaben und zur Generierung von ersten und zweiten Ableitungen mit Anwendungen bei Optimierungsaufgaben in Chemie und Verfahrenstechnik. Dissertation, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR) der Universität Heidelberg, 1999.

[6] Wunderlich L.: Analysis and Numerical Solution of Structured and Switched Differential-Algebraic Systems, PhD Thesis, TU Berlin, 2008.

[7] Li S., Petzold L. R.: Design of New DASPK for Sensitivity Analysis, Technical Report: TRCS99-28, 1999.

An XML Representation of DAE Systems Obtained from Modelica Models

Francesco Casella[a] Filippo Donida[a] Johan Åkesson[b,c]
[a]Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 35/5, 20133 Milano, Italy
[b]Department of Automatic Control, Lund University, Lund, Sweden
[c]Modelon AB, Sweden

Abstract

This contribution outlines an XML format for representation of flat Modelica models. The purpose is to offer a standardized model exchange format which is based on the DAE formalism and which is neutral with respect to model usage. Many usages of models go beyond what can be obtained from an execution interface offering evaluation of the model equations. Several such usages arise in the area of control engineering, where Linear Fractional Transformations (LFTs), derivation of robotic controllers, optimization, and real time code generation are some examples. The choice of XML is motivated by its defacto standard status and the availability of free and efficient tools. Also, the XSLT language enables specification of transformation of the XML model representation into other formats.

Keywords: DAE representation; XML standard; modeling

1 Introduction

The Modelica language allows to build complex models of physical systems, described by differential-algebraic equations (DAE). These models can be used for different purposes: simulation, analysis, optimization, model transformation, control system synthesis, real-time applications and so forth. Each one of these activities involves a specific handling of the corresponding differential algebraic equations, by both numerical and symbolic algorithms. Moreover, specialized software tools which implement these algorithm may already exist, and only require the equations of the model to be input in a suitable way.

The goal of this paper is to define an XML-based representation of the DAEs of Modelica models, which can then be easily transformed into the input of such tools, e.g. by means of XSLT transformations. On one hand, this representation must be as close as possible to the mathematical equations, therefore without any aggregation, inheritance, and complex data structures left. On the other hand, it must be as general as possible with respect to the possible usage of the equations, which should not be limited to simulation.

This representation could then be used as a standard interface between the front-end of any Modelica compiler, and any possible back-end for simulation, optimization, analysis, etc.

In addition, the XML representation could also be very useful for treating other information concerning the model, for example using an XML schema (DTD or XSD) for representing the simulation results, or the parameter settings. In those cases, using a well-accepted standard will result in great benefits in terms of interoperability for a very wide spectrum of applications.

The paper is structured as follows: in Section 2, the abstract structure of the DAE representation is informally described, motivating the structure of the formal XML schema definition. Section 3 discusses some of the possible usages of such a representation. Section 4 briefly describes test implementations in the OpenModelica and JModelica.org compilers, while Section 5 ends the paper with concluding remarks and future perspectives.

2 Abstract representation of the DAE system

To the best of the authors' knowledge, the optimum representation for defining a DAE system should be as close as possible to the mathematical definition. Provided that a DAE system consists of a system of dif-

DOI: 10.3384/ecp09430073

ferential algebraic equations, it can be expressed as:

$$f(\dot{x}, x, y, u, v, t, p) = 0 \qquad (1)$$

where $\dot{x}$ is the derivative of state, x is the states, y are the outputs, u are the inputs, v are the algebraic variables, t is the time and p is the set of the parameters.

For the rest of this paper, we assume that the DAEs (1) describing the model have index 1. This restriction is necessary to give to the x variables the meaning of *states*, i.e., variables whose initial values can be arbitrarily selected. Most applications for DAE models (and all the applications discussed in this paper) require an index-1 DAE as input, so it is reasonable to discuss a representation limited to this class of equations. In case the equations of the original Modelica model have higher index, such DAEs can be obtained by symbolic index reduction, which is an available feature in most Modelica tools, so this is not a drastic limitation to the range of applicable models. In this case, however, we must assume that the index reduction procedure gives a fixed selection of states.

Even though the representation provided in equation 1 is very general, and is very appreciable for viewing the problem as one could see it written on paper, it cannot be directly used for inter-tools exchange in an efficient way.

The main idea is then to provide a standardized mathematical representation of the DAE system that relies on standard technology and is application-oriented. This justifies the adoption of the XML standard as the base framework. It can be noted that while XML is generally not suited for manual inspection, an XSLT transformation translating an XML description into, e.g., a flat Modelica representation is easily defined.

As an additional requirement, we must consider that the DAE systems we are dealing with are derived from the Modelica models. Even if this can be seen as a restriction, this is not, since the Modelica language specification interprets a superset of the problems that are object of this paper, providing a textual definition to describe the physical systems, concerning also the variables types and the expressions operators definitions.

Previous efforts have been registered to define standard XML-based representations of Modelica models, including [14, 15]. A standard representation of process engineering models is described in [2] and a standard for Modelica-derived simulation models is presented in [11]. In addition, a standard representation for API implementation for Modelica is given in [16], and standard representation for simulation libraries [9]. A recent initiative is the Functional Model Interface (FMI)[1], [6], which is aimed at creating a standard for a Modelica execution API.

The aim of this work is to take advantage from all this studies and try to define a simple and general representation which is not tailored for a particular usage, but rather aims at covering the largest possible problems that can be formulated starting from a Modelica model. Particular care has been exercised in order to define concepts and structures which are general enough to be usable in different contexts.

In the remainder of this section, the different parts of the proposed XML representation will be described.

2.1 Variables

The `Variables` entity corresponds to the set of scalar variables (real, integer, boolean) that are present in the equations of the DAE. In particular, taking into account the continuous-time representation (1), five types of variables are needed:

- Time-invariant, i.e., the constants and the parameters.
- Input variables, conceptually given from the outside.
- Algebraic variables corresponding to algebraic equations in the matching algorithm. This category also includes dummy derivatives obtained after index reduction. This set of variables could also be identified as the set of the time-variant variables not contained in the set of states or inputs.
- State variables.
- Derivatives of state variables.

As one could easily imagine, some constraints are present on the variables set. Firstly, a one-to-one relationship is defined between the set of the state variables and the set of derivatives. Secondly, the outputs are a subset of the state and algebraic variables, in the sense that the state and algebraic variables might also be marked as output variables, i.e., have an output attribute.

Associated with a variable is also a set of attributes, corresponding to the attributes specified by the Modelica language. These include the start attribute, min and

[1] The work on FMI is done within the ITEA2 project MODELISAR.

max, unit, nominal etc. These attributes are essential to include in a model specification format since they provide information about, for example, start values of variables, model validity regions and unit information.

Note that string parameters could also be considered, with constant binding to constant literal strings. The string binding equations will not be considered in the equation sets, because are irrelevant from a mathematical point of view.

2.2 Expressions

`Expressions` represent all the mathematical expression of the system and can be formed by aggregating identifiers and literals through:

- Unary operators. This class contains all the mathematical functions that require only one argument as input. Possible examples are the trigonometric functions (`sin`, `cos`, `tan`, `asin`, `acos`, `atan`), the hyperbolic functions (`sinh`, `cosh`, `tanh`), the exponential functions (`exp`), the logarithmic functions (`log`, `log10`), and the square root function (`sqrt`).
- Binary operators. This set contains all the algebraic operators like +, *, −, /, the factor function ^, and the `atan2` function.
- Function calls referring to user-defined functions.

XML encoding of expressions is straightforward by introducing elements corresponding to an abstract grammar specification used in a compiler. This approach renders the DAE XML representation format to provide abstract syntax trees (ASTs) for expressions, which simplifies the development of XSLT transformations. Also, translating this representation into other formats for representing mathematical expressions, e.g., MathML, [5], would be trivial.

2.3 Functions

A `Function` is conceptually equivalent to an algorithm, i.e. a part of a procedural code and, in this sense has:

- Input variables (possibly with default values)
- Output variables
- Protected variables (i.e. variables visible only within the function)
- An algorithm to compute outputs from inputs, possibly using protected variables.

The DAE representation contains only scalar variables. If any vector or array variable is present within the original object oriented model, it is flattened to their fundamental scalar elements by the compiler before producing the XML. This is also the case for the other data structures (e.g. records) and for the functions. In particular, functions returning vectors or records are split into separate scalar functions, each one corresponding to the computation of a single scalar element of the outputs. For example, `(x,y)=f(u,v)` is converted to the scalar equations:

```
x1 = f1(u1,u2,v1,v2)
x2 = f2(u1,u2,v1,v2)
y1 = f3(u1,u2,v1,v2)
y2 = f4(u1,u2,v1,v2)
```

where each function `fj()` is defined in Modelica as the original function `f()`, save that all outputs except the j-th are declared as protected variables instead of outputs. The `fj()` functions should retain a reference to the original function `f()`, allowing an efficient computation scheme, where required. As an example, a simulation applications might cache the results of calling `f(u,v)` when encountering the first call to any `fj(u,v)`, and then use this to get the results of the other scalar functions calls with the same arguments, without re-executing the algorithm.

If the compiler performs function in-lining before generating the XML representation, the corresponding function calls disappear from the model; on the other hand, sophisticated in-lining of non-trivial functions could be performed by a post-processing tool, whose input is the XML code. The subject of in-lining is thus completely transparent and orthogonal to the DAE representation discussed here.

The algorithm in a function is formulated by an imperative language equivalent to Modelica algorithms, expressed as an XML translation of the corresponding abstract syntax tree.

The abstract syntax tree representation for functions is conceptually a superset of the `Expression` entity defined in subsection 2.2. More precisely, it is necessary to add three main classes of entities. Firstly, the program flow constructs (such as the `if-then-else`, the `for-loop`, the `while-loop` and others) are necessary. Secondly, there are some Modelica-specific constructs, e.g., `sample()` and `initial()`. Finally, the basic operators provided by a majority of programming languages, i.e., the boolean relation operators including <, <=, ==, <>, > and >=, and finally type conversion primitives such as `floor` and `edge`.

Again, XML elements are conveniently introduced to represent functions and related quantities. The advantages of having such kind of abstract representation are evident, for example, it can be easily converted to any imperative programming language (C, Matlab, Mathematica, Maxima, Maple, Scilab, Python, Java, etc) by means of XSLT transformations.

2.4 Equations

In many cases, a Modelica model includes parameters depending on other parameters. For simulation, it is necessary to solve the corresponding equations numerically at initialization. The numerical values can then be used for dynamic simulation. For other purposes, it may be necessary to keep some of these relationships in the model. For instance, in optimization problems there may be a free parameter `p1` and another parameter `p2 = f(p1)`. In this case, the parameter `p2` cannot be computed before the optimization procedure starts, but rather, the relationship needs to be included amongst the constraints in the optimization. When building LFT representations, `p1` might be an uncertain parameter, while `f()` might be well known; in this case, one wants to keep the dependency of `p2` from `p1` in the dynamic model.

When dealing with simulation problems, equations for parameters are conceptually part of the initialization section. However, they may play a special role in non-simulation problems, in particular when they all have fixed = true. In the case of LFT transformations there are no initial equations, but it is still necessary to consider the relationships between uncertain parameters and all other parameters when formulating the uncertain dynamic equations.

In fact, there is a whole class of problems for which the initial equations are irrelevant. As a first example, consider the LFT representation of an uncertain dynamical system. This system only involve the dynamic equations, and the initial values of the states are not required for the transformation. Also, when dealing with the derivation of inverse kinematics, computed torque and inverse dynamics in robot models, the resulting problems are purely algebraic: there are no initial equations involved once the appropriate BLT has been performed and the irrelevant parts of the model have been discarded.

However, there are still many problems where the values of initial states are an essential part of the problem. The initial variable values are generally not known, but need to be solved from the initial equations. There are also problems where additional initial equations are required to determine the values of some parameters (set with fixed=false). An additional example is given by the so-called trimming problems, where the values of the inputs are determined by prescribing certain steady values for the outputs.

Therefore, three separate sets of equations need be defined:

1. Dynamic equations. This set is composed of equations specified in equation sections and binding equations for variables. These equations are matched to algebraic variables (algebraic equations) and to state derivatives (differential equations). Each equation is given in residual form <expression> = 0.

2. Binding equations for parameters with fixed=true. These equations are matched to fixed=true parameters. The equations are in the form <parameter> = <expression>, and can be solved through assignments. The latter statement follows since it is illegal to define models with cyclic dependencies between parameters in Modelica.

3. Initial equations. This set is composed of equation given in initial equation sections and binding expressions for variables with fixed=true. These equations are matched to state variables, parameters with fixed=false, and possibly to inputs, if there are any (see example below). The initial equations should be in the form expression = 0.

```
model M
  input Real u;
  output Real y;
  Real x;
equation
  der(x) = -x + u;
  y = 4*x;
initial equation
  der(x) = 0; // Implies x(0) = u(0)
  y = 4; // This equation determines
         // x(0) = 1, and therefore
         // u(0) = 1;
end M;
```

Depending on the application, these three sets can be used in different ways, as will be discussed in Section 3.

2.5 Additional information

As introduced above, the range of applications that could directly use as input an XML DAE representation or any *ad hoc* description (derived from the more general one through XSLT transformation) is extremely variegated [3]. The common aspect is that the majority of tools for these usages require an index 1 DAE, as described in this section. Other information could be available from the Modelica tool, which could be relevant for some applications. For instance, information about the BLT structure of the dynamic simulation problem (compute the derivatives and algebraic variables, given the inputs, states, parameters, and time) could also be included, as well as information about the index reduction process in case the DAEs are the result of some index reduction algorithm such as [12]. This is however beyond the scope of the present paper.

3 Application examples

3.1 Simulation

The simulation tools are generally following the same approach. Firstly, the parameters and constants within each equation are numerically evaluated, by solving all the three equation sections together to determine the initial values of everything. After that, the numerical values of parameters are fixed, and the dynamic equations are used to compute derivatives and algebraic variables at each time step in an integration algorithm. This is also the case when considering the parallel simulation problem. Functions are the “linked” as external functions if any function calls is present for the state derivative computation.

3.2 LFT Transformation

LFT is a widely used model description formalism in modern control and system identification theory, in which uncertain parameters and non-linearities are “pulled out” from the system, resulting in the feedback connection between a linear, time-invariant model and blocks representing the uncertain and/or nonlinear parts. The procedure for obtaining an LFT representation from Modelica models is fully described in [4] and is only briefly summarized here. Assuming an ODE system, the values of the parameters are given by the binding equations, which specify the value of each parameter either by a numerical value, or as a function of other parameters. At each time instant, the values of states and inputs are known; the numerical values of the parameters are not known explicitly, but they can be considered as known, given the binding equations. The goal is now to compute the state derivatives $\dot{x}$ and the algebraic variables v. To this end, the equations and the variables of the problem can be re-ordered so that the incidence matrix (equations on the rows, unknowns on the columns) is brought in Block-Lower-Triangular (BLT) form. This task is accomplished by using the well-known Tarjan algorithm [7], applied to the equations-variables bipartite graph, which is equivalent to the incidence matrix of the system. The strongly connected components of the graph correspond to the blocks on the diagonal, and a partial ordering among equations can be deduced from the graph after the algorithm has terminated. After reordering, the system can be formulated as

$$\Phi(x,u,\Xi,p_0) = 0, \tag{2}$$

where $\Phi(\cdot)$ is the set of re-ordered equation residuals and Ξ is the re-ordered set of the system unknowns (i.e., all the elements of vectors $\dot{x}$ and v). By defining $\Phi_j(\cdot)$ as the j-th sub-set of equations corresponding to one block of the BLT form, Ξ_j as the corresponding sub-set of unknown variables, and q as the number of blocks on the diagonal of the BLT incidence matrix, the re-ordered system equations (2) can be formulated as

$$\Phi_1(x,u,\Xi_1,p_0) = 0 \tag{3}$$
$$\Phi_2(x,u,\Xi_1,\Xi_2,p_0) = 0 \tag{4}$$
$$\dots \tag{5}$$
$$\Phi_q(x,u,\Xi_1,\dots,\Xi_{q-1},\Xi_q,p_0) = 0. \tag{6}$$

The system expressed in the form (3)-(6) can be executed within a suitable environment, which supports the symbolic manipulation of LFTs. Summing up, in the case of LFTs, the binding equations for parameters are solved by keeping the uncertain parameters as symbolic objects, and the resulting expressions are symbolically substituted in the dynamic equations; then, the relationship between the states and the inputs on one hand and the derivatives and the outputs on the other hand, is transformed into an LFT.

3.3 Derivation of robotic controllers

The design of controllers for robotic systems with N degrees of freedom usually starts with the equations of motion obtained from the Euler-Lagrange equations:

$$B(q)\ddot{q} + H(q,\dot{q})\dot{q} + E(q) = \tau \tag{7}$$

$$y_p = K(q) \tag{8}$$

$$y_v = \frac{\partial K}{\partial q}\dot{q}, \tag{9}$$

where q is the N-element vector of Lagrangian coordinates, which usually correspond to the rotation angles of the actuator motors, $\dot{q}$ is the vector of the corresponding generalized velocities, y_p is the vector of the Cartesian positions of selected points of the robot, τ is the vector of generalized applied forces corresponding to each degree of freedom (the torques applied by the actuators), $B(q)$ is the inertia matrix, $H(q,\dot{q})$ is the matrix corresponding to the centripetal, Coriolis, and viscous friction forces, $E(q)$ accounts for the effects of the gravitational field.

The classical approach to write (7) requires to compute the so-called direct kinematics, i.e. how the values of q and $\dot{q}$ translate into the position and motion of the robot's links, then to compute the Lagrange function, i.e. the difference between kinetic and potential energy, and apply the Euler-Lagrange equations. This can be done manually, or using one of the specialized tools available for this task.

With an object-oriented approach the original model is usually an index-3 DAE. This model is then brought into index-1 form

$$F(x,\dot{x},y,u) = 0 \tag{10}$$

where

$$x = \begin{bmatrix} x_p \\ x_v \end{bmatrix} = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}, \quad y = \begin{bmatrix} y_p \\ y_v \end{bmatrix}, \quad u = \tau, \tag{11}$$

by means of connection tree analysis, change of state variables, and index reduction algorithms; this model is mathematically equivalent to the Lagrange model (7)-(9). Currently available Modelica tools solve the simulation problem by producing an efficient procedure to solve it for $\dot{x}$ and y given x and u. This procedure effectively brings the system into state-space form, which can then be linked to any ODE/DAE solver. In fact, there are other things that can be done with the model (10), which are very useful for the design of control system.

Robot trajectories are originally defined in Cartesian space as functions of time $y_p^0(t)$. Obtaining the corresponding reference trajectories in Lagrangian coordinates for the low-level robot joint controllers requires solving the problem to obtain the the so-called inverse kinematics:

$$q^0(t) = K^{-1}(y_p) \tag{12}$$

$$\dot{q}^0(t) = \left(\frac{\partial K}{\partial q}\right)^{-1} y_v; \tag{13}$$

the Jacobian of $K(q)$ is also needed to numerically invert (8). Furthermore, two interesting approaches to model-based robot control are based on the direct use of (7): the pre-computed torque approach and the inverse dynamics approach.

The pre-computed torque approach is a feed-forward compensation scheme, which requires to solve (7) backwards, i.e. compute the (theoretical) torque required to follow the reference trajectory:

$$\tau = B(q^0)\ddot{q}^0 + H(q^0,\dot{q}^0)\dot{q}^0 + E(q^0), \tag{14}$$

and then feed it directly to the actuators; some decentralized feedback action is also included to deal with uncertainties and disturbance.

The inverse dynamics approach is a feedback compensation scheme, that uses the model in order to transform the non-linear control problem into a linear problem with constant coefficients. Using this method, a virtual input variable v is defined which satisfies

$$\tau = B(q^0)v + H(q,\dot{q})\dot{q} + E(q). \tag{15}$$

Since the inertia matrix B is assumed to be structurally non-singular, it is always possible to solve (15) for $v(t)$, given the generalized velocities $q(t)$ and $\dot{q}(t)$, that are sensor outputs. Using this virtual input, the robot dynamics (7) can then be formulated as a set of double integrators:

$$\ddot{q} = v \tag{16}$$

For the robotic applications, the parameter binding equations are solved numerically; their numerical values are then substituted into the dynamic equations. Depending on the specific robotic problem: direct kinematic, inverse kinematic, pre-computed torques or inverse dynamics approach, these equations (or a part of these) are then solved.

3.4 Optimization

The needs for solving optimization problems based on Modelica models usually goes beyond what is typically offered by a simulation oriented execution API, see [1]. In some cases, the initial conditions are free variables in the optimization, which implies that the initial equations must be explicitly available. The same situation holds for dependent parameters since such a parameter may be dependent on another parameter which is free in the optimization. In addition, quantities derived from the model equations such as first and second order derivatives and sparsity patterns may be required by numerical algorithms.

The availability of a standardized XML-based model exchange format is useful in an optimization context since it enables transformation of a model into various formats suitable for different algorithms. Also, having access to expression syntax trees is useful for deriving derivatives, e.g., by means of automatic differentiation. It is worth noticing, however, that in order to completely specify an optimization problem, quantities such as cost function and constraints must also be taken into account. This is not part of the specification proposed in this paper. For a discussion on representation of optimization problems derived from Modelica models and Optimica specifications, see [10].

3.5 Real time code generation

Real time code could be directly generated by in-lining the discretization method within the equations of the XML file (e.g., forward or backward Euler), thus obtaining the core of the real time code. This could also be directly obtained from the XML formulation of the DAEs through the usage of an XSLT transformation if no symbolic manipulation is required. The simulation problem is then formulated by solving the dynamic equations for the next states and for the algebraic variables and then bringing it into BLT form.

If all the equations in the BLT form are linear, or can at least be solved explicitly in symbolic form, then it is straightforward to generate simulation code with fixed execution time. Otherwise, if there are implicit nonlinear equations, iterative solvers will be needed and there might be convergence problem that require proper handling.

4 Test implementations

There are two prototype implementations available. The implementation of the XML module within the OpenModelica compiler started the past year and is now included in the latest release of the compiler. This functionality allows dumping of a flattened Modelica model after performing the index reduction (if necessary), the BLT transformation and the matching algorithm. The API method provided by the OpenModelica compiler offers the possibility to specify several inputs parameters, such as if to add or not the information for solving the system, and if to dump the equations as residuals or add MathML representation for all the equations. This XML schema is available at [8].

The JModelica.org platform currently supports generation of variable meta data, as described in Section 2.1, in XML format, see [13, 10]. It is intended that this functionality is extended to include also equations an functions as well as cost functions and constraints for optimization.

As for the actual specification of a DAE XML schema, the objective is to build on what is done in the FMI initiative concerning model meta data and to merge this with the existing schema [8].

5 Conclusions and future perspectives

In this paper, we have outlined an XML representation of DAEs. This will allow easy coupling of Modelica compiler front-ends with diverse application back-ends that require the system equations as inputs. This format is not limited to Modelica models, but could be used as a lingua franca to represent continuous-time dynamical systems originally described with other modelling languages, such as, e.g., gPROMS or VHDL-AMS. This would allow developers of application back-ends (e.g. for optimal controller generation) to support multiple modelling platform easily.

The proposed format does not support all features of Modelica. Notably, description of hybrid constructs are lacking. However, there is a number of interesting applications where this is not needed, as demonstrated by the examples outlined in this paper.

The XML description format outlined in this paper might be extended in several respects. First of all, support for discontinuous expressions (e.g. *if-expressions*) could be added, possibly by including an explicit representation of the root functions that many tools need in order to handle discontinuities properly. In order to support hybrid models, it would also be necessary to introduce the concepts of discrete variables, discrete equations (those within *when* statements), time events and state events. Another extension might be to support variables declared as vectors and array equations, without reducing all equations to their scalar form; this might be useful for sophisticated symbolic processing at the vector level. Support of index-1 models with dynamic sets of states (such as those resulting from the dummy derivative algorithm [12] in some cases) might be added, as well as support for the description of higher index models. Finally, it would be interesting to investigate how this kind of formalism could be employed to describe sub-models that could then be aggregated at a higher level, by introducing some kind of connector concept; this might allow some form of separate compilation strategy, at least for a certain class of problems that do not lead to higher index DAEs when connecting the submodels.

References

[1] Johan Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *In 6th International Modelica Conference 2008*. Modelica Association, March 2008.

[2] Christian H. Bischof, H. Martin Bücker, Wolfgang Marquardt, Monika Petera, and Jutta Wyes. Transforming equation-based models in process engineering. In H. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors, *Automatic Differentiation: Applications, Theory, and Implementations*, Lecture Notes in Computational Science and Engineering, pages 189–198. Springer, 2005.

[3] F. Casella, F. Donida, and M. Lovera. Beyond simulation: Computer aided control system design using equation-based object oriented modelling for the next decade. In *2nd International Workshop on Equation-Based Object-Oriented Languages and Tools*, July, 8 2008.

[4] F. Casella, F. Donida, and M. Lovera. Automatic generation of lfts from object-oriented non-linear models with uncertain parameters. In *6th Vienna International Conference on Mathematical Modeling*, February, 11-13 2009.

[5] D. Suliman D. Draheim, W. Neun. Searching and classifing equations on the web, zib report 04-22. Technical report, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, 2004.

[6] DLR, Dynasim, ITI and QTronic. The functional model interface. Draft.

[7] I. S. Duff and J. K. Reid. An implementation of Tarjan's algorithm for the block triangularization of a matrix. *ACM Transactions on Mathematical Software*, 4(2):137–147, 1978.

[8] Filippo Donida. DAE XSD schema, 2009. `http://home.dei.polimi.it/donida/Projects/AutoEdit/Images/DAE.xsd`.

[9] P. A. Fishwick. Using xml for simulation modeling. In *Winter simulation conference*, December, 8-11 2002.

[10] J. Åkesson, T. Bergdahl, M. Gäfvert, and H. Tummescheit. The JModelica.org Open Source Platform. In *7th International Modelica Conference 2009*. Modelica Association, 2009.

[11] J. Larsson. A framework for simultion-independent simulation models. *Simulation*, 82(9):563–379, 2006.

[12] S. E. Mattsson and G. Söderlind. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM Journal on Scientific Computing*, 14(3):677–692, 1993.

[13] Modelon AB. JModelica Home Page, 2009. `http://www.jmodelica.org`.

[14] A. Pop and P. Fritzson. Modelicaxml: A modelica xml representation with applications. In *3rd Modelica conference*, November, 3-4 2003.

[15] U. Reisenbichler, H. Kapeller, A. Haumer, C. Kral, F. Pirker, and G. Pascoli. If we only had used xml... In *5th Modelica conference*, September, 4-5 2006.

[16] M. Tiller. Implementation of a generic data retrieval api for modelica. In *4th Modelica conference*, March, 7-8 2005.

Parallel Simulation of Equation-based Object-Oriented Models with Quantized State Systems on a GPU

Martina Maggio [*] [‡], Kristian Stavåker [†], Filippo Donida [*], Francesco Casella [*], Peter Fritzson [†]

Abstract

This work focuses on the use of parallel hardware to improve the simulation speed of equation-based object-oriented Modelica models. With this intention, a method has been developed that allows for the translation of a restricted class of Modelica models to parallel simulation code, targeted for the Nvidia Tesla architecture and based on the Quantized State Systems (QSS) simulation algorithm. The OpenModelica Compiler (OMC) has been extended with a new back-end module for automatic generation of the simulation code that uses the CUDA extensions to the C language to be executable with a General Purpose Graphic Processing Unit (GPGPU). Preliminary performance measurments of a small example model have been done on the Tesla architecture.

Keywords: Parallel Simulation, QSS algorithm, CUDA architecture, OpenModelica compiler, GPGPU

1 Introduction

Recent increases in the continuing growth of computing power predicted by Moore's law are mainly due to increased parallelism, rather than to increased clock frequency [16]. A challenge in the field of dynamic system simulation is to exploit this trend, reducing computation time via the use of parallel architectures [3, 12, 13].

Traditionally the majority of the parallel programming techniques are based on multi-CPU architectures. Recently, parallel execution of general purpose code has become cheaply available through the use of Graphic Processing Units (GPUs) that allows for general code execution, also known as General Purpose Graphic Processing Units (GPGPUs). The use of this particular hardware has been widely encouraged in recent years; in fact many applications have been developed, see for example [4, 10, 15].

The aim of this work is two-fold; as a first point the possibility of parallelization of the QSS algorithm *per se* together *with the chosen architecture* is investigated, while, as a second step, the parallel performance of the QSS integration method via automatically generated CUDA code is studied and some test are conducted to evaluate the chosen approach.

Since the Modelica language is used to describe many different classes of systems, in this work the test models have been restricted only to a subset:

- continuous time, time-invariant systems (with no events),
- index-1 DAE (if the index is greater than 1 the index reduction algorithm should be used before processing the model),
- initial values of states and values of parameters known at compile time, and inserted into the generated code as numbers,
- no implicit systems of nonlinear equations to be solved numerically.

The QSS integration method is a Discrete Event System (DEVS) method that was introduced in [5, 8], where the author suggested that it could be suitable for parallel execution. However, to the best of the authors' knowledge, no attempts have previously been made to deeply investigate the possibility of parallel implementations. In this work, a general discussion on the parallel QSS algorithm is done and a possible implementation for a particular shared-memory parallel architecture is presented.

The generated code, in fact, has been targeted for the Nvidia Tesla architecture, that "is useful to manage general purpose computation" [2]. To obtain speed improvement through fine-grained parallelism, the C language extension CUDA has been used, taking low level implementation details into account.

[*] Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milano, Italy

[†] PELAB - Programming Environment Lab, Dept. Computer Science Linköping University, S-581 83 Linköping, Sweden

[‡] Corresponding author email: maggio@elet.polimi.it

DOI: 10.3384/ecp09430032

This work is structured as follows. Section 2 describes the parallel architecture chosen to test the approach, highlighting the advantages and the disadvantages of the particular hardware. In section 3 the subset of the Modelica models targeted for the automatic generation of CUDA code is defined. The implementation of the parallel simulation through the available language and the strategies used to parallelize the Quantized State Systems simulation algorithm are treated in section 4. Section 5 describes the changes applied to the OpenModelica compiler to enable the code generation. Experimental results from model simulations are described in section 6 while in section 7 the conclusions of this work are explained and some proposals for future developments are sketched.

2 Parallelism with a Graphic Card

CUDA stands for *Compute Unified Device Architecture*, it is a C language extension developed by Nvidia with the intention of making it possible to exploit the massive parallelism found in GPUs for general purpose computing. Beyond the large number of computing cores available with the GPU architecture, the most interesting advantage is the presence of fast threads and a fast shared memory region, that leads to improved performance in memory writes and readbacks to and from the GPU.

On the other hand there are also some strong limitations: first of all, the language is a recursion-free, function-pointer-free subset of the C language, plus some simple extensions for managing the parallelism and allowing a single process run spread across multiple disjoint memory spaces. The memory management has to be taken in serious consideration, since there are strong limitations on the available address space. The bus bandwidth and the latency between the CPU and the GPU may be a bottleneck. Moreover, threads should be run in groups of at least 32 for best performance, with the total number of threads numbering in the thousands. Branches in the program code do not impact the performance significantly, provided that each of 32 threads (in a group) takes the same execution path. The SIMD (single instruction, multiple data) execution model of all thread in a group becomes a significant limitation for every inherently divergent task, in fact when taking a diverging branch the code execution will be significantly slowed down since each different code variant has to be executed in sequence.

The SIMT (single instruction, multiple thread) architecture inserts a new element in the Flynn taxonomy [6], since the groups of threads execute the same code among the core components in a single cluster and not among all the processing units as in the classical SIMD method; in fact MIMD parallelism can be achieved with a careful allocation of the threads to the clusters. Nonetheless, the parallelism exploitation is not trivial since the code needs to be designed *ad hoc* for the specific hardware to limit diverging branches. Specifically, the objective is to make threads run as long as possible over the same portion of the code. This is somehow in contrast with the concept of "parallel architecture" where every processing component can perform different operations, on the same or on different data. As stated, some code portions should be processed with a MIMD (multiple instruction, multiple data) method and CUDA partially allows it through the thread distribution to the available multiprocessors.

In the following, the architecture is described in detail; each graphic card is made up of common core components. The Tesla architecture, see figure 1, is based on a scalable processor array (SPA), with a certain number of streaming-processor (SP) cores. These SP cores are organized in sets of streaming multiprocessors (SMs) and in processor clusters (TPCs), i.e., independent processing units.

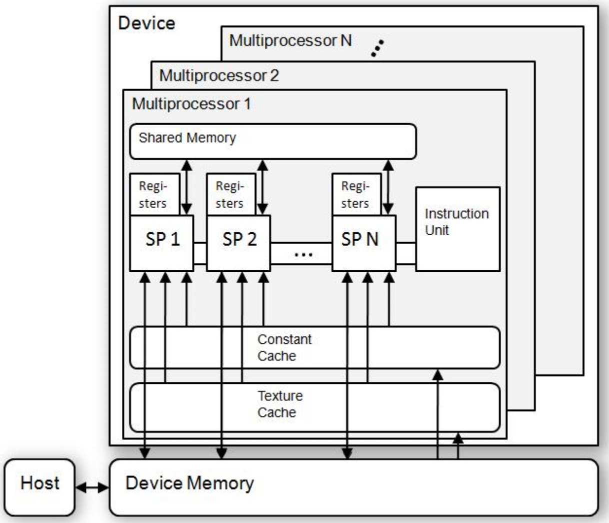


Figure 1: The Nvidia Tesla architecture.

The graphic unit interface communicates with the host processor, replying to commands, fetching data from system memory, checking command consistency, and performing context switching. The work distribution units forward the input assembler's output stream to the array of processors. The processor array executes thread programs and provides thread control and management. The number of clusters determines the processing performance and scales from one processor cluster in a small graphic card to twenty or more in high-level hardware.

The streaming multiprocessors consist of eight streaming-processor cores, a multi threaded instruction fetch and issue unit, an instruction cache, a read-only constant cache, and some shared memory. The shared memory holds input buffers or shared data for parallel computing. A low-latency interconnect network between the streaming-processors and the shared-memory banks provides shared memory access.

In this work we are using two different graphic cards, the more powerful one is the *Nvidia Tesla C1060*, which features 240 stream processors organized in 30 clusters of 8 SIMD processors, supports single and double precision, has 4 GB of memory and a memory bandwidth of 102 GB/s. According to the specification [2] this hardware has a “Compute Capability 1.3”, this means that the maximum number of threads per block is 512, the maximum number of active blocks per multiprocessor is 8 and each multiprocessor is composed of eight processors, so that a multiprocessor is able to process the 32 threads of a warp in four clock cycles. It also supports some features like warp voting, that are not used in this work.

The results obtained are compared with data obtained from an *Nvidia GeForce 8600*, which has just 32 stream processors, organized in 4 clusters, only supports single precision, has 512 MB of memory and a memory bandwidth of 57.6 GB/s. Its “Compute Capability” is just 1.1; this means that this hardware does not support double precision and has not the additional features of the previous. In order to compare the behaviour with different numbers of clusters, single precision numbers are used for both tests.

The memory management instructions access three read/write memory spaces:

- local memory for per-thread, private, temporary data (implemented in external DRAM);
- shared memory for low-latency access to data shared by cooperating threads in the same SM;
- global memory for data shared by all threads of a computing application (implemented in external DRAM).

A more detailed survey of the architecture features can be found in [11].

In order to exploit parallelism with the CUDA architecture, a programmer has to write a serial program that calls parallel kernels, which can be simple functions or full programs. The CUDA program executes serial code on the CPU and executes parallel kernels across a set of parallel threads on the GPU. The programmer has to organize these threads into a hierarchy of thread blocks in order to obtain SIMD, SIMT and MIMD parallelism. In fact, when a CUDA program on the host CPU invokes a kernel parallel execution, the thread blocks are enumerated and distributed to free multiprocessors on the device. The threads of a thread block execute concurrently on one multiprocessor. As thread blocks terminate, new blocks are launched on the vacated multiprocessors. Figure 2 shows the execution flow of the code.

3 A Restricted set of Models: the Parallelizable Modelica Models

Even if the long term goal is to be able to automatically generate parallel code for every possible Modelica model, in this work the study is restricted to a subset of purely continuous-time, time-invariant systems with time-varying external inputs.

This subclass of systems can be brought into standard form by applying index reduction and BLT algorithms, thereby generating code corresponding to:

$$\left\{ \begin{array}{rcl} \dot{x} & = & f(x,u) \\ y & = & g(x,u) \end{array} \right. \tag{1}$$

In order to simulate the system (1) it is necessary to implement functions for calculating the derivative of each state variable, as well as the output variables. This should be done within the graphic card kernel space. A function for the thread management is also needed: this function should be able to start a new thread and assign it to one of the core components preserving the load balance. For the current implementation, the load balance could be improved considering for example the estimated load of each new thread.

In addition some structural information about the mathematical representation of the model is required, i.e., the number of the state variable of the index reduced model and the number of outputs. It is also important to stress that the output computation should be executed within the card kernel space, thus resulting in a minimal overhead.

Another important aspect for simulation of the model is the QSS integration step. In this work we used a constant quantization step, unchanged for all

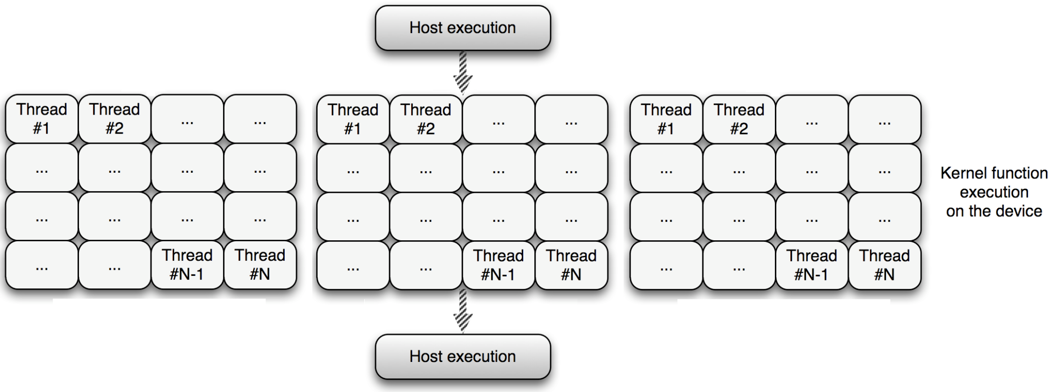


Figure 2: Execution flow example; the host machine asks the graphic device to compute the parallel execution of some CUDA threads, divided in blocks and assigned to the clusters of multiprocessors. After all thread terminations the control is returned to the host which can run the next instruction.

the state variables (for more detail see section 4) but a different quantization step can be used for each state variable, with minor modification to the code. Finally, input variables should be known *a priori* for the correct QSS algorithm execution. The input signals are pre-processed to compute the QSS inputs, expressed as piecewise constant trajectories.

4 Quantized State System Simulation and Parallelism

The QSS algorithm is a method to solve ODE systems; there are different ODE solvers, varying in approximation orders or in time slicing (i.e., how often they compute new state values). Moreover, there are explicit and implicit algorithms to compute the values of the state variables at the next discrete time instant, given current and past state and derivative information. Rather than making use of the concept of time slicing to reduce a continuous-time problem to an (in some way equivalent) discrete-time problem, the QSS method employs the concept of state quantization for the same purpose.

Given the current value of a state variable, $x_i = Q_i \in X_i$ where X_i is an ordered increasing set of discrete values that the state variable may assume; the QSS algorithm calculates when is the earliest time instant at which this state variable shall reach either the next higher or the next lower discrete level in the set.

The algorithm transforms a continuous time system in a Discrete Event System (DEVS) [18, 17]. The QSS algorithm has been studied in depth, and it has been proved by mathematical theorems that a limited boundary error exists when transforming a continuous time system into a DEVS one, i.e.:

$$\dot{x} = f(x,u) \longrightarrow \dot{x} = f(q,u) \tag{2}$$

where the state vector x becomes a "quantized state vector" q where state values are in the corresponding set. The quantized state vector is a vector of discretized states where each state varies according to an hysteretic quantization function [8]. Suppose u are described by a piecewise constant trajectories (i.e., are described by events that at a certain time makes the value of u_i change from $u_{i\,old}$ to $u_{i\,new}$).

Simulate a system with the QSS algorithm means applying a variable-step techniques. The algorithm adjusts the time instant at which the state variable is re-evaluated to the speed of change of that state variable, and it is naturally asynchronous. This means that different state variables update their state values separately and independently of each other at different instants of time.

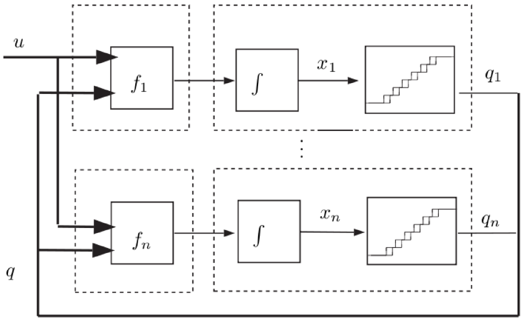


Figure 3: The scheme of a QSS model.

Specifically, the QSS algorithm consists in the creation of a coupled DEVS model, similar to the one in figure 3, where each state variable has an associated DEVS subsystem and the subsystem interconnection is based on the dependency between state variables and derivative equations. The events of the DEVS model are fired when the hysteretic quantization threshold are reached. The simulation therefore consists of three different steps:

- Search the DEVS subsystem that is the next to perform an internal transition, according to its internal time and to the derivative value. Suppose that the event time is t_{next} and the associated state variable is x_i. If $t_{next} > t_{inputevent}$ than set $t_{next} = t_{inputevent}$ and perform the input change.

- Advance the simulation time from current time to t_{next} and execute the internal transition function of the model associated to x_i or the input change associated to u_i.

- Propagate the new output event produced by the transition to the connected state variable DEVS models.

This approach is very interesting for parallel simulation since "*due to the asynchronous behavior, the DEVS models can be implemented in parallel in a very easy and efficient way*" [8]. As noticed, the QSS algorithm is naturally keen to be parallelized, because of the possibility to separately compute the derivatives state variables and the time events schedule; however some considerations are indeed.

The interested reader can refer to [9] for a detailed treatment on the matter, however, for the purposes of this work, the QSS integration method can be briefly described as follows.

For the first step, assume that the initial values of the state variables are known, the derivative of the state variables are computed using the model equations; this part of the code advantages from the MIMD execution model. After that calculation, the time of the new event is calculated; this code section exploits completely SIMD parallelism because all the computing threads execute the same code on different data portion. The second step consists in the time advance, a new event is registered if the values of one of the inputs changes or if one of the bounds of the quantized state function is reached. To verify the second possibility the minimum time advance for the state variable vector is taken into account. When an event occurs, each value of the state variables is re-computed, according to the new values of the inputs and/or the state variables and the quantized integrators are updated. Here the SIMD parallelism is exploited as well as in the previous part, due to the same reason (the same code executes on different data element). The last algorithm step does not need further explaination within the chosen architecture due to the fact that data are saved in the shared memory without need for propagation.

As shown, the specific architecture cannot be neglected when trying to asses the parallelization performance. A very careful analysis is needed to exploit the architecture dependent features. The first difference to be considered is the one between a *message passing* and a *shared memory* architecture. In [7] the authors make a comparison between these different architecture models.

For our application a message passing architecture would be interesting, but has some limitations. Each processor can manage a single or a group of DEVS subsystems, receiving events from the connected one. This is not particularly flexible, in fact, while the number of processors is fixed, the number of subsystems depends on the particular model. The grouping itself should be performed according to subsystems connection; in order to minimize the number of exchanged messages.

A shared memory architecture, as the Nvidia Tesla is, is more flexible but much attention has to be given to the algorithm definition. Since the Nvidia TESLA architecture requires all the computing cores in the same group to compute the same instruction at the same time, good performance can be achieved via the definition of a state vector array. Each derivative state value is calculated within a separate thread.

In this case the code to compute such values is different for each state variable, therefore the SIMD model is not performing well. A MIMD-fashion code should be produced. The speed-up is limited from the number of clusters present in the architecture, the execution is in fact parallel for each group of clusters. When all threads finish, the derivative values have been calculated and the threads execute the same portion of code (therefore speeding up) to calculate the next time event for each variable. This part of the code should gain an advantage from the SIMD model as every thread execute the same code on different data portion (i.e., following the *single instruction, multiple data* technique). After doing that the next time event of the QSS simulation is determined and processed.

In summary, for the particular architecture and programming technique, the *derivative calculation* part of the code is not completely parallel, while the *system advance* part takes full advantage of the hardware possibilities.

5 Extracting the Model from Modelica Code

The OpenModelica Compiler (OMC) [1] is an open source compiler and development environment for the Modelica language that can be used for, among other things, research in language technology and code generation. In this work we have extended the back-end of the compiler with a new module GPUpar that generates simulation code according to the specifications given in this paper. This module can be turned on and off with a compiler flag. If this module is instructed to run it will take the equation system right after the matching and index reduction phases and generate CUDA C-code.

5.1 Overview

A brief overview of the interesting internal call chain in the compiler can be seen in figure 4.

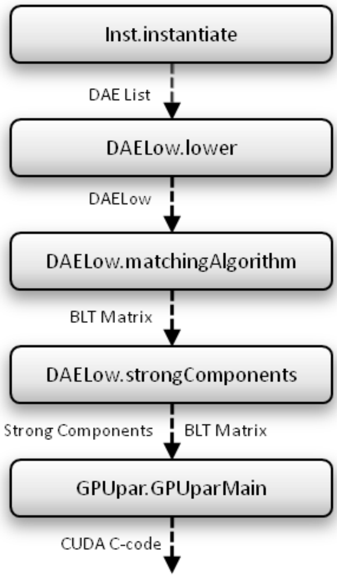


Figure 4: Internal call chain in the OpenModelica compiler to obtain parallel CUDA code.

The flattening phase takes the abstract syntax representation of the initial code and instantiates it (flattening, type checking, etc.) and the result is a list of so-called DAE elements. Here we are only interested in DAE elements that are equations. The list of DAE elements/equations is then transformed into a more suitable form called DAELow by DAELow.lower. The DAELow form contains the equations as well as all the variables and parameters. After this sorting, index reduction, strong component gathering, etc. is performed. The resulting data structures - BLT Matrix, strong components, DAELow form, etc. - are then passed into our new GPUpar module (in the normal case with serial simulation code we would call the module Simcodegen instead at this point).

5.2 GPUpar Module

In this module different kernel and header files are generated in succession. In order to generate the model-specific files, some data have to be computed from the DAELow form. The most important things to consider are:

- A *derivative* function which contains the algorithm for the time derivative computation is generated in the CUDA C-code for each state variable. If the time derivative calculation relies on other equations, they are also added to the *derivative* function.
- An *output* function for computing the output values is generated in the CUDA C-code for each output variable. As for the *derivative* function, each of them can also contain other equations if necessary.
- Initial variable (and parameter) values must be gathered from the list of variables in the DAELow form.

The additional equations necessary for the single *derivative*/*output* functions, where present, form a subtree having the main equation as the root node. An existing function (DAELow.markStateEquations) was slightly modified to handle with this problem. All the equations are also brought into solved form (explicit form) by calling Exp.solve and the equations are sorted by using information obtained in the sorting phase (which was run before GPUpar was called). The initial values are gathered in a rather straight-forward manner by traversing the list of variables. Finally, in the generated code some of the variables are stored in different arrays: xd (derivatives), x (state variables), y (output variables), u (input variables) and p (parameters). At the beginning of the GPUpar module an environment is created that contains a mapping between each variable/parameter and the array name plus the index number in this array. This environment is then used when the CUDA C-code is generated to find the correct array and index to print for a given variable.

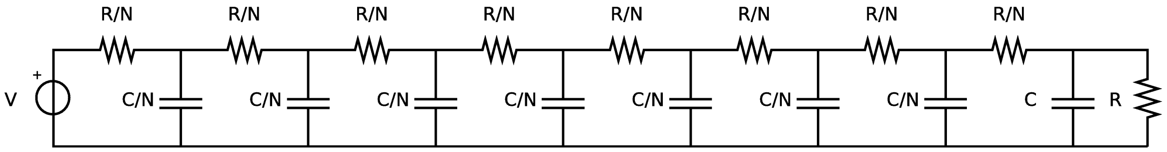


Figure 5: Test case example.

Appendix A contains a Modelica model and a part of the code necessary for simulating that model with the Nvidia architecture. In particular the missing files are model independent and can be found in [14].

6 Experimental Results

In this section a test case is presented, to evaluate the CUDA code performances. The two mentioned graphic cards are tested and a summary of the comparison between them is reported. The execution times are measured using the `clock()` function provided by the CUDA library. The initial time is obtained at the beginning of the program, before the memory allocation, in order to evaluate the architecture properly. The end time is measured when the simulation stops with the same function call and the difference between them is divided by the `CLOCKS_PER_SEC` constant, to compare architectures with different clock periods. The *parallel* algorithm is compared to the *sequential* one, where a single thread is executed on the graphic card and takes care of the computation sequentially.

The code for the circuit model of figure 5 is generated and executed. The depicted model has eight state variables that stands for the voltages in the eight capacitors. The model is then extended to sixteen, thirty-two and sixty-four state variables while keeping the same structure to prove the method scalability.

The following considerations apply to the model with N state variables. The circuit consists of a generator voltage that comprises $N-1$ different branches; each of them is composed by a resistor with resistance R/N and of a capacitor with capacitance C/N. The last branch is made up of the resistor with resistance R/N and a capacitor with capacitance C together with a resistor with resistance R in parallel. The only input of the system, in the following referred as u, is the voltage V, that is supposed to be a square wave with rise time and fall time of 1s and voltage of 1Volt. The equation model with $N = 8$ is therefore

$$\left\{\begin{array}{lcl}
\dot{x}_0(t) & = & \frac{N^2}{RC}(-2x_0(t)+u+x_1(t)) \\
\dot{x}_1(t) & = & \frac{N^2}{RC}(-2x_1(t)+x_0(t)+x_2(t)) \\
\dot{x}_2(t) & = & \frac{N^2}{RC}(-2x_2(t)+x_1(t)+x_3(t)) \\
\dot{x}_3(t) & = & \frac{N^2}{RC}(-2x_3(t)+x_2(t)+x_4(t)) \\
\dot{x}_4(t) & = & \frac{N^2}{RC}(-2x_4(t)+x_3(t)+x_5(t)) \\
\dot{x}_5(t) & = & \frac{N^2}{RC}(-2x_5(t)+x_4(t)+x_6(t)) \\
\dot{x}_6(t) & = & \frac{N^2}{RC}(-2x_6(t)+x_5(t)+x_7(t)) \\
\dot{x}_7(t) & = & \frac{N}{RC}(-R(\frac{N+1}{N})x_7(t)+x_6(t))
\end{array}\right. \qquad (3)$$

and can be easily generalized to $N = 8 \times i$ with i being an integer value ($i = 1, 2, 3, \dots$). The tests are conducted with $R = 1k\Omega$, $C = 1m$F and with a quantum of 0.001 for the QSS algorithm execution.

The results with the Nvidia Tesla GeForce 8600 can be seen in Table 1. Table 2 contains the results with the Nvidia Tesla C1060 when just one cluster is used to compute the derivative values, while Table 3 reports the data with the same graphic card when all the available clusters are used.

	parallel [s]	**sequential** [s]	**speed-up**
8-statevar	6.26	7.07	1.129
16-statevar	8.04	10.27	1.277
32-statevar	27.02	45.55	1.685
64-statevar	103.18	507.38	4.917

Table 1: Execution times and speed-up with the GeForce 8600.

	parallel [s]	**sequential** [s]	**speed-up**
8-statevar	1.06	5.71	5.387
16-statevar	8.11	9.07	1.118
32-statevar	22.91	47.30	2.065
64-statevar	208.76	711.00	3.406

Table 2: Execution times and speed-up with the C1060 using one cluster for the derivative calculation.

In figure 6 a summary of the obtained speed-up values is presented.

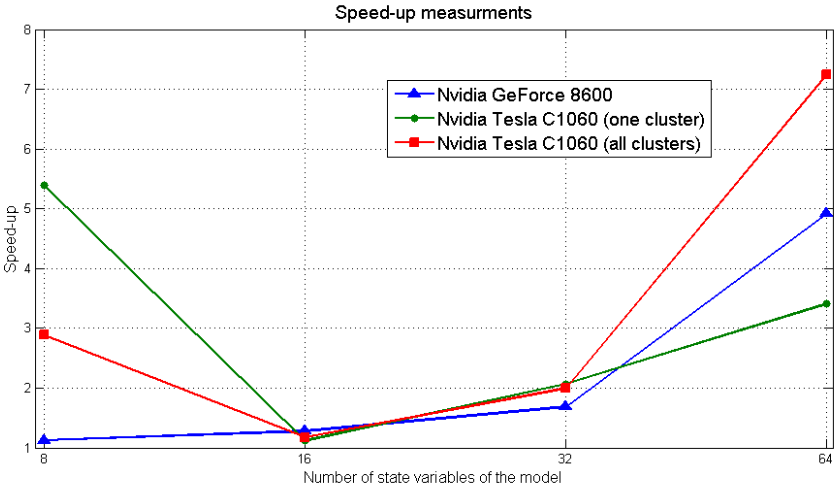


Figure 6: Speed-up measurements: comparison between a GeForce 8600 and an Nvidia Tesla C1060 when the number of state variables in the model changes.

The results presented are promising but should investigated further in order to understand the scalability of the proposed solution. Moreover, the new tools available from Nvidia (e.g., a profiler) give the possibility of a more careful analysis of the performances.

7 Conclusions and Future Research

This work considers the parallelization of the QSS algorithm using a GPGPU. As shown in section 4 the implementation of the algorithm can not neglect the particular hardware architecture. In this case the difficulties are essentially related to the fact that the Nvidia Tesla GPGPU is not a completely general parallel architecture. The memory consumption should also be taken into account. In particular, a problem with 256 state variables requires more than $\frac{(5\times 64+1\times 32)\times 256}{8}[Bytes] = 11[Mb]$, while a case with 1024 state variables would require $43[Mb]$.

Surely, the side effects of the diverging branches has to be furthermore reduced. A comparison between the code that uses just one cluster of multiprocessors and a complete has been performed; however, further studies are still necessary to investigate possible extensions, e.g. for exploiting the computational power of each processor within the cluster.

	parallel [s]	**sequential** [s]	**speed-up**
8-statevar	1.98	5.71	2.884
16-statevar	7.73	9.07	1.173
32-statevar	23.73	47.30	1.993
64-statevar	98.09	711.00	7.248

Table 3: Execution times and speed-up with the C1060 using all the clusters for the derivative calculation.

Despite this, the results are promising albeit preliminary. Future work will compare QSS-based parallel method with other parallel implementations and investigate how the Tesla architecture thread manager allocates threads to the different computing cores. A profiling analysis is needed too, in order to understand if the limitations in the speed-up are caused by physical limits of the architecture or due to the non-exploitable hardware facilities.

References

[1] The OpenModelica project webpage: http://www.openmodelica.org.

[2] *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide*, 2008.

[3] P. Aronsson. *Automatic Parallelization of Equation-Based Simulation Programs*. PhD thesis, Linköping University, Department of Computer and Information Science, 2006.

[4] P. Bailey, J. Myre, S.D.C. Walsh, D.J. Lilja, and M.O. Saar. Accelerating lattice boltzmann fluid flow simulations using graphics processors. In *Processing the 2009 International Conference on Parallel (ICPP)*, 2009.

[5] F.E. Cellier and E. Kofman. *Continuous System Simulation*. Springer, 2006.

[6] M. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, C-21:948–960, 1972.

[7] A. C. Klaiber and H. M. Levy. A comparison of message passing and shared memory architectures for data parallel programs. *SIGARCH Comput. Archit. News*, 22(2):94–105, 1994.

[8] E. Kofman. *Discrete Event Based Simulation and Control of Continuous Systems*. PhD thesis, School of Electronic Engineering - FCEIA Universidad Nacional de Rosario, 2003.

[9] Ernesto Kofman and Sergio Junco. Quantized-state systems: a DEVS approach for continuous system simulation. *Trans. Soc. Comput. Simul. Int.*, 18(3):123–132, 2001.

[10] H. Li and L. Petzold. Efficient parallellization of stochastic simulation algorithm for chemically reacting systems on the graphics processing unit. Technical report, Dept. Computer Science, University of California, Santa Barbara, 2008.

[11] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA tesla: A unified graphics and computing architecture. *Micro, IEEE*, 28(2):39–55, 2008.

[12] H. Lundvall. Automatic paralleliztion using pipelining for equation-based simulation languages, 2008. Lic. Thesis.

[13] H. Lundvall, K. Stavåker, P. Fritzson, and C. Kessler. Automatic parallelization of simulation code for equation-based models with software pipelining and measurements on three platforms. *Computer architecture news, Special issue MCC08 - Multicore computing 2008*, 36(5), 2008.

[14] M. Maggio. Simulazione di modelli orientati agli oggetti su architetture parallele tramite algoritmo QSS. Master thesis. Politecnico di Milano, Dipartimento di Elettronica ed Infomazione, 2008.

[15] Michael Schwarz and Marc Stamminger. Fast GPU-based adaptive tessellation with CUDA. *Computer Graphics Forum*, 28(2):365–374, 2009.

[16] H. Shutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3).

[17] Bernard P. Zeigler, Tag G. Kim, and Herbert Praehofer. *Theory of Modeling and Simulation*. Academic Press, London, January 2000.

[18] Bernard P. Zeigler, Hae Sang Song, Tag Gon Kim, and Herbert Praehofer. DEVS framework for modelling, simulation, analysis, and design of hybrid systems. In *In Proceedings of HSAC*, pages 529–551. Springer-Verlag, 1996.

Appendix A: Code References

The following code example contains the model dependent part of the code for the generation of the experiment presented in Section 6 with 8 state variables, where three output variables are defined.

```
model Test_Model
 parameter Integer N = 8;
 input Real inputVars[1](start = 0.0);
 Real stateVars[N](start = 0.0);
 output Real outputVars[3];
equation

 der(stateVars[1]) = N*N * (-2.0*stateVars[1] +
   stateVars[2] + inputVars[1]);

 for i in 2:(N-1) loop
 der(stateVars[i]) = N*N * (-2.0*stateVars[i] +
   stateVars[i-1] + stateVars[i+1]);
 end for;

 der(stateVars[N]) = N * (stateVars[N-1] -
   1000 * ((N+1)/N) * stateVars[N]);

 outputVars[1] = stateVars[1];
 outputVars[2] = stateVars[4];
 outputVars[3] = stateVars[N];
end Test_Model;
```

The translation phase produces two output files: `model.h` and `model.cu`. The first one is the C-CUDA header and contains the function prototypes of the routine contained in the second one.

```
/********************************
 * MODEL.H
 ********************************/
#ifdef _MODEL_H
#define _MODEL_H

#define NUMBER_STATES 8
#define NUMBER_INPUTS 1
#define NUMBER_OUTPUT 3
#define NUMBER_EVENTS 10
#define SIMULATION_TIME 10
#define SIMULATION_STEP 0.001

/* Initializations */
void initializeSystem(float* x, float* u);

void initializeEvents(float* t, unsigned* i, float* v);
```

```
/* Derivative calculation */
__global__ void derivative
 (float* dx, float* x, float* u, float* t, unsigned* c);
__device__ void dx7
 (float* dx, float* x, float* u, float* t, unsigned* c);
__device__ void dx6
 (float* dx, float* x, float* u, float* t, unsigned* c);
__device__ void dx5
 (float* dx, float* x, float* u, float* t, unsigned* c);
__device__ void dx4
 (float* dx, float* x, float* u, float* t, unsigned* c);
__device__ void dx3
 (float* dx, float* x, float* u, float* t, unsigned* c);
__device__ void dx2
 (float* dx, float* x, float* u, float* t, unsigned* c);
__device__ void dx1
 (float* dx, float* x, float* u, float* t, unsigned* c);
__device__ void dx0
 (float* dx, float* x, float* u, float* t, unsigned* c);

 /* Output calculation */
__global__ void output
 (float* y, float* x, float* u, float* t, unsigned* c);
__device__ void y2
 (float* y, float* x, float* u, float* t, unsigned* c);
__device__ void y1
 (float* y, float* x, float* u, float* t, unsigned* c);
__device__ void y0
 (float* y, float* x, float* u, float* t, unsigned* c);

#endif

/*********************************
 * MODEL.CU
 ********************************/
#include "inclusion.h"
#include "model.h"

/* Initializations */
void initializeSystem(float *x, float* u) {
  int i;
  u[0]=0.0;
  for(i=0;i<NUMBER_STATES;i++) x[i]=0.0;
}

void initializeEvents(float* t, unsigned* i, float* v) {
  t[0] = 1; i[0] = 0; v[0] = 1;
  t[1] = 2; i[1] = 0; v[1] = 0;
  t[2] = 3; i[2] = 0; v[2] = 1;
  t[3] = 4; i[3] = 0; v[3] = 0;
  t[4] = 5; i[4] = 0; v[4] = 1;
  t[5] = 6; i[5] = 0; v[5] = 0;
  t[6] = 7; i[6] = 0; v[6] = 1;
  t[7] = 8; i[7] = 0; v[7] = 0;
  t[8] = 9; i[8] = 0; v[8] = 1;
  t[9] = 10;i[9] = 0; v[9] = 0;
}

/* Derivative calculation */
__global__ void derivative
(float* dx, float* x, float* u, float* t, unsigned* c) {
int i = threadIdx.x;
switch(i) {
  case 7: dx7(dx, x, u, t, c); break;
  case 6: dx6(dx, x, u, t, c); break;
  case 5: dx5(dx, x, u, t, c); break;
  case 4: dx4(dx, x, u, t, c); break;
  case 3: dx3(dx, x, u, t, c); break;
  case 2: dx2(dx, x, u, t, c); break;
  case 1: dx1(dx, x, u, t, c); break;
  case 0: dx0(dx, x, u, t, c); break;
  }
}
__device__ void dx7
(float* dx, float* x, float* u, float* t, unsigned* c) {
dx[7] = 8.0 * (x[6] - 1000 * 1.0625 * x[7]);
}
__device__ void dx6
(float* dx, float* x, float* u, float* t, unsigned* c) {
dx[6] = 16384.0 * (-2.0 * x[6] + x[5] + x[7]);
}
__device__ void dx5
(float* dx, float* x, float* u, float* t, unsigned* c) {
dx[5] = 16384.0 * (-2.0 * x[5] + x[4] + x[6]);
}
__device__ void dx4
(float* dx, float* x, float* u, float* t, unsigned* c) {
dx[4] = 16384.0 * (-2.0 * x[4] + x[3] + x[5]);
}
__device__ void dx3
(float* dx, float* x, float* u, float* t, unsigned* c) {
dx[3] = 16384.0 * (-2.0 * x[3] + x[2] + x[4]);
}
__device__ void dx2
(float* dx, float* x, float* u, float* t, unsigned* c) {
dx[2] = 16384.0 * (-2.0 * x[2] + x[1] + x[3]);
}
__device__ void dx1
(float* dx, float* x, float* u, float* t, unsigned* c) {
dx[1] = 16384.0 * (-2.0 * x[1] + x[0] + x[2]);
}
__device__ void dx0
(float* dx, float* x, float* u, float* t, unsigned* c) {
dx[0] = 16384.0 * (-2.0 * x[0] + x[1] + u[0]);
}
/* Output calculation */
__global__ void output
(float* y, float* x, float* u, float* t, unsigned* c) {
int i = threadIdx.x;
switch(i) {
  case 2: y2(y, x, u, t, c); break;
  case 1: y1(y, x, u, t, c); break;
  case 0: y0(y, x, u, t, c); break;
  }
}
__device__ void y2
(float* y, float* x, float* u, float* t, unsigned* c) {
y[2] = x[7];
}
__device__ void y1
(float* y, float* x, float* u, float* t, unsigned* c) {
y[1] = x[3];
}
__device__ void y0
(float* y, float* x, float* u, float* t, unsigned* c) {
y[0] = x[0];
}
```

Modeling and Control of a Parallel Robot Using Modelica

Isolde Dressler[1] Johannes Schiffer[1,2] Anders Robertsson[1]
[1]Department of Automatic Control, Lund University
Box 118, 22100 Lund, Sweden
[2] Institute for System Theory and Automatic Control
University of Stuttgart, Germany

Abstract

A new type of high-performance robots has been developed by ABB Robotics, the Robotics Lab at Lund University and Güdel AG, Switzerland. In all parts of the project, ranging from the simulation of the kinematic configuration and reachable workspace, and kinematic and dynamic calibration/grey-box identification, and to code generation of controllers and optimal switching strategies for hybrid control, Modelica and Optimica provide very valuable functionality. We will make a short overview of the different aspects used during the development.

Keywords: Robotics, Multi Body Systems, Dual motor control

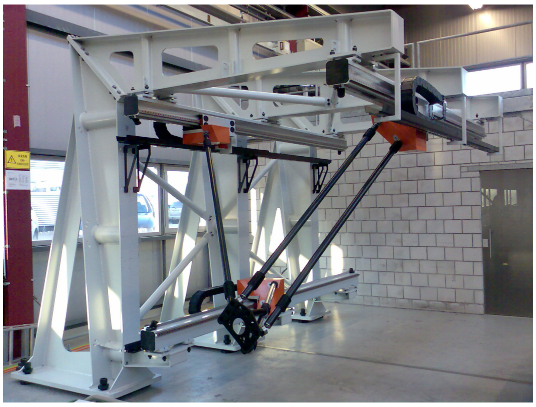


Figure 1: Full size Gantry-Tau prototype developed within the SMErobot$^{\text{TM}}$project. The carts (red) are controlled in a coordinated way along the three rails to move the tool/end-plate along a desired trajectory.

1 Introduction

A new type of high-performance manipulator, the Gantry-Tau robot [1], has been developed within the EU FP-6 project SMErobot$^{\text{TM}}$[2] by ABB Robotics, the Robotics Lab at Lund University and Güdel AG, Switzerland. The new concept, which is based on the parallel configuration of the robot's joints (parallel robots), see Fig. 1, is modular, has a large open workspace, is easy to scale and has the inherent benefit of very low inertia of the moving robot parts. This, together with high stiffness of joints and arms, makes it possible to build high-performance robots with respect to accuracy, speed, stiffness and mechanical bandwidth.

Modelica and the MultiBody Library [3] can advantageously be used for modeling and control of robots. In [4] we have reported on how the dynamic model equations of the Gantry-Tau robot were extracted from a MultiBody Modelica model of the robot. The control of a parallel robot using an inverse dynamic model generated by Dymola is presented in [5].

This article presents how different functionalities of Modelica have been used for modeling, simulation, identification and controller generation of the Gantry-Tau manipulator during the different project phases. Two main areas will be discussed: The first is the calibration of the robot's kinematics using Optimica [6], the second the optimization of the actuator control.

In [4], the authors carried out kinematic calibration of the Gantry-Tau robot using a scripting language. In this work, it is shown how a Modelica model for optimization is generated of a subset of the original MultiBody model of the robot. The kinematic parameters are then optimized using Optimica.

To reduce backlash and improve the actuator positioning, the usage of two motors for each cart has been investigated. The actuator system was

DOI: 10.3384/ecp09430118

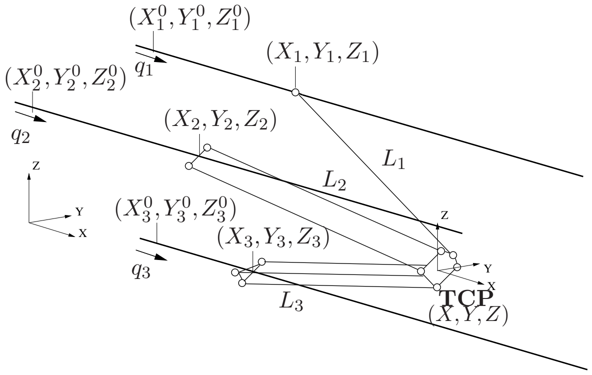


Figure 2: Gantry-Tau schema with variable and parameter notation

modeled in Modelica and a hybrid switching control concept has been tested and optimized using Dymola.

The article is structured as follows: In Sect. 2, the Gantry-Tau manipulator is presented, Sect. 3 describes the kinematic calibration and Sect. 4 the dual motor control. In Sect. 5 the results and methodology are discussed and Sect. 6 concludes the article.

2 The Gantry-Tau Robot

The 3 degree-of-freedom (DOF) parallel Gantry-Tau robot (Figs 1 and 2) consists of three kinematic chains. A prismatic actuator, implemented as a cart moving on a track is connected to an end-effector plate via a link cluster. The altogether 6 links, mounted with passive spherical joints on cart and plate, are distributed in a 3-2-1 configuration to the 3 link clusters. The spherical joint placement on carts and plate is such that links belonging to one cluster form parallelograms, which assures a constant end-effector orientation.

The actuation for the linear motion of the carts along the rails are provided by a so called *rack-and-pinion system*. In all transmissions friction and backlash may severely degrade the performance and accuracy. However, with a rack-and pinion system several motors/carts/robots may be mounted on the same rail and can be controlled independently of each other, which is not the case if the linear motion transmission is made by e.g., ball screws. In Sect. 4 this property will be used for dual motor control and backlash reduction, see also Fig. 5.

As the end-effector orientation is constant for all cart positions, it is sufficient to consider one link per kinematic chain. In addition to a full model with 6 links according to Fig. 2, a simplified Gantry-Tau model has been implemented using the Modelica MultiBody Library (Fig. 3). Here the end-effector orientation is kept constant by a block (blue rectangle in the bottom) which contains 3 passive, serially connected prismatic joints aligned with the 3 coordinate axes. Each of the 3 kinematic chains visible in Fig. 3 consists of a model for track and cart positioned in the base coordinate system by a `FixedTranslation` block and a link connected to the end-effector plate. Input signals of the model are the cart positions.

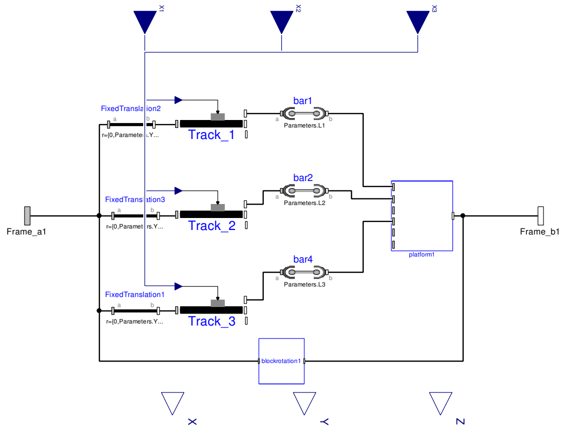


Figure 3: Modelica model of a Gantry-Tau PKM

3 Kinematic Calibration using Optimica

To determine the kinematic parameters, the end-effector position (X, Y, Z) was recorded with a laser tracker for a number of actuator positions (q_1, q_2, q_3). The altogether 21 parameters to optimize are link lengths L_i, the vectors in track direction c_i and the track offsets $(X_i^{\text{offset}}, Y_i^{\text{offset}}, Z_i^{\text{offset}})$, which accumulate the start positions (X_i^0, Y_i^0, Z_i^0) and the offsets between spherical joints and tool center point (TCP) on the end-effector plate, $i = 1, 2, 3$ (see Fig. 2).

The calibration with Optimica is divided in several steps. As the MultiBody Library is not yet compatible with Optimica, a flat Modelica model for optimization has to be generated. For that, the model equations are extracted automatically from the MultiBody model. With a subset of these equations, the kinematic constraint equations, a model for optimization is then generated. After the optimization, the results are validated.

The extraction of the model equations was first presented in [4]. When translating the MultiBody model in Dymola, the `dsmodel.mof` file with a listing of the translated Modelica code can be generated. This file contains a section with the relevant model equations and assignments relating the large number of variables and parameters that the MultiBody model contains. A script written in python parses this file, extracts the model equations and uses the assignments to successively substitue variables and parameters until the equations are expressed in a desired and previously determined set of parameters and variables. The equations for the kinematic constrains are:

$$\begin{aligned} 0 &= L_i^2 - \Big((X_i - X)^2 + (Y_i - Y)^2 + \\ &\quad +(Z_i - Z)^2\Big),\ i = 1,2,3, \qquad (1) \end{aligned}$$

where the cart position (see Fig. 2) $(X_i, Y_i, Z_i)^T = (X_i^{\text{offset}}, Y_i^{\text{offset}}, Z_i^{\text{offset}})^T + q_i \cdot c_i$.

The remaining 9 equations can be found in [4].

Using the measurement data and the kinematic constraint equations among the extracted equation system, a new Modelica model for optimization is then generated:

```
model GTPKinCalib

  parameter Real q1[N] = {data};
  parameter Real q2[N] = {data};
  parameter Real q3[N] = {data};

  parameter Real X[N] = {data};
  parameter Real Y[N] = {data};
  parameter Real Z[N] = {data};

  parameter Real L1;
  parameter Real X1offset;
  parameter Real Y1offset;
  parameter Real Z1offset;
  parameter Real c1[3];
  parameter Real L2;
  parameter Real X2offset;
  ...

  Real f1[N];
  Real f2[N];
  Real f3[N];
  Real cost;
equation
  for i in 1:N loop
    f1[i] = kinematic constraint link 1;
    f2[i] = kinematic constraint link 2;
    f3[i] = kinematic constraint link 3;
  end for;

  cost = f1[1]^2+f2[1]^2+f3[1]^2+ ...;

end GTPKinCalib;
```

The variables `fi[N]` in the model `GTPKinCalib` are the residuals for equation (1) for the given measurement data and parameter values. The variable `cost` is then minimized using Optimica.

3.1 Results

For kinematic calibration, the TCP position (X, Y, Z) was recorded for 176 robot poses with known actuator positions (q_1, q_2, q_3) with a laser tracker. Every second measurement was used for calibration, the remaining ones for the validation of the optimization results.

Figure 4 shows the validation results of the calibration. The calibrated model has a mean absolute positioning error of about 140 μm. Very similar results for parameters and positioning accuracy can be obtained with the Matlab script used in [4].

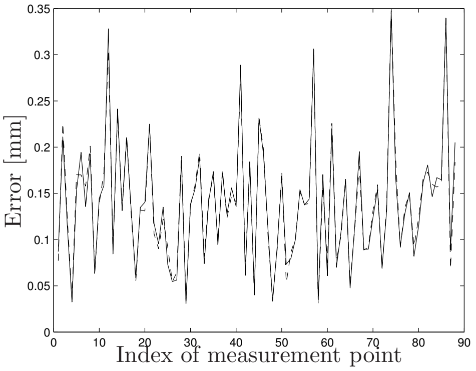


Figure 4: Positioning accuracy of the Gantry-Tau after calibration: absolute positioning error of TCP for the validation measurement points. The model calibrated with Optimica (solid) and the model obtained with the Matlab script used in [4] (dashed) give very similar results.

4 Dual Motor Control

To meet the demands on the system, kinematic sensitivity analysis shows that the robot needs backlashfree gearboxes to achieve the desired positioning accuracy of the robot's TCP. The actuator and drive-line of the robot are based on the rack-and-pinion principle which has been simulated in Modelica.

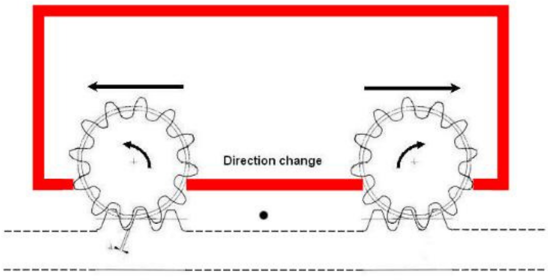


Figure 5: Backlash is present in gearboxes and connection to rails (rack-and-pinion); By using two motors on each cart, which work in opposite directions, improved positioning accuracy and stiffness can be achieved.

In the following, a model representing the robot's actuator drive-line and including a control law for the dual motor control is implemented in Dymola. We will then show, how the Optimization Function of the Dymola Design Library both can be used to optimize the parameters of the motor control law and how to find the the optimal switching instant for when to change direction between the motors..

4.1 Model and control law

A model representing the robot's actuator drive-line consisting of two driving motors and one cart is implemented in Modelica as shown in Fig. 6. Each motor is connected to the load by a free shaft inertia. The backlash is modeled by the 'ElastoBacklash'-block of the Mechanics-package.

This kind of system represents an extension of a two-mass system exhibiting backlash. The latter has been well-studied in literature since the 1940s, due to the fact that in most cases the considered plants which exhibit backlash-effects may be modeled as such a two-mass system. Some older and more recent research on this standard plant can be found in [7], [8]. In the present case, the additional second motor is considered by introducing a third mass, which leads to a three-mass system with two nonlinearities representing the backlash between each motor and the cart.

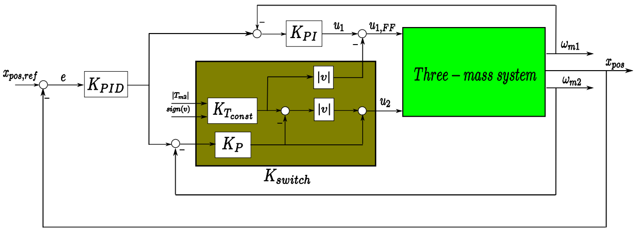


Figure 7: Nonlinear dual motor control structure of a three-mass system. K_{switch} defines the switching of the slave motor and provides the master motor with additional information through a feed-forward structure.

The implemented control structure aims to use both motors for the motion drive while the system is not in backlash and to switch the operating direction of the second motor when the system gets into backlash for fast closing the backlash gap and improving the position accuracy of the carts, and thus of the robot TCP. In the traditional case of a cart driven by just one motor limit cycles may occur in the system due to the backlash. These can be eliminated by the dual motor control. Therefore a nonlinear, smooth switching control law based on a switching variable $v \in [-1, 1]$ ('direction') and a switching function K_{switch} is designed (see Fig. 7). For first simulations nominal parameters for the controllers and the switching function are used. Herewith, the results shown in Fig. 8 are obtained. As desired, no limit cycles occur.

4.2 Parameter Optimization

This section aims to improve the system performance by optimizing the controller parameters and the switching of the second motor. For this purpose, the Optimization Function of the Dymola Design Library is used. This design tool provides several optimization algorithms and allows to optimize parameters of a Dymola model with respect to certain criteria. An introduction to the function is given in [9].

4.2.1 Controller parameter optimization

In a first step, we aim to optimize parameters for the outer-loop PID-controller, controlling the cart position x_{pos}. Therefore the following cost function is defined

$$f = \min\left(\max\left(\frac{1}{W_1}\text{riseTime}\left(x_{pos}\right) + \frac{1}{W_2}\text{overshoot}(x_{pos}) + \frac{1}{W_3}\text{settlingTime}(x_{pos})\right)\right)$$

with weights $W_1 = 1.7473$, $W_2 = 10^{-4}$ and $W_3 = 3.957$. Blocks to determine the rise time, the overshoot and the settling time are provided by the Dymola Design Library.

For the optimization an operation with a constant torque on the second motor is chosen, thus $v = -1$. The values of the weighting parameters correspond to the obtained results of the characteristics when simulating with the nominal controller parameters. The available tuning parameters are K_p, T_i and T_d of the PID-controller. The optimization is carried out using the different algorithms implemented in the Optimization Function. As starting values the nominal values of the PID-controller

$$K_P = 200, T_i = 1.5, T_d = 0.05$$

are used. The start value of the cost function is then 3.00354. The Optimization Function allows also to set bounds on parameters. We set the following bounds

$$Kp \in [100,300],\ Ti \in [0.5,2],\ Td \in [0.01,0.1].$$

An overview of the results is provided in Table 1. The best results are obtained by Pattern Search and Genetic Algorithm.

4.2.2 Switching parameter optimization

In this section, we consider the switching strategy, that is when to change direction of the second motor (slave motor), see Fig. 7. The switching is based on the relative position error

$$e_{abs} = \frac{|x_{ref,new} - x_{pos}|}{|x_{ref,new} - x_{ref,old}|}.$$

A schematic view of the switching v is depicted in Fig. 9, where the parameters e_{max} and e_{min} parametrize the curve and thus can be used as tuning parameters for the optimization. For a more detailed description, see [10].

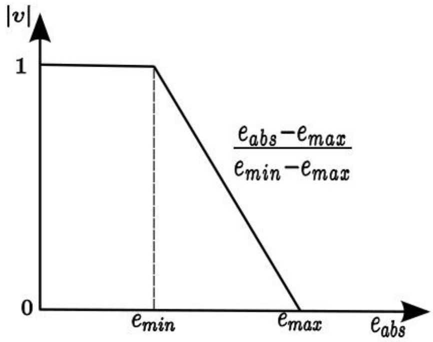


Figure 9: Switching function $v = f(e_{abs})$. $|v|$ takes the value 1, when $e_{abs} < e_{min}$. If $e_{abs} > e_{max}$, v takes the value 0.

To minimize energy and reduce overshoot for a step response in the position reference, the following cost function is defined

$$f_1 = \min(\max(\frac{1}{W_1}\int_0^T (u_1^2 + u_2^2)d\tau + \frac{1}{W_2}\text{overshoot}(x_{pos}))), \quad (2)$$

with $W_1 = \psi \cdot 2165$ and $W_2 = 3 \cdot 10^{-4}$. The weighting parameters correspond to the values obtained for the characteristic parameters of the optimization function, when operating with the previously chosen values $e_{max} = 0.25$ and $e_{min} = 0.01$, as well as an input step reference of $x_{pos,ref} = 0.1$ m and a simulation time of $t = 10$ s. As the optimization goal consists of minimizing the energy input by avoiding any overshoot in the system's step response, the energy input is additionally weighted with a factor $\psi = 10$.

For the optimization different start values and optimization methods are chosen. As initial values for the switching parameters two sets are chosen, [$e_{max} = 0.5$, $e_{min} = 0.25$] and [$e_{max} = 0.25$, $e_{min} = 0.01$]. The cost function has then a start value of $f_1(start) = 9.54322$ and $f_1(start) = 10.9464$ respectively. The tuning parameters are limited to $e_{max} \in [0.1,1]$ and $e_{min} \in [0.01,1]$ in order to avoid switchings in the area where limit cycles occur. An overview of the different setups and the corresponding results is given in Table 2.

All algorithms give similar results for the optimal switching parameters. These lie in a range of $e_{max} \in [0.62,0.69]$ and $e_{min} \in [0.40,0.45]$. Only the SQP and the Simplex-method lead to different results when starting with [$e_{max} = 0.25$, $e_{min} = 0.01$]. However, the Genetic Algorithm and the Pattern Search seem to give more reliable results, as the value of the cost function and the optimal switching parameters are almost identical for both initial sets.

To recheck the optimization results, the cost function f_1 is reformulated to f_2

$$f_2 = \min\left(\max\left(\frac{1}{W_1}\int_0^T (u_1^2+u_2^2)d\tau,\right.\right.$$
$$\left.\left.\frac{1}{W_2}\text{overshoot}(x_{pos})\right)\right). \qquad (3)$$

Then the start values are $f_2(start) = [8.8531, 0.6901]$ and
$f_2(start) = [10.02246, 0.9240]$. The results of this optimization are shown in Table 3. For the cases with initial values $[e_{max} = 0.5,\ e_{min} = 0.25]$ the results are similar to the ones obtained with the previous cost function. However, for the initial set $[e_{max} = 0.25,\ e_{min} = 0.01]$ there seem to exist at least two different minima, one in the neighbourhood of $[e_{max} = 1.0,\ e_{min} = 0.1]$ and one around $[e_{max} = 0.6,\ e_{min} = 0.4]$. Again the Genetic Algorithm gives the best results. The optimal values for the switching parameters obtained with this method are almost identical to the ones obtained with the previous cost function f_1. Thus, one can conclude that a pair of parameters $[e_{max} \approx 0.6,\ e_{min} \approx 0.4]$ may satisfy the optimization goal best.

As a consequence, the switching parameters are set to $[e_{max} = 0.6,\ e_{min} = 0.4]$. Then the integrated square sum of the required input signals for a reference step of $x_{pos,ref} = 0.1$ m and a simulation time of $t = 10$ s is reduced from 2165 to 1845, which represents an energy saving of about 15 %.

5 Discussion

The authors showed that the kinematic calibration method presented gives accurate results. In comparison to the Matlab script used in [4], the method is more flexible.

Changes in the MultiBody model of the Gantry-Tau robot, which would make a cumbersome reprogramming of a calibration script necessary, can be handeled with minor changes. Such changes may include kinematic error models (e.g. to consider all 6 links in a slightly non-ideal configuration so that the end-effector orientation varies) or new robot components (e.g. to increase the robot's DOF).

A similar procedure can be used for calibrating the dynamic model of the Gantry-Tau robot or models of a different robot.

6 Conclusion and Future Work

This article shows how different functionalities of Modelica were used for modeling, identification and controller generation for the parallel kinematic Gantry-Tau robot. The work focuses on two aspects, kinematic calibration with Optimica and the evaluation and optimization of the actuator system control.

A method for kinematic calibration of the Gantry-Tau robot using Modelica and Optimica was presented and shown to give accurate results.

A nonlinear three-mass system representing the robots actuator drive-line and a previously designed switching control law has been implemented in Dymola, which Optimization Function of the Dymola Design Library has then been used to optimize the control and switching parameters.

In the future, the flexibility of the calibration method presented can be used for calibrating a kinematic error model. With a similar procedure, the dynamic model of the Gantry-Tau will be calibrated and the inverse dynamic model used for feedforward control. The dual motor control tested successfully in simulations will be implemented and tested in practice. The possibility of code generation for hardware-in-the-loop simulations from the Gantry-Tau Modelica models presented here will be considered.

7 Acknowledgements

This work has partially been funded by the European Commission's Sixth Framework Programme under grant no. 011838 as part of the Integrated Project SMErobot^{TM}. The authors would like to thank Johan Åkesson for discussions about kinematic calibration with Optimica and ABB Robotics for supporting the laser tracker measurements.

References

[1] L. Johannesson, V. Berbyuk and T. Brogårdh, "Gantry-Tau – A New Three

Degrees of Freedom Parallel Kinematic Robot", *in Parallel Kinematic Machines in Research and Practice; The 4th Chemnitz Parallel Kinematics Seminar*, 2004, pp. 731-734.

[2] SMErobot$^{\text{TM}}$ homepage: `http://www.smerobot.org` (2009).

[3] M. Otter, H. Elmqvist, S.-E. Mattson, "The New Modelica MultiBody Library", *in Proc. of the 3rd International Modelica Conference*, Linköping, Sweden, 2003, pp. 311-330.

[4] I. Dressler, A. Robertsson and R. Johansson, "Accuracy of Kinematic and Dynamic Models of a Gantry-Tau Parallel Kinematic Robot", *in Proc. International Conference on Robotics and Automation (ICRA'07)*, Rome, 2007.

[5] M. Krabbes and C. Meißner, Dynamic modeling and control of a 6 DOF parallel kinematics. In: Proceedings of the 5th Modelica Conference 2006, Vienna, Austria, Modelica Association, 2006.

[6] J. Åkesson, Optimica–An Extension of Modelica Supporting Dynamic Optimization. In: Proceedings of the 6th Modelica Conference 2008, Bielefeld, Germany, Modelica Association, 2008.

[7] M. Nordin and P.-O. Gutman (2002). Controlling mechanical systems with backlash - a survey. In *Automatica* 38 (pp. 1633-1649).

[8] P. Rostalski, T. Besselmann, M. Baric, F. Van Belzen and M. Morari (2007). A hybrid approach to modelling, control and state estimation of mechanical systems with backlash. In *International Journal of Control* Vol.80, No. 11 (pp. 1729-1740).

[9] H. Elmqvist, H. Olsson, S.E. Mattsson, D. Brück, C. Schweiger, D. Joos, M. Otter (2005). Optimization Design and Parameter Estimation. The Modelica Association.

[10] J. Schiffer (2009), Dual motor control for backlash reduction. Master's Thesis report, Department of Automatic Control, Lund University, Sweden, ISRN LUTFD2/TFRT--5841--SE.

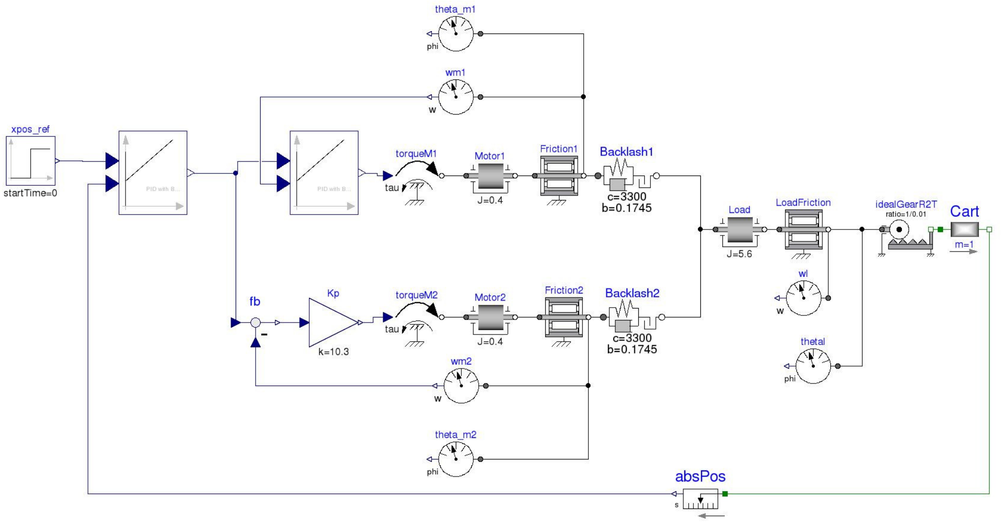


Figure 6: Three-mass system representing the robots actuator drive-line implemented in Dymola. The backlash is represented using 'ElastoBacklash'-block of the Mechanics-package.

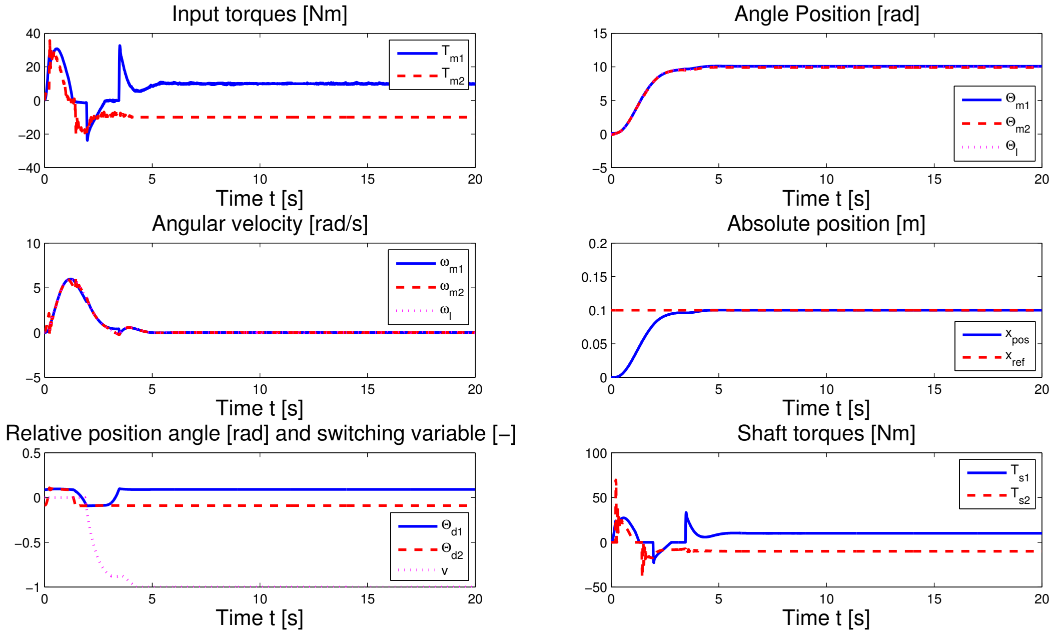


Figure 8: Dual motor control with $v = f(e_{abs})$ simulated with Dymola. The limit cycles are oppressed and the required controller energy is distributed on both motors in the beginning of the motion. Furthermore a smooth response is obtained and the strategy is robust against disturbances.

Table 1: Optimization of controller parameters. The Genetic algorithm gives the best result. The start value of the cost function is 3.00354.

Optimal values $[K_p,T_i,T_d]$	**Optimization method**	**Optimal value f**
[292.01, 1.24, 0.01]	Pattern Search	1.67645
[229.32, 1.38, 0.05]	SQP	1.82310
[223.38, 1.4, 0.073]	Simplex	1.88427
[281.92, 1.25, 0.087]	Genetic Algorithm	1.67730

Table 2: Optimization of switching parameters with cost function f_1 of Eq.(2). The Genetic algorithm gives the best results. The start value of the cost function is 10.9464.

Start values	**Optimization method**	**Optimal value** f_1	**Optimal values**
[0.5, 0.25]	Pattern Search	9.08898	[0.69, 0.41]
[0.5, 0.25]	SQP	9.06205	[0.62, 0.44]
[0.5, 0.25]	Simplex	9.08140	[0.63, 0.45]
[0.5, 0.25]	Genetic Algorithm	9.06148	[0.62, 0.45]
[0.25, 0.01]	Pattern Search	9.08444	[0.68, 0.4]
[0.25, 0.01]	SQP	9.47738	[1, 0.01]
[0.25, 0.01]	Simplex	9.19005	[1, 0.24]
[0.25, 0.01]	Genetic Algorithm	9.06148	[0.62, 0.45]

Table 3: Optimization of switching parameters with cost function f_1 of Eq.(3). The Genetic algorithm gives again the best results. The start values of the cost function are [10.02246, 0.924029].

Start values	**Optimization method**	**Optimal value** f_1	**Optimal values**
[0.5, 0.25]	Pattern Search	8.50622, 0.51865	[0.67, 0.40]
[0.5, 0.25]	SQP	8.60525, 0.51835	[0.55, 0.33]
[0.5, 0.25]	Simplex	8.49155, 0.57118	[0.60, 0.44]
[0.5, 0.25]	Genetic Algorithm	8.49167, 0.57253	[0.61, 0.45]
[0.25, 0.01]	Pattern Search	8.80359, 0.65697	[1, 0.033]
[0.25, 0.01]	SQP	8.81529, 0.66332	[1, 0.01]
[0.25, 0.01]	Simplex	8.55782, 0.58015	[0.98, 0.28]
[0.25, 0.01]	Genetic Algorithm	8.49013, 0.57150	[0.62, 0.44]

Modelling and Simulating the Efficiency and Elasticity of Gearboxes

F.L.J. van der Linden   P.H. Vazques de Souza Silva
German Aerospace Center (DLR)
Institute of Robotics and Mechatronics, Oberpfaffenhofen, Germany

Abstract

Two elastic gearbox models with friction losses are presented; a 1-Degree Of Freedom (DOF) and a 3-DOF model.

The presented models have advantages over the existing lossy gear model [7] of the Modelica Standard Library when gear vibrations are of interest. Care has been taken that also in the case of gear locking, the elasticity effects are treated adequately.

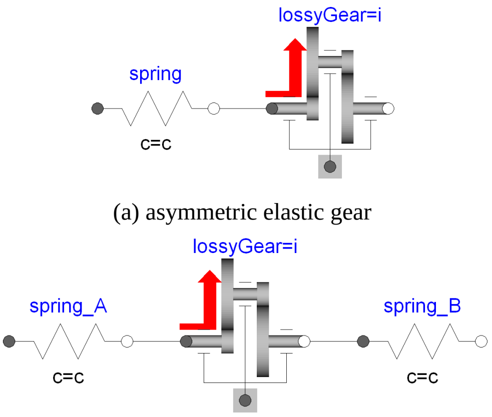
In addition to external excitations, it is now also possible to model internal excitations of the gearbox caused by the varying stiffness and/ or damping. This varying stiffness can be specified by the user for each gear wheel. This feature can for instance be used to model tooth interaction or broken gears.

Furthermore, the 3-DOF elastic model can simulate the elasticity of the support bearings in the load direction, which is impossible in the standard lossy gear model.

Keywords: Elastic Gearbox, Efficiency, Gearbox

1 Introduction

Gearbox vibrations and losses can affect the performance of a mechanical system as a whole. For example in wind turbines, vibrations of gearboxes often cause undesired behaviour or even fatigue failures. Moreover, in other applications elastic effects of the gearbox can influence the performance of the system, especially in low weight - high gear ratio applications, such as lightweight robots and aircraft applications. In this article it is presented how the lossy gear model from the Modelica Standard Library `Modelica.Mechanics.Rotational` addressed by Pelchen et al. [7] is extended to a full elastic model, without losing the symmetry of the model. Two models have been developed; a 1-DOF model and a 3-DOF model.



(a) asymmetric elastic gear

(b) symmetric elastic gear

Figure 1: Lossy gear from the Modelica Standard Library, extended with springs to create an elastic gear model

2 Overview of Available Models

The lossy gear model can simulate the efficiency of the gearbox depending on load direction and speed as well as stick slip effects. Bearing friction effects are also included in the model. The standard lossy gear model is a rigid model. When elastic effects are needed for correct modelling, constructions as in Figure 1 can be made to simulate elasticity. Figure 1a yields in the case of a locked gear a non-symmetric model since the elasticity is lumped on one side. In many cases this leads to non-realistic model behaviour.

The model shown in Figure 1b has problems to be simulated at all. See Appendix A for an in-depth analysis of this problem.

Sing and Houser [8] have developed an elastic gear model that can simulate torsional as well as transverse vibrations. Mesh- and bearing losses, however, are not taken into account. Howard et al. [1] have been working with FEM models. These models are highly complex and the simulation times are high. Moreover the geometry and material properties have to be known, which is usually not the case.

In this report, the work of Pelchen et al. [7] on the standard lossy gear model is combined with the work of Sing and Houser [8]. The goal of this paper is to present a low order symmetric model that can simulate mesh- and bearing losses as well as an elastic contact. The parameters needed for the model can either be found in vendor catalogues or can be measured without spending much time and resources.

3 The Elastic Lossy Gear Model

Two elastic gear models are developed; a torsional elastic model (Figure 2a, 1-DOF) and a translational-torsional model (Figure 2b, 3-DOF). With the first model the effect of elastic teeth in a gearbox can be simulated. The second model adds elasticity of the bearings in the direction of the gear load.

A schematical overview of the power flow through the lossy gear model is shown in Figure 3. The symbols and their description are listed in Table 1.

The mesh is modelled using a spring and a damper on each side of the contact position y_{mesh}. The mesh forces F_{mA} and F_{mB} are related to the mesh torques τ_{mA} and τ_{mB} as:

$$F_{mA} = \frac{\tau_{mA}}{r_A} = \frac{\tau_{gA} - \tau_{lossA}}{r_A} \tag{1}$$

$$F_{mB} = -\frac{\tau_{mB}}{r_B} = -\frac{\tau_{gB} - \tau_{lossB}}{r_B} \tag{2}$$

And the resulting driving forces are (These are the forces left after all friction losses):

$$F_{gA} = \frac{\tau_{gA}}{r_A}, \qquad F_{gB} = -\frac{\tau_{gB}}{r_B} \tag{3}$$

The spring forces F_{mA} and F_{mB} in the load direction are for the 1-DOF model (see Figure 2a):

$$F_{mA} = k_{hA}\left(y_{mesh} - r_A\theta_A\right) + c_{hA}\left(v_{mesh} - r_A\dot{\theta}_A\right) \tag{4}$$

$$F_{mB} = k_{hB}\left(r_B\theta_B - y_{mesh}\right) + c_{hB}\left(r_B\dot{\theta}_B - v_{mesh}\right) \tag{5}$$

For the 3-DOF model (see Figure 2a) it results:

$$\begin{aligned} F_{mA} = k_{hA}\left(y_{mesh} - (r_A\theta_A + y_A)\right) + \\ c_{hA}\left(v_{mesh} - (r_A\dot{\theta}_A + \dot{y}_A)\right) \end{aligned} \tag{6}$$

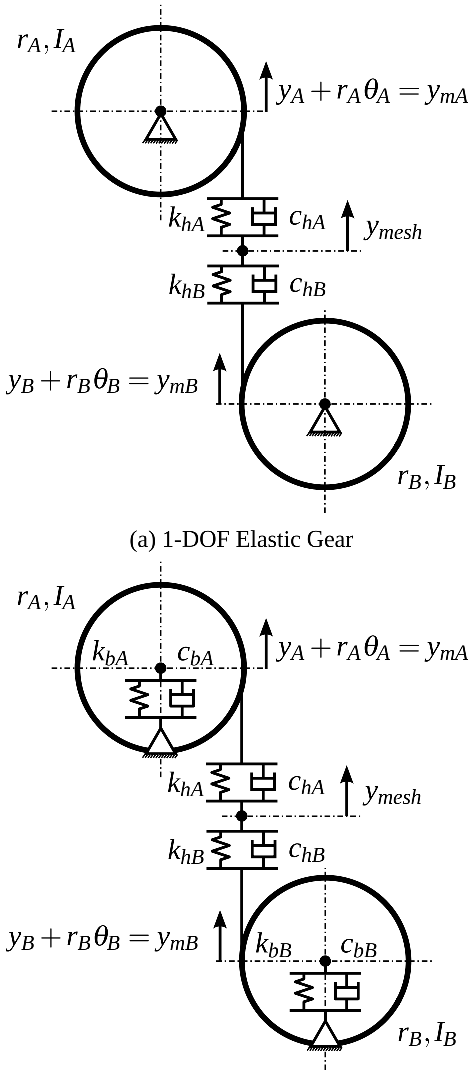
$$\begin{aligned} F_{mB} = k_{hB}\left((r_B\theta_B + y_B) - y_{mesh}\right) + \\ c_{hB}\left((r_B\dot{\theta}_B + \dot{y}_A) - v_{mesh}\right) \end{aligned} \tag{7}$$

Since in a gear mesh the gear moduli of the meshing teeth have to be equal, the following assumption is postulated:

Symbol	Description
$m_{\mathscr{I}}$	Mass wheel $\mathscr{I}$
$r_{\mathscr{I}}$	Radius of wheel $\mathscr{I}$
$I_{g,\mathscr{I}}$	Mass moment of inertia of wheel $\mathscr{I}$
$k_{h,\mathscr{I}}$	Gear contact spring constant wheel $\mathscr{I}$
k_h	Total gear contact spring constant
$k_{b,\mathscr{I}}$	Bearing stiffness wheel $\mathscr{I}$
$\Delta k_{h,\mathscr{I}}$	normalized stiffness profile
$k_{h,base}$	$k_h = k_{h,base}\,\Delta k_{h,\mathscr{I}}$
$c_{h,\mathscr{I}}$	Gear contact damping constant wheel $\mathscr{I}$
c_h	Total gear contact damping constant
$c_{b,\mathscr{I}}$	Bearing damping wheel $\mathscr{I}$
$\Delta c_{h,\mathscr{I}}$	normalized damping profile
$c_{h,base}$	$c_h = c_{h,base}\,\Delta c_{h,\mathscr{I}}$
$y_{\mathscr{I}}$	Displacement of wheel $\mathscr{I}$
$y_{m,\mathscr{I}}$	$y_{\mathscr{I}} + r_{\mathscr{I}}\theta_{\mathscr{I}}$
y_{mesh}	Displacement in load direction of the contact point
v_{mesh}	$\dot{y}_{mesh}$
$\theta_{\mathscr{I}}$	Angular position of wheel $\mathscr{I}$
$\tau_{\mathscr{I}}$	Input torque on shaft $\mathscr{I}$
$\tau_{m,\mathscr{I}}$	Mesh torque of shaft $\mathscr{I}$
$\tau_{g,\mathscr{I}}$	Resulting driving toque of shaft $\mathscr{I}$
$\tau_{loss,\mathscr{I}}$	Mesh loss torque on shaft $\mathscr{I}$
$\tau_{bf,\mathscr{I}}$	Bearing friction torque on shaft $\mathscr{I}$
$\tau_{loss,max,\mathscr{I}}$	Maximal loss torque of gear $\mathscr{I}$
$\tau_{loss,min,\mathscr{I}}$	Minimal loss torque of gear $\mathscr{I}$
$F_{m,\mathscr{I}}$	Mesh force of gear wheel $\mathscr{I}$
$F_{g,\mathscr{I}}$	Resulting driving force of gear $\mathscr{I}$
$F_{loss,\mathscr{I}}$	Mesh loss force on shaft $\mathscr{I}$: $\frac{\tau_{g,\mathscr{I}}}{r_{\mathscr{I}}}$
$F_{b,\mathscr{I}}$	Bearing force wheel $\mathscr{I}$
ΔF_h	Force difference over the mesh $\Delta F_g = F_{gB} - F_{gA}$
$F_{loss,max,\mathscr{I}}$	Maximal loss force of gear $\mathscr{I}$
$F_{loss,min,\mathscr{I}}$	Minimal loss firce of gear $\mathscr{I}$
P_{loss}	Power loss of the gear mesh
$P_{loss,\mathscr{I}}$	Power loss of the gear $\mathscr{I}$
$P_{m,\mathscr{I}}$	Power flow into mesh $\mathscr{I}$

$\mathscr{I}$ can be substituted for respectively A or B to indicate a certain gear wheel

Table 1: List of symbols.



(a) 1-DOF Elastic Gear

(b) 3-DOF Elastic Gear

Figure 2: The One Degree and the Three Degrees of Freedom Elastic Gear Models

Assumption 1 *The mesh stiffness and the mesh damping are equal on both gear wheels*

This leads to:

$$k_h = 2k_{hA} = 2k_{hb} \tag{8}$$

$$c_h = 2c_{hA} = 2c_{hb} \tag{9}$$

The bearing forces are given by:

$$F_{bA} = -\left(k_{bA}y_A + c_{bA}\dot{y}_A\right) \tag{10}$$

$$F_{bB} = -\left(k_{bB}y_B + c_{bB}\dot{y}_B\right) \tag{11}$$

In Figure 3 the torques and forces on both gear wheels are shown. Using the sign conventions from this figure, the rotational and translational equations of motion can be obtained.

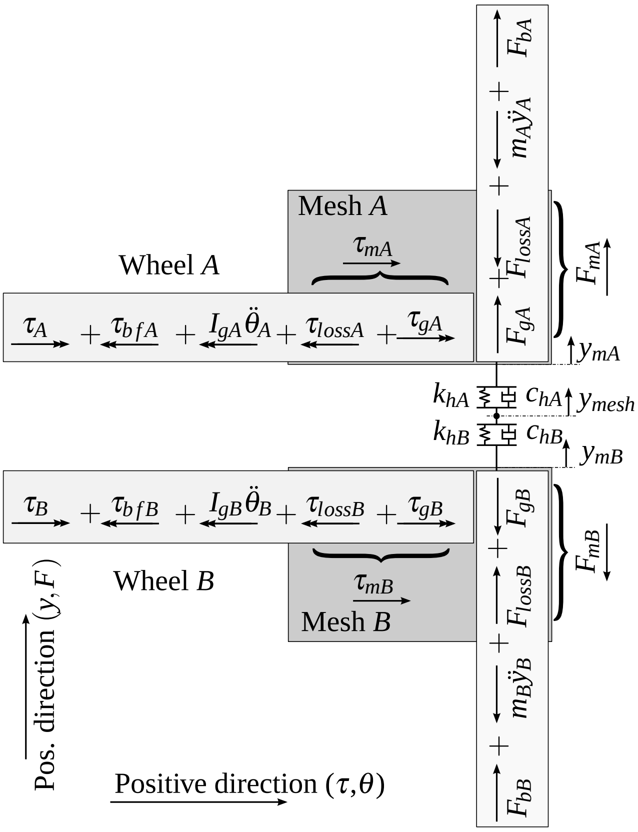


Figure 3: Forces and moments on the gearbox. The Torque/ Force convention is also shown.

$$\tau_A - \tau_{bfA} - I_{gA}\ddot{\theta}_A - \tau_{lossA} + \tau_{gA} = 0 \tag{12}$$

$$\tau_B - \tau_{bfB} - I_{gB}\ddot{\theta}_B - \tau_{lossB} + \tau_{gB} = 0 \tag{13}$$

$$F_{bA} - m_A\ddot{y}_A - F_{lossA} + F_{gA} = 0 \tag{14}$$

$$F_{bB} - m_B\ddot{y}_B + F_{lossB} - F_{gB} = 0 \tag{15}$$

Equations 14 and 15 reduce for the 1-DOF model to:

$$F_{lossA} - F_{gA} = 0 \tag{16}$$

$$-F_{lossB} + F_{gB} = 0 \tag{17}$$

For a moving gear (not stuck) this coupling equation between hull A and B is defined by the resultant drive forces:

$$F_{gA} = F_{gB} \tag{18}$$

In stuck mode though, hull A and B are uncoupled. Since the gear is stuck, the constraint equation is:

$$v_{mesh} = 0 \tag{19}$$

3.1 Gear Mesh Losses

For the gear mesh efficiency it is important to identify the power in- and outflows of the gear mesh. These

	$F_{hY} > 0$	$F_{hY} < 0$
$v_{mesh} > 0$	Quadrant 1	Quadrant 2
$v_{mesh} < 0$	Quadrant 4	Quadrant 3

Table 2: Gear operational modes and quadrants. The power flow in quadrant 1 and 3 is from Gearwheel *A* to Gearwheel *B*, in quadrant 2 and 4 from Gearwheel *B* to Gearwheel *A*

powers can be obtained by rearranging Equation 12 and 13 and multiplying them by the rotational velocity. The power that flows into the gear hull is therefore:

$$\begin{aligned} P_{m,A} &= \left(\tau_A - \tau_{bfA} - I_{gA}\ddot{\theta}_A\right)\dot{\theta}_A \\ &= \left(\tau_{lossA} - \tau_{gA}\right)\dot{\theta}_A \end{aligned} \tag{20}$$

$$\begin{aligned} P_{m,B} &= \left(\tau_B - \tau_{bfB} - I_{gA}\ddot{\theta}_B\right)\dot{\theta}_B \\ &= \left(\tau_{lossB} - \tau_{gB}\right)\dot{\theta}_B \end{aligned} \tag{21}$$

The total mesh loss is:

$$P_{loss} = P_{lossA} + P_{lossB} \tag{22}$$

To distribute the losses over both gear wheels the following assumption is made:

Assumption 2 *The mesh power losses are equally distributed over both gear wheels.*

This assumption leads to:

$$\tau_{lossA}\omega_A = \frac{P_{loss}}{2} = \tau_{lossB}\omega_B \tag{23}$$

Using the sign conventions from Figure 3 this leads to the conclusion that a power flow into the gearbox is positive. Therefore the power loss is defined as:

$$P_{loss} = P_{mA} + P_{mB} \tag{24}$$

Gearbox efficiency η depends on the power flow through the gear[1]. Using operational quadrants (see also Table 2), the efficiency in each quadrant is defined by following definitions:

Definition 1 *The efficiency of the gearbox in quadrant 1 and 3 is:*

$$\eta = -\frac{P_{mB}}{P_{mA}} = \eta_1 \tag{25}$$

Definition 2 *The efficiency of the gearbox in quadrant 2 and 4 is:*

$$\eta = -\frac{P_{mA}}{P_{mB}} = \eta_2 \tag{26}$$

Combining Assumption 2 with Definitions 1 and 2, the friction moment on the axis for quadrant 1 and 3 is for $\dot{\theta} \neq 0$:

$$\tau_{lossA} = -\frac{1-\eta_1}{1+\eta_1}\tau_{gA} \tag{27}$$

$$\tau_{lossB} = -\frac{1-\frac{1}{\eta_1}}{1+\frac{1}{\eta_1}}\tau_{gB} \tag{28}$$

For quadrant 2 and 4 the power loss and friction moment on the axis is:

$$\tau_{lossA} = -\frac{1-\frac{1}{\eta_2}}{1+\frac{1}{\eta_2}}\tau_{gA} \tag{29}$$

$$\tau_{lossB} = -\frac{1-\eta_2}{1+\eta_2}\tau_{gB} \tag{30}$$

For $v_{mesh} = 0$ (gear mesh can get stuck), the loss moment working on the gear wheel is set to zero, since the position where the loss is generated is fixed. This leads to:

$$\tau_{lossA} = 0, \qquad \tau_{lossB} = 0 \tag{31}$$

3.2 State Switching

In order to define in which quadrant the gearbox is operating or if the gearbox is stuck, a state machine is developed. It switches based on the mesh velocity v_{mesh}, the sum of the total mesh loss forces and the force difference over the mesh.

The total mesh loss force depends on the operational quadrant. Since for $v_{mesh} = 0$, it is unknown if the gearbox is operating in quadrant 1 or 3 (motor mode) or in quadrant 2 or 4 (generator mode). As the mesh forces (F_{gA} and F_{gB}) are known it is possible to develop a maximum and minimum loss torque for each hull. The quadrants 1 and 2 lead to $\tau_{loss,max}$ using quadrant 1 for $F_h > 0$ and quadrant 2 for $F_h < 0$. The quadrants 3 and 4 $\tau_{loss,min}$ using quadrant 4 for $F_h > 0$ and quadrant 3 for $F_h < 0$. This leads to the total loss forces:

$$F_{loss,max} = \frac{\tau_{loss,maxA}}{r_{gA}} + \frac{\tau_{loss,maxB}}{r_{gB}} \tag{32}$$

$$F_{loss,min} = \frac{\tau_{loss,minA}}{r_{gA}} + \frac{\tau_{loss,minB}}{r_{gB}} \tag{33}$$

The force difference over the mesh is:

$$\Delta F_g = -\left(\frac{\tau_{gA}}{r_{gA}} + \frac{\tau_{gB}}{r_{gB}}\right) = F_{gB} - F_{gA} \tag{34}$$

Figure 4 shows how the mode switching takes place.

[1] A good example is a worm wheel drive. In such a drive the efficiency from worm to gearwheel is usually > 0.5. However, the efficiency from gearwheel to worm is in some cases zero.

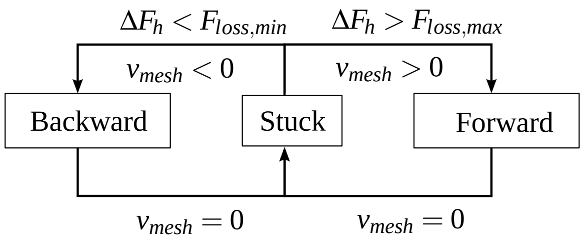


Figure 4: Mode switching of the elastic gearbox

3.3 Mesh Stiffness- and Damping Variations

Kar and Mohanty [3], Li et al. [4] as well as Kahraman and Singh [2] report that internal gearbox vibrations are caused by the variation of the mesh stiffness between two adjacent teeth in contact. Nevzat et al. [5] report that also damping is an important factor in gear vibrations. Moreover Li et al. [4] note that gear root cracks lead to a lower local stiffness.

To model these variations, a normalized gear stiffnes- and damping profile ($\Delta k_h, \Delta c_h$) is introduced for each gearwheel that specifies the local stiffness and damping over the circumference of each gear wheel. Using this profile the stiffness and damping of the contact point is calculated using:

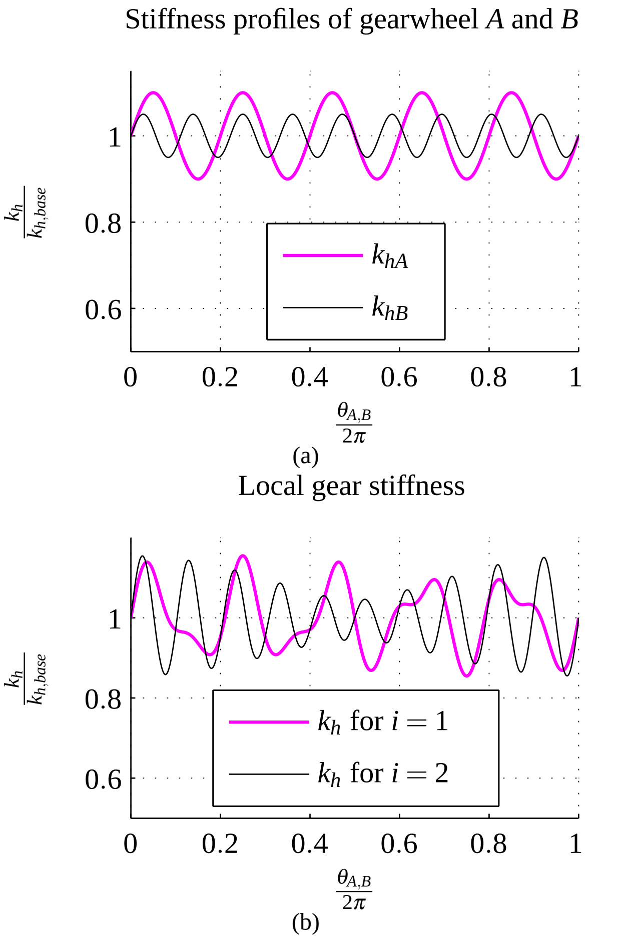
$$k_h = k_{h,base} \cdot \Delta k_{hA}(\theta_A) \cdot \Delta k_{hB}(\theta_B) \tag{35}$$

$$c_h = c_{h,base} \cdot \Delta c_{hA}(\theta_A) \cdot \Delta c_{hB}(\theta_B) \tag{36}$$

In Figure 5 an example of the two normalized stiffness profiles and the total local stiffness is given for two gear ratios $i = 1$ and $i = 2$.

3.4 Bearing Losses

In the elastic lossy gear model it is possible to have a stuck mesh and at the same time moving gearwheels. Therefore the mesh- and bearing losses cannot be lumped like Pelchen et al. [7] do. Instead two bearings are modelled, one on each gear wheel side. Since usually the bearings from gearboxes are not identical, each bearing can have individual friction characteristics. The same approach as Otter et al. [6] is used to model the bearing friction (this is in fact the bearing friction model from the Modelica Standard Library).


Stiffness profiles of gearwheel A and B

(a)

Local gear stiffness

(b)

Figure 5: Local stiffness of two contacting gear wheels for gear ratios $i = 1$ and $i = 2$

4 Modelica Model

The elastic lossy gear model as developed in Section 3 can be implemented straightforward into a Modelica model. The parameters for the simulation of a 1-DOF model are the gear wheel radii r_A and r_B, the moments of inertia of the gearwheels I_{gA} and I_{gB} and the nominal stiffness k_h and damping c_h of the mesh. Tabulated values of η_{n1}, η_{n2} as a function of v_{mesh}, τ_{bfA} and τ_{bfB} have to be given as a function of $\dot{\theta}_A$ respectively $\dot{\theta}_B$. The last inputs are the profile tables Δk_{hA}, Δk_{hB}, Δc_{hA} and Δc_{hB} which are a function of the normalized circumference of the gear wheel.

5 Simulation Results

To check if the simulation results of the elastic lossy gear model correspond with the simulation results of the standard lossy gear model (extended with a dummy mass and two spring and damper elements) a simulation is executed with both models. A schematic

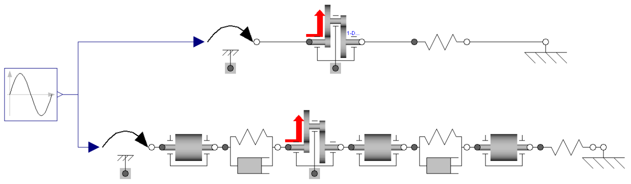


Figure 6: Elastic 1-DOF lossy gear model and standard lossy gear model extended with dummy mass and two spring- damper combinations.

overview of the models can be found in Figure 6. Both gears are driven by a sinusoidal torque (left). The loading of the gearbox takes place by a spring (right) with very low stiffness (1N/m). Note that this dummy mass is a work-around to avoid simulation problems. It introduces higher dynamics and moreover increases the simulation order. For a smooth simulation, the standard “Dassl” integrator requires the dummy mass to be maximal 10^6 times smaller than the main masses. Implementing a 3-DOF system using the Modelica Standard Library is hardly possible and therefore not worked out.

5.1 Gearbox Sticking

The simulation results of the gearbox getting stuck, are shown in Figure 7 [2]. This figure demonstrates that the simulation results are almost identical for both torsional models, leading to the conclusion that the 1-DOF elastic lossy gear model delivers the right results. The simulation results also show that the eigenfreuquency of the 3-DOF model is lower than the 1-DOF model. This can be explained by the lower stiffness and damping of the 3-DOF model than the 1-DOF model, caused by the extra spring-damper combination at the bearings. Note that the bearing stiffness is set to a low value to make the differences extra clear.

In addition the simulation results show clearly that in the case of a blocked gearbox, both sides of the gearbox are uncoupled; the eigenfrequency of wheel *A* is higher than of wheel *B*. This seems not logical at first sight, since the inertia of wheel *A* is higher than of wheel *B*. Yet the stiffness of a gearbox has a quadratic relation with the gear ratio, leading to a 9 times higher stiffness of wheel *A* with respect to wheel *B*. Since the inertia of wheel *A* is only 4 times higher than of wheel *B*, the eigenfrequency of wheel *A* is $\sqrt{\frac{k}{m}} = \sqrt{\frac{9}{4}} = 1.5$ times higher than wheel *B*.

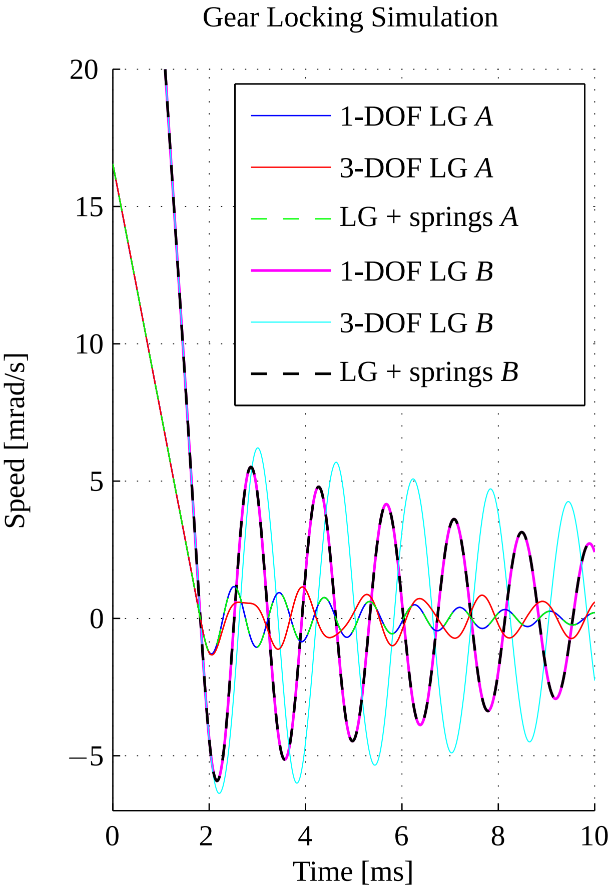


Figure 7: Simulation of the elastic 1-DOF and 3-DOF lossy gear model together with the standard lossy gear model extended with dummy mass and springs.

5.2 Internal Gearbox Vibrations

As concluded in Section 3.3 the stiffness variation of the gear mesh between the gear teeth is an important source of gearbox vibrations. To simulate this behaviour a gearbox is modelled using realistic parameters for radii, stiffness and damping. The mesh stiffness variation is modelled by using the stiffness profile from Section 3.3. To simulate two gear wheels with 60 teeth, both gearwheels are given a stiffness variation profile (Δk_{hA} and Δk_{hB}) with a sinusoidal variation with 60 periods and an amplitude of 2.5% of $k_{h,base}$. The average of the profile is one. Combining both gear wheels lead to a 10% fluctuation of the gear stiffness varying 60 times each rotation.

The simulation setup is shown in Figure 8. In this figure the gear is driven by a constant speed block (left) and loaded by a constant load block (right). The elas-

[2] The simulation parameters are: $k_h = 1e6Nm^{-1}$, $c_h = 10Nsm^{-1}$, $\eta_1 = \eta 2 = 0.5$, $I_{gA} = 4e-3kgm^2$, $I_{gB} = 1e-3kgm^2$, $r_{gA} = 0.3m$ and $r_{gB} = 0.1m$.

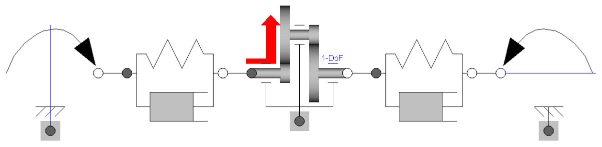
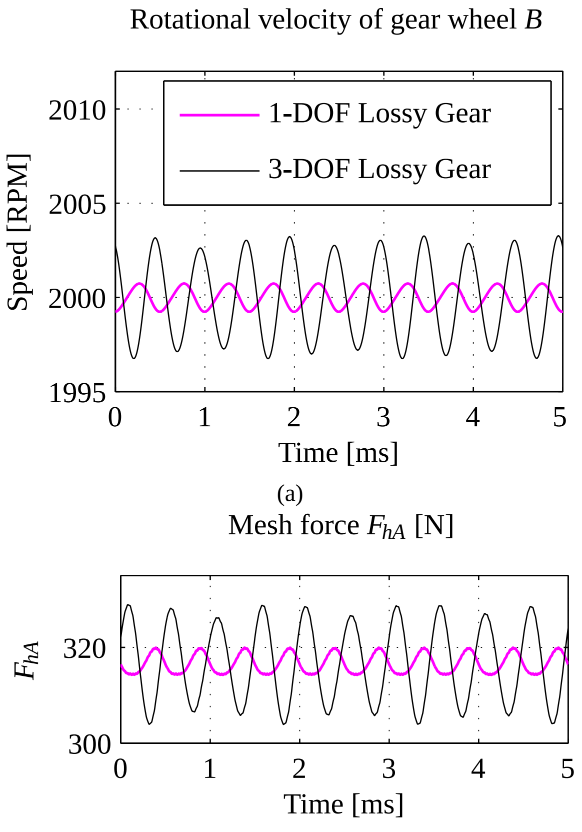


Figure 8: Simulation environment for the internal gear vibration test.



(a)

(b)

Figure 9: Simulation of the Elastic lossy gear model with a varying tooth stiffness at 2000 RPM:

tic gearbox is coupled using two relatively stiff couplings of 20kNM/rad to a constant speed and a constant torque block. The simulation results for the gear running stationary at 2000 RPM are illustrated in Figure 9 [3]. The extra elasticity of the bearings lowers the eigenfrequency of the vibrations (2000 RPM is close to the eigenfrequency of the 3-DOF lossy gear model). Just like in Section 5.1, the bearing stiffness and damping is chosen relatively low to show the effect of bearing stiffness.

6 Conclusion

The lossy gear model of the Modelica Standard Library is extended with two models; a 1-DOF model, simulating tooth stiffness and a 3-DOF model, simulating tooth and bearing stiffness. Elasticity of the gearbox is dealt with in an appropriate way without the need for dummy masses. Just like the standard lossy gear model, chattering is avoided in this model by the state switching algorithm.

With the 1-DOF and 3-DOF elastic lossy gear models it is now possible to model the torsional as well as the translational (in load direction) vibrations of gearboxes. In addition it is possible to simulate the change of stiffness and/ or damping between the two adjacent teeth of a gear. This facilitates the modelling of vibrations that are internally generated. Furthermore, the 3-DOF model can simulate the effect of elastic bearings. The extra elasticity caused by the bearings will decrease the lowest eigenfrequency, which can cause huge problems in high velocity gear applications. The possibility to easily simulate these problems makes it possible to identify problems in an early design stage.

[3] $k_h = 1e8Nm^{-1}$, $c_h = 50Nsm^{-1}$, $\eta_a = \eta_2 = 0.9$,$I_{gA} = 9e-5kgm^2$, $I_{gB} = 9e-5kgm^2$, $r_{gA} = 30mm$ and $r_{gB} = 30mm$.
The extra parameters for the 3-DOF model are:
$m_A = 0.3kg$, $m_A = 0.5kg$, $k_A = k_B = 1e8Nm^{-1}$, $c_A = c_B = 5Nsm^{-1}$

Appendices

A Lossy Gear Simulation Problems

The lossy gear model from the Modelica Standard Library can not simulate when it is directly coupled with two springs. In this appendix an example will be used to demonstrate where the simulation problems originate.

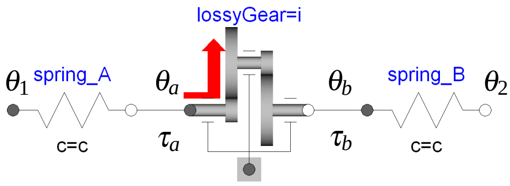


Figure 10: Lossy gear model extended with two springs

The equations of motion for the model in Figure 10 using gear ratio $i = 1$ are shown in equation 37 to 40:

$$\theta_a = i\theta_b = \theta_b \tag{37}$$
$$\tau_a = c\,(\theta_a - \theta_1) \tag{38}$$
$$\tau_b = c\,(\theta_2 - \theta_b) \tag{39}$$

$$\tau_{loss} = \begin{cases} \text{stuck} & : -(\tau_a + \tau_b) \text{ so that } \ddot{\theta}_a = 0 \\ \text{sliding} & : k(\dot{\theta}_a)\ \dot{\theta}_a \end{cases} \tag{40}$$

In Equation 40, $k(\dot{\theta}_a)$ is a variable defining the efficiency of the lossy gear model (which can be dependant on $\dot{\theta}_a$).

Combining equations 37 to 40 yield the following differential equations for the stuck mode as well as for the sliding mode:

$$\text{stuck:}\quad c\,(\theta_a - \theta_1) + c(\theta_2 - \theta_a) + k\left(\dot{\theta}_a\right) = 0 \tag{41}$$

$$\text{sliding:}\quad \begin{cases} \tau_{loss} & = -(c(\theta_a - \theta_1) + c(\theta_2 - \theta_a)) \\ \ddot{\theta}_a & = 0 \end{cases} \tag{42}$$

Comparing Equation 41 and 42 shows that the equation in stuck mode (Eq 41) is a differential equation of first order in θ_a (θ_1 and θ_2 are input signals to this equation). On the contrary, the differential equation for sliding (Eq 42) is of second order in θ_a.

Summing up, this leads to a changing number of differential equations while switching between stuck and sliding mode. Dymola (and also other Modelica tools) cannot handle cases in which the number of differential equations changes during simulation. Therefore the model as shown in Figure 10 cannot simulate any switching between stuck and sliding.

A method to fix this problem would be to replace $\ddot{\theta}_a = 0$ from Equation 42 by $\dot{\theta}_a = 0$, yielding a model that does not change states. However this is non-trivial because all switching conditions (how to switch between sliding and stuck mode) would change.

References

[1] I. Howard, S. Jia, and J. Wang. The dynamic modelling of a spur gear in mesh including friction and a crack. *Mechanical Systems and Signal Processing*, 15:831–853, 2001.

[2] A. Kahraman and R. Singh. Interactions between time-varying mesh stiffness and clearance nonlinearities in a geared system. *Journal of Sound and Vibration*, 142:49–75, 1990.

[3] C. Kar and A. Mohanty. Determination of time-varying contact length, friction force, torque and forces at the bearings in a helical gear system. *Journal of Sound and Vibration*, 309:307–319, 2008.

[4] C. J. Li, H. Lee, and S. H. Choi. Estimating size of gear tooth root crack using embedded modelling. *Mechanical Systems and Signal Processing*, 16:841–852, 2002.

[5] H. Nevzat, Özgüven, and D. Houser. Dynamic analysis of high speed gear by using loaded static transmission error. *Journal of Sound and Vibration*, 125:71–83, 1988.

[6] M. Otter, H. Elmqvist, and S. E. Mattsson. Hybrid modeling in modelica based on the synchronous data flow principle. In *CACSD*, Hawaii, USA, August 1999.

[7] C. Pelchen, C. Schweiger, and M. Otter. Modeling and simulating the efficiency of gearboxes and of planetary gearboxes. In *2nd International Modelica Conference*, pages 257–266, 2002.

[8] R. Sing and D. R. Houser. *Non-Linear Dynamic Analysis of Geared Systems*. PhD thesis, Ohio State University, February 1990.

Performance Analysis of VON MISES' Motor Calculus within Modelica

Tobias Zaiczek Olaf Enge-Rosenblatt
Fraunhofer Institute for Integrated Circuits
Design Automation Division
Dresden, Germany
{Tobias.Zaiczek,Olaf.Enge}@eas.iis.fraunhofer.de

Abstract

This paper presents an alternative concept of modelling multibody systems within Modelica, the so-called motor calculus. This approach was introduced by R. VON MISES in 1924 and can be used to describe the dynamical behaviour of spatial multibody systems in a very efficient way. While the equations clearly take a very simple form in terms of motor algebra, the numerical efficiency is still an open question.

In the paper, first some fundamentals of motor calculus are summarized. An experimental implementation of motor algebra is used to measure and analyse the numerical efficiency and performance regarding the simulation time of VON MISES' approach. Therefore, some components of the Modelica Multibody Standard Library were modified in order to compare both implementations. Finally, some examples are given to prove the applicability and correctness of the concept but also to serve as a basis for a discussion of the numerical performance. The chosen approach utilizes all object-oriented features provided by the modelling language. Besides, it gives reason for the present endeavours to introduce the possibility of operator overloading within Modelica.

Keywords: motor calculus, screw theory, rigid multibody system, Modelica, performance

1 Introduction

The motion of mechanical systems in three-dimensional space has been examined for hundred of years. In 1924 R. VON MISES suggested an approach, the so-called motor calculus, to describe rigid body motion in 3D mechanics in a very clear and efficient way [6, 7]. Inspired by previous contributions (e.g. [2, 3, 12]), he introduced the motor as a six-tuple of scalar quantities and developed a special algebra for these mathematical objects, called the motor calculus. Though his approach is not well known throughout all branches of mechanical engineering, in the field of robotics VON MISES' ideas were rediscovered during the last decades [1, 5, 10, 11, 13], since they seem to be well suited to investigate the behaviour of spatial multibody systems. However, in the context of the modelling language Modelica (see e.g. [4, 14]), the motor calculus has not been taken into account up to now.

Meanwhile, many researchers apply the Modelica Multibody Standard Library ([9]) to model different kinds of – partially very complex – multibody systems (see proceedings of the Modelica conferences [8]). Hence, this library has proven to be a well suited resource to modelling such systems. Nevertheless, applying the motor calculus, the equations of motion for a rigid body become more concise and clearer, e.g.

$$\dot{\mathfrak{p}} = \mathfrak{f}$$

($\mathfrak{p}$ – momentum motor, $\mathfrak{f}$ – force motor). Despite the formal equivalence to Newton's Second Law for a point mass, this equation fully describes the three-dimensional mechanics of a rigid body.

In our first publication [15], we were already able to show the possible simplifications of the resulting equations within some components of the Modelica Multibody Standard Library using the motor calculus. Furthermore, we compared both approaches e. g. with respect to numerical correctness. So, the motivation to follow further up the motor calculus in the Modelica context is now to investigate the performance of the simulation of mechanical systems with regard to simulation time.

An extended test realization within the Modelica Multibody Standard Library has been carried out by changing some components of this library. These modifications take advantage of the built-in feature of inheritance. Hence, it is possible to compare both approaches e. g. with respect to numerical effectiveness.

In the following section, some fundamentals of motor calculus are shortly sketched. Some of the most important mathematical operations are defined. The test implementation is presented in section 3. The performance of the motor calculus approach will be evaluated and compared to the performance of the Modelica Multibody Standard Library using some examples in section 4.

2 Fundamentals of motor calculus

A *motor*

$$\mathfrak{h} = \begin{pmatrix} \boldsymbol{g} \\ \boldsymbol{h}_o \end{pmatrix}$$

is an ordered pair of vectors, $\boldsymbol{h}_o$ and $\boldsymbol{g}$, that define a vector field

$$\boldsymbol{h}(\boldsymbol{r}) = \boldsymbol{h}_o + \boldsymbol{g} \times \boldsymbol{r} \qquad (1)$$

in the three-dimensional Euclidean space. In this definition, $\boldsymbol{r}$ is the position vector of any point in space, while the vectors $\boldsymbol{h}$ and $\boldsymbol{g}$ are called the *moment* and the *resultant vector* of the motor, respectively. Accordingly, $\boldsymbol{h}_o$ stands for the *moment* of the motor at the origin O of the reference coordinate system.

For every motor, an infinite number of points exists, for which the moment of the motor $\boldsymbol{h}$ is parallel to the resultant vector $\boldsymbol{g}$. All these points exhibit the same moment $\boldsymbol{h}_n$ and lie on a straight line $\mathcal{N}$ given by

$$\boldsymbol{r}_n(\lambda) = \frac{\boldsymbol{g} \times \boldsymbol{h}_o}{|\boldsymbol{g}|^2} + \lambda \boldsymbol{g} , \qquad \lambda \in \mathbb{R}.$$

Geometrical interpretation. A very strong goal of the motor calculus is the fact that motors and all operations with motors (that will be defined later on) can be interpreted as geometrical objects or constructions. Hence, all motors can be seen as abstract objects that do not depend on the choice of a reference frame. Details can be found in [7, 15].

2.1 Motor calculus

In the following, some computational rules of the motor calculus are recalled.

Let $\mathfrak{h}$, $\mathfrak{h}_1$, and $\mathfrak{h}_2$ be three motors given by

$$\mathfrak{h} = \begin{pmatrix} \boldsymbol{g} \\ \boldsymbol{h}_o \end{pmatrix}, \quad \mathfrak{h}_1 = \begin{pmatrix} \boldsymbol{g}_1 \\ \boldsymbol{h}_{o1} \end{pmatrix}, \quad \mathfrak{h}_2 = \begin{pmatrix} \boldsymbol{g}_2 \\ \boldsymbol{h}_{o2} \end{pmatrix}.$$

Then, according to VON MISES, the following mathematical operations are defined:

$$\mathfrak{h}_1 + \mathfrak{h}_2 = \begin{pmatrix} \boldsymbol{g}_1 + \boldsymbol{g}_2 \\ \boldsymbol{h}_{o1} + \boldsymbol{h}_{o2} \end{pmatrix} \qquad \text{(addition)}$$

$$\alpha\mathfrak{h} = \begin{pmatrix} \alpha\boldsymbol{g} \\ \alpha\boldsymbol{h}_o \end{pmatrix} \qquad \text{(multiplication with a scalar } \alpha \in \mathbb{R}\text{)}$$

$$(\mathfrak{h}_1, \mathfrak{h}_2) = (\boldsymbol{g}_1, \boldsymbol{h}_{o2}) + (\boldsymbol{g}_2, \boldsymbol{h}_{o1}) \qquad \text{(inner product)}$$

$$\mathfrak{h}_1 \times \mathfrak{h}_2 = \begin{pmatrix} \boldsymbol{g}_1 \times \boldsymbol{g}_2 \\ \boldsymbol{g}_1 \times \boldsymbol{h}_{o2} + \boldsymbol{h}_{o1} \times \boldsymbol{g}_2 \end{pmatrix} \qquad \text{(outer product)}$$

In analogy to the vector calculus, VON MISES declared dyads for the motor calculus by linear vector functions mapping motors to motors. Referred to a concrete coordinate system, such a dyad can be represented as a (6 × 6) matrix.

The mapping can be described in the following manner:

$$\begin{aligned} \mathfrak{T} \circ \mathfrak{h}_1 &= \begin{pmatrix} \boldsymbol{T}_{11} & \boldsymbol{T}_{12} \\ \boldsymbol{T}_{21} & \boldsymbol{T}_{22} \end{pmatrix} \circ \begin{pmatrix} \boldsymbol{g}_1 \\ \boldsymbol{h}_{o1} \end{pmatrix} \\ &= \begin{pmatrix} \boldsymbol{T}_{11}\boldsymbol{h}_{o1} + \boldsymbol{T}_{12}\boldsymbol{g}_1 \\ \boldsymbol{T}_{21}\boldsymbol{h}_{o1} + \boldsymbol{T}_{22}\boldsymbol{g}_1 \end{pmatrix}. \end{aligned} \qquad (2)$$

Now, all calculation rules for the motor calculus can be derived readily. For details we refer to [7, 15]. In [15], it is also shown that, due to the definition of addition and scalar multiplication, motors span a vector space over the field of real numbers. Additionally, by the introduction of the outer product, motors form a *Lie-Algebra*[1].

2.1.1 Differentiation with respect to real-valued parameters

Consider a motor $\mathfrak{h}$ that depends on a real parameter t (e. g. the time) with differentiable components $\boldsymbol{g}$ and $\boldsymbol{h}_o$ with respect to t. Then, the first derivative of this motor with respect to t can be computed componentwise:

$$\frac{d\mathfrak{h}}{dt} = \begin{pmatrix} \frac{d\boldsymbol{g}}{dt} \\ \frac{d\boldsymbol{h}_o}{dt} \end{pmatrix}.$$

2.1.2 Differentiation in moving frames

If frame $\mathcal{F}_1$ moves relatively to a reference frame $\mathcal{F}_0$, the observed temporal change of a motor is then in general different in the two frames. The relative motion of the origin of frame $\mathcal{F}_1$ measured in frame $\mathcal{F}_0$

[1] Named after the mathematician SOPHUS LIE (*1842, †1899).

shall be given by the velocity vector $\underline{\boldsymbol{v}}_o$, while the angular velocity vector of frame $\mathcal{F}_1$ with respect to frame $\mathcal{F}_0$ is denoted by $\underline{\boldsymbol{\omega}}$. Then, the equation

$$\dot{\mathfrak{h}} = \overset{\circ}{\mathfrak{h}} + \begin{pmatrix} \underline{\boldsymbol{\omega}} \\ \underline{\boldsymbol{v}}_o \end{pmatrix} \times \mathfrak{h} \tag{3}$$

holds for the derivation with respect to time observed in frame $\mathcal{F}_0$. In Equ. (3), $\overset{\circ}{\mathfrak{h}}$ denotes the derivation w. r. t. time of the motor $\mathfrak{h}$ observed in frame $\mathcal{F}_1$.

2.2 Applications of motor calculus

The most important application of motor calculus is the description and analysis of the static and dynamic behaviour of rigid bodies subject to external forces and torques.

All forces and torques acting on a rigid body can be combined to one single force vector $\boldsymbol{f}$ and one torque vector $\boldsymbol{d}_o$. Similarly, the movement of a rigid body can be fully described by the movement of a special reference point O on the body (i. e. by its velocity vector $\underline{\boldsymbol{v}}_o$) and the angular velocity vector $\underline{\boldsymbol{\omega}}$, the body is turning with (see Fig. 1).

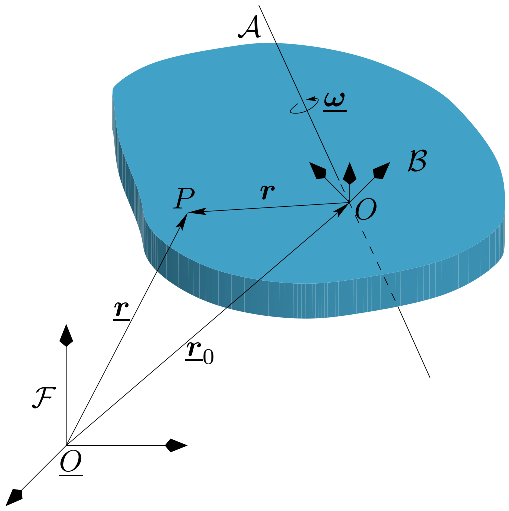

$\mathcal{A}$
$\underline{\boldsymbol{\omega}}$
$\mathcal{B}$
P
$\boldsymbol{r}$
O
$\underline{\boldsymbol{r}}$
$\underline{\boldsymbol{r}}_0$
$\mathcal{F}$
$\underline{O}$

Figure 1: Definition of vectors at the rigid body

The following paragraphs aim to show that, by introducing physically motivated motors, the motor calculus is well suited to describe rigid body movements.

2.2.1 Definition of physically motivated motors

Here, we introduce some motors that are able to describe the motion sequence of a rigid body as well as the acting torques and forces in a physically meaningful manner.

The first motor is called the *force motor* $\mathfrak{f}$ combining the resulting force $\boldsymbol{f}$ and torque $\boldsymbol{d}_o$ (referred to the reference point O) acting on the rigid body, i. e.

$$\mathfrak{f} = \begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{d}_o \end{pmatrix} .$$

Hence, the torque referred to any other point with the position vector $\boldsymbol{r}$ is calculated by

$$\boldsymbol{d}(\boldsymbol{r}) = \boldsymbol{d}_o + \boldsymbol{f} \times \boldsymbol{r} .$$

A second motor, the so-called *velocity motor*, is able to describe the whole motion of a rigid body. It consists of the velocity vector $\underline{\boldsymbol{v}}_o$ of the chosen reference point O and the angular velocity vector $\underline{\boldsymbol{\omega}}$ representing the rotation of the body w. r. t. an inertial frame:

$$\mathfrak{v} = \begin{pmatrix} \underline{\boldsymbol{\omega}} \\ \underline{\boldsymbol{v}}_o \end{pmatrix} .$$

This motor is able to describe the velocity $\underline{\boldsymbol{v}}$ of any point $\boldsymbol{r}$ of the rigid body by the equation

$$\underline{\boldsymbol{v}}(\boldsymbol{r}) = \underline{\boldsymbol{v}}_o + \underline{\boldsymbol{\omega}} \times \boldsymbol{r} .$$

Two other important vectors in the description of dynamic mechanical systems are the momentum vector $\boldsymbol{p}$ and the angular momentum vector $\boldsymbol{l}_o$. Both are combined in the *momentum motor* $\mathfrak{p}$ with

$$\mathfrak{p} = \begin{pmatrix} \boldsymbol{p} \\ \boldsymbol{l}_o \end{pmatrix} .$$

Similar to the force motor, the representation of momentum motor depends upon the chosen reference point. Between the angular momentum $\boldsymbol{l}_o$ referred to O and the angular momentum vector $\boldsymbol{l}(\boldsymbol{r})$ referred to any other point at position $\boldsymbol{r}$, the relationship

$$\boldsymbol{l}(\boldsymbol{r}) = \boldsymbol{l}_o + \boldsymbol{p} \times \boldsymbol{r}$$

holds. The proof of this statement can be found in [15].

2.2.2 Some fundamental laws of mechanics in terms of motor calculus

With the definitions above, a relationship between the velocity motor $\mathfrak{v}$ and the momentum motor $\mathfrak{p}$ can be derived by introducing the inertia dyad $\mathfrak{M}$ for the motor calculus:

$$\mathfrak{p} = \underbrace{\begin{pmatrix} m\boldsymbol{I} & -m\boldsymbol{R_s} \\ m\boldsymbol{R_s} & \boldsymbol{\Theta}_o \end{pmatrix}}_{\mathfrak{M}} \circ \mathfrak{v} . \tag{4}$$

The new symbol $\boldsymbol{R_s}$ describes the cross product dyad of the vector $\boldsymbol{r}_s$ pointing to the centre of mass.

With the help of the foregoing motor relations, the main mechanical laws can be rewritten in terms of motors.

The first law describes the change of momentum and angular momentum in the presence of external forces and torques in a very efficient and short way, namely

$$\boxed{\dot{\mathfrak{p}} = \mathfrak{f}}\,.$$

Here, $\dot{\mathfrak{p}}$ denotes the time derivative of the momentum motor $\mathfrak{p}$ observed in an *inertially fixed* reference frame.

A much more applicable form for concrete calculations can be derived using (3) to express the time derivation w. r. t. the body frame

$$\boxed{\overset{\circ}{\mathfrak{p}} + \mathfrak{v} \times \mathfrak{p} = \mathfrak{f}}\,, \tag{5}$$

where $\mathfrak{p}$, $\mathfrak{v}$, and $\mathfrak{f}$ are referred to the origin of the body frame.

Replacement of the momentum motor with the help of Equ. (4) yields the following relationship

$$\boxed{\mathfrak{M} \circ \overset{\circ}{\mathfrak{v}} + \mathfrak{v} \times (\mathfrak{M} \circ \mathfrak{v}) = \mathfrak{f}}$$

if all components are given in the body frame.

The kinetic energy of a rigid body can be expressed by means of motor calculus as follows:

$$\boxed{T = \frac{1}{2}\,(\mathfrak{v}, \mathfrak{p}) \qquad \text{with} \qquad \mathfrak{p} = \mathfrak{M} \circ \mathfrak{v}}\,.$$

Again, this expression agrees formally with the equation of the kinetic energy of a mass point, if therein the mass is substituted by the inertia dyad $\mathfrak{M}$ and the vectors are substituted by their corresponding motors.

Similarly, the equation for the power performed by the applied forces and torques is given by

$$\boxed{P = (\mathfrak{f}, \mathfrak{v})}\,.$$

2.2.3 Applications to multibody systems

The use of the motor calculus introduced above can also be very beneficial when describing multibody systems. These systems are often modelled as an interconnection structure of rigid bodies and ideal joints.

Exemplarily, two types of ideal joints, the revolute and the prismatic joint, will be analysed in this paper. Therefore, the necessary equations will be derived in this paragraph.

Both joints set up five constraint equations on the relative motion of the rigid bodies interconnected. By defining the unit vector $\boldsymbol{e}$ as the joint axis, one can write the velocity motor of the relative motion for the prismatic and the revolute joint as

$$\mathfrak{v}_r = \dot{x}\mathfrak{e}_P = \begin{pmatrix} \mathbf{0} \\ \dot{x}\boldsymbol{e} \end{pmatrix} \quad \text{and} \quad \mathfrak{v}_r = \dot{x}\mathfrak{e}_R = \begin{pmatrix} \dot{x}\boldsymbol{e} \\ \mathbf{0} \end{pmatrix}. \tag{6}$$

Here $\dot{x}$ denotes the translational velocity along or the rotational velocity around the joint axis $\boldsymbol{e}$. The cut forces and torques within the joint are merged in the force motor $\mathfrak{f}$. Since friction is neglected, the dissipated power of the joint vanishes and hence the applied power reads

$$P = (\mathfrak{f}, \mathfrak{v}_r)\,.$$

3 Object-oriented implementation

The test implementation presented here is based on the Modelica Multibody Standard Library. Due to some still existing limitations of the Modelica language in terms of operator overloading, compromises had to be made during implementation of the motor calculus.

3.1 Motor library

The first step of the implementation towards a description of rigid body motion by means of motor calculus is the realization of a general motor class. From the view of data structure, motors are nothing more than a combination of six scalars.

A clear structured class `motor` with two vectors, the resultant vector and the moment vector, would have been desirable. Due to the missing possiblity of operator overloading in our simulation tool (Dymola 7.1), an alternative implementation has been chosen. All six scalars are stored within one vector which is called `Motor`:

```
type Motor = Real[6]
    "Motor: [Resultant;Moment at r0]";
```

The reason for the chosen implementation was the ability to keep at least the operators "+" and "−" as well as the multiplication with scalars for the motor calculus in its original sense. One drawback is that, within the context of inheritance, no real specialization concerning the physical units of the quantities can be made. Hence, the child classes of velocity motor, force motor, and momentum motor have also a quite simple definition, namely:

```
type VelocityMotor= Motor "Velocity motor";
type ForceMotor    = Motor "Force motor";
type MomentumMotor= Motor "Momentum motor";
type DerMomMotor   = Motor "Time Derivative
                             of Momentum motor";
```

All the other calculation rules introduced in section 2.1 had to be implemented using Modelica functions.

The first function has been written to perform the inner product between two motors. It is denoted by `dot`:

```
function dot "Inner product of motor
              calculus"
  input Motor m1 "First motor";
  input Motor m2 "Second motor";
  output Real r3 "Resulting scalar";
algorithm
  r3 := m1[1:3]*m2[4:6] + m1[4:6]*m2[1:3];
end dot;
```

Similarly, the outer product has been implemented using the function `'x'`:

```
function 'x' "Outer product of motor
              calculus"
  input Motor m1 "First motor";
  input Motor m2 "Second motor";
  output Motor m3 "Resulting motor";
algorithm
  m3 := vector([ cross(m1[1:3],m2[1:3]);
                 cross(m1[1:3],m2[4:6])
                +cross(m1[4:6],m2[1:3])]);
end 'x';
```

The preceding reasons for the simple implementation of the motor class apply for the implementation of the motor dyads, too. Hence, a motor dyad given w. r. t. a given frame can be expressed as a (6×6) matrix:

```
type MotorDyad = Real[6,6] "Motor Dyad";
```

To apply a motor dyad to a motor, another function has been created. Referring to Equ. (2), the function has been defined by:

```
function times "Application of a Motor Dyad
                on a Motor"
  input MotorDyad m1
    "Motor dyad to be applied";
  input Motor      m2 "Input motor";
  output Motor     m3 "Output motor";
algorithm
  m3 :=  m1[:,1:3]*m2[4:6]
      + m1[:,4:6]*m2[1:3];
end times;
```

Finally, there exist two functions that enable to transform the components of a motor from one frame to another and vice versa.

```
function coordChange1 "Transforms motor
  from frame b to frame a"
  import F = Modelica.Mechanics.MultiBody.
    Frames;
  input Modelica.SIunits.Position[3] r_0
    "Vector pointing from origin of frame a
     to origin of frame 2, resolved in
     frame 1";
  input F.Orientation R "Orientation object
    of frame 2 resolved in frame 1";
  input Motor m1 "Motor res. in frame 2";
  output Motor m2 "Motor res. in frame 1";
algorithm
  m2 := vector([transpose(R.T)*m1[1:3];
        transpose(R.T)*m1[4:6]
        + cross(r_0,
                transpose(R.T)*m1[1:3])]);
end coordChange1;

function coordChange2 "Transforms motor
  from frame 1 to frame 2"
  import F = Modelica.Mechanics.MultiBody.
    Frames;
  input Modelica.SIunits.Position[3] r_0
    "Vector pointing from origin of frame a
     to origin of frame 2, resolved in
     frame 1";
  input F.Orientation R "Orientation object
    of frame 2 resolved in frame 1";
  input Motor m1 "Motor res. in frame 1";
  output Motor m2 "Motor res. in frame 2";
algorithm
  m2 := vector([R.T*m1[1:3];
                R.T*mom(m1,r_0)]);
end coordChange2;
```

3.2 Multibody implementation

The existing implementations of several parts of the Modelica Multibody Standard Library were adapted to the motor algebra. First of all, the connectors `frame_a` and `frame_b` were changed by substituting the vectors `force` and `torque` by the force motor `force` in the connector class `frame`. Hence, all other classes of the Multibody library used in this paper had to be adjusted as well. In the following, some important changes to the most relevant classes will be explained in detail.

3.2.1 Changes to the `body` class

The first changes were the replacements of important motion variables by some physically motivated motors. While the angular velocity vector as well as the velocity vector of `frame_a` were removed, the velocity motor and the momentum motor were introduced. Also, all acceleration vectors and all inertia dyades have been replaced by the time derivative of the momentum motor `dmom` and the motor inertia dyad, respectively.

```
// Motors
// ------
VelocityMotor velB   (start=[\dots])
  "Velocity motor wrt. frame a";
MomentumMotor mom    (start=[\dots])
  "Momentum motor wrt. frame a";
DerMomMotor dmom     (start=[\dots])
  "Time Derivative of Momentum motor wrt.
   frame a";
ForceMotor f_g
  "Force Motor due to gravitation";
// Motor Dyads
// -----------
final parameter MotorDyad I_mot =
  [diagonal({m, m, m}),-skew(m*r_CM);
   skew(m*r_CM), [I_11, I_21, I_31;
                  I_21, I_22, I_32;
                  I_31, I_32, I_33]
      + m*(diagonal(r_CM*r_CM*ones(3))
            -[r_CM]*transpose([r_CM]))]
  "Motorial Inertia Tensor";
```

Afterwards, all declared motors and motor dyads had to be defined using the following statements:

```
// Motors
// ------
velB   = vector([frame_a.R.w;
            frame_a.R.T*der(frame_a.r_0)]);
mom    = times(I_mot, velB);
dmom   = der(mom);
f_g    = vector([ m*frame_a.R.T*g_0;
          cross(r_CM, m*frame_a.R.T*g_0)]);
```

Finally, the equations of motion originally implemented according to

```
frame_a.f = m*(Frames.resolve2(frame_a.R,
                               a_0 - g_0)
           + cross(z_a, r_CM)
           + cross(w_a, cross(w_a, r_CM)));
frame_a.t = I*z_a + cross(w_a, I*w_a)
                 + cross(r_CM, frame_a.f);
```

have been replaced by Equ. (5):

```
frame_a.f = der(mom) + 'x'(velB,mom) - f_g;
```

Because of the object-oriented structure of the Modelica Standard Library, the changes had to be implemented only once. All subclasses of the Body class, like BodyShape, BodyBox, or BodyCylinder inherit the changes automatically.

3.2.2 Changes to the revolute class

Also, in the class revolute some changes had to be carried out. Firstly,

```
frame_a.f = -Frames.resolve1(R_rel,
                                frame_b.f);
frame_a.t = -Frames.resolve1(R_rel,
                                frame_b.t);
```

had to be replaced by

```
frame_a.f = -coordChange1(zeros(3),R_rel,
                                frame_b.f);
```

and

```
frame_b.f = -Frames.resolve1(R_rel,
                                frame_a.f);
frame_b.t = -Frames.resolve1(R_rel,
                                frame_a.t);
```

was substituted by

```
frame_b.f = -coordChange1(zeros(3),R_rel,
                                frame_a.f);
```

Last of all, the constraint equation was reformulated according to (6) as

```
  tau=-dot(frame_b.f,vector([e;zeros(3)]));
```

3.2.3 Changes to the prismatic class

In the class prismatic the following lines

```
zeros(3)  = frame_a.f + frame_b.f;
zeros(3)  = frame_a.t + frame_b.t
     + cross(e*(s_offset + s), frame_b.f);
// d'Alemberts principle
f = -e*frame_b.f;
```

were replaced by the two lines

```
zeros(6)  = frame_a.f
        + coordChange1(e*(s_offset + s),
        Frames.nullRotation(),frame_b.f);
// d'Alemberts principle
f = -dot(frame_b.f,vector([zeros(3);e]));
```

4 Examples and performance analysis

On the basis of the following examples, different performance tests were carried out in order to evaluate the numerical effectiveness of the two different modelling approaches. Two of the examples can also be found in [15] where the authors already showed the applicability and correctness of some of the implementations.

The following first three subsections describe the chosen examples and show some simulation results. The last subsection introduces a performance criterion and evaluates the performance of the simulations.

4.1 Movable double pendulum

As a first example, the movable double pendulum (Fig. 2) was chosen to compare the simulation time of the implemented body classes based on motor calculus to the implementation of the Modelica Standard

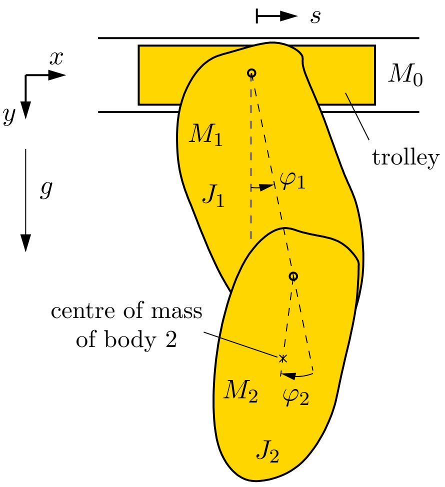


Figure 2: Sketch of double pendulum

Library. The pendulum consists of a trolley with the mass M_0 and two rigid bodies with masses M_1 and M_2. The trolley is able to move horizontally. The first body is suspended on the trolley by a revolute joint. The second body is suspended on the first body via a revolute joint, too. Both axes of rotation are parallel to the z-axis which lies perpendicular to the xy-plane (see Fig. 2). The moments of inertia of both bodies around the axis of rotation w. r. t. their particular centre of mass are given by J_1 and J_2. The distance between both axis of rotations is denoted by l_1.

The pendulum moves from an initial deflection of $\varphi_1(0) = 90\,\text{deg}$ and $\varphi_2(0) = 0\,\text{deg}$ due to the earth's gravity field. A viscous friction, acting in every joint, damps the motion of the pendulum.

As a reference, the same pendulum system has been implemented using the Modelica Standard Library. In Fig. 3, the trajectory for the position s of the trolley for both simulations is displayed. Both curves are nearly congruent. Hence, Fig. 3 shows only one curved line.

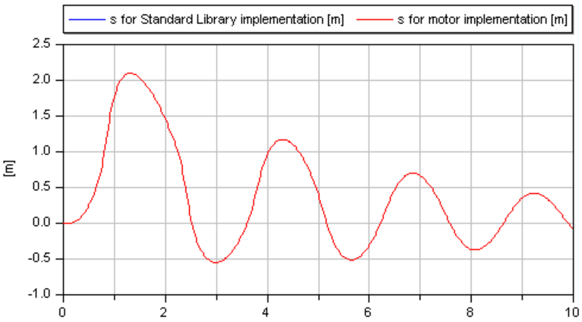


Figure 3: Trajectory of the trolley position s

The time histories of the revolute joint angles φ_1 and φ_2 are depicted in Fig. 4. The differences between both simulation results for an integration tolerance of 10^{-4} are shown in Fig. 5. Apparently, the deviation of the position stays smaller than $6 \cdot 10^{-12}$m for the given simulation time of 10s. The deviations of both pendulum angles are also very small. They do not exceed 10^{-11}rad. Hence, these differences can be interpreted as numerical errors of the simulator, since they depend on the integration tolerance.

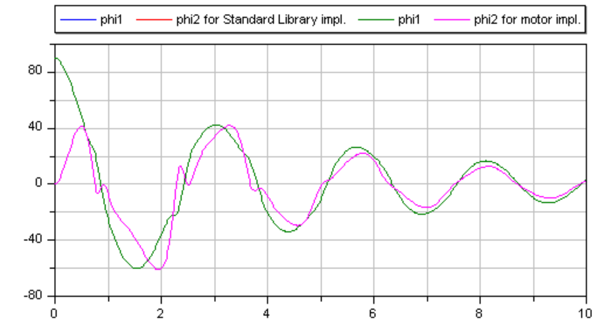


Figure 4: Trajectory of the pendulum angles φ_1 and φ_2

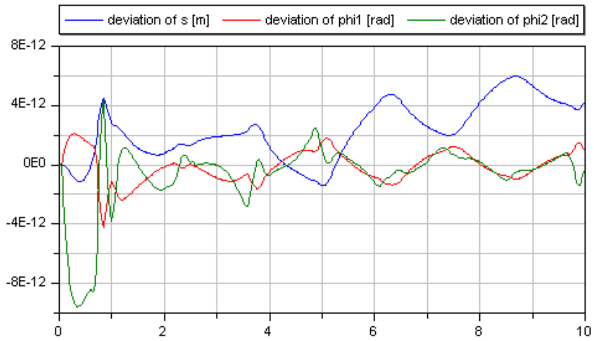


Figure 5: Deviations between both simulations for trolley position s and both angles φ_1 and φ_2

4.2 Fourfold pendulum on two movable sliders

The second example is a fourfold pendulum. It consists of two trolleys and a chain of four rigid bodies between them. Both trolleys are guided along straight tracks (see Fig. 6). Hence, this example contains a closed kinematic loop. Similar to the foregoing example, the pendulum moves due to the gravity field of the earth. The motion starts with an initial deflection (see Fig. 7) and is damped by a viscous friction in ev-

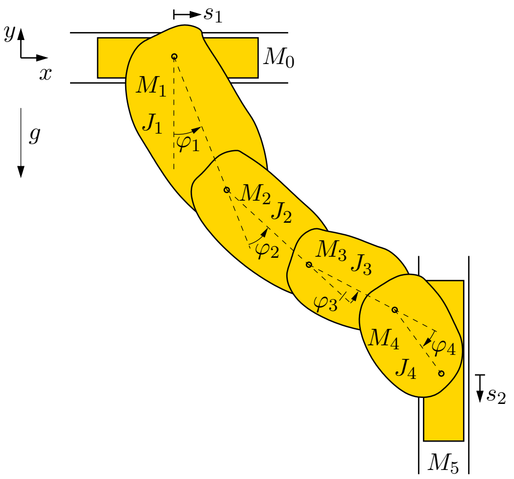


Figure 6: Sketch of fourfold pendulum

ery joint except the last one connecting the bodies with masses M_4 and M_5. The initial values for the pendulum angles are

$$\varphi_1(0) = 45\,\text{deg}, \qquad \varphi_2(0) = -15\,\text{deg},$$
$$\varphi_3(0) = 30\,\text{deg}, \qquad \varphi_4(0) = -37.5\,\text{deg}.$$

Fig. 7 shows the initial configuration of the pendulum system. Here, the length proportions between the four bodies of the pendulum are illustrated.

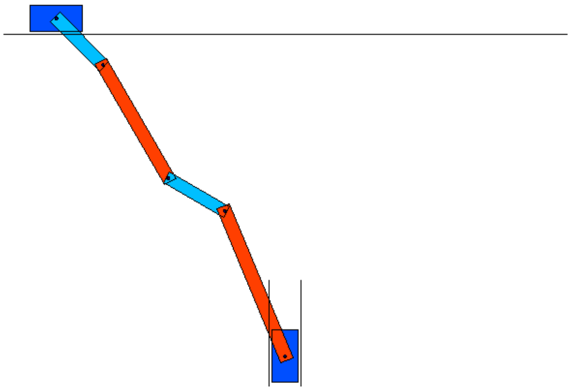


Figure 7: Start configuration of fourfold pendulum

Like before, the pendulum system was implemented twice using two different simulation models. The first implementation is based on the Multibody Standard Library and serves as a reference. The second model uses the modified Multibody Library on the basis of the motor algebra.

In order to compare both approaches concerning their simulation results, one instance of the first model and one instance of the second model were calculated simultaneously. This way, the deviations of both simulations can be calculated. For reasons of compactness only the maximum deviation for all prismatic joints and for all revolute joints are plotted in Fig. 8. They have the same order of magnitude as in the example before and can thus be explained by numerical errors.

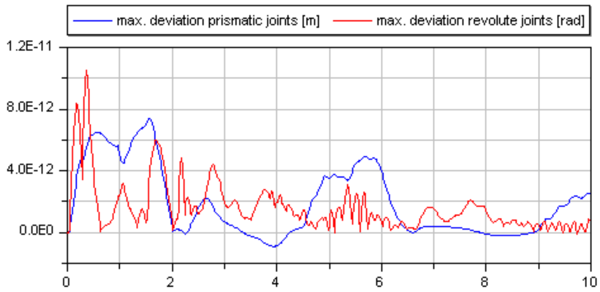


Figure 8: Maximal deviations between both simulations for all prismatic joints and all revolute joints

4.3 Rotating wheel on a movable axis

The last example is a rotating wheel that is fixed on a movable axis (see Fig. 9). There are three revolute joints within this mechanism. The first one allows a rotation of the rack (the long cylinder posing upright in Fig. 9) around the z-direction. A second revolute joint is the bearing of the wheel that enables the wheel to turn around its axis. Between them, there is a revolute joint enabling a rotation of the axis orthogonal to the rotation of the spinning wheel. Hence, in this example, the rigid bodies do not only perform planar motions.

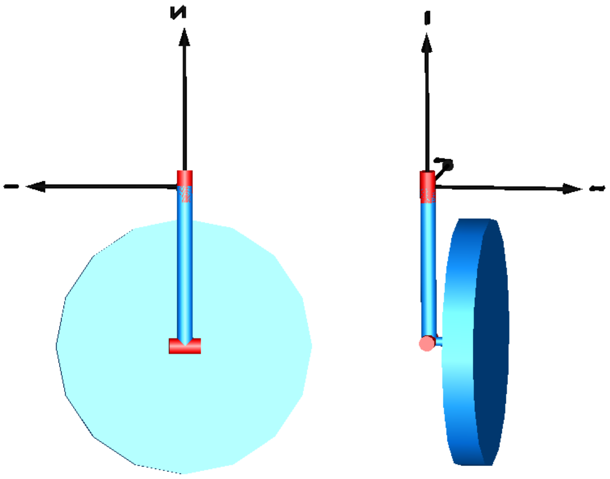


Figure 9: Sketches of a rotating wheel on movable axis

Again, the bodies move under the influence of the earth's gravitational field that acts in the negative z-direction.

At the beginning, the wheel turns with a speed of $\omega_3 = 50\,{}^{\mathrm{rad}}\!/\!{}_{\mathrm{s}}$ around its axis while the rack and the wheel's axes stand perpendicular to each other. In opposit to the foregoing examples, this system is completely undamped. Fig. 10 depicts the trajectory of the intersection point between the wheel axis and a unit sphere. Obviously, the resulting motion of the mechanism is a superposition of a precession and a free nutation. In order to illustrate this characteristic motion in more detail, Fig. 11 shows the same trajectory for an initial speed of only $10\,{}^{\mathrm{rad}}\!/\!{}_{\mathrm{s}}$.

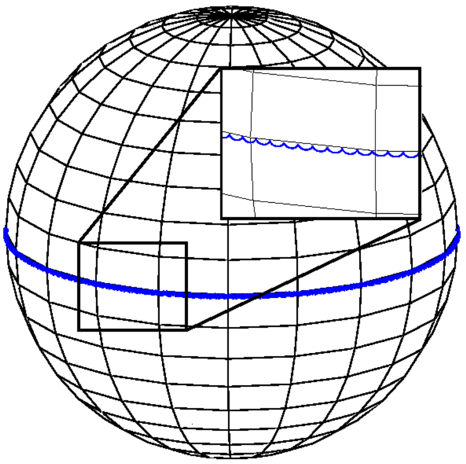


Figure 10: Trajectory of the intersection point between the rotation axis and a unit sphere for $\omega_3 = 50\,{}^{\mathrm{rad}}\!/\!{}_{\mathrm{s}}$

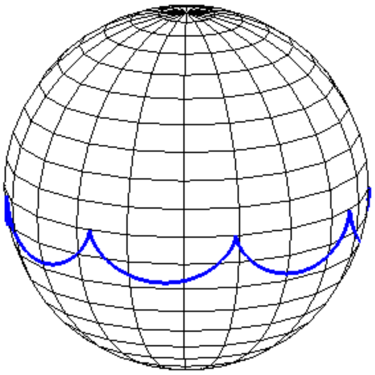


Figure 11: Trajectory of the intersection point between the rotation axis and a unit sphere for $\omega_3 = 10\,{}^{\mathrm{rad}}\!/\!{}_{\mathrm{s}}$

As in the paragraphs before, the example was implemented twice to compare Modelica Standard implementation with modified motor implementation. The deviations for all revolute joint angles between both implementations do not exceed $6 \cdot 10^{-12}$ for a simulation time of 10s. However, since the system is undamped, the deviations of both simulations increase with continuing time.

4.4 Performance analysis

All simulation tests were performed using the simulation tool Dymola in the version 7.1. The analysis of the performance requires the definition of a performance indicator. Even though Dymola provides a lot of information on the translation as well as the simulation process, we decided to evaluate the performance by the simulation time, since this might be the most interesting indicator for many users. For all simulations, we used a PC running the operating system Windows XP Professional. All simulations has been carried out ten times for each model while no other application was running on the system. A comparison of the average values of the simulation time for these implementations can be seen in Fig. 12.

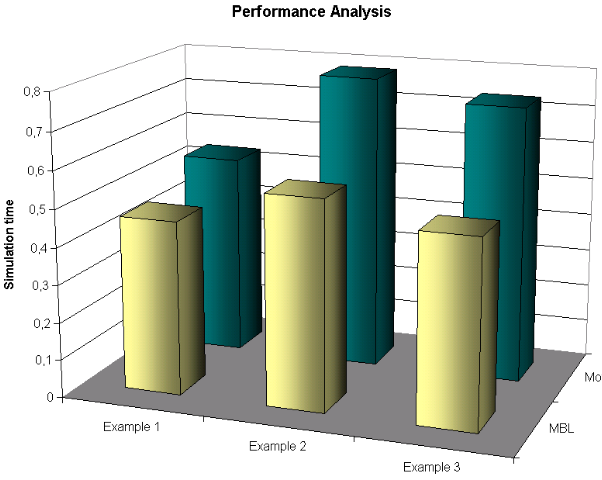


Figure 12: Simulation time of all examples for both implementations

Hence, the performance analysis on the chosen simulation system revealed that the modified multibody library on the basis of the motor algebra shows a performance which is inferior compared to the one of the Modelica Standard Library. According to the translation information of Dymola the reason might be that Dymola was not able to reduce and simplify the system equations of the motor implementation as much as the equations of the Modelica Standard Library. That seems reasonable due to the broad use of functions and vectorised quantities within the motor implementation. Indicated by this insight, further investigations with different simulation tools seem to be necessary for the future to get results which are clearer and better comparable.

5 Summary and outlook

The paper shows an alternative approach to modelling spatial multibody systems in Modelica. This approach is characterized by a clear and concise formulation of the equations of motion.

To get some experiences in terms of numerical efficiency and limits of this approach, an extended test implementation was carried out. Appropriate modifications of the Modelica Multibody Standard Library enabled us to compare the Standard Library implementation and the motor calculus implementation with regard to simulation time.

The results presented here were determined using the Modelica simulator Dymola. These results seem to encourage the idea of testing the motor calculus within other Modelica simulator tools, too.

References

[1] J. Angeles. *Fundamentals of Robotic Mechanical Systems.*. Second Edition. NewYork, Springer-Verlag, 2003.

[2] R.S. Ball. *A Treatise on the Theory of Skrews*. Cambridge University Press, 1900.

[3] W.K. Clifford. Preliminary sketch of biquaternions. *Proc. London Math. Soc.*, 4:381–395, 1873.

[4] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Wiley-IEEE Press, 2003.

[5] C. Heinz. Motorrechnung im X_{1+3+3}. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 67(11):537–544, 1987.

[6] R. von Mises. Motorrechnung, ein neues Hilfsmittel der Mechanik. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 4(2):155–181, 1924.

[7] R. von Mises. Anwendungen der Motorrechnung. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 4(3):193–213, 1924.

[8] `http://www.modelica.org/events`. seen on August 10th, 2009.

[9] M. Otter, H. Elmqvist, and S. E. Mattsson. The New Modelica MultiBody Library. In *3rd International Modelica Conference, Linköping, Sweden, November 3–4, 2003, Proc.*, pages 311–330. The Modelica Association, 2003.

[10] B. Roth. Screws, motors, and wrenches that cannot be bought in a hardware store. In M. Brady and R. Paul (eds.): *The First Internal Symposium on Robotic Research.*, MIT Press, Cambridge (MA), pp. 679–693,.

[11] K. Sugimoto. Kinematic and Dynamic Analysis of Parallel Manipulators by Means of Motor Algebra. Journal of mechanisms, transmissions, and automation in design, vol. 109(1), pp 3–7, 1987.

[12] E. Study. *Geometrie von Dynamen. Die Zusammensetzung von Kräften und verwandte Gegenstände der Geometrie.* Teubner, Leipzig, 1903,

[13] H. Stumpf and J. Badur. On the non-abelian motor calculus. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 70(12):551–555, 1990.

[14] M.M. Tiller. *Introduction to Physical Modeling with Modelica*. Springer, 2001.

[15] T. Zaiczek, O. Enge-Rosenblatt. Towards an Object-oriented Implementation of VON MISES' Motor Calculus Using Modelica. In *2nd International Workshop on Equation-Based Object-Oriented Languages and Tools, Paphos, Cyprus, July 3, 2008, Proc.,* pages 131–140.

Implementation of the Contensou–Erismann Model of Friction in Frame of the Hertz Contact Problem on Modelica

Ivan Kosenko Evgeniy Aleksandrov
Russian State University of Tourism and Service, Department of Engineering Mechanics
Glavnaya str. 99, Cherkizovo-1, Moscow reg., 141221, Russia

Abstract

An approximate model to compute resulting wrench of the dry friction tangent forces in frame of the Hertz contact problem is built up. An approach under consideration develops in a natural way the contact model constructed earlier. Generally an analytic computation of the integrals in the Contensou–Erismann model leads to the cumbersome calculation, decades of terms, including rational functions depending in turn on complete elliptic integrals. To implement the elastic bodies contact interaction computer model fast enough one builds up an approximate model in the way initially proposed by Contensou.

To verify the model built results obtained by several authors were applied. First the Tippe-Top dynamic model is used as an example under testing. It turned out the top revolution process is identical to one simulated with use of the set-valued functions approach.

In addition, the ball bearing dynamic model was also used to verify different approaches to the tangent forces computational implementation in details. A model objects corresponding to contacts between balls and raceways were replaced by ones of a new class developed here. Then the friction model of the approximate Contensou type embedded into the whole bearing dynamic model was thoroughly tested.

Keywords: Hertz contact model; Contensou simplified model; Contensou–Erismann model; Vil'ke model; Tippe-Top; ball bearing model

1 Introduction

To make a contact model for the multibody dynamics more accurate and simultaneosly more efficient using the Hertz contact problem as a frame one has to develop an approach taking into account nature of the tangent forces acting along a contact spot area. The simplest case one could encounter in this way is one of the dry friction forces distributed over the elliptic area arising in the Hertz model. It is known as the Contensou–Erismann friction model [1, 2].

The model assumes the resulting wrench of the dry friction tangent forces. The wrench consists of the total friction force and the drilling friction torque. An approach under consideration continues in a natural way the contact model development started earlier [3]. The normal contact force distribution is determined by the Hertz model while the tangent forces on an elemental level satisfy the Amontons–Coulomb law for dry friction.

The dry friction force and torque are integrated over the contact elliptic spot thus composing the resulting wrench. Generally an analytic computation of the integrals mentioned leads to the cumbersome calculation including decades of terms depending on rational functions depending in turn on complete elliptic integrals.

To keep an accuracy and to make the model fast enough an approach proposed initially by Contensou [1] is built up. The model under construction is one derived from the Contensou simplified model in the following directions: (a) the model is anisotropic: total friction forces along the contact ellipse axes are different; (b) for the translatory and almost translatory relative motions one uses the Amontons–Coulomb friction law regularization [4]; (c) the approximate model for the drilling torque also is under construction.

2 Problem Formulation

The Hertz problem solution [5] to a normal pressure distributed over the contact area of elliptic shape is defined [6] by the formula

$$\sigma(x,y) = \frac{3N}{2\pi ab}\sqrt{1 - \frac{x^2}{a^2} - \frac{y^2}{b^2}},$$

where N is the total force of normal pressure, a and b are the contact spot ellipse semi-major and semi-minor

DOI: 10.3384/ecp09430006

axes respectively, see Figure 1, $Pxyz$ is the contact local coordinate frame oriented such that the x-axis is directed along the ellipse semi-major axis. All three values: N, a, and b supposed already computed by the Hertz algorithm [3].

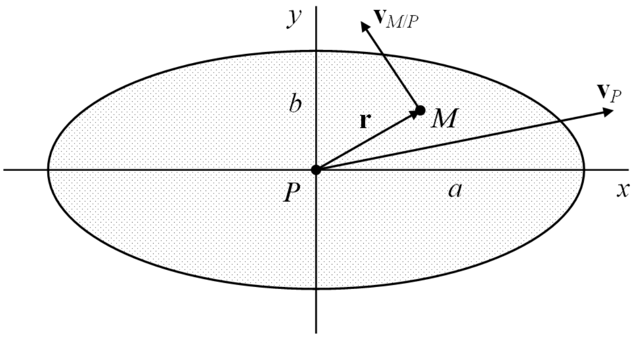


Figure 1: The contact spot area

The body A supposed “below/behind” the picture of Figure 1 plane while the body B supposed “above/in front of” it. In addition, all the forces under computation here supposed to act to the body B from the body A. Consider a method to compute a wrench consisting of the tangent force $\mathbf{F} = F_x\mathbf{i} + F_y\mathbf{j}$ and the resulting torque $\mathbf{T} = T_z\mathbf{k}$ arising due to couple of dry friction forces distributed over the contact area. This latter one usually called a drilling friction torque.

According to the dry friction definition the tangent stress $\boldsymbol{\tau}(x,y)$ at the contact spot any point $M(x,y)$ is computed by the formula

$$\boldsymbol{\tau}(x,y) = -f\sigma(x,y)\frac{\mathbf{v}_M(x,y)}{|\mathbf{v}_M(x,y)|}, \qquad (1)$$

where f is the dry friction coefficient, $\mathbf{v}_M(x,y)$ is the relative slip velocity of the body B with respect to (w. r. t.) body A at the geometric point M. The right hand side of Eqn. (1) isn’t regular. Because of that the set-valued functions calculus is applied frequently to the problems including dry friction [7]. Let us try to build up a computational procedures for the dry friction problems staying in frame of classical calculus and using the known results [8, 9] on asymptotic closeness for an exact and an approximate problems.

Assuming the bodies A and B to be rigid from the kinematic viewpoint the body B relative motion, along the contact spot plane, is an instant planar (the relative velocity normal component supposed to participate only in the normal force computation), and subsequently field of velocities over the spot is defined by the known Euler formula

$$\mathbf{v}_M(x,y) = \mathbf{v}_P + \mathbf{v}_{M/P}(x,y) = \begin{pmatrix} v_{Px} - \omega_z y \\ v_{Py} + \omega_z x \\ 0 \end{pmatrix}, \qquad (2)$$

where $\mathbf{v}_{M/P}(x,y) = [\boldsymbol{\omega}, \mathbf{r}(x,y)]$, and $\mathbf{r}(x,y) = x\mathbf{i} + y\mathbf{j}$ is the current point $M(x,y)$ radius vector within the contact spot, see Figure 1. The ellipse central point relative slip velocity $\mathbf{v}_P$ is represented as follows

$$\mathbf{v}_P = v\begin{pmatrix} \alpha \\ \beta \\ 0 \end{pmatrix} = v\mathbf{w}, \quad \alpha = \cos\psi, \quad \beta = \sin\psi,$$

where v is a relative slip velocity value at the point P, and ψ is the angle between the axis Px and vector $\mathbf{v}_P$. According to the Contensou–Erismann model [1, 2] to compute the dry friction total force and torque vectors one has to evaluate integrals over the contact elliptic area in the following way

$$\mathbf{F} = \iint \boldsymbol{\tau}(x,y)dxdy, \ \mathbf{T} = \iint [\mathbf{r}(x,y), \boldsymbol{\tau}(x,y)]\, dxdy. \qquad (3)$$

For the further use it is suitable to introduce the dimensionless velocity $u = v/a\omega_z$ of relative slipping at the point P instead of a dimensioned one.

3 Theoretical Background

It turned out the friction total force and drilling friction torque components are regular functions of the relative sliding (dimensionless) velocity u, relative angular velocity supposed fixed parameter here, at a center of the ellipse such that for the exact force $\mathbf{F}$ and torque $\mathbf{T}$ we have

$$\mathbf{F}(u) = \mathbf{F}_0^\infty + \mathbf{O}\left(u^{-2}\right), \quad \mathbf{T}(u) = \mathbf{T}_1^\infty u^{-1} + \mathbf{O}\left(u^{-2}\right) \qquad (4)$$

as $u \longrightarrow \infty$, and

$$\mathbf{F}(u) = \mathbf{F}_1^0 u + \mathbf{O}\left(u^2\right), \quad \mathbf{T}(u) = \mathbf{T}_0^0 + \mathbf{O}\left(u^2\right) \qquad (5)$$

as $u \longrightarrow 0$. Here $\mathbf{F}_0^\infty$, $\mathbf{T}_1^\infty$, $\mathbf{F}_1^0$, $\mathbf{T}_0^0$ are constant vectors defining the approximate model. The vectors $\mathbf{F}_1^0$, $\mathbf{T}_0^0$ depend on complete elliptic integrals of the first and the second kind depending in turn on the contact ellipse eccentricity in the following way

$$F^0_{1\,x} = -\alpha A u, \quad F^0_{1\,y} = -\beta B u, \quad T^0_{0\,z} = -C$$

with the constants

$$\begin{aligned} A &= \frac{3}{2}\frac{\mathbf{K}(e) - \mathbf{E}(e)}{e^2}, \\ B &= \frac{3}{2}\left(\mathbf{K}(e) + \frac{\mathbf{E}(e) - \mathbf{K}(e)}{e^2}\right), \\ C &= \frac{3}{8}\mathbf{E}(e), \end{aligned}$$

where $\mathbf{K}(e)$, $\mathbf{E}(e)$ are complete elliptic integrals of the first and second kind respectively. Remark that really the values A, B, C are a variable functions of time because the contact ellipse eccentricity e can vary while the simulation process.

Note here the dry friction total wrench simplified model doesn't require any noticeable computational resources because complete elliptic integrals mentioned are already calculated while computing the total normal force according to the Hertz model.

As Contensou [1] remarked the main effect in the Contensou–Erismann dry friction model if the contact area is non-zero besides the drilling friction torque arises is that the total friction force decreases monotonically to zero as a function of u.

On the other hand one can easily note from (5) that a steepness of the total friction force change, as a function of v, grows as $a\omega_z \longrightarrow 0$. In this case either contact spot area decreases to zero or the drilling angular velocity vanishes. Finally, for the value $a\omega_z$ small enough and $A, B \geq a\omega_z/\delta$, where δ is a regularization parameter for the case of dry friction, we have the almost point contact case already implemented earlier [4] as a dry friction model regularization. Thus in the current simplified Contensou model resulting computer model always uses the "regular" case of the friction force decreasing, though sometimes steep, to zero. Taking into account that according to (4) for u large enough the simplified friction force differs from its Coulomb's value by the magnitude of the second order of smallness and following the Contensou proposal [1] let us simplify our model such that the friction force supposed to be of the Amontons–Coulomb type for $u \in \left[u^*_{x,y}, \infty\right)$ and linear one for $u \in \left[0, u^*_{x,y}\right)$. Note, the friction force has an anisotropy here: constants u^*_x and u^*_y along axes Px and Py respectively are in general different. Evidently, we can find these values from equations

$$Au^*_x = 1, \quad Bu^*_y = 1.$$

Now we can represent the Contensou approximate model for the (dimensionless) functions $F_x(u)$, $F_y(u)$, $T_z(u)$ as follows

$$X_C(u) = -\alpha \begin{cases} Au & \text{for} \quad u \in [0, u^*_x), \\ 1 & \text{for} \quad u \in [u^*_x, \infty), \end{cases}$$

$$Y_C(u) = -\beta \begin{cases} Bu & \text{for} \quad u \in \left[0, u^*_y\right), \\ 1 & \text{for} \quad u \in \left[u^*_y, \infty\right), \end{cases}$$

$$T_C(u) = \begin{cases} -C & \text{for} \quad u \in \left[0, u^*_z\right), \\ T^\infty_{1z} u^{-1} & \text{for} \quad u \in \left[u^*_z, \infty\right), \end{cases}$$

where u^*_z is a sewing point for the horizontal "shelf" of height C and a branch of the hyperbola decreasing at infinity and being defined by the function $T^\infty_z(u)$. An equation specifying the value u^*_z has the form

$$C = -\frac{T^\infty_{1z}}{u^*_z}.$$

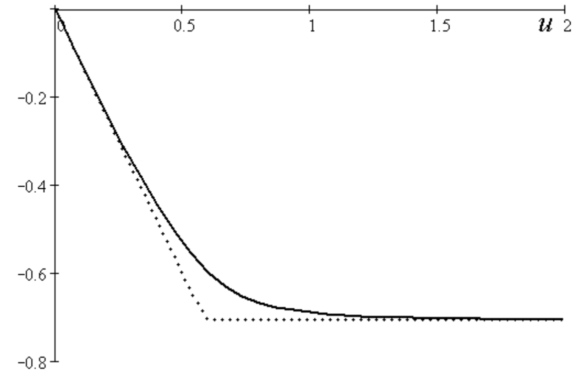


Figure 2: x-components of the force vector for exact and approximate models (similar picture for y-components)

Comparison of graphs for the Contensou–Erismann model functions is represented in Figures 2 and 3. Functions of the exact model correspond to solid lines, and ones of the approximate model correspond to the dotted lines.

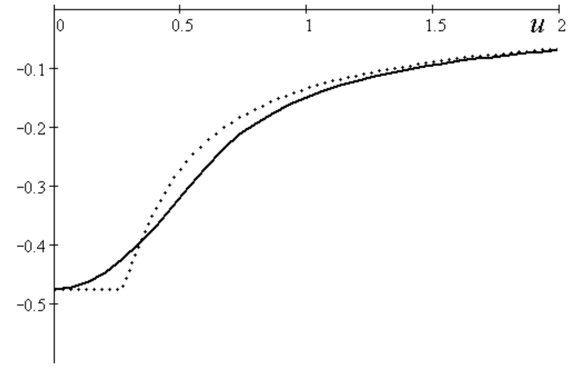


Figure 3: The drilling friction torque for exact and approximate models

If while the simulation process all the values u^*_x, u^*_y, u^*_z found become less than $\delta/a\omega_z$ then we arrive at the regularized Coulomb model implemented earlier. Thus the approximate Contensou model implemented here really is a simplest generalization of the regularized Coulomb one mentioned regularizing it even more by introducing the parameters u^*_x, u^*_y, u^*_z enhancing initial use of the parameter δ. Such an improvement simply is a consequence of the contact spot existence in the exact model. Thus as a result, with the exception of the cases of $a = 0$ and $\omega_z = 0$, we can avoid use the set-valued functions being able to apply the procedures of classical calculus.

The approximate model under construction here has several differences from a piece-wise linear approximation built up in the paper [1]:

(a) the model is anisotropic and is suitable for the elliptic contact area of any eccentricity;

(b) for the cases of instant translatory and almost instant translatory bodies in the contact relative motion with the conditions

$$u_x^* a\omega_z < \delta, \quad u_y^* a\omega_z < \delta, \quad u_z^* a\omega_z < \delta$$

fulfilled simultaneously we apply the dry friction regularization proposed in [8] and [11];

(c) the approximate model used also for the drilling friction torque.

Computations show the force/torque expressions represented here give an approximation of the Contensou–Erismann model more accurate in compare with the linear-fractional approximation satisfying boundary conditions at zero and infinity. If we use the Pade approximations with the polynomials of the second and third degrees [12] then the resulting accuracy is improved but computations become more significant.

It is known [13] the V. G. Vil'ke formula gives an approximation for the contact interaction normal elastic force decent enough in a wide range of eccentricities. Computer implementation of such a model runs noticeably faster than the implementation of the exact Hertz model. The main reason for that is a necessity in the latter case to resolve the transcendental equation

$$\frac{1}{2}\frac{\mathbf{K}(c)}{\mathbf{K}'(c)} - (1-c) = g \quad (0 \le c < 1, 0 < g \le 1) \qquad (6)$$

w. r. t. $c = e^2$ which is the contact spot eccentricity squared. Here we use the elliptic integral modulus squared c as an argument of complete elliptic integral of the first kind, as it has been done in [14].

The V. G. Vil'ke algorithm to compute the normal contact force doesn't require to know the current value of c but the Contensou–Erismann anisotropic friction model does. To keep the gain has been gotten while the normal force calculation and don't waste the computer time to resolve the equation (6) this time to compute the tangent friction force it turned out to be possible that the solution mentioned can be reduced to the explicit linear formula once applied.

After the value $c = c_*$ needed has been computed then to find the values A, B, C mentioned above we should calculate complete elliptic integrals of the first and second kind using theta-functions [14]. First of all for any $c_* \in [0,1)$ one can use the expansion

$$\theta_3(q) = 1 + 2\sum_{n=1}^{\infty} q^{n^2}$$

with fast conversion such that complete elliptic integral of the first kind can be computed by the formula

$$\mathbf{K}(c) = \frac{\pi}{2}\theta_3^2(q),$$

where nome q is calculated with a very high accuracy using equations [14, 15]

$$\varepsilon = \frac{1}{2}\cdot\frac{1-(1-c)^{1/4}}{1+(1-c)^{1/4}},$$

$$q = \varepsilon + 2\varepsilon^5 + 15\varepsilon^9 + 150\varepsilon^{13} + 1707\varepsilon^{17} + \ldots,$$

and the terms enumerated above are sufficient for the accuracy level of order not less than one for the value $1/2^{21}$.

It is convenient for complete elliptic integral of the second kind to use the formula [16]

$$\mathbf{E}(c) = \frac{2-c}{3}\mathbf{K}(c) + \frac{\pi^2}{\mathbf{K}(c)}\left[\frac{1}{12} - 2\sum_{n=1}^{\infty}\frac{q^{2n}}{(1-q^{2n})^2}\right].$$

If the value of c is small then to regularize the expression

$$\frac{\mathbf{E}(c)-\mathbf{K}(c)}{2c} = \frac{d\mathbf{E}}{dc}$$

one can use hypergeometric expansions converging well enough in this case [17].

4 Implementation Specifics

According to experience while developing the models for elastic contacting of rigid bodies interactions in the multibody dynamics a flexibility provided by Modelica can be used to utilize a wide variety of different properties concerning a contact of solids. The properties are mainly of the following categories:

(a) geometric properties for surfaces in vicinity of the contact spot (gradients of the functions defining surfaces, their Hesse matrices);

(b) a model to compute the contact area dimensions and normal elastic force;

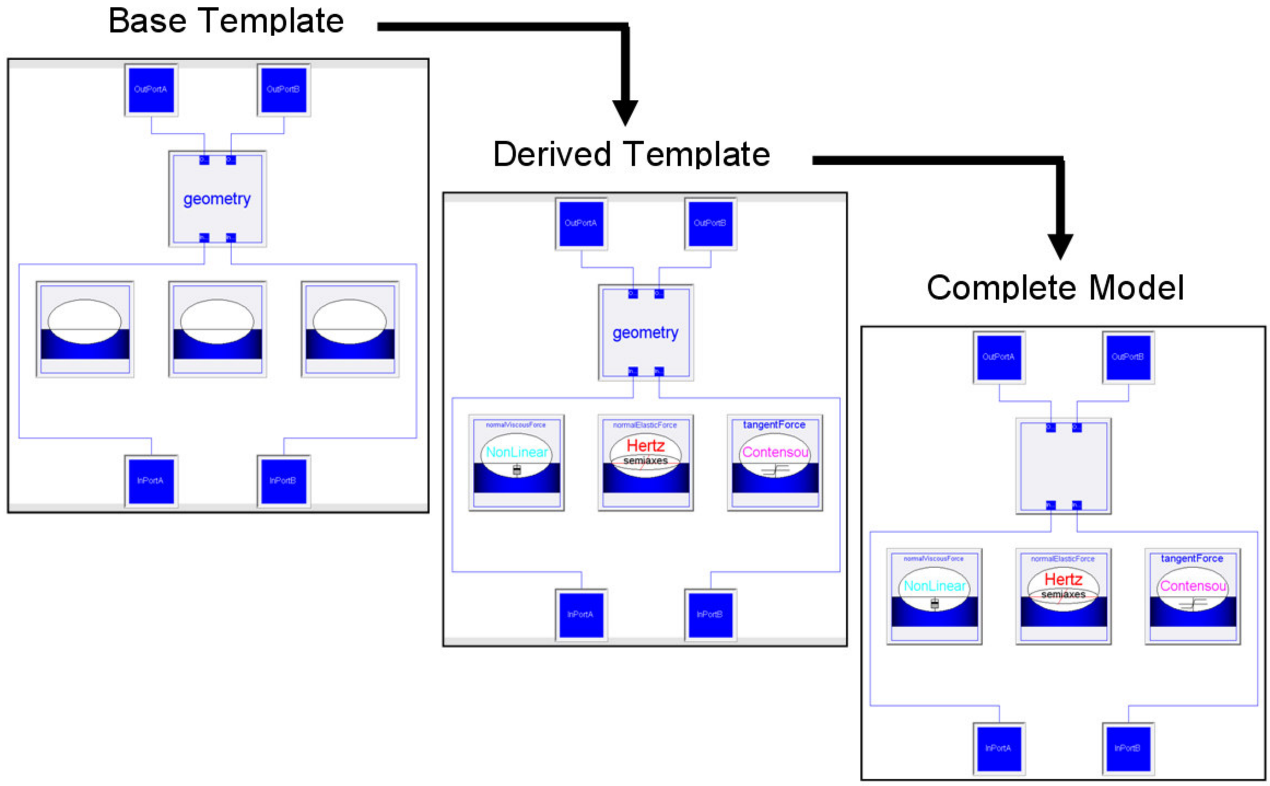


Figure 4: The model of mechanical contact by stages of inheritance.

(c) model for the normal viscous force of resistance;

(d) model for the tangent forces along the plane of the contact area.

A submodel of the geometry properties is to describe analytically algebraic surfaces of the structure complex enough. To implement the normal force computation one can choose from at least two approaches: the Hertz model and its volumetric modification. Force of viscous resistance also can be modeled in several different ways: linear, non-linear, etc. In the models for tangent forces one can adopt either “simplest” approaches based on the Amontons–Coulomb friction or more complex ones represented by the Contensou–Erismann, and other models.
While developing a mechanical contact model architecture we used the base class `Constraint` described earlier [18] as a starting point to construct its inheritor `ContactConstraintTemplate` being simultaneously a base class of new family of models to simulate mechanical contacts. Really this class is a base template represented as a container having four “sockets” to instantiate there the specific parameter classes of four types enumerated above, see its visual model in Figure 4 at a top left corner, and its Modelica code can be outlined as follows

```
partial model ContactConstraintTemplate
  extends Constraint;
 ...
 replaceable
   NormalElasticForce
     normalElasticForce;
 replaceable
   NormalViscousForce
     normalViscousForce;
 replaceable
   TangentForce
     tangentForce;
 replaceable
   SurfacesOfConstraintDifferential
     geometry;
 ...
end ContactConstraintTemplate;
```

To develop complete model one can move along different ways. Class parametrization implemented in Modelica is the facility in line to apply to the problem under description. In our case we have four class parameters corresponding to the submodel categories enumerated above. An example to construct specific contact interaction model see in Figure 4. The example includes two stages of inheritance:

1. to derive a template with the forces models, namely: the Hertz model for normal force, nonlinear viscous force, the Contensou–Erismann model for the dry friction forces (to “fill in” three sockets in the middle of the base template visual

model, see the derived template visual model at a central position of the Figure 4);

2. to complete the whole construct one should define a specific geometry submodel for the surfaces in contact (to "seal" the socket for geometry properties, see the complete visual model at a bottom right corner of the Figure 4).

The Modelica code for the intermediate derived template can be represented in the following way

```
partial model
  ContactConstraintTemplate...
  extends ContactConstraintTemplate(
    redeclare
      NormalElacticForceHertzDiff
        normalElasticForce,
    redeclare
      NormalViscousForceNonLinear
        normalViscousForce,
    redeclare
      TangentForceContensou
        tangentForce);
  ...
end
  ContactConstraintTemplate...;
```

On all the stages of inheritance the templates considered have an internal information interconnections between the submodels to be instantiated. These interconnections are implemented via the set of equations hidden behind the visual models and can vary for different models requiring different variables for the algorithms to compute normal and tangent forces of the complete model. So the whole picture remind us known construct of a card with the sockets and the interconnection wiring in its internal layers as a base template, and a chips to be instantiated in the sockets as a models of four types from above. With one exclusion: we have the derived template playing a role of additional card with its own additional wiring servicing already instantiated models "covering the card" of the base template.

One can remark finally an approach under presentation allows us to create and to change fast enough different types of an elastic contact models while developing the multibody dynamics systems simulators.

5 Numeric Experiments

The tangent forces model under presentation here has been verified by two stages: (a) for the case of circular contact; (b) for the case of elliptic non-circular contact. The known Tippe-Top dynamical model was investigated as an example of the first case. All the parameters and initial conditions are exactly the same as in the paper [19] whose authors got these data in turn from the work [20]. The only difference is that in our case we considered an unrestricted problem with the contact ellipse, including depth of penetration and normal force, being computed dynamically.

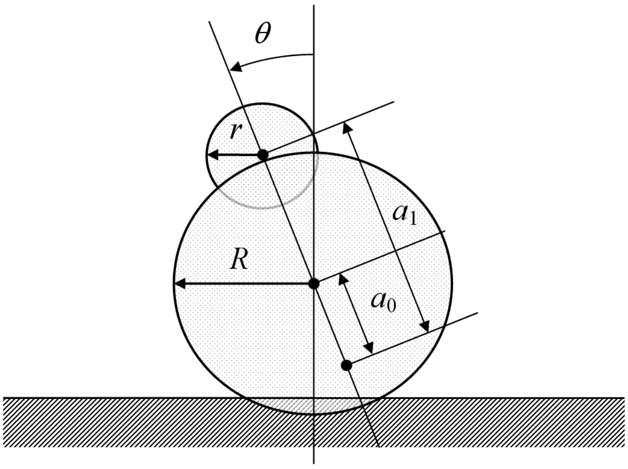


Figure 5: The Tippe-Top geometric properties

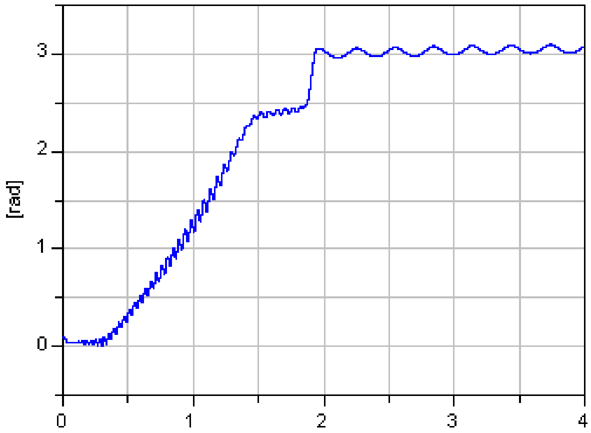


Figure 6: The top axis of symmetry evolution

The top body, supposed geometrically rigid, composed by two balls, Figure 5, one of larger radius $R = 1.5 \cdot 10^{-2}$m, and another, smaller, one of the radius $r = 0.5 \cdot 10^{-2}$m. The top mass center location supposed "under" the larger ball geometric center on its axis of symmetry at a distance of $a_0 = 3 \cdot 10^{-3}$m and at the distance of $a_1 = 16 \cdot 10^{-3}$m to the smaller ball center. The top mass is equal to $m = 6 \cdot 10^{-3}$kg. The top body supposed dynamically symmetric, and the central principal moments of inertia are the following: an equatorial moment equals to $8 \cdot 10^{-7}$kg $\cdot$ m^2, and a polar one has the value of $7 \cdot 10^{-7}$kg $\cdot$ m^2. A material the top and the horizontal floor the top rolls on are made

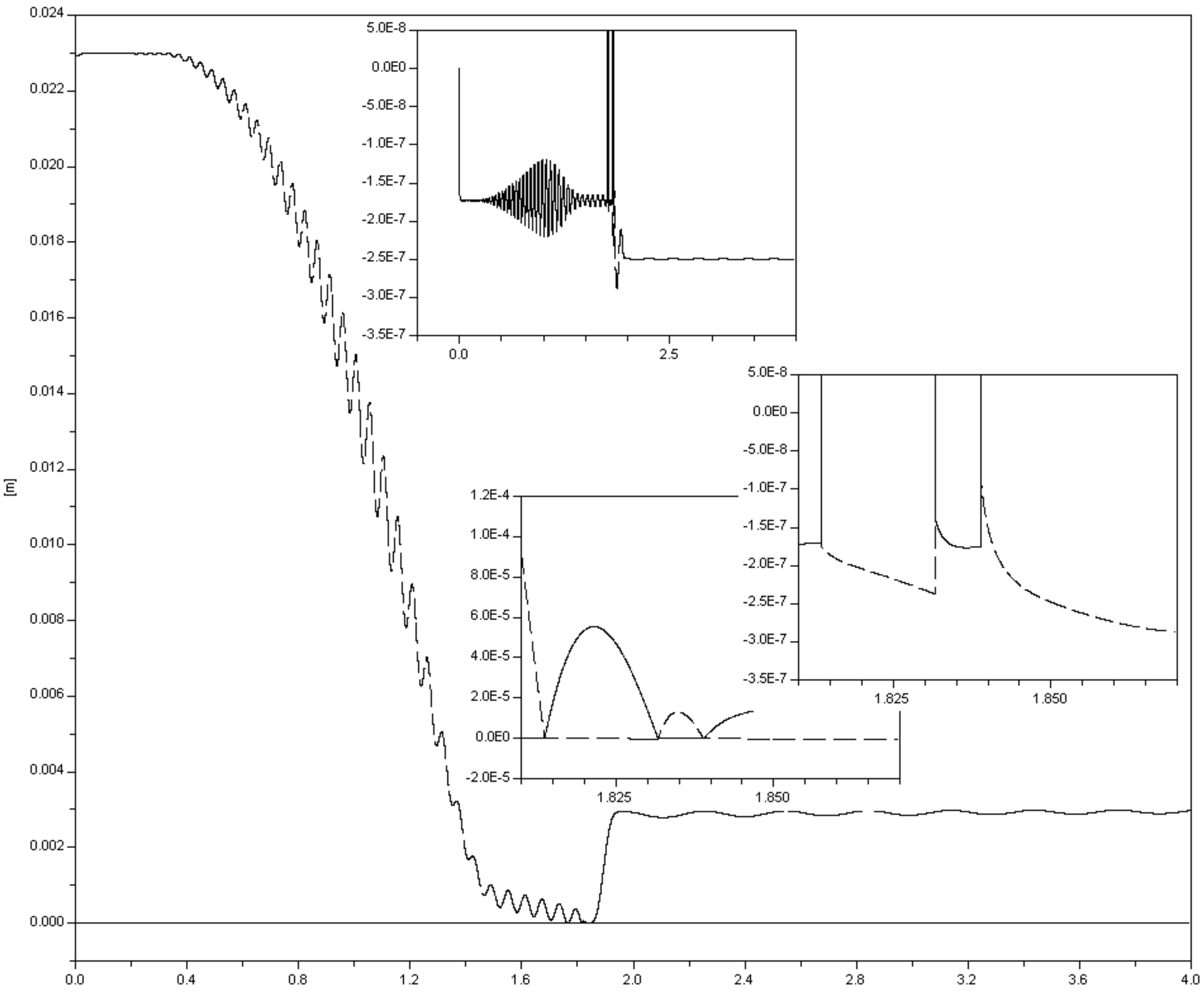


Figure 7: The contact indicators evolution. Some fragments zoomed in and rescaled.

of the wood with Young's modulus $E = 9.1 \cdot 10^9 \mathrm{N/m^2}$. If we suppose the Poisson ratios as 0.3 then an effective Young's modulus has to be $E_* = 5 \cdot 10^9 \mathrm{N/m^2}$ just as in [19]. The dry friction coefficient supposed to be equal to the value $f = 0.3$.

The top center of mass supposed resting at initial instant of motion. Besides the top itself, more accurately its larger ball, assumed without any initial penetration with the horizontal surface. The smaller ball is located on the upper hemisphere of the larger ball, and initially the top axis of symmetry bends w. r. t. vertical by the angle $\theta_0 = 0.1$rad. Initial angular velocity $\omega_0 = 180 \mathrm{s}^{-1}$ is the same as in [19] and directed along axis of the top symmetry.

Note that in [19] contact problem is interpreted as usual in so to speak "restricted" sense: the contact area supposed constant and predefined corresponding to the normal force of the static equilibrium. This assumption concerns the contacts for both the balls with the same contact area radius. Actually, while motion the normal reaction force, being implemented here by elastic forces, changes. Then the contact spot changes also, and so for its radius. Thus the top really undergoes the vertical microvibrations. And, as one can see from the above simplified model, the drilling friction torque also changes causing in general a consequences for the top motion.

In the model under development here we consider an unrestricted contact problem that is the normal force is computed from the Hertz (or V. G. Vil'ke) model with addition of some nonlinear viscous term. Simultaneously the contact area is computed too. Then all the data have been gotten are used to calculate the tangent force and the drilling friction torque in frames of the simplified Contensou model.

Remarkably, a computational experiment showed the top revolution from "feet", the larger ball in contact, to "head", the smaller ball in contact, scenario obtained in [19] using another approach to the problem, based on the set-valued functions calculus, repeated in our model with a high degree of accuracy. One can get an access to the paper [19] visiting, for instance, the

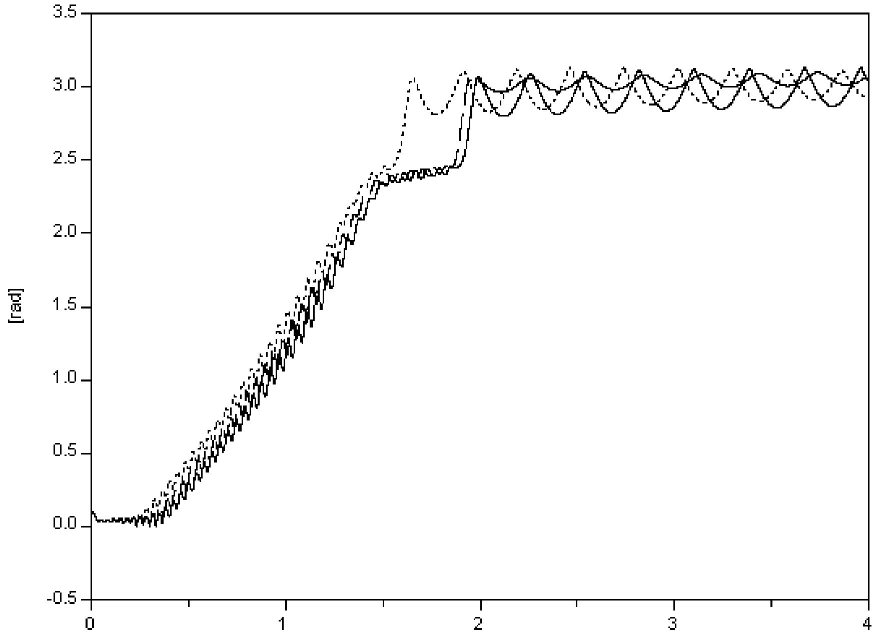


Figure 8: Comparison of three models

one of the authors Home Page, see [21]. Graph in Figure 6 illustrates well the Tippe-Top revolution process: similar to [19] it shows the $\theta(t)$ angle evolution. If we compare this plot with one from [19] then soon the complete identity can be observed. The only difference is that in Figure 6 one can find additional vibrations of small amplitude evidently existing due to elastic compliance in direction normal to the contact area. Similar identity show the curves of the contact indicators for the balls the top composed of and the horizontal surface, see Figure 7. The indicator for the pair (larger ball, floor) marked by the solid line while the (smaller ball, floor) contact indicator pictured as a dashed curve.

Really an indicators are the distances between an opposite points for the surfaces being tracked for contact. The indicator is strictly positive if contact is absent. Otherwise it is less than (if the bodies are in a state of mutual penetration) or equal to (if the bodies touch one another exactly at one point) zero. Let us describe the Figure 7 in more details. Initially the top smaller ball is out of contact, and corresponding indicator is positive, dashed curve. But other indicator is not equal to zero. Instead it is negative, see the vertically scaled subfigure at the upper edge of Figure 7. Here at the very left side we see that initially indicator set to be of the zero value. Then the penetration develops and the whole top sinks into the floor by very small depth until the vertical quasi-equilibrium is reached. After that we can observe the vertical micro-oscillations develop into the modulated pulse decreasing afterwards. One can match the problem parameters such that the pulse amplitude will grow and the top can start to bounce over the floor thus distorting all the following dynamical predictions of its revolution. In the upper subfigure we can observe also the change of the balls at contact, before the instant of time = 2 seconds. Then for the case of the smaller ball contacting the floor we observe the larger depth of penetration. Indeed, in this case we have a smaller area of the contact spot.

The bottom subfigure reflects the revolution process inself. Here the whole graph zoomed in vicinity of the time instant of 1.825 seconds, and we see that the revolution process is implemented by two attempts: two times the solid humps alternate the dashed ones. Thus first time the Tippe-Top “head” touches the floor then it once more is forced to loose a contact temporarily, and only then the head–floor contact becomes permanent. The right subfigure illustrates the depth of penetration for the larger and smaller balls by the vertical rescaling over the same interval as for the bottom subfigure.

In addition, yet another verification procedure has been performed, this time using the results of the paper [10] (one can access the paper [10] using the ScienceDirect on-line library [22]). Namely, exact formulae for the friction force and for the drilling friction torque, case (a), were applied to the top dynamics computer model implemented on Modelica language in frame of the unrestricted, in sense mentioned above,

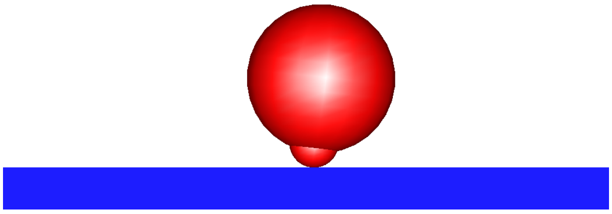


Figure 9: The Tippe-Top 3D-animation

contact model. In the same dynamical frame the simplified Contensou model, case (b), as well as a linear-fractional Pade approximation for the friction force and torque, case (c), were also implemented. The results of the inclination angle evolution are shown in Figure 8, where the cases (a), (b), and (c) correspond to the solid, dashed, and dotted curves respectively. One can see easily the revolution scenarios are mutually closest in cases (a) and (b). The 3D-animation shot is shown in Figure 9.

Note in addition, one can easily obtain a behavior typical to the Tippe-Top, revolution to "head", in frame of the "regularized" Amontons–Coulomb friction. One has to understand regularization in a sense proposed in the works [8, 11] and used in [4] in case of the point contact. We only have to "bend" graph for the friction force dependence on the relative slip velocity in vicinity of zero replacing its discontinuity by the linear function. The more flat slope of the graph the sooner one can find out the Tippe-Top revolution effect. As the simplified Contensou model shows that just this slope appears in the corresponding graph for the friction force dependence on the velocity, this time in frame of the exact Contensou–Erismann model.

The dynamical model of the ball bearing was considered in a way similar to the paper [3] while the verification second stage. This time the contact area is essentially elliptic one. The main goal for the numeric simulations was to compare two approaches: (a) the standard Hertz model for the normal force plus the Contensou simplified model for the friction forces; (b) the simplified model of V. G. Vil'ke for the normal elastic force plus the Contensou simplified model for the friction forces. As it was observed in [3] for the case of the regularized Coulomb friction force here dynamical models of the cases (a) and (b) differ one from another in a slightest degree too. Simultaneously, the model (b) is faster than (a) by 20% meaning the CPU time needed. To illustrate this in Figure 10 we compare the cases (a) and (b) for one component of the tangent friction force at a contact between one of the bearing ball and its inner raceway. The solid curve corresponds to the case (a) while the dashed one represents the case (b). A values of the contact spot eccentricity squared appeared to be constant equal to 0.687 in the case (a) and 0.643 in the case (b).

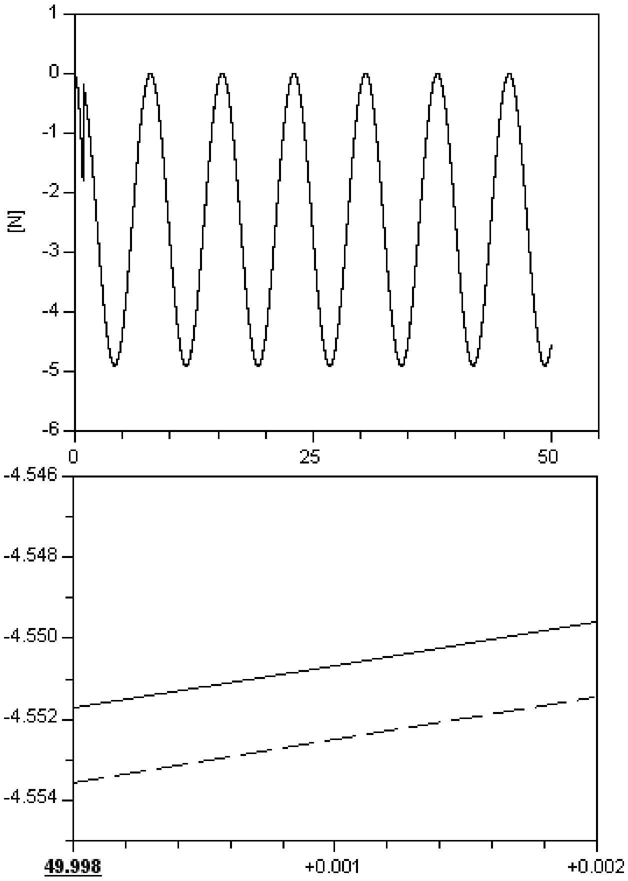


Figure 10: One of the friction force component evolution. A final stage of the simulation zoomed in.

6 Conclusions

Summarizing the results described above we can remark the following.

- The Tippe-Top "on head" revolution effect is caused completely by the dry friction force "regularization" in vicinity of zero value for the velocity of relative slip. Such a regularization takes place exactly in the Contensou–Erismann model. Numeric experiments showed if the slope of friction force graph in vicinity of the zero velocity in the regularized Coulomb model is steep enough then the Tippe-Top effect either isn't observed at all or arising during short time after a long evolution then vanishes quickly. And only noticeable decreasing of the slope mentioned immediately causes the top revolution on the "head" with the subsequent long precession in this position.

The Contensou–Erismann model creates a property just as one described above. Note the drilling friction torque role is reduced to a dissipative effect with subsequent gradual "fall" of the top approaching it to the static stable configuration.

- Since complete elliptic integrals used in the Contensou simplified model are already found in frame of the Hertz algorithm while computing the normal force then from the computational viewpoint application of this model is practically "cost free". If, in addition, we will take into account an effect of the regularization provided by the Contensou–Erismann model then we arrive at a unexpected from the first sight result: a numeric simulation of the Hertz model for the normal force and the Contensou–Erismann for the tangent force and the drilling friction torque turned out to be faster than the combination of the Hertz model and the "simple" Amontons–Coulomb dry friction. It is evident such a deceleration in latter case surely concerns the large stiffness of the problem while the almost rolling mode.
- Though for isotropic case, one of the circular contact area, the tangent forces average values for the Amontons–Coulomb and Contensou–Erismann models differ not so much, however in anisotropic case the first model becomes inadequate while the second one continues to serve correctly the contacting process simulation. Such a property has an importance for instance in case of the ball bearing simulation with the contact areas of essentially elliptic form.

Regarding the directions of a future work one can enumerate possible development and testing for different kinds of the contact properties combinations: normal-elastic-force / normal-viscous-force / tangent-force+drilling-friction-torque to match various engineering applications. It would be for instance different types of lubrication, or any new types of the normal elastic volumetric models etc.

7 Acknowledgement

The paper was prepared with partial support of Russian Foundation for Basic Research, projects 08-01-00600-a, 08-01-00718-a, 08-08-00553-a.

References

[1] Contensou, P., Couplage entre frottement de glissement et frottement de pivotement dans la théorie de la toupie. In: Kreiselprobleme Gyrodynamics: IUTAM Symposium Celerina, 1962, Berlin: Springer, 1963, pp. 201–216.

[2] Erismann, Th., Theorie und Anwendungen des echten Kugelgetriebes. Z. angew. Math. Phys., 1954, Vol. 5, No. 5, pp. 355–388.

[3] Kosenko I. I., Alexandrov E. B., Implementation of the Hertz Contact Model and Its Volumetric Modification on Modelica. In: Bachmann, B. (Ed.) Proceedings of the 6th International Modelica Conference, Bielefeld, Germany, March 3–4, 2008, Bielefeld: The Modelica Association, and University of Applied Sciences Bielefeld, 2008, pp. 203–212.

[4] Kossenko, I. I., Implementation of Unilateral Multibody Dynamics on Modelica. In: Schmitz, G. (Ed.) Proceedings of the 4th International Modelica Conference, Hamburg–Harburg, Germany, March 7–8, 2005, Hamburg–Harburg: The Modelica Association, and The Department of Thermodynamics, Hamburg University of Technology, 2005, pp. 13–23.

[5] Hertz, H., Über die Berührung fester elastischer Körper. J. reine und angewandte Mathematik, 1882, B. 92, S. 156–171.

[6] Landau, L. D. and Lifshitz, E. M., Theory of Elasticity. 3rd Edition. Landau and Lifshitz Course of Theoretical Physics. Volume 7. Oxford – Boston – Johannesburg – Melbourne – New Delhi – Singapore: Reed Educational and Professional Publishing Ltd., 1999.

[7] Leine, R. I. and Nijmeijer, H., Dynamics and Bifurcations of Non-Smooth Mechanical Systems. Berlin – Heidelberg – New York: Springer Verlag, 2004.

[8] Novozhilov, I. V., Conditions of Stagnation in Systems with the Coulomb Friction. Mechanics of Solids, 1973, Vol. 8, No. 1, pp. 8–14.

[9] Novozhilov, I. V., Fractional Analysis: Methods of Motion Decomposition, Boston: Birkhauser, 1997.

[10] Zhuravlev, V. F., The Model of Dry Friction in the Problem of the Rolling of Rigid Bodies. J. Appl. Math. Mech., 1998, Vol. 62, No. 5, pp. 705–710.

[11] Rooney, G. T. and Deravi, P., Coulomb Friction in Mechanism Sliding Joints. Mechanism and Machine Theory, 1982, Vol. 17, Iss. 3, pp. 207–211.

[12] Kireenkov, A. A., Three-Dimensional Model of Combined Dry Friction and Its Application in Non-Holonomic Mechanics. In: van Campen, D. H., Lazurko, M. D. van den Oever, W. P. J. M., (Eds.) Proceedings of ENOC-2005, Fifth EUROMECH Nonlinear Dynamics Conference, Eindhoven, August 7–12, 2005, Eindhoven, The Netherlands: Eindhoven University of Technology, 2005.

[13] Aleksandrov, E. B., Vil'ke, V. G. Kosenko, I. I., Hertzian Contact Problem: Numerical Reduction and Volumetric Modification. Computational Mathematics and Mathematical Physics, 2008, Vol. 48, No. 12, pp. 2226–2240.

[14] Whittaker, E. T., Watson, G. N., A Course of Modern Analysis, Cambridge – New York – Melbourne – Madrid – Cape Town: Cambridge University Press, 2002.

[15] Janke, E., Emde, F., Lösch, F., Tafeln Höherer Funktionen, Stuttgart: B. G. Teubner Verlagsgesellschaft, 1960.

[16] Milne-Thomson, L. M., Elliptic Integrals, Abramowitz, M., Stegun, I. A., (Eds) Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables, New York: Dover Publications Inc., 1972.

[17] Bateman, H., Erdélyi, A., Higher Transcendental Functions. Volume 3. New-York – Toronto – London: Mc Graw-Hill Book Company, Inc., 1955.

[18] Kosenko, I. I., Loginova, M. S., Obraztsov, Ya. P. and Stavrovskaya, M. S., Multibody Systems Dynamics: Modelica Implementation and Bond Graph Representation. In: Kral, Ch. and Haumer, A., (Eds.) Proceedings of the 5th International Modelica Conference, Vienna, Austria, September 4–5, 2006, Vienna: The Modelica Association, and arsenal research, 2006, pp. 213–223.

[19] Leine, R. I. and Glocker, Ch., A Set-Valued Force Law for Spatial Coulomb–Contensou Friction. Europian Journal of Mechanics A/Solids, 2003, Vol. 22, No. 2, pp. 193–216.

[20] Friedl, C., Der Stehaufkreisel. Master's thesis. Augsburg: Institut für Physik, Universität Augsburg, 1997.

[21] `http://www.zfm.ethz.ch/~leine/publications.htm`

[22] `http://www.sciencedirect.com/science/journal/00218928`

Evaluation of Different Compressor Control Concepts for a Swash Plate Compressor

Norbert Stulgies[⋈] Manuel Gräber[⋈] Wilhelm Tegethoff[⋈] Sven Försterling[◇]
[⋈] Technical University Braunschweig, Institut für Thermodynamik (IfT)
38106 Braunschweig - Germany
[◇] TLK Thermo GmbH, 38106 Braunschweig - Germany
n.stulgies@tu-bs.de

Abstract

Due to the development of high efficiency R744 air conditioning systems, the main aim of this paper is the investigation of a control concept for swash plate compressors.

This paper presents three different control concepts for a swash plate compressor using a built-in control valve. Therefore a model for a one-phase R744 expansion valve was developed and adapted to detailed measurement data. To achieve high reliability in the simulation, the entire R744 refrigerant cycle was validated using analyses of measurement data from an IfT test bench. The simulation of the refrigeration cycle components was realised using TIL (**T**LK-**I**fT-**L**ibrary). The main focus was set as the description of the compressor with its internal and external mass flow rates. The internal mass flow, which is directed through the crankcase, directly affects the crankcase pressure. It is also called the control mass flow. As a result of the crankcase pressure an adjustable mechanism regulates the displacement as shown in figure 1. The greater the displacement, the greater is the inclination angle. This is caused by different load incidence points on the swash plate e.g. by springs, pistons and pressure states. A comparison of different control concepts shows the characteristic and control behaviour of each of them relating to control time and control mass flow rate. Whenever a control mass flow occurs, it implicates throttle losses. The dissipated energy can be minimised using another control concept.

Keywords: CO2; compressor; control; R744; refrigeration; simulation; valve

1 Introduction

In case of different motor speeds, the refrigerant compressor needs to adjust the displacement in order to control the cooling capacity. The adjustment is reached by inclining the swash plate by an angle α as seen in figure 1. The inclination itself is a function of the balance of forces. Besides the friction, inertia and spring forces, the pressure difference between the crankcase and the cylinder capacity has the greatest influence. Thus the target is to control the aforementioned pressure difference by controlling the crankcase pressure. The control mechanism of a swash plate compressor separates a part of the cooling mass flow of the refrigerant cycle at the discharge chamber and leads it through a valve to the crankcase. Another valve seated between the crankcase and suction chamber completes the control path of the compressor. Depending on the throttle devices used in the two throttle locations, different control strategies can be realised. Throttle devices can be either simple orifices or proportional valves.

2 TIL

TIL is a component model library for thermodynamic systems that was developed by the Institute for Thermodynamics (IfT) and TLK-Thermo-GmbH, and allows steady-state and transient simulation of thermodynamic systems. The library provides miscellaneous models of thermal and fluid technology components, as well as transport phenomena. Media properties can be included by using the TILMedia library or Refprop media data. The structure of TIL is an independent library-modelling concept, which due to its shallow inheritance structure, permits designers

DOI: 10.3384/ecp09430035

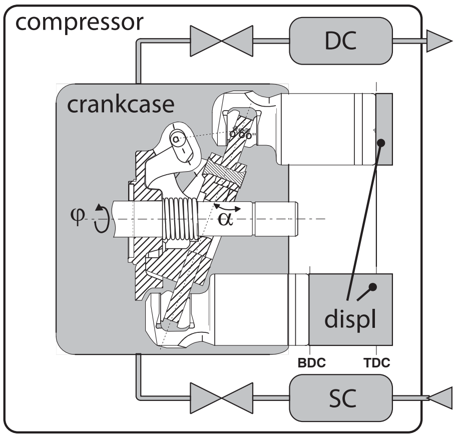


Figure 1: Swash plate mechanism with driven shaft, pistons, suction chamber, discharge chamber and displacement

as well as simulation specialists to achieve fast, suitable models.

3 Cycle Layout

In order to model a cycle, some components used from TIL, and some were made for the specific problem. In the following section, the components are described in more detail. All components can run in both flow directions. More specific information about TIL is in Richter2008 [Ric08]. The complete cycle is shown in figure 2. The implemented models for the external heat exchangers use a cross flow scheme. To create a heat exchanger having more than one path it is possible to use the required number of serial connected models. In the investigated cycle, a gas cooler with two paths was used. The discretisation level on the refrigerant side was ten cells per path. The internal heat exchanger (IHX), was realised by using a tube in tube construction with discretisation level 10, and connecting the heat port of every cell in one tube with another heat port from the second tube in such a manner that a counter flow heat exchanger is realised. The flow through the expansion valve can be specified using the Bernoulli equation. The accumulator offers the possibility to store refrigerant at different operating points. The compressor model describes a variable displacement compressor which can be characterised by an efficiency based approach. The used efficiencies are the volumetric efficiency, isentropic efficiency and quality grade. The efficiencies are accessible in a compressor characteristic plot. The calculation of the displacement is an iterative process of pressure states in the crankcase, suction chamber and discharge chamber. In the model considered, perturbances are caused by internal leakage that means the blowing by of gas through the piston clearance, which shows ordinary values of 0.0004 in/in of diameter for steel pistons up to 0.002 in/in of diameter for aluminium pistons without piston rings [AAD00]. Although these flows are very small compared with the cooling mass flow rate, the influence on the control mass flows is not negligible. A characteristic property is the non-continuous behaviour of these flows.

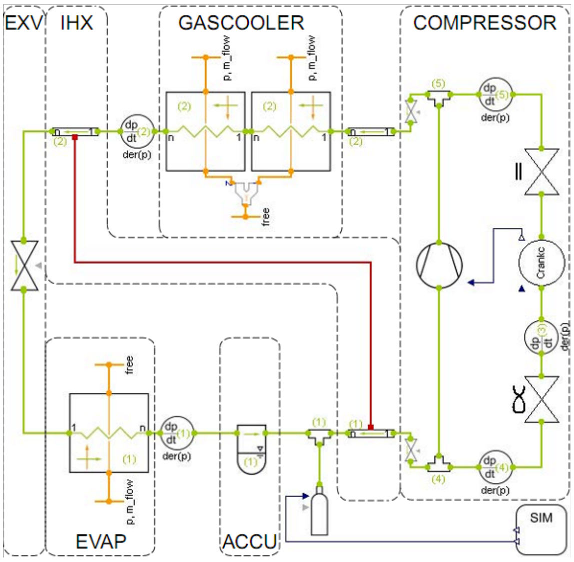


Figure 2: Schematic diagram of the Modelica model of concept A

4 Compressor Submodels

The simulation of the thermodynamical and mechanical behaviour of a mobile air conditioning compressor is computationally very intensive. The first reason for this is the complicated equations of motion of the swash plate mechanism. The second and far more significant reason for this condition is the media data, which have to be calculated in every of the several (in this case 7) displacement chambers with a very small time constant. The complexness of all is the strong causal dependence of these two effects. The interdepence of the pistons was in this model neglected. An complete consideration can be find in [Cav08].To simplify the model, detailed calculations of the swash plate mechanism behaviour were done beforehand. The analysis of the calculation leads to simple trigonometric equations which describe the mechanism of the compressor. There are four main effects which have an impact on the mechanism and its moment balance system. At any time the moment balance of the swash plate has to be zero.

- Adjusting springs
  The moment due to the springs, with one displacement increasing and another decreasing, is the sum of the spring forces multiplied by a constant lever. The function has the following appearance. $M_{springs} = f(\alpha)$

- Deviation moment of the swash plate
  The swash plate is fixed to the driving shaft by a swing bearing which allows inclination by an angle α. Depending on the inclination angle, the moment which forces the swash plate to a position perpendicular to the driving shaft increases with an increasing speed. The expression which is described as well in Valeo [Val] is shown here:

$$M_{dev} = \frac{m_{SWP}}{12}\ cos(\alpha)\ sin(\alpha)\ (3r_a^2 + 3r_i^2 - h^2)$$

- Piston inertia
  The piston inertia contributes a moment to the swash plate that increases the angle of the swash plate and hence the displacement of the compressor. The observation of the moment caused by a single piston shows a periodic moment to the swash plate related to one turn. The moment resulting from all the pistons together shows an almost constant moment to the swash plate. The only influencing variables are the compressor speed and the angle of the swash plate, so that the moment due to inertia is a function like this: $M_{piston\ inertia} = f(\alpha(t), \dot\alpha(t), \ddot\alpha(t), \varphi(t), \dot\varphi(t), \ddot\varphi(t))$.
  Assuming $\ddot\varphi(t)$ is negligible leads to the following expression where M is a linear function of the swash plate angle and has a quadratic dependency on the speed:

$$M_{p\ inert} = \sum_i m_i \cdot \ddot{x}(\alpha(t), \varphi(t)) \cdot r_i(\varphi)_i$$

  A chosen piston mass of 55g including the piston slide shoes results in the implemented expression in Modelica:

$$\begin{aligned} M_{inertia} =& (c_0\ \alpha + c_1)\ n^2 + (c_2\ \alpha + c_3)\ n\ + \\ & (c_4\ \alpha + c_5) \\ & where\ c_0 = -1.01\mathrm{e} - 9 \\ & \quad c_1 = 5.68e - 11 \\ & \quad c_2 = 1.01e - 17 \\ & \quad c_3 = -5.32E - 1 \\ & \quad c_4 = -7.67e - 15 \\ & \quad c_5 = 4.76e - 16 \end{aligned}$$

- Pressure loads
  The pressure loads consider all forces or moments caused by the pressure differencees over the pistons. These pressure differences are given by the pressure in the crankcase and the currently obtaining pressure in the displacement chamber. In this model an expression was derived, which describes the torque to the swash plate without calculating the state variables in the displacement chamber. The thus found equation depends only on the quite slowly changing state variables of the cooling circuit or the much more idle volume of the crankcase.

$$\begin{aligned} M_{dp} =& ((-0.5742\ \pi + 0.3449)p_s + \\ & (arctan((\pi + k_0)k_1)k_2 + k_3)) + \\ & 3.82536\ (p_{crankcase} - p_s) \\ & where \\ & k_0 = 0.0045\ \alpha^2 - 0.1372\ \alpha - 2.2008 \\ & k_1 = 25 \\ & k_2 = -0.0195\ \alpha^2 + 0.5915\ \alpha + 0.606 \\ & k_3 = -0.0316\ \alpha^2 + 0.9524\ \alpha + 1.471 \end{aligned} \tag{1}$$

5 Control Concepts

In figure 3, three different control concepts are shown. Concept A presents the combination of an orifice at the high pressure side, i.e. between the discharge chamber and crankcase, and a proportional valve at low pressure side, which is situated between the crankcase and suction chamber. Concept B shows the opposite assembly. In the third example, concept C, the application of two proportional valves is shown.

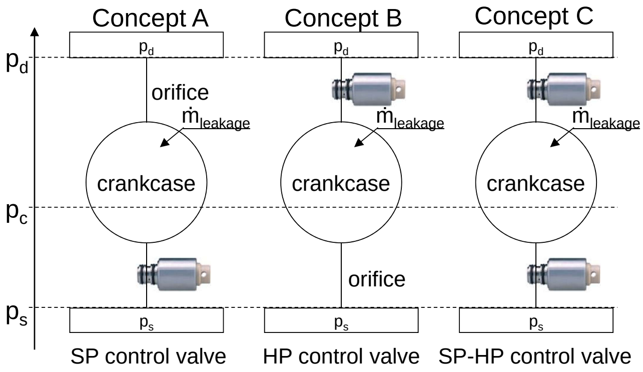


Figure 3: Visualisation of the three control concepts [NS08]

It is quite simple to come to the conclusion, that concept C has to be the best one. The explanation for that is that concept A and B have a continuous control mass flow rate of in steady state conditions, whereas concept C shows no control mass flow rate for this case. To simulate more realistic conditions, perturbances such as blowing by flows at the pistons are considered. These are shown in figure 3 as leakage mass flow rates.

A comparison of the three different control concepts was carried out using an identical maximum mass flow rate through the control valves. For concept A and B, simulation results are shown in figure 4 and figure 5.

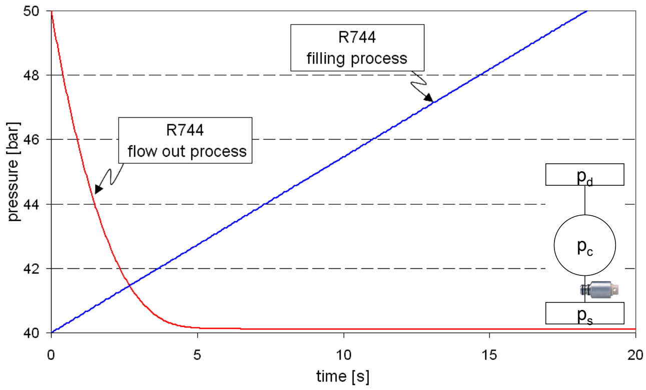


Figure 4: Filling process (proportional valve is closed) and flow out with open valve

In figure 6 the sequence flow for the control of the evaporator air outlet temperature is shown. The above written equations are all included in the compressor. More exactly in the adjusting process of the swash plate.

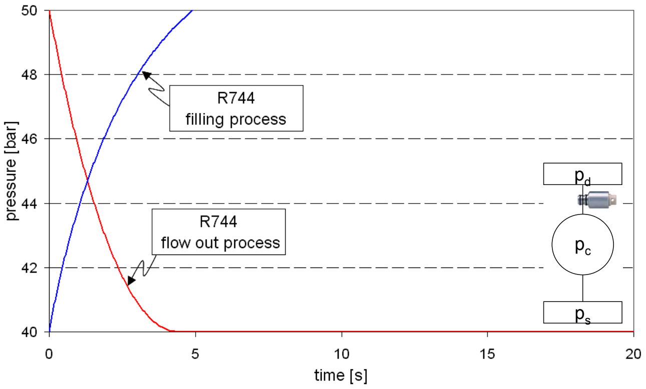


Figure 5: Filling process (proportional valve is open) and flow out with closed valve

6 Transient Simulation of the System

In the following, results of the transient simulation of the above mentioned CO_2-system are presented. The transient data of the CO_2-cycle was measured by using the input data of speed and air velocities from CADC (Commen Artemis Driving Cycle) [BM07]. The CADC shows much more transient behaviour than the commen NEDC, and thus the influence of the different control concepts show different results in control behaviour.

In figure 7 the simulation results of the evaporator air outlet temperature controlled CAD-Cycle are shown. The controller of the outlet temperature was realised using an integrator controller. The necessary control mass flow related to the effective cooling mass flow shows effective cooling losses of about 6.1%.

In figure 8 the simulation results of the compressor sensitivity are shown. The proportional control valve of the compressor (concept B) was regulated by current. The first simulation shows the response characteristic of the compressor in idle mode and a constant control current. The second simulation shows the influence of a sine-shaped varying control mass flow. A modification of the control mass flow shows a gain in influence of the swash plate angle. The variation of the mass flow by approximately 1% leads to responses of the relative displacement up to 10%.

7 Conclusion

In this paper a refrigeration cycle was modeled and validated by measurement data. The focus

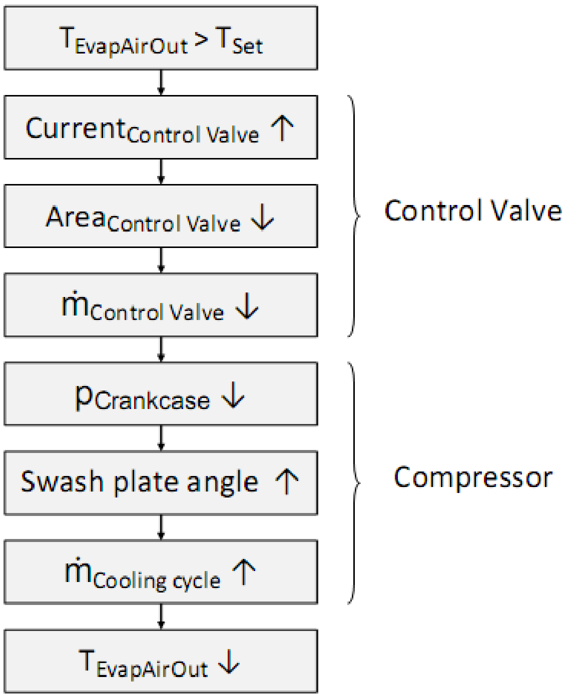

$T_{EvapAirOut} > T_{Set}$
$Current_{Control\ Valve}$ ↑
$Area_{Control\ Valve}$ ↓
$\dot{m}_{Control\ Valve}$ ↓
$p_{Crankcase}$ ↓
Swash plate angle ↑
$\dot{m}_{Cooling\ cycle}$ ↑
$T_{EvapAirOut}$ ↓
Control Valve
Compressor

Figure 6: Sequence flow of an evaporator air outlet temperature controlled cooling cycle. The control Valve is an normally opened valve mounted in between the discharge chamber and the crankcase (concept B)

was set to the compressor especially to the adjusting mechanism in the crankcase. For that, equations for the momentum balance of the swash plate were derived. With this model the presented concepts can be compared related to efficiency and comfort.

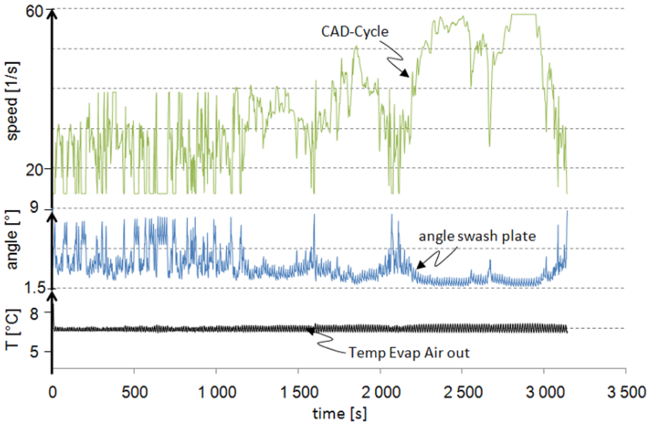


Figure 7: Simulation results of Concept B. The green shape shows the speed of the compressor the blue one the resulting swash plate angle. The black curve shows the simple controlled temperature outlet response.

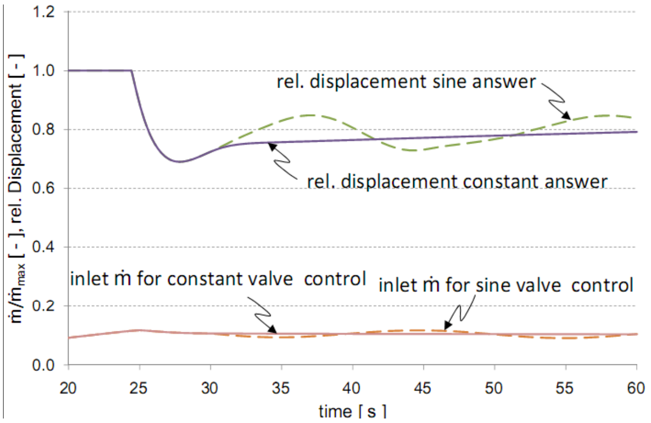


Figure 8: Identifying the swash plate angle response due to a control mass flow variation

References

[AAD00] Albert A. Domingorena, Dean H. Rizzo, John H. Roberts Rudy Stegmann Jaroslav Wurm: *The 2000 ASHRAE Handbook*, 34 - Compressors, 34.1 – 34.36. ASHRAE, 2000.

[BM07] Boulter, Paul Ian McCrae: *Assessment and reliability of transport emission models and inventory systems.* , TRL Limited, October 2007.

[Cav08] Cavalcante, Peterson: *Instationäre Modellierung und Sensitivitätsanalyse regelbarer CO2-Axialkolbenverdichter.* , 2008.

[NS08] Norbert Stulgies, Axel Müller, Horst Kappler Wilhelm Tegethoff Sven Försterling Jürgen Köhler: *Proposal for efficient characterization of (R)744 compressor control valves.* 2008.

[Ric08] Richter, Christoph: *Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic Systems.* , 2008.

[Val] Valeo: *Verfahren zum Regeln des Kältemittel-Massenstroms eines Verdichters.*

Investigation of Energy Dissipation in an Ejector Refrigeration Cycle

Christian Tischendorf [1] Denise Janotte [2] Ricardo Fiorenzano [1] Wilhelm Tegethoff [2]
[1] Technical University Braunschweig, Department of Thermodynamics
[2] TLK Thermo GmbH
Hans-Sommer-Straße 5, 38106 Braunschweig Germany
c.tischendorf@tu-bs.de

Abstract

The presented work focuses on the differences in energy dissipation in each cycle component compared to the energy dissipation of the whole ejector refrigeration cycle. With help of this analysis, improvement of energetic efficiency by using an ejector can be set in relation to the potential improvement in efficiency of other components such as heat exchangers. Information about entropy production associated with energy dissipation allows for an objective estimation of the optimization potential of each component within an ejector refrigeration cycle. In addition, the improvement due to the specific process control of the ejector cycle compared to the conventional heat pump cycle can be analyzed. The energetic benefit gained using an ejector depends on the refrigerant used. The refrigerants R134a and R744 (CO_2) were compared in regard to the entropy production of the heat pump system.

In order to simulate an ejector refrigerant cycle and to evaluate the energy dissipation by means of entropy production, existing models for cycle components were modified. Applying the second law of thermodynamics, local distribution of entropy production as well as the overall entropy produced in each component was determined. The analysis showed that entropy production is caused by two types of effects. One part results from real effects such as pressure drop and heat transfer, the other part is due to the modeling assumptions made. Thus, the investigation of energy dissipation leads to a deeper understanding of the model.

The simulated amount of entropy produced is summarized in a record, so that the results can be read easily by other programs, e.g. programs that visualize energy and entropy flows. In the presented investigation the entropy flow and dissipation effects were analyzed by means of diagrams, such as Sankey diagrams.

The complete heat pump system has been simulated using the Modelica library TIL (TLK-IfT-Library) in order to determine the energy dissipation in each cycle component. With the modified TIL models, other process controls can also be investigated. This approach offers the opportunity to analyze the energy dissipation in detail, and differs in that sense from the commonly used technique of integrated energy balances and COP determinations.

Keywords: entropy analysis; refrigeration; compression cycle; simulation; CO2; R134a

1 Introduction

The pressure difference between the high and low pressures in CO_2 refrigerant systems is high compared to other refrigerants, e.g. R134a. Previous investigations by other authors have shown that in a conventional refrigeration cycle, the pressure differences cause significant throttling losses. Using an expansion valve results in an isenthalpic throttling process, which means that the kinetic energy is completely dissipated and the evaporation enthalpy is reduced compared to an isentropic process.

Using an ejector is one way to recover part of the lost kinetic energy and to increase refrigeration capacity. As a result, the energetic efficiency (COP) of the refrigerant system will be improved. In addition to the improvement caused by an ejector, the COP can be raised by optimization of other cycle components. The key issues are the comparative cost effectiveness of the modifications, as well as the question of which components have the most optimization potential. Therefore, an analytical technique enabling one to recognize optimization potential is needed in order to assess the alternative solutions. The required technique was developed during the project presented in this work.

One approach to investigate the potential is to ap-

DOI: 10.3384/ecp09430090

ply the second law of thermodynamics to determine the produced entropy in each refrigeration cycle component, so that a basic analysis of the component efficiencies is possible. This approach was introduced in [Franke04]. In the presented work, the investigation was carried out using the Modelica library TIL to simulate the refrigeration cycles. TIL is a component library for steady-state and transient simulation of fluid systems such as heat pump, air conditioning, refrigeration or cooling systems, developed by TLK-Thermo GmbH and TU Braunschweig, Institute for Thermodynamics [for a detailed description see [Richter08]]. The advantage of TIL is that it has a very shallow inheritance structure, which makes the models easy to understand and extend. The results of the entropy analysis were visualized by bar and Sankey charts using the newly developed software EnergyViewer by TLK-Thermo GmbH, in order to simplify the interpretation of the effects.

2 Simulated Refrigeration Cycle

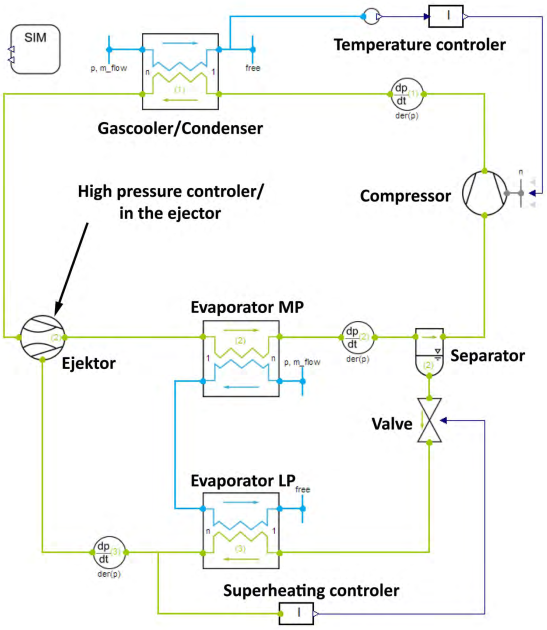


Figure 1: Object diagram of the simulated cycle

In the presented work, an ejector heat pump used for heating water for domestic use and floor heating using R744 or R134a as refrigerant was simulated. The principal functionality of a common ejector refrigerant cycle is described in the following literature [Elbel06]. For this investigation, the common ejector refrigeration cycle was modified. An object diagram of the modified ejector refrigeration cycle is shown in figure 1. The cycle consists of the following components: gas cooler (R744)/ condenser (R134a), medium pressure evaporator (MP), low pressure evaporator (LP), valve, separator, compressor and ejector. In addition, a temperature controller and a super-heating controller were added in the cycle. As well, a high pressure controller was added to the ejector component model. The heat pumps were simulated at the following conditions.

heating capacity for both operation modes:
5000 W
temperature floor heating water:
30 °C to 35 °C
temperature domestic hot water:
10 °C to 60 °C
overall heat transfer capability gas cooler/condenser:
1400 W/K
overall heat transfer capability evaporator LP/MP:
500 W/K
heat exchanger pressure drop R744 refrigerant:
1 bar
heat exchanger pressure drop R134a refrigerant:
0,2 bar
water mass flow rate evaporator
equal for all simulations
water temperature of heat source
10 °C
high pressure
set to optimize the conditions in the gas cooler/condenser (low mean driving temperature difference)

The water that serves as a heat source first flows through the evaporator MP and afterward through the evaporator LP. The temperature controller controls the speed of the compressor, such that the desired output temperature is achieved by a constant water mass flow rate through the gas cooler or condenser. The controller integrated in the ejector controls the high pressure level by setting the mass flow rate through the driving nozzle. The controller determines a high pressure level, which can be adjusted such that optimal conditions are found in the gas cooler or condenser. The super-heating controller is used to create optimal conditions in the evaporator LP.

3 Effects Causing Entropy Production in the Cycle Components

An important distinction must be made between entropy production caused by numerical errors based on modeling assumptions and the entropy produced due to real physical effects. The numerical errors depend on the mathematical model as well as on the degree of discretization when using a discretized model. The real physical effects depend on the quality and the construction of the refrigeration cycle components and can be influenced by the specific process control. In addition to this, the produced entropy is dependent on the refrigerant used. In the presented work the following entropy producing effects generated in the cycle components were investigated:

- **Heat Exchanger**
  pressure drop, heat transfer and numerical errors (modeling)
- **Valve**
  pressure drop (isenthalpic expansion)
- **Compressor**
  efficiency based model
- **Ejector**
  special efficiency based model
- **Separator**
  mixing effects

3.1 Heat Exchanger

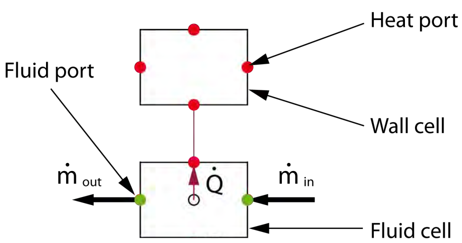


Figure 2: Illustration of cell structure of a tube

The heat exchangers used for the investigation consist of tubes connected via heat ports. The number of tubes and the direction in which the medium flows through them can vary. The manner of connection specifies the kind of heat exchanger. For this paper, counterflow heat exchangers consisting of one liquid and one refrigerant tube were used. The medium used in the liquid tube was water. The tubes are divided into cells, and each tube is comprised of two types of cells: wall cells and fluid cells (see figure 2). The fluid cells can be either liquid or refrigerant cells, depending on the type of medium flowing through them. The connection of fluid and wall cells via heat ports allows the exchange of heat between the cells. The temperature of the connected heat ports of two cells are set equal. The heat transfer inside the cell between the medium and the heat port is determined by the heat transfer relation and the heat transfer coefficient. In the cell model, equations to determine the heat flow are implemented. A similar modeling of heat transfer and fluid flow is presented in [Patankar80]. The number of wall and fluid cells in each tube determines the degree of discretization (finite volume approach). An introduction to the finite volume approach can be found in [Baumann06]. In order to model the entropy production within the heat exchangers, a hierarchical approach was followed. First the entropy production due to the aforementioned effects was determined for each cell of the heat exchanger. The second law of thermodynamics for a transient system yields:

$$\frac{d(sm)}{dt} = \sum_{i=1}^{n} \dot{m}_i s_i + \dot{S}_Q + \dot{S}^{Q}_{prod} + \dot{S}^{\Delta p}_{prod} + \dot{S}^{M}_{prod} \qquad (1)$$

With $\frac{d(sm)}{dt}$ being the change of entropy of the cell. The terms $\sum_{i=1}^{n} \dot{m}_i s_i$ and $\dot{S}_Q$ specify the entropy conveyed by mass and heat flows. $\dot{S}^{Q}_{prod}$, $\dot{S}^{\Delta p}_{prod}$, $\dot{S}^{M}_{prod}$ represent the entropy production rate due to heat transfer, pressure drop and modeling respectively. Heat transfer and pressure drop are real effects that cause entropy production, which can be determined by formulas presented later. The last production term, however, is due to the modeling of the cell as a volume with constant medium properties such as enthalpy or temperature. For a **fluid cell** this can be illustrated by the image of an agitator stirring the medium inside the cell so that it is perfectly mixed. Because of the modeling assumptions, the enthalpy of the medium inside the cell matches the enthalpy at the outlet port of the cell. Likewise, the temperature of the medium inside the cell matches the temperature at the outlet port of the cell. If there is a temperature change, this mixing of the cell content produces entropy. This entropy production $\dot{S}^{M}_{prod}$ is numerical error and differs from $\dot{S}^{Mi,T}_{prod}$ the entropy production due to mixing of two streams of ideal gas $\dot{m}_a$ and $\dot{m}_b$ with temperature T_a and T_b respectively which is presented in [Cerbe07] as follows:

$$\dot{S}^{Mi,T}_{prod} = \dot{m}_a c_{ma}|_{t_a}^{t_{Mi}} ln\frac{T_{Mi}}{T_a} + \dot{m}_b c_{mb}|_{t_b}^{t_{Mi}} ln\frac{T_{Mi}}{T_b} \qquad (2)$$

with T_{Mi} being the mixing temperature and c_m the mean specific heat capacity of stream a or b. Hence $\dot{S}^{M}_{prod}$ has to be determined via the second law. Transforming equation 1 and setting $\frac{dm}{dt} = \sum_{i=1}^{n} \dot{m}_i$ yields:

$$\dot{S}^{M}_{prod} = m\frac{ds}{dt} - \sum_{i=1}^{n} \dot{m}_i(s_i - s) - \dot{S}_Q - \dot{S}^{Q}_{prod} - \dot{S}^{\Delta p}_{prod} \quad (3)$$

The index i labels the variables of stream i flowing in or out of the cell. The variables of the medium inside the cell do not have index labels. Each fluid cell has only one inlet $\dot{m}_{in}$ and one outlet stream $\dot{m}_{out}$. The modeling assumption yields $s = s_{out}$ so that the equation can be simplified further.

$$\dot{S}^{M}_{prod} = m\frac{ds}{dt} - \dot{m}_{in}(s_{in} - s) - \dot{S}_Q - \dot{S}^{Q}_{prod} - \dot{S}^{\Delta p}_{prod} \quad (4)$$

Without heat transfer and pressure drop, there can still be entropy produced by continuously mixing the fluid inside the cell. In the case that the specific entropy at inlet and inside the cell are not equal, the last term in the equation does not reduce to zero. This occurs if the temperatures are not equal because the specific entropy depends on pressure and temperature.

$$\dot{S}^{M}_{prod} = m\frac{ds}{dt} - \dot{m}_{in}(s_{in} - s) \quad (5)$$

Each fluid cell emits a heat flow $\dot{Q}$ at the heat port with the temperature T_{hp}. This flow conveys entropy that can be determined by the following equation.

$$\dot{S}_Q = \frac{\dot{Q}}{T_{hp}} \quad (6)$$

Before being emitted, the heat flow is transferred from the medium inside the cell (temperature T) to the heat port (temperature T_{hp}). The temperature gradient between T_{hp} and T is determined by the heat transfer relation and the heat transfer coefficient. In the cell model, the equations for the heat transfer phenomena are implemented. This heat transfer causes entropy production, which can be determined according to [Bejan88] as follows.

$$\dot{S}^{Q}_{prod} = \frac{\dot{Q}}{T} - \frac{\dot{Q}}{T_{hp}} \quad (7)$$

Pressure drop between the inlet and outlet port also leads to entropy production. It can be determined as presented in [Bejan88].

$$\dot{S}^{\Delta p}_{prod} = \dot{m}\int_{out}^{in} \frac{v}{T}_{h=const} dp \quad (8)$$

This formula can be linearized

$$\dot{S}^{\Delta p}_{prod} = \dot{m}\frac{v}{T}\Delta p \quad (9)$$

Δp represents the pressure drop.

Equation 1 has to be adapted for the determination of the entropy production inside a **wall cell**. Since there is no medium flowing through the wall cell, there are no terms for entropy transportation via mass flow, and no pressure drop occurs. The entropy production due to the modeling of the wall cell can be determined according to the following equation:

$$\dot{S}^{M}_{prod} = m\frac{ds}{dt} - \dot{S}_Q - \dot{S}^{Q}_{prod} \quad (10)$$

Heat can be emitted at each of the heat ports i with the overall entropy conveyed because of heat flow being $\dot{S}_Q = \sum_{i=1}^{n} \dot{S}_{Q,i}$. Each of the summands $\dot{S}_{Q,i}$ can be determined according to equation 6. The heat transfer from the medium inside the cell to one heat port or vice versa causes entropy production. Each production term $\dot{S}^{Q}_{prod,i}$ can be determined in accordance with equation 7 and has to be summed in order to determine the overall entropy production due to heat transfer, $\dot{S}^{Q}_{prod} = \sum_{i=1}^{n} \dot{S}^{Q}_{prod,i}$.

The entropy production resulting from modeling assumptions $\dot{S}^{M}_{prod}$ (see equation 10) represents only small numerical errors. There is no mixing inside the wall cell and the term $\dot{S}^{M}_{prod}$ can be neglected.

In order to calculate the entropy production due to each effect for the whole tube, the results of the entropy production in the cells are summed. The results of the refrigerant and the liquid tube are then summarized to determine the entropy produced within the whole heat exchanger.

3.2 Valve

In the valve model (control valve) the refrigerant is throttled adiabatically. It is assumed that the kinetic energy of the flowing refrigerant is completely dissipated. Since the valve has been modeled as a component without volume, the second law for the valve has be applied in the steady state form.

$$\sum_{i=1}^{n} \dot{m}_i s_i + \dot{S}^{T}_{prod} = 0 \quad (11)$$

As the mass flow rate $\dot{m}_i$ and the specific entropy s_i at the inlet and outlet ports are known, this equation is used to determine the overall produced entropy.

$$\dot{S}^{T}_{prod} = -\sum_{i=1}^{n} \dot{m}_i s_i \tag{12}$$

3.3 Compressor

The compressor model is based on an efficiency model with isentropic, volumetric and effective isentropic efficiency. Furthermore, a heat flow from the housing surface to the surroundings is considered, but is set to zero in the simulation. The main entropy production in the compressor is caused by the following loss mechanisms, which result from the compression process of a reciprocating type compressor. These are: friction losses due to mechanical components such as pistons and piston rings and swash plate, throttling losses that take place in the valves, flow channels and chambers, the heat transfer between the different compressor components, e. g. the suction and discharge chamber, the leakage losses and losses caused by the control valve. Since the compressor is modeled as a volumeless component, the second law for the compressor has to be applied in steady state form:

$$\sum_{i=1}^{n} \dot{m}_i s_i + \dot{S}_Q + \dot{S}^{T}_{prod} = 0 \tag{13}$$

$\dot{S}_Q$ specifies the entropy flow that is discharged with the heat flow.

3.4 Ejector

When a medium is throttled in a conventional valve, friction losses occur and the kinetic energy of the medium is completely dissipated. In an ejector, however, the kinetic energy of a primary flow at high pressure can be partly used to compress a secondary flow and thus to diminish the compression work done in the compressor. Figure 3 illustrates the functionality of an ejector.

The driving flow exits the driving nozzle with a high velocity and carries with it a flow from the suction nozzle. Since the cross section of the suction nozzle becomes progressively narrower, the suction flow is accelerated and the pressure drops. Both the suction and driving flows exchange momentum, are mixed in the mixing tube and flow through a diffuser before leaving the ejector. The pressure inside the diffuser increases with decreasing velocity.

The ejector is modeled using an analogous model consisting of several separate components. Because the exchange of momentum is not ideal and entropy will be produced, the ejector can partly recover the kinetic energy. However, the entropy produced has not been determined in detail for this work. A simplified approach has been used instead, and the entropy production rate for steady state has been determined as the difference between inlet and outlet entropy flow rates according to the second law in steady state.

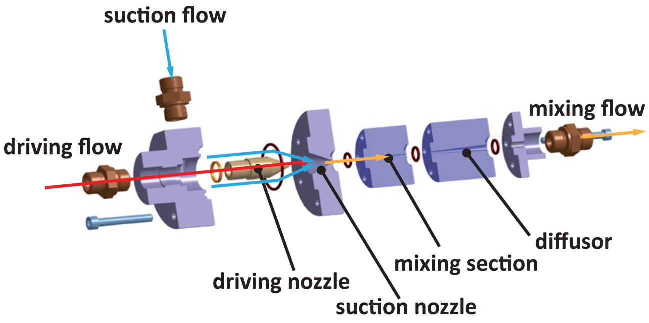


Figure 3: Exploded view of an ejector

$$\dot{S}^{T}_{prod} = -\sum_{i=1}^{n} \dot{m}_i s_i \tag{14}$$

3.5 Separator

The separator model is similar to the refrigerant cell model apart from the fact that no heat transfer or pressure loss is considered, and that the refrigerant is not mixed, but separated into liquid and gaseous phase. The entropy production rate is determined by the following equation.

$$\dot{S}^{T}_{prod} = m\frac{ds}{dt} - \sum_{i=1}^{n} \dot{m}_i s_i \tag{15}$$

As there are no effects producing entropy $\dot{S}^{T}_{prod}$ is equal to zero.

3.6 Implementation of Entropy Production in TIL

In the Modelica library TIL, the specific entropy s is implemented as a function of p and h. In order to reduce the index of the differential algebraic equation system, the terms in the entropy equation have to be expressed by means of state variables used for the description of the system. This implies that the term $\frac{ds}{dt}$ (derivative of the specific entropy with respect to time) had to be rewritten in terms of p and h. To transform the equation Bridgeman tables, which can be found in [Bejan88] were used. This made the implementation more numerically effective by not relying on Dymola for the rearrangement of the equation system. The discussed mathematical equations to determine the entropy production were implemented in each model. The collection of the results in the summary records simplifies the readout by other programs used to analyze the results.

4 Visually Supported Analysis of Simulation Results

The simulation results were analyzed with the help of 2d-plots which show the temperature relations, and Sankey diagrams which show the entropy flow beside the production rates. Therefor the entropy flows at the inlet and outlet port of each component were determined. The entropy production caused by heat transfer, pressure drop, numerical errors due to modeling and the overall entropy production were calculated according to the aforementioned equations and illustrated by bar charts. The visualized entropy flows were normalized for each medium such that the flow with the lowest specific entropy is equal to zero. With the help of the Sankey diagrams, the entropy shift within the flows is visualized, and for each medium the relation of the entropy flows between the components becomes clear. Sankey diagrams are a powerful visualization method to analyze all kind of flows. A detailed discussion of the application of Sankey diagrams can be found in [Schmidt06].

4.1 Entropy Production in Heat Exchangers

The diagrams in figures 4 and 5 illustrate the temperature curves in the gas cooler (R744) and the condenser (R134a) of a steady-state ejector refrigeration cycle for domestic hot water. It can be clearly seen that the temperature glide in the gas cooler (R744) leads to a lower mean driving temperature difference in the gas cooler than in the condenser (R134a). In the evaporators, the refrigerant is evaporated at a nearly constant temperature and exits the heat exchangers with low superheating, which is controlled by the controller. This results in a lower mean driving temperature difference and lower entropy production due to heat transfer in the evaporators. See figure 7 and 9 for entropy production in the evaporators in relation to entropy production in the gas cooler / condenser. In figure 6 the profile of entropy production due to different effects is illustrated for the cells of the gas cooler. The corresponding temperature profile is shown in figure 5. Wherever high temperature differences occur, the produced entropy due to heat transfer and mixing (modeling assumption) is high. The entropy production rate due to pressure drop is low in comparison to the production rate due to heat transfer and mixing.

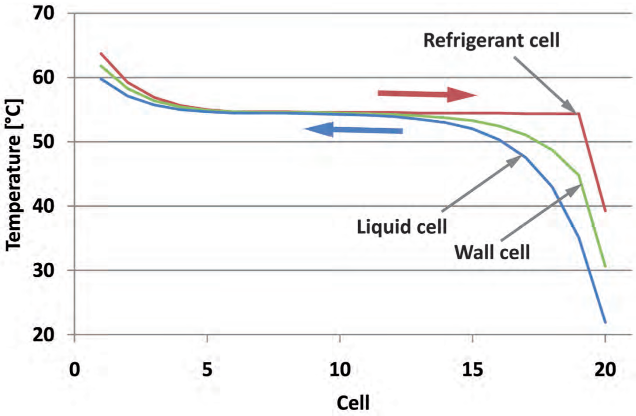


Figure 4: Temperature profile of refrigerant, liquid and wall cells inside the condenser (R134a domestic hot water)

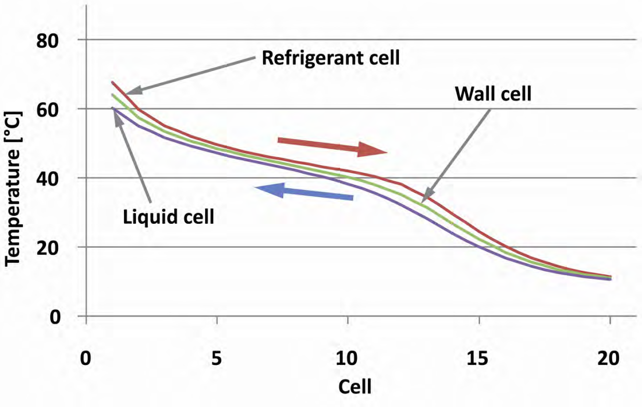


Figure 5: Temperature profile of refrigerant, liquid and wall cells inside the gas cooler (R744 domestic hot water)

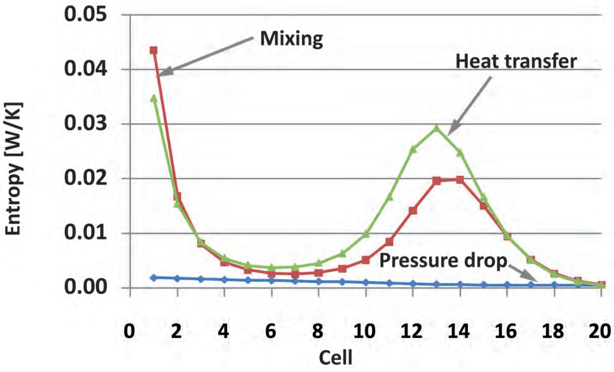


Figure 6: Profile of entropy production inside the gascooler (R744 domestic hot water)

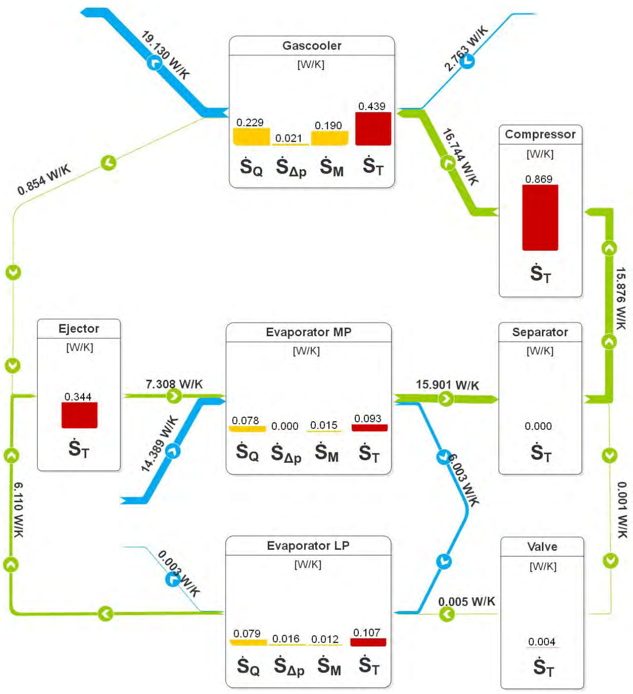


Figure 7: Entropy flow and production rate in an R744 heat pump cycle used to heat up hot domestic water. Entropy production rate due to heat transfer ($\dot{S}_Q$), pressure drop ($\dot{S}_{\Delta p}$), modeling ($\dot{S}_M$) and total entropy production rate ($\dot{S}_T$) for each component are represented by bar charts. The entropy flow is normalized to the lowest specific entropy of each medium. Entropy production and flows are represented in different scales.

4.2 Comparison R744 ejector cycle for domestic hot water and floor heating

The figures 7 and 8 show the Sankey diagrams of a steady state R744 ejector refrigeration cycle for domestic hot water and floor heating operation respectively. First, looking at the gas cooler, it is shown that the entropy production caused by heat transfer in the case of floor heating operation is higher than in the case of domestic hot water operation. This is due to the higher driving temperature difference in the gas cooler in the floor heating operation. However, the entropy production caused by mixing (modeling assumption) in the case of floor heating operation is lower than in the case of domestic hot water operation, despite the greater refrigerant and water mass flow rates. The reason for this is the lower temperature gradients between the inlet and outlet of the refrigerant as well as liquid tubes. With the heating capacities equal in both the floor heating and domestic hot water operation modes, in the floor heating mode, the compressor and the ejector produce more entropy, although the water mass flow rate and the refrigerant mass flow rate are greater. That shows that the R744 ejector refrigeration cycle is better suited to heat water up to higher temperatures.

4.3 Comparison R744 and R134a Ejector Cycle for Domestic Hot Water

The Sankey diagrams of a steady state R744 and R134a ejector refrigeration cycle for domestic hot water mode are illustrated in figure 7 and 9. The results show that entropy production caused by heat transfer in the condenser (R134a) is significantly higher than in the gas cooler (R744). This is due to the higher mean driving temperature difference between the refrigerant and the water in the condenser. The temperature curves in the condenser and the gas cooler are illustrated in figure 4 and 5. Because of the temperature glide in the gas cooler, the mean driving temperature difference is lower. For the same reason, the entropy production resulting from mixing (modeling assumption) is also higher in the condenser. In the R744 refrigeration cycle, production of entropy in the compressor and ejector is higher, the reason for this being the different refrigerant properties and process controls. Because of this, the pressure difference between the high and low pressure level in the R744 refrigeration cycle is higher. An ejector heat pump using R744 as refrigerant is suited to supply domestic hot water better than a heat pump using R134a, if particular attention is directed to the entropy production in the condenser or gas cooler. In addition, the energy saving potential with an ejector is higher in an R744 cycle than in an R134a cycle.

5 Conclusions and Outlook

The presented work shows how an analysis of the dissipation effects in thermodynamic systems can be done with the help of simulation. For this purpose the equations are presented which are needed to mathematically describe the observed entropy-producing phenomena. In particular, the entropy production in the heat exchangers is examined. Entropy production resulting from heat transfer, pressure drop and numerical error due to modeling are observed. Using the example of an ejector heat pump, it is shown how the resulting simulated entropy production can be visualized and used in the dissipation analysis. The analysis is carried out using bar diagrams, Sankey diagrams and 2d-plots. Heat pumps using R134a and R744 were compared for both domestic water heating and floor

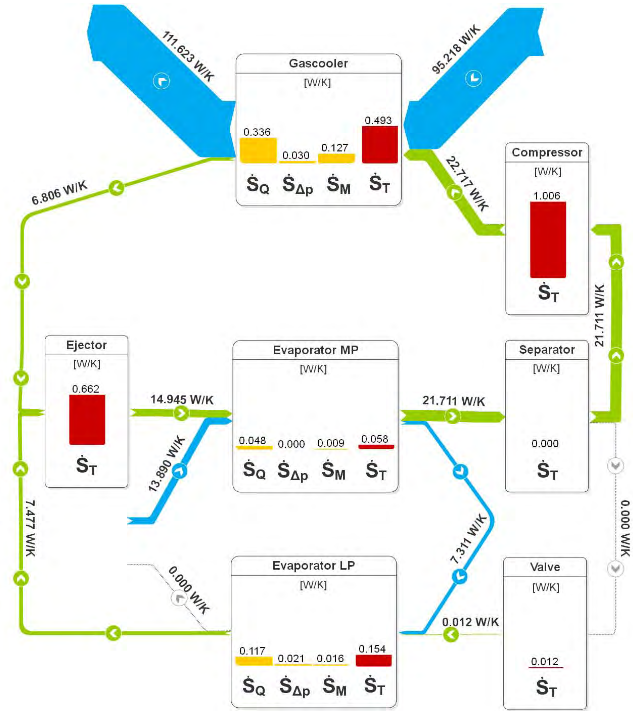


Figure 8: Entropy flow and production rate in an R744 heat pump cycle used to heat water for floor heating.

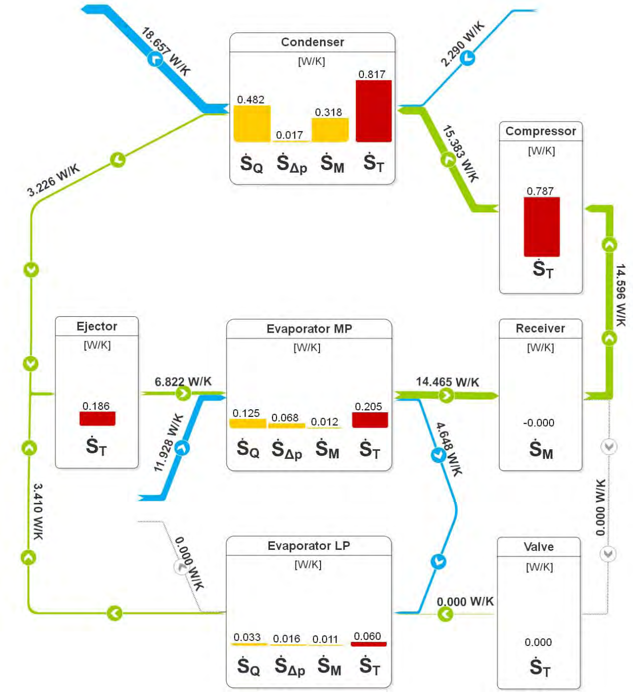


Figure 9: Entropy flow and production rate in a R134a heat pump cycle used to heat domestic hot water.

heating. The investigation shows that the heat pump with R744 is better suited for domestic hot water operation. It is shown that this analysis method is suitable for investigation of thermodynamic systems on the basis of entropy production. In the future, it is planned to research whether the boundaries of the system can be altered to produce an even better analysis or not. In addition, it is planned to carry out a more detailed analysis of entropy production within the ejector model.

References

[Baumann06] Baumann W., Bunge U., Fredrich O., Schatz M., Thiele F. Finite-Volumen-Methode in der Numerischen Thermofluiddynamik. Technische Universität Berlin, Institut für Strömungsmechanik und technische Akustik: Vorlesungsmanuskript, Berlin, 2006.

[Bejan88] Bejan A. Advanced Engineering Thermodynamics. John Wiley & Sons, New York, 1988.

[Bejan02] Bejan A. Fundamentals of Exergy Analysis, Entropy Generation Minimization, and the Generation of Flow Architecture. In: International Journal of Energy Research vol.26 no.7 p.545-565, 2002.

[Cerbe07] Cerbe G., Wilhelms, G. Technische Thermodynamik, Hanser Verlag, München, 2007.

[Elbel06] Elbel S., Hrnjak P. Development of a Prototype Refrigerant Ejector used as Expansion Device in a Transcritical CO2 System,Presentation VDA Alternative Refrigerant Winter Meeting Saalfelden, 2006.

[Franke04] Franke U. Thermodynamische Prozessanalyse: Ursachen und Folgen der Irreversibilität. Shaker, Aachen, 2004.

[Patankar80] Patankar S. Numerical Heat Transfer and Fluid Flow. Hemisphere Publ. Co, New York, 1980.

[Richter08] Richter C. Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic systems. Braunschweig, Germany: PhD Thesis, Department of Mechanical Engineerging, Institute of Thermodynamics, TU Braunschweig, 2008.

[Schmidt06] Schmidt M. Der Einsatz von Sankey-Diagrammen im Stoffstrommanagement. Beiträge der Hochschule Pforzheim, Nr. 124, Pforzheim, 2006.

Simulation of an absorption chiller based on a physical model

Christian Fleßner Stefan Petersen Felix Ziegler
Technische Universität Berlin, Fachgebiet Maschinen- und Energieanlagentechnik,
Marchstr. 18 10587 Berlin

Abstract

Previous works on simulation of air conditioning systems with absorption chillers in conjunction with detailed experimental analysis have shown a need for a more detailed and generalized modelling and simulation of heat and mass transfer processes in absorption chillers. An existing model for absorption is adapted to be applicable for subcooled or superheated liquids and for the desorption process. New classes compatible with the Modelica_Fluid library (beta 2) for these sub-processes are developed. A media model for evaporating aqueous salt solutions based on Modelica.Media is developed and implemented accordingly. Subsequently, simulations of a complete absorption chiller are conducted and compared with experimental data. The comparison of simulations under stationary conditions show a good agreement with experimental data while the transient behaviour of the plant is not yet fully implemented in the model.

Keywords: heat and mass transfer; falling film; aqueous salt solutions; Modelica.Media; Modelica_Fluid

1 Introduction

Absorption chillers are an advantageous option for reduction of primary energy demand for air-conditioning. The necessary heat can be provided by solar thermal collectors for example. To increase the market share of these cooling systems, more efficient and more compact systems with low driving temperature requirements are necessary.

Previous works on building system simulations with absorption chillers [1] have used simple linear models for absorption chillers based on empirical coefficients. For newly designed systems or systems yet to be designed these parameters are often not readily available. Also, the results of a detailed experimental analysis show significant deviations from the simple assumptions made during design [2]. Physical simulation enables a more reliable design process and improvement of absorption chillers with reduced experimental effort. The models used for the simulation must be as generalized as possible to be able to vary parameters without the need for preliminary experiments. To reach this, models based on fundamental physical properties are a favourable option. On the other hand the complexity of the model must not exceed certain limits to enable the simulation of complete systems within acceptable time limits.

In the current work a thoroughly examined sorption chiller is modelled to enable an evaluation of model quality for a complete system. The focus of modelling is on the absorption and desorption process since these are critical for overall process efficiency.

The investigated system is a compact absorption chiller with a nominal refrigerating capacity of 10 kW and the working pair water/lithium bromide. All heat exchangers are built as falling film units with the external media passing through horizontal tubes in counter-cross-flow to the internal (process side) falling film, flowing on the outside of the tubes. Detailed experimental data and the complete specification of this chiller is presented in [2].

The simulation tool used in this work is Dymola 6.1 with version 2.2.1 of the Modelica standard library and the beta 2 Version of the Modelica_Fluid library.

2 Model library

A new library for the simulation of absorption chillers was developed. The library is based on and is compatible with the Modelica_Fluid beta 2. It consists of some modified components of Modelica_Fluid, extended media models based on Modelica.Media and newly developed models for absorption and desorption or evaporation and condensation respectively in falling film flow as the mostly used unit operation in absorp-

DOI: 10.3384/ecp09430071

tion chillers. The most significant adapted models were the integration of heat transfer models with variable heat transfer coefficients with the the DistributedPipe model.

Modelling concentrates on the thermodynamic modelling of heat and mass transfer in falling film flow. The pressure losses in the steam phase and the falling film are considered to be negligibly small. Therefore hydrodynamical aspects generally are considered on a significantly simplified level.

Some parts of the absorption chiller including the sumps of the falling film heat exchangers, the pumps and much of the external piping are not considered in this model. The neglect of these parts is expected to lead to inaccuracies of the model's dynamic behaviour. Control of the chiller was not considered as well since the current work is mainly concerned with the modelling and simulation of the internal process.

2.1 Media model

For the Media model a new partial model PartialMixtureTwoPhaseMedium was derived from the Modelica.Media partial models PartialTwoPhaseMedium and PartialMixtureMedium to allow for evaporating mixtures analogously to the model developed for aqueous sodium chloride solution in [3]. The Correlations for density, specific enthalpy, specific isobaric heat capacity and specific entropy are taken from [4], while heat conductance and dynamic viscosity are computed according to Lee et. al [5]. The final model for the aqueous lithium bromide solution assumes pure water in the steam phase since the salt has a negligible vapour pressure within the valid range of the medium model. This allows to refer to the Modelica.Media.Water models for this part. The final model is not well suited for full two phase flow simulation since it is explicit in pressure, temperature and mass fraction which does not allow for a proper description of the two phase dome. For the scope of this study the possibility of surface evaporation is sufficient as it is reasonable to assume pure vapour without liquid droplets in all units. In the following parts only the newly developed models are described since most adaptations of existing models includes only minor modifications.

2.2 Film model

The model for absorption and desorption is derived from an existing model [6]. The full derivation of this model is to be found there. In the current work only the basic ideas and the main equations can be shown. Both the original model and the further development shown here assume that absorption and desorption only take place during the falling film mode along the horizontal tube with no mass transfer occurring in the droplet formation and falling droplet modes. Furthermore ideal mixing during droplet formation is assumed allowing for simple connections between single tubes. The geometry of the film is simplified to a straight one-dimensional falling film.

The calculation of heat transfer coefficients in the model is based on the stagnant film theory of Nußelt resulting in eq. (1)

$$\alpha_{film} = \frac{\lambda}{\delta} = \sqrt[3]{\frac{\lambda^3 g \rho_s^2}{3\dot{\Gamma}\eta}} \,, \tag{1}$$

with the mass flow rate per one side tube length $\dot{\Gamma}$, the thermal conductance of the solution λ, the dynamic viscosity η and the density of the solution ρ.

By means of a coordinate transformation from the running length z to a running time t_r, the mass transfer can be regarded as instationary diffusion into a semi-infinite body since the concentration in the bulk phase is constant along the falling film on each horizontal pipe. A uniform entrance concentration can be derived form the assumption of ideal mixing during droplet formation. This results in eq. (2) to describe the concentration profile along the film thickness.

$$\frac{c_{H_2O}(y,t) - c_{H_2O}(y,t=0)}{c_{H_2O}(y=0,t) - c_{H_2O}(y,t=0)} = \mathrm{erfc}\left(\frac{y}{2\sqrt{Dt_r}}\right) \tag{2}$$

The average mass transfer coefficient for a single tube is defined by eq. (3)

$$\beta = \frac{m_{abs}}{\bar{\tau}A\left(x'_{LiBr} - x_{LiBr,i}\right)} \,, \tag{3}$$

with the overall absorbed mass for a single tube m_{abs}, the mean time of exposure $\bar{\tau}$ and the mass fractions x. The local mass flow rate into the film is described with Fick's first law. The Introduction of eq. 2) into Fick's law along with the assumption that the density at the interface ρ_i and the density on entry can both be approximated with an average density $\bar{\rho}$ allows the integration of the mass flow rate over the time of exposure. This leads to the calculation of the mass transfer coefficient with eq. (4)

$$\beta = \frac{2}{\sqrt{\pi}} \cdot \bar{\rho} \cdot \sqrt{\frac{D}{\bar{\tau}}} \,. \tag{4}$$

The original model assumes a stationary temperature profile and a latent heat that is large compared to the sensible heat. Therefore only the heat of absorption has to be transferred from the interface to the cooling water which determines the coupling of heat and mass transfer as given by eq. (5)

$$\frac{m_{abs}}{\bar{\tau}}\Delta h_{abs} = A \cdot \alpha_{film} \cdot (T_i - T_w) \ . \tag{5}$$

The interface temperature T_i and the wall temperature T_w are to be assumed as constant. The interface mass fraction and temperature are connected by the equilibrium relation $T_i = T_{sat}\left(x_{H_2O}(y=0,t),p\right)$ with the constant pressure p.

This model is extended to be applicable to subcooled and superheated conditions in the bulk liquid phase of the film. To reach this the film is divided into two layers, where the heat transfer from the film surface to the core of the film is coupled with mass transfer from or to the vapour phase while the heat transfer in the inner layer between the film's core and the pipe wall is assumed to be independent from mass transfer on the surface. Heat transport in both layers is calculated separately according to Nußelt's theory and are coupled with an interface temperature between the layers. As a first approximation this temperature T_S is constantly set to be the mixed cup temperature T'' at the outlet derived from the energy balance. In this model the the distribution of the film thickness between the layers is an arbitrary constant that has to be fitted according to experimental data. A similar assumption was already made by Jeong and Garimella [7] for a laminar falling film. They assumed a linear temperature profile with a sharp increase in temperature because the heat of absorption is released locally at the surface therefore inducing a steeper gradient. In contrast to the still greatly simplified model detailed here they considered a variable depth of the bending profile. Furthermore they did neither make the simplified assumption of a constant temperature profile over the complete tube nor did they assume the surface concentration as constant. This leads to an extremely complex model requiring a discretisation of a single tube into 100 elements, making the model unsuitable for the simulation of a complete chiller. In the current work the thickness of the outer layer is set to be 10% of the overall film thickness. Jeong and Garimella [7] have calculated a similar distribution for a flow angle of 90°.

The schematic of a single absorber tube with its concentration and temperature profiles is shown in Figure 1 with the sharper increase of temperature towards the phase interface highlighted. The desorption process is described in the same way, leading to inverse profiles.

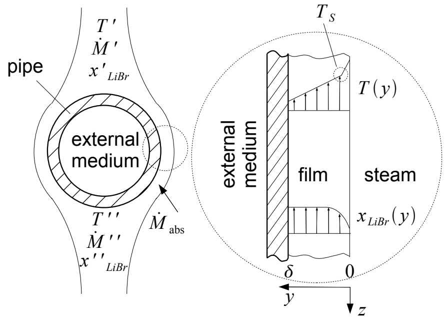


Figure 1: Schematic of a single absorber tube, concentration and temperature profile

The average absorbed or desorbed mass flow rate for a single tube $\dot{M}_{abs} = m_{abs}/\bar{\tau}$ is calculated with eq. (6)

$$\dot{M}_{abs} = \beta \cdot A \cdot \left(x'_{LiBr} - x_{LiBr,Ph}\right) \ . \tag{6}$$

The heat flow transferred to or from the tube wall and from there to the cooling water is given by eq. (7)

$$\dot{Q}_{cool} = A \cdot \alpha_{film,inner} \cdot (T_S - T_w) \ . \tag{7}$$

In the model presented here the coupling between heat and mass transfer is set up slightly different to eq. (5) since it is assumed that coupling between heat and mass transfer occurs only in the upper layer which is directly influenced by the heat of absorption resulting in eq. (8)

$$\frac{m_{abs}}{\bar{\tau}}\Delta h_{abs} = \alpha_{film,upper} \cdot A \cdot (T_i - T_S) \ , \tag{8}$$

with the relevant heat transfer occurring between the interface temperature T_i and the Temperature at the bend of the temperature profile T_S.

No mass storage in the liquid phase is assumed. Therefore the overall mass balance is defined in eq. (11)

$$0 = \dot{M}' + \dot{M}'' + \dot{M}_{abs} \ . \tag{9}$$

Accordingly the component mass balance of lithium bromide is given as eq. (10)

$$0 = x'_{LiBr} \cdot \dot{M}' + x''_{LiBr} \cdot \dot{M}'' \ . \tag{10}$$

The energy balance eq. (11) is instationary to allow for heat storage in the balance volume.

$$\begin{aligned} \frac{\mathrm{d}U}{\mathrm{d}t} &= h_L(T',x') \cdot \dot{M}' + h_L(T'',x'') \cdot \dot{M}'' \\ &\quad + h_{v,sat}(p) \cdot \dot{M}_{abs} + \dot{Q}_{cool} \end{aligned} \tag{11}$$

The vapour phase is considered to consist entirely of saturated pure water steam. Mass transfer resistance in the vapour phase and on the phase boundary is neglected. Thermal equilibrium on the interface is assumed for the liquid phase. The mathematical representation of the Generator in the current model is completely identical to that of the absorber and is therefore not shown here.

Evaporator and condenser are completely described with Nußelt's stagnant film theory. No mass transfer resistances are accounted for. Equilibrium is assumed in both the vapour and the liquid phase. The heat transfer and balance equations are identical to absorber and generator. No composition has to be considered since pure water is used as cooling agent.

The implementation in Modelica includes partial models for film heat and mass transfer and final models based on the equations above. These models are selectable from separate models for falling film flow including the balance equations and the coupling between heat and mass transfer. Flow reversal is implemented for the vapour phase connectors to allow the usage of the film models for absorption/desorption and evaporation/condensation respectively.

2.3 System models

The models for the falling film are coupled with the modified tube models via HeatPort connectors from Modelica.Thermal as shown in Figure 2. These pipe segments are then connected with FluidPorts according to each units flow path. A more flexible solution allowing for selection of different flow paths and number of passes is desirable but is not implemented yet.

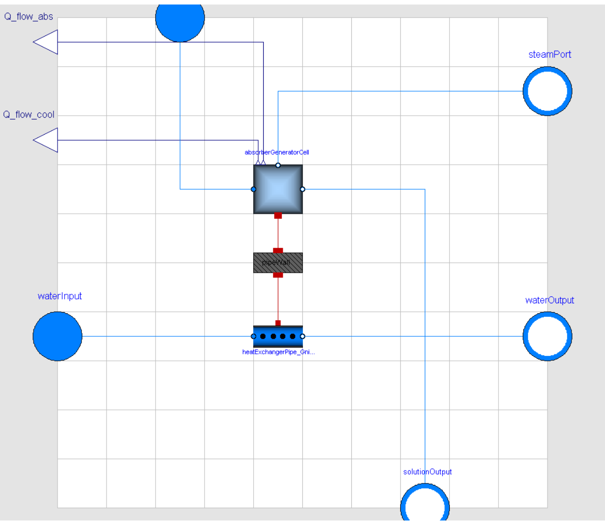


Figure 2: Diagramm of AbsorberGeneratorPipe

Each vapour phase is represented by a MixingVolume which is connected to the heat exchangers by FluidPorts. Constant mass flow rates are prescribed for internal and external circuits. The temperatures of the external flows are kept constant, while the entry temperatures and concentrations of the internal solution circuit and the entry temperature of the evaporator circulation are fed back from the respective output values. The representation of the overall model in Dymola is shown in Figure 3.

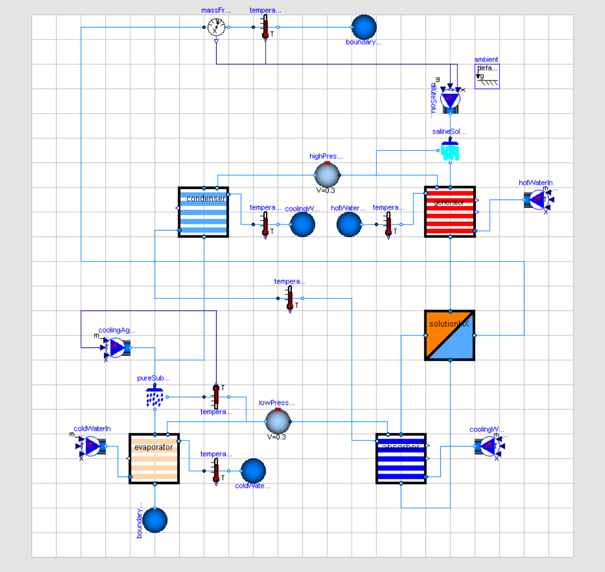


Figure 3: Diagramm of complete model

3 Comparison with experimental results

In most cases the simulation is conducted with constant external conditions. From a cold start the whole system becomes fully stationary after less than 100 s simulated time. This relatively fast reaction is reasonable as some major points of heat and mass storage in the plant (heat exchanger sumps an external piping for instance) are not included in the model and mass storage in the falling film is generally neglected. Simulations with ideal steps in one of the three external temperature levels have been conducted showing a qualitatively correct response. Figure 4 exemplary shows the response of the model to an instantaneous increase in driving temperature with all other external conditions left constant. The response is qualitatively plausible with all final values identical to the stationary simulations. Due to the reasons mentioned above the simulated delay is much shorter than the delay observed in the real plant, where [8] reports measured delays 10 times as high as in the current simulations.

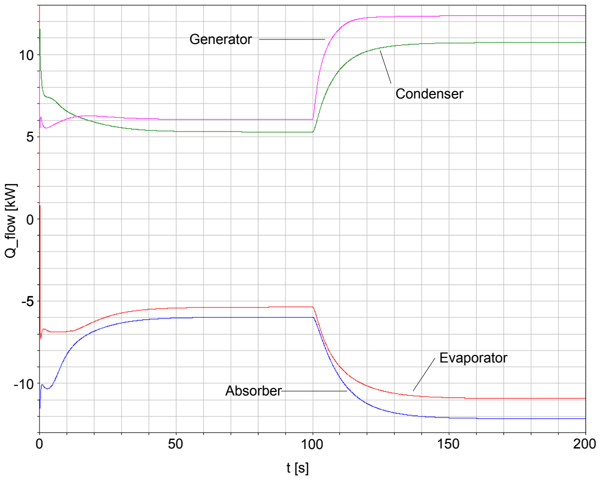


Figure 4: Response of external heat flows to step in driving temperature (55°C to 75°C at 100 s)

The simulation of the complete systems yields stationary heat fluxes which show a positive deviation of 5% to 15% from experimental data. The deviation increases with the driving temperature. the results for nominal operating conditions (external conditions: 75°C generator inlet, 27°C cooling water inlet, 18°C cold water inlet) are shown in Figure 5. One plausible reason for this systematic deviation is that incomplete wetting of the heat exchanger surfaces is not considered in the model. In [2] an average wetting between 80% and 90% with significant deviations between operating conditions was reported. A rough approximation where the internal heat exchanger surface was reduced by 20%, while the external area was constant, led to significantly improved results for nominal conditions. The generator shows a larger deviation than the other units. This indicates that the generalisation of assumptions concerning heat and mass transfer as well as wetting for absorber and generator is not fully adequate. Nevertheless, the reults are generally in good agreemant with the experimental data.

The internal temperatures and concentrations in the falling film including superheating and subcooling that were measured in [2] could also be approximated in the simulation giving further credibility to the assumptions made in the model. Figure 6 shows the arithmetic average of the calculated and measured internal temperatures for better comparison.

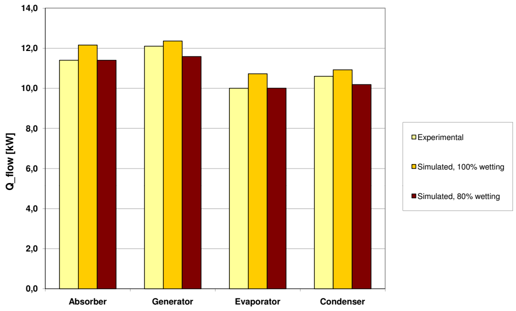


Figure 5: Stationary heat flow at nominal conditions

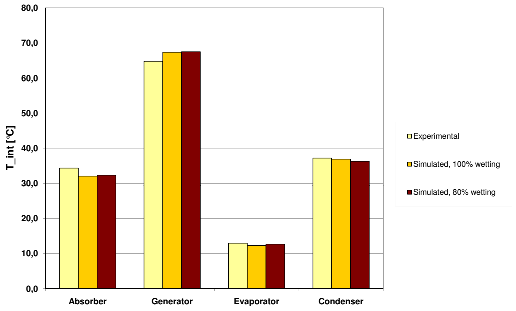


Figure 6: Stationary internal temperatures (arithmetic average of inlet and outlet) at nominal conditions

4 Conclusion

Overall the accuracy of the stationary simulations with this simple model which is nonetheless not strongly dependent on empirical fitting parameters is rather good. The comparison of the simulated results with experimental data shows good agreement in overall performance though an evaluation of the accuracy of the predicted local transfer coefficients has yet to be done by more detailed experiments. Pending issues include the yet incomplete modelling of the dynamic behaviour of the whole system and the long simulation times required for simulation of a complete chiller. Since the implementation of flow reversal and the structure of the media model generate large amounts of non-linear equations, a single simulation run over 100 s simulated time with constant external conditions takes more than an hour of simulation time. In its current state the model is more suitable for parameter generation for simpler models than for system simulation. Therefore further development will be conducted including the adaptation of the library to the Modelica_Fluid 1.0 Library to resolve current performance issues. The hydrodynamical modelling also needs some improvement as a systematic and physically founded consideration of incomplete wetting seems to be worthwhile for generalisation of the model. Here care has to be taken to avoid an overly complex model making simulation of complete chillers unfeasible.

References

[1] Annett Kühn, José Luis Corrales Ciganda, Felix Ziegler. (2008): Comparison of control strategies of solar absorption chillers, Proceedings of the 1st International Conference on Solar Heating, Cooling and Buildings (Eurosun), 7-10 October 2008, Lisbon, Portugal

[2] Annett Kühn, Lukas Enke, Felix Ziegler (2008): Detailed Analysis of A 10 kW H_20/LIBR Absorption Chiller, International Sorption Heat Pump Conference 2008, 23-26 September 2008, Seoul

[3] Katja Poschlad, Manuel A. Pereira Remelhe, Martin Otter (2006): Modeling of an experimental Batch Plant with Modelica, Proceedings of the 5th International Modelica Conference 2006, Vienna

[4] Günther Feuerecker (1994): Entropieanalyse für Wärmepumpensysteme: Methoden und Stoffdaten, Ph.D.-Thesis Technische Universität München

[5] R.J. Lee, R.M. DiGuilio, S.M. Jeter, A.S. Teja (1990): Properties of Lithium Bromide-Water Solutions at High Temperatures and Concentrations - II Density and Viscosity in ASHRAE Transactions, Paper 3381, RP-527, pp. 709-714 Atlanta: American Society of Heating, Refrigerating and Air-Conditioning Engineers

[6] Hein Auracher, Arnold Wohlfeil, Felix Ziegler (2008): A simple physical model for steam absorption into a falling film of aqueous lithium bromide solution on a horizontal tube, Heat and Mass Transfer 44; 1529-1536

[7] Siyoung Jeong, Srinivas Garimella (2002): Falling-film and droplet mode heat and mass transfer in a horizontal tube LiBr/water absorber: International Journal of Heat and mass Transfer 45; 1445-1458

[8] Paul Kohlenbach (2006): Solar Cooling with absorption chillers: Control strategies and transient chiller performance: Ph.D.-Thesis Technische Universität Berlin; DKV-Forschungsbericht Nr. 74

Effects of Tool Coupling on Transient Simulation of a Mobile Air-Conditioning Cycle

Roland Kossel[1] Nils Christian Strupp[2] Wilhelm Tegethoff[1]
[1]TLK-Thermo GmbH
[2]TU Braunschweig, Institut für Thermodynamik
Hans-Sommer-Str. 5, 38106 Braunschweig
r.kossel@tlk-thermo.de

Abstract

Results of numerical simulations more and more provide a basis for design decisions in an automotive context. When simulating complex systems, one of two approaches can be chosen: The modeling in one multi-domain language like Modelica or the utilization of different specialized simulation programs.

This paper demonstrates the simulation of the Heating Ventilation and Air-Conditioning system (HVAC) of a car. The different components are modeled individually and validated with measurement data in separate test benches. A co-simulation using one Dymola instance per component model is then created to represent the whole refrigeration cycle taking into account the inter-component dependencies.

To evaluate the effects introduced by the tool coupling, the results are compared to those of a single Modelica model composed of all component models.

Keywords: tool coupling; co-simulation; refrigeration

1 Introduction

Results of numerical simulations more and more provide a basis for design decisions in an automotive context. This also applies for the thermodynamic subsystems for example the Heating Ventilation and Air-Conditioning system (HVAC).

This paper discusses dynamic simulations of an automotive refrigeration cycle with Modelica using the TIL library and Dymola. A R134a cycle with detailed components is used. Each component model is validated separately using measurement data from a broad range of ambient conditions.

There are two approaches for modeling and simulating complex systems composed of multiple components: Use a suitable language to describe the complete system in one model or divide the system into submodels, then employ different simulation programs specialized for the respective subproblems and use co-simulation to create a model of the complete system.

The required level of detail plays an important role in the decision for either approach. If for example a simulation of the HVAC unit and the passenger's compartment of a car shall be conducted, the models could be created using just Modelica. If however the goal of the whole simulation is an evaluation of the temperature distribution within the compartment, a 3d simulation tool must be used; because the HVAC unit can be represented only poorly by 3d tools, a co-simulation makes sense [5].

Under specific circumstances it is even practical to create a co-simulation with multiple instances of one tool. The decision must be made considering two main points: The simulation speed and the numerical stability of the simulation. For both points no general rule can be given to decide in favor or against tool-coupling. When considering small or numerically simple models, the simulation time is most likely to increase when splitting them into several parts (see e.g. [9]). Looking at large or complex models, splitting these into submodels can greatly enhance the speed.

Especially solving systems of equations with significantly different time constants can be greatly improved by decoupling these time constants. While simulation tools could support this internally by employing multi-rate solving techniques, co-simulation enables the user to create a "distributed multi-rate simulation" using tools without a multi-rate solver.

Considering all points mentioned above it becomes clear that the model partitioning is an important part of the model design. Normally aspects like time constants or required computing time have to be inspected. This step can be omitted for this paper, since each

DOI: 10.3384/ecp09430064

component shall be simulated separately.

2 Co-simulation

For the tool coupling this paper uses the co-simulation environment TISC®. This environment is divided in two layers: The *Control-Layer* and the *Simulation-Layer* (figure 1). To ensure platform independency, TCP-sockets are used for communication between all distributed components.

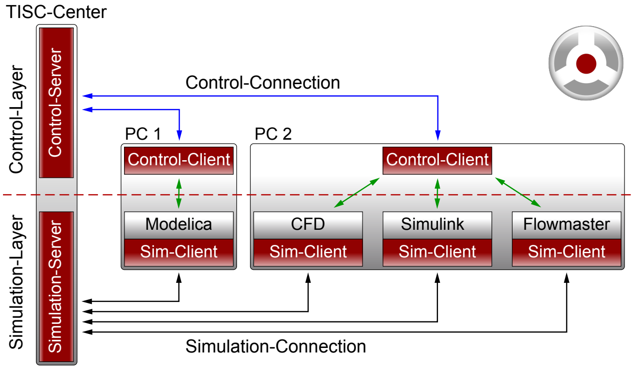


Figure 1: Layer structure of TISC

The Control-Layer consists of the central Control-Server and several Control-Clients – one on each computer participating in the simulation. Through the Control-Server the user can trigger the start of the simulation. The Control-Server sends the appropriate commands to the respective Clients using the Control-Connection. Besides these start commands, also status messages and stop commands can be sent.

The started model instances connect to the Simulation-Server through the Simulation-Client, which has been integrated into each model. The used integration techniques differs depending on the used language and tool. In case of Modelica and Simulink input and output blocks are added to the model, in case of the 1d tools Flowmaster and Kuli the information is accessed through interfaces available through COM, other tools (e.g. CFD) require still other techniques.

To make it more convenient for the user to configure, run and evaluate the simulation, the two Servers are united in the TISC-Center.

During the simulation, the Simulation-Server's tasks are the data transfer between and synchronization of the single models. While it is possible to use sequential (or "explicit") synchronization, the parallel (or "implicit") synchronization (see [11]) is used most of the time in TISC. As main advantage of this over the sequential synchronization, the different models are being calculated in parallel. This benefit is amplified with an increasing number of coupled models leading to a major increase of simulation speed for complex systems.

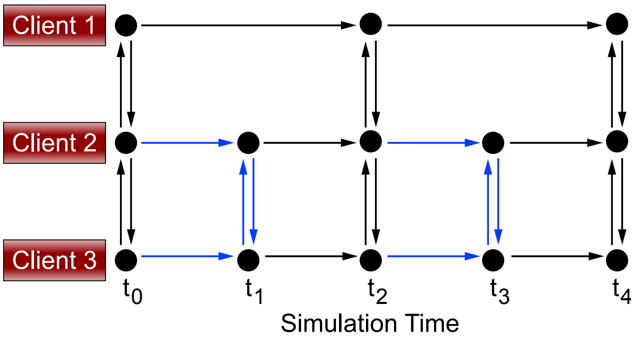


Figure 2: TISC synchronization scheme

The required time for the co-simulation is heavily influenced by the exchange rate between the single models. At synchronization time, every client has to be stopped, the data transmitted and the solvers reinitialized. While the time needed for reinitialization heavily depends on the employed solver, the other delays are directly proportional to the number of synchronization points. As shown in figure 2, the implemented synchronization allows for different time step sizes for the simulation clients. Therefore the overall simulation speed can be improved by increasing the exchange rate for complex systems with relatively large time constants thus reducing the overall number of synchronization events.

At synchronization time the reinitialization of the solvers is being hampered by the value patterns of the variables exchanged through TISC. Since only the value of the variables can be transferred, the variable is a discrete one on the receiving side. The higher the step at synchronization time, the harder it is for the solver to find a consistent solution for the system of equations – it is even possible that the solver fails to find a solution. TISC includes extrapolation and smoothing possibilities on the receiving side to cancel this effect. Figure 3 shows the values of a sine sent through TISC with a period time of $1\,s$ and an exchange rate of $0.1\,s$. Cubic polynomials are used to extrapolate received values to the respective next time step. By using a 5th degree polynomial to switch between the polynomial built from the 4 values before synchronization time and the polynomial built from the last 4 values including the synchronization time, the curve is smoothed resulting in the line shown in figure 3 ("Extrapolated"). The described technique leads to a function which is two times continuously differentiable, hereby helping the solving process.

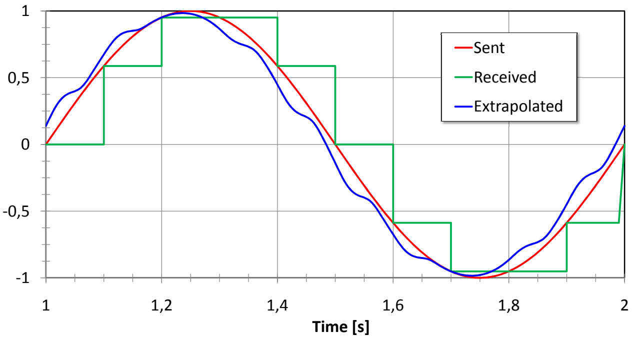


Figure 3: Value pattern with and without extrapolation

3 Investigated System

Figure 4 depicts the design of a car's HVAC unit which consists of five components: a compressor, a condenser, a receiver, an expansion device, an evaporator and an internal heat exchanger.

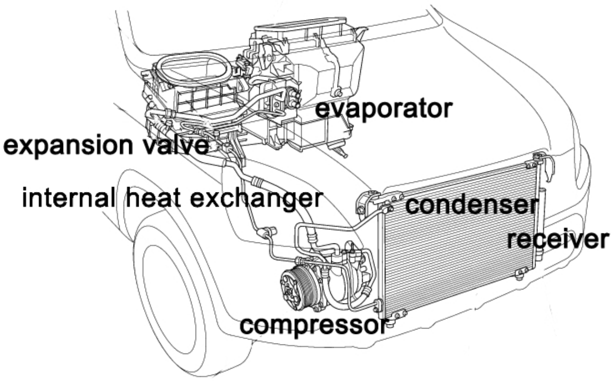


Figure 4: HVAC unit of a car

The HVAC unit uses a circulating refrigerant which enters the compressor and is compressed to a higher pressure, resulting in a higher temperature as well. Afterwards the refrigerant enters the condenser, where it rejects heat to the environment. In the internal heatexchanger the refrigerant is furthermore subcooled by rejectig heat to the low pressure side. Hereafter it is expanded to a lower pressure by an expansion device, e.g. a thermostatic expansion valve. Then the refrigerant flows through the evaporator where it is evaporated. During the process of evaporation the refrigerant absorbs heat from the passenger compartment decreasing its temperature. The absorbed heat is also called cooling capacity, which is a characteristic value of the performance of an HVAC unit. Finally, inside the low pressure side of the internal heat exchanger, the refrigerant is superheated before reentering the compressor.

3.1 Component modeling

The R134a vapor compression cycle is modeled using component models from the TIL library. TIL is a component model library for thermodynamic systems developed by the Institute for Thermodynamics (IfT) and the TLK-Thermo GmbH. It allows for steady-state and transient simulation of thermodynamic systems.

Heat transfer and pressure drop correlations for each component model are validated with measurement data from a set of more than 15 different ambient conditions.

The condenser is a flat-tube heat exchanger with four refrigerant flow passes. Each of the four passes is discretized into five control volumes ("cells") representing the manifoldness of flat-tubes of the respective pass, thereby considering different flow cross sectional areas. Since the receiver is integrated into the condensor, the component is also called "CondReceiver".

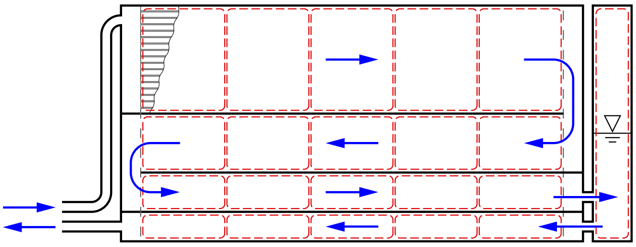


Figure 5: Schematic diagram of CondReceiver

Air-side heat transfer coefficient and pressure drop are modeled using correlations from literature [10] capturing influences of geometry as well as ambient conditions. The refrigerant side heat transfer coefficient is preestimated using a correlation specific for condensation in minichannels [2] but is set constant during the simulation to the preestimated value of $\alpha = 4300\,W/m^2K$. Wall heat conduction is modeled one-dimensional and perpendicular to both fluids, where characteristic lengths are calculated from the geometric parameters. The integrated receiver is considered as a separator with a characteristic curve accounting for changes in outlet vapor fraction at very low and very high filling levels.

Due to the object oriented approach, the evaporator model is built from the same basic elements as the condenser – with different geometric parameters. The evaporator modeled has a two layer design with three passes per layer. Each of the passes is discretized into five cells. Condensation and evaporation of moisture are taken into account by means of an analogy of heat and mass transfer. The heat transfer and pressure drop correlations are developed analogous to those of the condenser. The refrigerant side heat transfer coefficient is also set to a preestimated value ($\alpha = 4300\,W/m^2K$) during simulation.

The internal heat exchanger is modeled as a tube in tube heat exchanger, using heat transfer correlations from [13]. Each tube is represented by five cells.

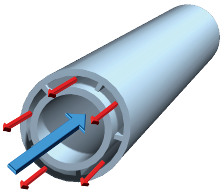


Figure 6: Schematic drawing of internal heat exchanger

The compressor model is mapped using a quasi-steady state model based on measurement data for full and partial load as proposed by [3]. Three efficiency functions are used to characterize the compressor efficiencies, namely volumetric efficiency, effective isentropic efficiency and isentropic compressor efficiency.

The thermostatic expansion valve is modeled using Bernoulli´s equation for compressible and incompressible flow [4].

3.2 Model validation

Even though data validity is usually not considered part of model verification and validation, it is addressed here, as suggested by [12], as measurement data validation sets the baseline for the achievable model accuracy. Therefore only a small proportion of the available measurement data was used for validation purposes.

Simulations in test bench environments have been run for all component models. In these test benches, each component model is provided with mass flow, specific enthalpy and pressure by boundaries at the inlet or the outlet. These boundary conditions are extracted from measurement data for over 15 measurement points. Extreme-condition Tests were run to define the scope of each component model.

Exemplarily the validation results of the condenser model are depicted in figure 7 in terms of measured rejected heat over simulated rejected heat.

The measurement data can be reproduced with a deviation of ±10% by using empirical, physically motivated correlations without any correction factors. These points cover a range of thermal efficiency of 65% to 100%.

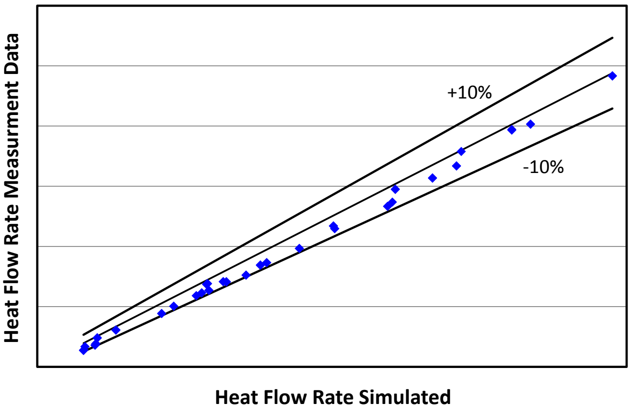


Figure 7: Comparison of measurement data and simulated heat flow rate with validated condenser model

After combining the single models to a closed model of the refrigeration cycle, p,h-diagrams were used as further means of validation. Figure 8 depicts the slight deviations of the simulation results compared to the measurement data stating the validity of the system model.

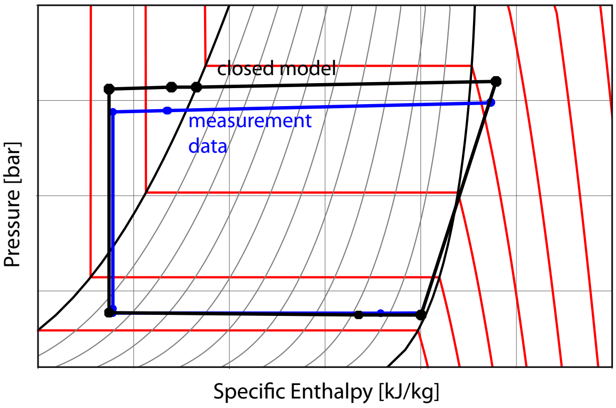


Figure 8: Comparison of measurement data and simulation results of a steady state condition

4 Simulation setup

In order to determine the effects of tool coupling on the simulation of the closed refrigeration cycle, the single validated models presented in section 3 are connected in two different ways:

1. Closed model in Modelica
2. Co-simulation of separate models

When splitting a model into submodels in preparation of a co-simulation, several possibilities exist. Various considerations may influence the partitioning.

When different tools are to be employed – e.g. coupling a 3d-model of a heat exchanger with 1d-models of the remaining components – the cutting points are obvious. Very detailed component models can be separated in order to use more processors and memory. Models with considerably different time constants can be decoupled creating a sort of distributed multirate method increasing the overall simulation speed [6, 7].

The system presented in this paper is composed of detailed models (especially the heat exchangers) with many interconnections. Since the intention was to reuse the models of section 3 without modifications, they were in the first step included in the co-simulation as independent systems. Since the small time constant of the internal heat exchanger induces a tight coupling of the two pressure levels, the complete system is prone for oscillation of the thermodynamic state variables. In addition, the complex heat transfer and pressure drop correlations impede the simulation. In a second step therefore the internal heat exchanger was split into two parts in order to get closed models of the two pressure levels which correspond to one pressure state variable each in TIL [8]. Figure 9 shows the structure of the closed refrigeration cycle, the colored areas corresponding to the four coupled models Compressor (green), CondReceiver and high pressure side of the Internal Heat Exchanger (blue), Valve (brown) and Evaporator and low pressure side of the Internal Heat Exchanger (red).

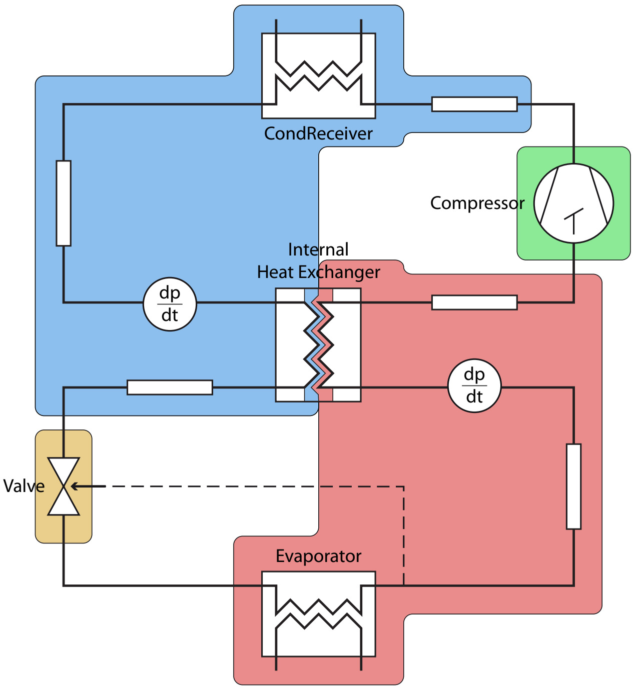


Figure 9: Structure of simulated cycle

As figure 10 shows, the internal heat exchanger is represented by tubes with heat ports. The heat flow rate is read on the high pressure side and imposed on the low pressure side. The temperature is treated the same way in the opposite direction. Resistors are added to each side to increase the time constant of the subsystems. Furthermore a capacitor is used to create an artificial temperature state. While the effect of these three elements on the accuracy of the steady-state simulation result is negligibly small, they allow for a larger larger exchange rate in the co-simulation.

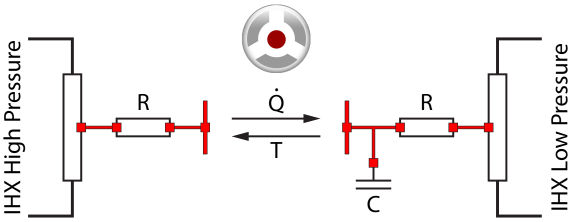


Figure 10: Splitting of Internal Heat Exchanger for co-simulation

To be able to compare the closed model with the co-simulation, the same component models with identical sets of parameters are used. The correlations for heat transfer and pressure drop are switched from constant values during initialization to geometrically and physically based correlations at different instances in time during the simulation. This procedure has shown to be necessary for the closed Modelica model to achieve a robust initialization. Although the co-simulation models can be started using the complex correlations from the very beginning, the same settings were used for the correlations for better comparability of the results.

5 Effects of co-simulation on steady-state simulation

The main differences for the user are development time and the time needed for the simulation. In figure 11 the simulation time for different simulations is presented. The switching of correlations is deductable from the pattern of the closed model (green line). The red and the blue line represent coupled simulations varying only in the employed solver (the time needed for the simulation using the dassl solver is about 5800 seconds wall clock time for 60 seconds simulation time).

Comparing the co-simulation with the closed model, a gain in speed is achieved during initialization. As the curves of the coupled simulations highlight, the co-simulation is able to initialize a lot faster

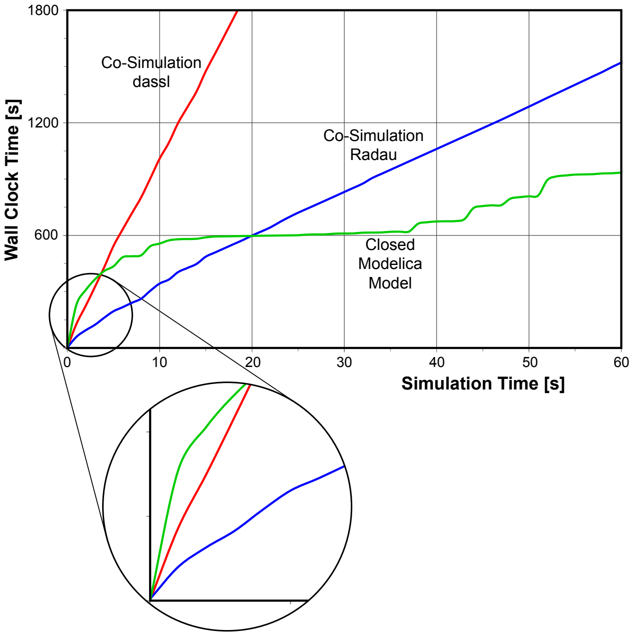


Figure 11: Simulation time of different simulations with constant boundary conditions

– regardless of the employed solver. Experience also shows that the co-simulation is far less vulnerable to ill-configured start and initial values. For this reason the development time needed for reaching a well behaving simulation is much lower when employing co-simulation.

As figure 12 highlights, the result of the co-simulation is consistent with the one from the closed model. Analogous to section 3.2, the co-simulation of the refrigeration cycle is considered as validated.

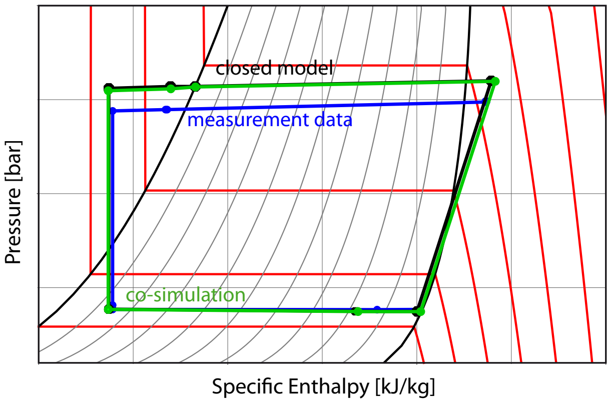


Figure 12: p,h-diagram of a simulated steady state condition

The data exchange rate not only has effect on the required simulation time as stated in section 2, it also heavily influences the robustness of the co-simulation – the smaller the exchange rate the more robust the simulation. The drawback of a small exchange rate is the decrease in simulation speed. Since the numerical solver has to be stopped at a specific point in time, not only the time event is generated, the solver also needs to be reinitialized which requires a significant amount of time [1]. As figure 11 shows, the chosen solver can also greatly influence the simulation speed. The size of the data exchange rate is limited by the time constants of the system's components. The simulation shown in figures 11 and 12 were conducted with an exchange rate of 0.1 seconds, which roughly equals the lowest time constant in the system.

6 Effects of co-simulation on transient simulation

The advantage of the co-simulation reacting friendly to ill-configured starting conditions can also be observed during transient simulations with highly dynamic boundary conditions. As an example a simulation using the NEDC (see figure 13) was conducted.

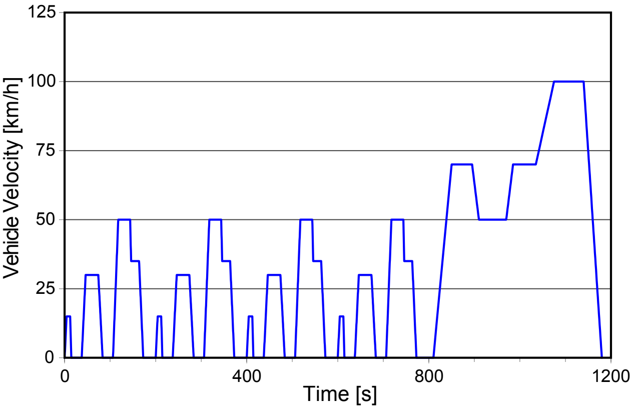


Figure 13: New European Driving Cycle (NEDC)

While the time constants can be easily determined under constant boundary conditions, close attention has to be paid in a changing environment. During periods with highly dynamic boundaries the smallest time constant was as low as 0.02 seconds. Therefore the data exchange rate had to be adjusted to these changing conditions since a slightly too large rate is immediately inducing instabilities.

7 Conclusion and Outlook

Modelica models for the different components of a car's HVAC unit have been modeled and validated in separate test benches. It could be shown, that a closed model of a refrigeration cycle employing the validated components as well as a co-simulation coupling separate simulation instances of the same components can be considered validated.

To reach a robust initialization for the closed model, heat transfer and pressure correlations were set to start with constant values, switching to physically motivated equations at different instances in time during the simulation. Furthermore, adjustments to start and initial values had to be made. The co-simulation required no simplified correlations and also initialized robustly with ill-configured start and initial values. Therefore the co-simulation can play it's trump cards when changing single component models of an existing system or when building a completely new system model. In addition, also the simulation of different constant and dynamic boundary conditions is simplified. In all cases, the start and initial values hardly ever need to be changed.

Different techniques can be utilized to further stabilize the solution process during a co-simulation. Decoupling of tight dependencies was presented by splitting the internal heat exchanger thus separating the two pressure state variables of the refrigeration cycle. Extrapolation and smoothing can be applied to avoid steps in the course of received values simplifying the reinitialization of the numerical solver. An investigation targeting the optimal extrapolation order has not been conducted for this paper but is interesting for future work since the critical data exchange rate is smaller at higher orders of extrapolation (see [7]). Even if not using extrapolation, smoothing the steps still helps the solver.

Close attention has to be paid to the data exchange rate within the co-simulation since it must never exceed the smallest time constant. While this time constant can easily be determined in simulations with constant boundary conditions, it is more difficult but not less important with dynamic boundaries like driving cycles. An automatic adjustment of the exchange rate is subject of future work.

References

[1] M. Arnold, Simulation Algorithms in Vehicle System Dynamics, Martin-Luther-Universität Halle-Wittenberg, 2004.

[2] T. M. Bandhauer, Measurement and Modeling of Condensation Heat Transfer Coefficients in Circular Microchannels, In: Transactions of the ASME, Vol. 128, 2006.

[3] S. Försterling, Vergleichende Untersuchung von CO2-Verdichtern in Hinblick auf den Einsatz in mobilen Anwendungen, TU Braunschweig, PhD-Thesis, 2004.

[4] D. W. Green, Perry's chemical engineers' handbook, The McGraw Hill Companies, ISBN 978-0-07-142294-9, 2007.

[5] R. Kossel et al., Simulation of Complex Systems using Modelica and Tool Coupling. In: Proceedings of the 5th International Modelica Conference 2006, Vienna, Austria, Modelica Association, 4-5 September 2006.

[6] R. Kossel et al., Einsatz hybrider Simulationstechnik für die Bewertung mobiler Heiz- und Kühlkonzepte. In: Wärmemanagement des Kraftfahrzeugs VI, Berlin, Germany, Haus der Technik, June 2008.

[7] R. Kübler, Modulare Modellierung und Simulation mechatronischer Systeme, Universität Stuttgart, PhD-thesis, 2000.

[8] N. Lemke, Untersuchung zweistufiger Flüssigkeitskühler mit dem Kältemittel CO2, TU Braunschweig, PhD-thesis, 2004.

[9] K. Nyström and P. Fritzson, Parallel Simulation with Transmission Lines in Modelica. In: Proceedings of the 5th International Modelica Conference 2006, Vienna, Austria, Modelica Association, 4-5 September 2006.

[10] Y.-G. Park et al., Air-Side Heat Transfer and Friction Correlations for Flat-Tube Louver-Fin Heat Exchngers, In: Journal of Heat Transfer, February 2009, vol. 131.

[11] W. Puntigam et al., Transient Co-Simulation of Comprehensive Vehicle Models by Time Dependent Coupling. In: SAE 2006 Transactions Journal of Passenger Cars: Mechanical Systems, ISBN 978-0-7680-1838-7, pages 1516 - 1525.

[12] Robert G. Sargent, A tutorial on validation and verification of simulation models, In: Proceedings of the 1988 Winter Simulation Conference, San Diego, USA, 12-14 December 1988.

[13] John R. Thome, Engineering Data Book III, 2004.

[14] J. R. Thome, Heat transfer model for evaporation in microchennels Part 1: presentation of the model, In: International Journal of Heat and Transfer, March 2004, vol. 47, pp. 3375-3385.

Dynamic modelling of the heat transfer into the cooling screen of the SFGT-Gasifier

Julia Kittel[1] Frank Hannemann[2] Friedemann Mehlhose[2] Sindy Heil[1] Bernd Meyer[1]

[1]Institut of Energy Process Engineering and Chemical Engineering
TU Bergakademie Freiberg
09596 Freiberg

[2]Siemens Fuel Gasification Technologie GmbH & Co.KG
Halsbrückerstraße 34
09599 Freiberg

Julia.Kittel@iec.tu-freiberg.de

Abstract

The paper deals with the transient modelling of the heat flux into the cooling screen of the SFGT-Gasifier. Therefore the modelling assumptions and the implementation in Modelica/Dymola were described.

Keywords: SFGT-Gasifier, heat transfer, slag layer modelling, cooling screen

1 Introduction

The SFGT-Gasifier is an entrained flow gasifier. Coal consisting of fixed carbon, volatiles, ash and water is converted at high pressure (about 40 bars) and high temperature (1400-1700 °C) conditions and by addition of oxygen into a synthesis gas (syngas) composed primarily of carbon monoxide (CO) and hydrogen (H_2).

An advantage of the SFGT-Gasifier is the utilization of a cooling screen instead of refractory lining allowing a fast start-up-process. The cooling screen is composed of a castables layer and a helical tube with water as cooling medium (Figure 1).

An entrained flow gasifier is operated at high temperatures well above the ash melting temperature (T > 1300 °C). The molten ash (called slag) accumulates on the internal walls of the reaction chamber due to drag forces. And hence, a liquid slag layer is formed. Between molten slag and cold castables a layer of solidified slag appears. The thickness of the slag layer depends on the process conditions.

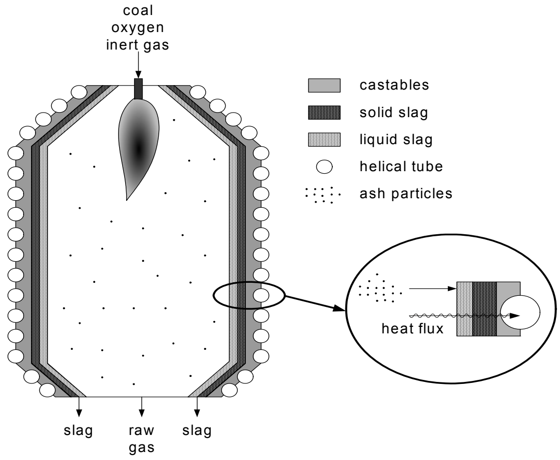


Figure 1: Schematic illustration of the reaction chamber of the SFGT-Gasifier and slag deposit

Hence any dynamical change of heat flux indicates variation in gasifier performance and can be used for better operational control. For this reason it is of great interest to simulate the slag layer formation since the slag layer is the limiting factor for the heat flux, due to the small thermal conductivity of the slag.

DOI: 10.3384/ecp09430011

2 Gasification fundamentals

2.1 Gasification in general

The gasification process is of great importance for the power and basic chemical industry as it coverts any carbon-containing material into a syngas composed primarily of carbon monoxide and hydrogen. This syngas can be used as a fuel in a combined cycle to generate electricity (Integrated Gasification Combined Cycle). But it can also be used as a feedstock for a large number of syntheses in the chemical industry, gaining products like methanol, methane, ammonia or hydrocarbons (Fischer Tropsch Synthesis).

Gasification means the thermo-chemical conversion of fuels with one or more reactants to a combustible gas, which is desirably rich of components CO, H_2 and methane (CH_4). The most proceeded reactions are the partial oxidations, which take place with oxygen in free (molecular) or bounded form (steam (H_2O), carbon monoxide (CO_2)). These partial oxidations are interfered in dependence on the process and the process parameters with pyrolysis or devolatilization and hydrogenation processes [1].

The gasification process can be classified into different types according to the heat supply (autothermic, allothermic or hydrogenating gasification), the gas-solid-contacting (fixed/moving bed, fluidized bed or entrained flow gasification) or concerning the process temperature (above or below the ash melting point).

In the gasification process a large number of reactions take place. Principle chemical reactions are those involving carbon (C), carbon monoxide, carbon dioxide, hydrogen, water (or steam) and methane [2]:

Combustion reactions:

$$C + 0.5\, O_2 \rightarrow CO \qquad -111 \text{ MJ/kmol}$$

$$CO + 0.5\, O_2 \rightarrow CO_2 \qquad -283 \text{ MJ/kmol},$$

$$H_2 + 0.5\, O_2 \rightarrow H_2O \qquad -242 \text{ MJ/kmol}$$

Boudouard reaction:

$$C + CO_2 \leftrightarrow 2\, CO \qquad +172 \text{ MJ/kmol},$$

Water gas reaction:

$$C + H_2O \leftrightarrow CO + H_2 \qquad +131 \text{ MJ/kmol},$$

Hydrogenation reaction:

$$C + 2\, H_2 \leftrightarrow CH_4 \qquad -75 \text{ MJ/kmol},$$

CO Shift reaction:

$$CO + H_2O \leftrightarrow CO_2 + H_2 \qquad -41 \text{ MJ/kmol},$$

Steam reforming reaction

$$CH_4 + H_2O \leftrightarrow CO + 3\, H_2 \qquad +206 \text{ MJ/kmol}.$$

Most fuels contain additional components beside carbon, hydrogen and oxygen, e.g. sulfur, nitrogen or minerals. Sulphur in the fuel is converted into H_2S and COS and the nitrogen into molecular nitrogen, NH_3 or HCN.

2.2 SFGT-Gasifier

The SFGT-Gasifier is a top fired, dry feed, autothermic, oxygen blown, entrained flow gasifier with temperatures in the gasification section well above the ash melting point. The slag and the hot gasification gas leave the gasification section together. After gasification section the hot gas is cooled down in the quench by injection of cold water.

Figure 2 shows the schematic design of the SFGT-Gasifier.

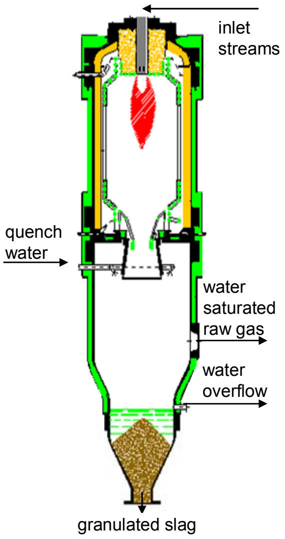


Figure 2: Schematic design of the SFGT-Gasifier

3 Theoretical background

3.1 Equilibrium calculation for the gasification process

For an entrained flow gasifier it can be assumed that the raw gas leaving the reaction chamber is in chemical equilibrium due to high temperatures.

There are two general alternatives to calculate a chemical equilibrium: equilibrium due to reaction equilibria or equilibrium due to minimization of the Gibbs free energy.

Here the minimization of the Gibbs free energy was adopted:

$$G\left(\{n_j\}\right)=\sum_{j=1}^{N_S}\mu_j\cdot n_j=\min!,\quad T,p=const \tag{1}$$

where G is the Gibbs free energy, μ_j is the chemical potential of chemical substance *j, n_j* is the mol quantity of chemical substance j and N_S is the number of chemical substances.

Under the side conditions:

$$\sum_{j=1}^{N_S}a_{ij}\cdot n_j=b_i,\quad i=1,\dots,N_E \tag{2}$$

where b_i is the quantity of chemical element i, $\{a_{ij}\}$ is the elemental matrix and N_E is the number of chemical elements.

For the modelling of the heat flux through the cooling screen of the SFGT-Gasifier only the typical chemical gasification substances CO, CO_2, CH_4, H_2, H_2O, H_2S, N_2, O_2 and fixed carbon have to be considered for the calculation of the chemical equilibrium.

The constrained optimization problem can be solved through conversion in an unconstrained minimization problem by adoption of Lagrange multipliers $\{\lambda_i\}$ [3]:

$$L\left(\{n_j\},\{\lambda_k\}\right)=\sum_{j=1}^{N_S}\mu_j\cdot n_j+\sum_{i=1}^{N_E}\lambda_i\left(b_i-\sum_{j=1}^{N_S}a_{ij}\cdot n_j\right) \tag{3}$$
$$=\min!$$

This can be transferred in a set of $\left(N_S+N_E\right)$ nonlinear equations:

$$\begin{aligned}\frac{\partial L}{\partial n_j}&=0=\mu_j+\sum_{i=i}^{N_E}a_{ij}\cdot\lambda_i && j=1,\dots,N_S\\ \frac{\partial L}{\partial\lambda_i}&=0=b_i-\sum_{j=1}^{N_S}a_{ij}\cdot n_j && i=1,\dots,N_E\end{aligned} \tag{4}$$

Nonlinear equation system (4) can be solved e.g. by application of the Newton algorithm.

The above introduced equations are only valid for constant temperature and pressure. But the equilibrium temperature of the gasification gas is unknown. Therefore the output temperature of the gasification gas is iteratively calculated by solving the energy balance equation:

$$\begin{aligned}\sum_k\dot{H}_{in,k}+H_{u,k}\cdot\dot{m}_{k,in}=\sum_j\dot{H}_{out,j}+H_{u,j}\cdot\dot{m}_{j,out}\\ +\Delta h_m\cdot\dot{m}_{ash,in}\end{aligned} \tag{5}$$

where k belongs to coal, gasification agent and additional input gases; j belongs to gasification gas, ash/slag and remaining fixed carbon. Furthermore H_u represents the lower heating value, $\dot{H}_{in}$ the entering enthalpy flow, and $\dot{H}_{out}$ the outgoing enthalpy flow. Δh_m is the melting enthalpy of the coal ash and $\dot{m}_{ash,in}$ the incoming coal ash mass flow rate.

3.2 Heat transfer

The heat transfer from the hot, particle loaded gasification gas to the slag layer is due to radiation and convection, whereupon the convective heat transfer can be neglected [4].

For calculation of radiative heat transfer the coupled gas and particle radiation has to be considered. Thereby CO, CO_2, CH_4 and H_2O are radiation absorbing gas components. Due to the fact that there is only less material about the calculation of the emission coefficients for CO under high pressure, CO is handled as CO_2 in the equations as *Fleischer* has done [5].

For the hot gasification process the radiation due to increased particle loading has to be regarded. Then the heat flux $\dot{Q}_{rad}$ owing to the coupled gas-particle radiation can be defined as [6]:

$$\begin{aligned}\dot{Q}_{rad}&=A\cdot\sigma\cdot\beta\cdot\left(\varepsilon_{G+P}\cdot T_G^4-\alpha_{G+P}\cdot T_{G\to S}^4\right)\\ \beta&=\frac{\varepsilon_S}{\alpha_{G+P}+\varepsilon_S-\alpha_{G+P}\cdot\varepsilon_S}\quad,\end{aligned} \tag{6}$$

where A is the heat transfer area, σ is the Boltzmann constant, T_G is the gas temperature, $T_{G\to S}$ is the surface temperature of the liquid slag layer, ε_S is the emission coefficient of the slag, ε_{G+P} and α_{G+P} are the emission and the absorption coefficient of the particle loaded gas, respectively. Modeling equations and parametric tables for ε_{G+P} and α_{G+P} can be found in *VDI Wärmeatlas* [6].

For the emission coefficient of slag the fixed value of $\varepsilon_S=0.83$ is assumed [7].

3.3 Helical tube

For calculation the heat flow due to water side convection the fluid flow conditions have to be known. With the Nusselt number Nu the heat transfer coefficient α can be calculated:

$$\mathrm{Nu}=\frac{\alpha}{\lambda}\cdot d_0 \tag{7}$$

where d_0 is the internal diameter of the pipe and λ is the thermal conductivity of the fluid.

Literature provides different equations for calculation of Nusselt numbers in helical tubes. An overview about some of them can be found in *Kumar et al* [8].

The following explanations refer to *VDI Wärmeatlas* [6].

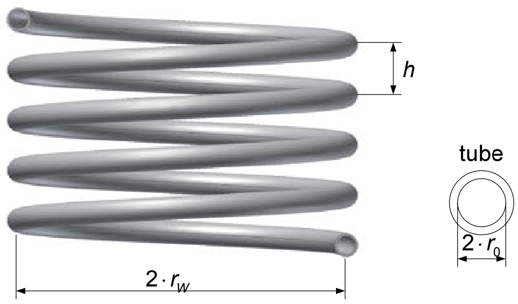


Figure 3: Helical tube

Figure 3 shows schematically a helical tube. The critical Reynolds number Re_{crit} to define the flow condition is defined as:

$$\mathrm{Re}_{crit} = 2300 \cdot \left[1 + 8.6 \cdot \left(\frac{d_0}{D} \right)^{0.45} \right] \tag{8}$$

D is the middle curve diameter of the helical tube.

For laminar flow conditions $\left(\mathrm{Re} \le \mathrm{Re}_{crit} \right)$ the Nusselt number is calculated as:

$$\mathrm{Nu}_l = \beta \cdot \left(\frac{\mathrm{Pr}}{\mathrm{Pr}_W} \right)^{0.14}$$

where:

$$\beta = \left(3.66 + 0.08 \cdot \left[1 + 0.8 \cdot \left(\frac{d_0}{D} \right)^{0.9} \right] \cdot \mathrm{Re}^m \cdot \mathrm{Pr}^{1/3} \right) \tag{9}$$

$$m = 0.5 + 0.2903 \cdot \left(\frac{d_0}{D} \right)^{0.194}$$

For turbulent flow conditions $\left(\mathrm{Re} \ge 2.2 \cdot 10^4 \right)$ the Nusselt Number is defined as:

$$\mathrm{Nu}_t = \frac{0.125 \cdot \xi \cdot \mathrm{Re} \cdot \mathrm{Pr}}{1 + 12.7 \cdot \sqrt{0.125 \cdot \xi} \cdot \left(\mathrm{Pr}^{2/3} - 1 \right)} \cdot \left(\frac{\mathrm{Pr}}{\mathrm{Pr}_W} \right)^{0.14}$$

$$\xi = \frac{0.3164}{\mathrm{Re}^{0.25}} + 0.03 \cdot \left(\frac{d}{D} \right)^{0.5} \tag{10}$$

And for the transition zone $\left(\mathrm{Re}_{crit} < \mathrm{Re} < 2.2 \cdot 10^4 \right)$:

$$\mathrm{Nu} = \eta \cdot \mathrm{Nu}_l \left(\mathrm{Re}_{crit} \right) + \left(1 - \eta \right) \mathrm{Nu}_t \left(\mathrm{Re} = 2.2 \cdot 10^4 \right)$$

$$\eta = \frac{2.2 \cdot 10^4 - \mathrm{Re}}{2.2 \cdot 10^4 - \mathrm{Re}_{crit}} \tag{11}$$

3.4 Slag properties

Coal slag is a multi-phase system. The main components are SiO_2, CaO, MgO, Fe_2O_3 and Al_2O_3.

To implement a slag building model the physical properties of the slag such as thermal conductivity or viscosity must be known. Most of the physical properties are dependent on temperature and composition of the coal ash.

3.4.1 Slag Viscosity

There are a lot of empirical viscosity models obtainable from literature. A summary of these models can be found in *Vargas et al* [9]. At this point only the *Kalmanovitch-Frank Model* shall be shortly introduced, because this model reflects the viscosity of coal slags with sufficient accuracy [9][10].

The *Kalmanovitch-Frank Model* is based on the Weymann-Correlation:

$$\log \eta = \log a + \log T + b / T \tag{12}$$

For calculation of the parameters *a* and *b* slag components were classified into glass builder (x_g), glass modifier (x_m) and amphoterics (x_a):

$$x_g = \zeta_{\mathrm{SiO_2}} + \zeta_{\mathrm{P_2O_5}}$$

$$x_m = \zeta_{\mathrm{FeO}} + \zeta_{\mathrm{CaO}} + \zeta_{\mathrm{MgO}} + \zeta_{\mathrm{Na_2O}} + \zeta_{\mathrm{K_2O}} + \zeta_{\mathrm{MnO}} + \zeta_{\mathrm{NiO}} + 2\left(\zeta_{\mathrm{TiO_2}} + \zeta_{\mathrm{ZrO_2}} \right) + 3\zeta_{\mathrm{CaF_2}}$$

$$x_a = \zeta_{\mathrm{Al_2O_3}} + \zeta_{\mathrm{Fe_2O_3}} + \zeta_{\mathrm{B_2O_3}}$$

where ζ_i is the mass fraction of component *i*.

With these mass fractions the parameters *a* and *b* can be computed as:

$$b = 10^3 \cdot \left(b_0 + b_1 \cdot \zeta_{\mathrm{SiO_2}} + b_2 \cdot \zeta^2_{\mathrm{SiO_2}} + b_3 \cdot \zeta^3_{\mathrm{SiO_2}} \right)$$

$$a = \exp\left(-0{,}2812 \cdot 10^{-3} \cdot b - 14{,}1305 \right)$$

With:

$$b_0 = 13.8 + 39.9355 \cdot \alpha - 44.049 \cdot \alpha^2$$

$$b_1 = 30.481 - 117.1505 \cdot \alpha + 129.9978 \cdot \alpha^2$$

$$b_2 = -40.9429 + 234.0486 \cdot \alpha - 300.04 \cdot \alpha^2$$

$$b_3 = 60.7619 - 153.9276 \cdot \alpha + 211.1616 \cdot \alpha^2$$

$$\alpha = \frac{x_m}{x_m + x_a}$$

3.4.2 Thermal conductivity

The thermal conductivity of slag is one of the physical properties with the largest influence on the heat flow rate through the slag layer [11]. Literature shows only some mathematical models available for the calculation of the thermal conductivity.

Here the following mathematical model which was also used by *Seggiani* [12] was implemented:

$$\lambda_S = \alpha \cdot c_p \cdot \rho$$

with:

$$\begin{aligned} \alpha &= 4.5 \cdot 10^{-7}\ \mathrm{m^2/s} \\ c_p &= 1100\ \mathrm{J/(m\ K)} \\ \rho &= 2500\ \mathrm{kg/m^3} \end{aligned} \tag{13}$$

4 Implementation of the Model in Modelica/Dymola

4.1 Model development

As base for the modelling of heat flux through the cooling-screen of the SFGT-Gasifier the Modelica Fluid 1.0 Library connectors were used. The components for modelling a gasifier do not exist in a Modelica library. This extension was developed.

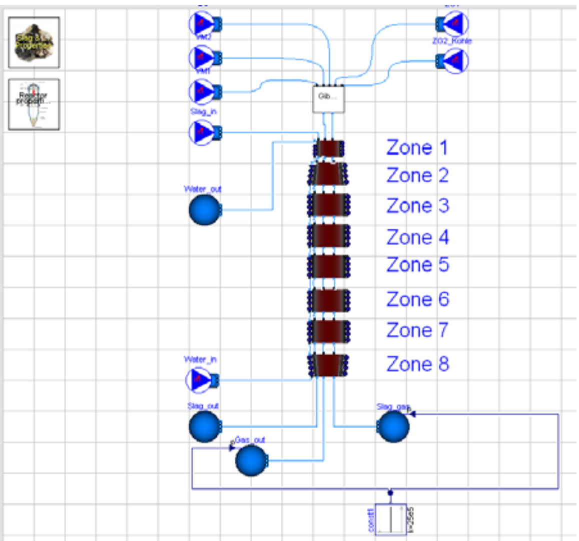


Figure 4: Implementation of the SFGT-Gasifier model in Modelica/Dymola

The gasification section including the cooling screen was modelled by division into several zones (Figure 4). In the first zone the thermo-chemical equilibrium is calculated by minimization of Gibbs free energy. Therefore, a Dynamic Link Library (DLL) was implemented in C and was inserted into the Modelica model as an external function.

In the following zones slag layer thickness and heat transfer from the hot raw gas to the cooling water (heat flux zones) are calculated. The number of heat flux zones depends on the size of the gasifier.

Furthermore, two system components have to be included in the simulation model. The system component "slag" comprises the composition of slag and coal in order to calculate the slag properties. The "slag" component provides also the opportunity to include experimentally determined correlations for slag properties.

In the component "reactor" the dimensions of the gasifier like diameters of pipes and the properties of wall materials are configured.

Each heat flux zone is built up of 3 sections (Figure 5): the gas compartment, the solid materials (liquid and solid slag layer, castables layer and helical tube material) and the cooling water. Between these sections occur heat and mass transfer as shown in Figure 6. For each section the energy and mass conservation equations are solved, the momentum conservation equations are neglected.

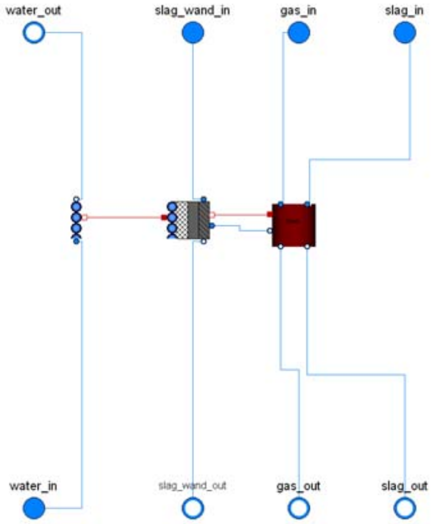


Figure 5: Implementation of one heat flux zone in Modelica/Dymola

It has to be noted that the composition of the gas leaving the last heat flux zone does not belong exactly to the equilibrium composition at the outlet temperature. But the differences in the equilibrium composition for the equilibrium state with and without heat loss, respectively, are only small due to the high temperatures.

4.2 Gas compartment

The gas compartment is assumed as a continuously stirred-tank reactor. The following mass balances are regarded:

$$\frac{\mathrm{d}\, m_{G,i}}{\mathrm{d}\, t} = \dot{m}_{G,in,i} + \dot{m}_{G,out,i} \tag{14}$$

$$\frac{\mathrm{d}\, m_{S,i}}{\mathrm{d}\, t} = \dot{m}_{S,in,i} + \dot{m}_{S,out,i} + \dot{m}_{S,wall,i} = 0 \tag{15}$$

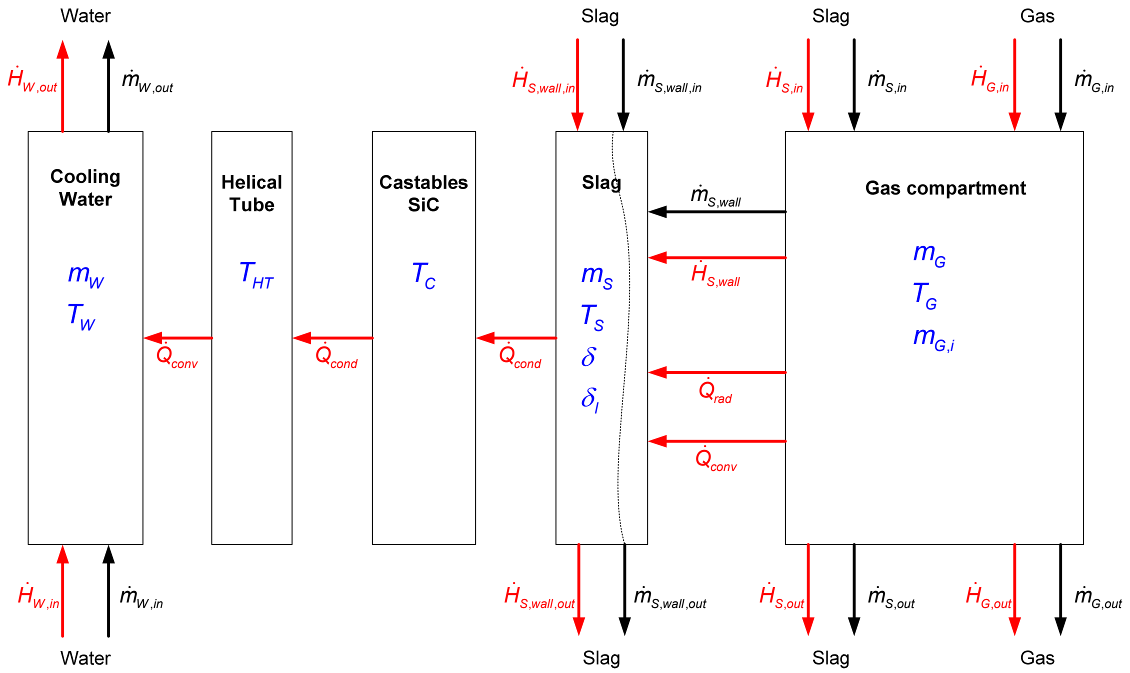


Figure 6: Heat and mass flow for one heat flux zone

where $m_{G,i}$ is the mass of gasification gas, $m_{S,i}$ is the mass of slag, $\dot{m}_{G,in,i}$ and $\dot{m}_{G,out,i}$ are the incoming and leaving gas mass flow rate and $\dot{m}_{S,in,i}$ and $\dot{m}_{S,out,i}$ are the incoming and leaving slag mass flow rate.

Equation (15) means no slag storage in the gas compartment. The fraction of incoming slag mass flow rate accumulating at liquid slag layer $\dot{m}_{S,wall,i}$ can be specified by the user.

For the energy balance of the gas compartment in addition to the in- and out-flowing streams the gas radiation heat flow $\dot{Q}_{G\to S,i}$ has to be considered:

$$\frac{\mathrm{d}U_{G,i}}{\mathrm{d}t} = \dot{H}_{S,in,i} + \dot{H}_{S,out,i} + \dot{H}_{S,wall,i} \qquad (16)$$

$$+\dot{H}_{G,in,i} + \dot{H}_{G,out,i} + \dot{Q}_{G\to S,i}$$

For the calculation of specific enthalpy for the slag mass flow rate accumulating at liquid slag layer the gas temperature is assumed.

4.3 Solid materials

4.3.1 Slag Layer

For the implementation of the slag layer modelling, assumptions of the slag building model by *Reid and Cohen* [12] were used:

(1) The transition temperature between the solid and the liquid slag layer is the temperature of critical viscosity.

(2) The flow of liquid slag is of Newtonian type and the flow at temperatures below T_{CV} is negligible.

(3) The shear stress between gas and slag layer is negligible.

(4) The temperature profile across the slag layer is linear.

(5) The heat transfer occurs perpendicularly to the surface.

(6) The model is written in linear coordinates, owning to a large difference between slag deposit thickness and gasifier radius.

(7) The density, specific heat and thermal conductivity of slag are independent on temperature.

Mass balance for the slag:

$$\frac{\mathrm{d}m_{S,i}}{\mathrm{d}t} = \dot{m}_{S,in,i} + \dot{m}_{S,out,i} + \dot{m}_{S,wall,i} \qquad (17)$$

where $m_{S,i}$ is the slag mass, $\dot{m}_{S,in,i}$ is the incoming slag mass flow rate, $\dot{m}_{S,out,i}$ is the discharging slag mass flow rate and $\dot{m}_{S,wall,i}$ is the mass flow rate impacting on the liquid slag layer.

The discharging mass flow rate is calculated due to the assumption that the slag can be considered as a Newtonian fluid. Then the weight F_w equals the friction force F_f in steady state:

$$F_f = F_w$$

Hence:

$$F_f = \eta \cdot A \cdot \frac{\mathrm{d}u_y}{\mathrm{dx}} = b \cdot h \cdot \eta \cdot \frac{\mathrm{d}u_y}{\mathrm{d}x} \qquad (18)$$

$$F_w = m \cdot g = (\delta_l - x) \cdot b \cdot h \cdot \rho \cdot g$$

where η is the viscosity, A is the area, b is the length of the slag layer, h is the height of the slag layer, u_y is the velocity in vertical direction, δ_l is the thickness of the liquid slag layer, m the mass of the slag, ρ the density of the slag and x the horizontal position.

Hence, the change in velocity at each horizontal location x can be defined as:

$$\frac{\mathrm{d}u_y}{\mathrm{d}x} = \frac{(\delta_l - x) \cdot \rho \cdot g}{\eta(T(x))}.$$

By integration of this equation under the boundary condition that the velocity at the boundary layer between liquid and solid slag layer equals zero the equation for velocity results to:

$$u_y(x) = \frac{\rho_S \cdot g}{\eta(T(x))} \cdot \left(\delta_{l,i} \cdot x - \frac{x^2}{2} \right). \qquad (19)$$

So the discharging slag mass flow rate can be calculated as:

$$\dot{m}_{S,out,i} = \rho_S \cdot b_i \cdot \int_{x=0}^{\delta_{l,i}} u_y(x) \mathrm{d}x. \qquad (20)$$

The thickness of the liquid slag layer is estimated under the assumption of linear temperature distribution as:

$$\delta_{l,i} = 0.5 \cdot \frac{T_{G \to S,i} - T_{crit}}{T_{G \to S,i} - T_{S,i}} \cdot \delta_i. \qquad (21)$$

Where $T_{G \to S,i}$ is the surface temperature of the liquid slag layer, $T_{S,i}$ is the middle slag layer temperature, T_{crit} is the temperature of critical viscosity and δ_i is the thickness of the slag layer.

Energy conservation equation for the slag

$$\frac{\mathrm{d}U_{S,i}}{\mathrm{d}t} = \dot{H}_{S,in,i} + \dot{H}_{S,out,i} + \dot{H}_{S,wall,i} + \dot{Q}_{G \to S} + \dot{Q}_{S \to C}$$

where for the temperature of discharging slag $T_{S,out,i}$ the middle temperature of the liquid slag layer is assumed. The heat flux from the slag layer to the castables layer is defined as:

$$\dot{Q}_{S \to C} = \frac{T_{S,i} - T_{C,i}}{R_{\lambda,i}}$$

$$\text{with:} \quad R_{\lambda,i} = \frac{1}{A_i} \cdot \left(\frac{0.5 \cdot \delta_i}{\lambda_S} + \frac{0.5 \cdot \delta_C}{\lambda_C} \right), \qquad (22)$$

where $T_{C,i}$ is the middle temperatures of the castables layer, δ_C is the thickness of the castables layer and λ_C is the thermal conductivity of the castables layer.

4.3.2 Castables layer and helical tube material

For the castables layer and the helical tube material only the energy conservation equations have to be considered:

$$\frac{\mathrm{d}E_C}{\mathrm{d}t} = \dot{Q}_{S \to C} + \dot{Q}_{C \to HT}$$

$$\frac{\mathrm{d}E_{HT}}{\mathrm{d}t} = \dot{Q}_{C \to HT} + \dot{Q}_{HT \to W} \qquad (23)$$

The heat flux from the castables layer to the helical tube material is defined as (linear temperature distribution):

$$\dot{Q}_{C \to HT} = \frac{T_{C,i} - T_{HT,i}}{R_{\lambda,i}}$$

$$\text{with:} \quad R_{\lambda,i} = \frac{1}{A_i} \cdot \left(\frac{0.5 \cdot \delta_{HT}}{\lambda_{HT}} + \frac{0.5 \cdot \delta_C}{\lambda_C} \right), \qquad (24)$$

where $T_{HT,i}$ are the middle temperatures of helical tube material, δ_{HT} is the thickness of helical tube material and λ_{HT} is the thermal conductivity of helical tube.

4.4 Cooling water

The cooling water in the helical tube in each heat transfer zone is implemented as a water volume with a heat port. The characteristic flow numbers are calculated due to the actual flow conditions. Then the heat transfer coefficient α_i for the convective heat transfer rate is calculated in a separate function. The following heat and mass balance equations were implemented for each water volume:

$$\frac{\mathrm{d}m_{W,i}}{\mathrm{d}t} = \dot{m}_{W,in,i} + \dot{m}_{W,out,i}$$

$$\frac{\mathrm{d}U_{W,i}}{\mathrm{d}t} = \dot{H}_{W,in,i} + \dot{H}_{W,out,i} + \dot{Q}_{HT \to W} \qquad (25)$$

where the heat flow rate from the helical tube to the cooling water is calculated as:

$$\dot{Q}_{HT \to W} = \alpha_i \cdot A_i \cdot (T_{HT \to W} - T_W) \qquad (26)$$

where $T_{HT \rightarrow W}$ is the surface temperature of the helical tube material and T_W is the temperature of the cooling water.

5 Simulation results

The Modelica/Dymola model could be steady state and transient validated with data of the Siemens test facility located in Freiberg.

Therefore, the input streams (e.g. coal mass flow rate, temperatures, gasification agent mass flow rate…) of the test facility were loaded to the model as a Modelica TimeTable.

As shown in Figure 7 the model provides good correlation with the test data for different evaluation points and various kinds of coal. By means of Figure 7 it can also be shown that the *Kalmanovitch-Frank Model* for calculation of the slag viscosity provides sufficient agreement for calculation of heat flux compared to experimentally determined viscosity.

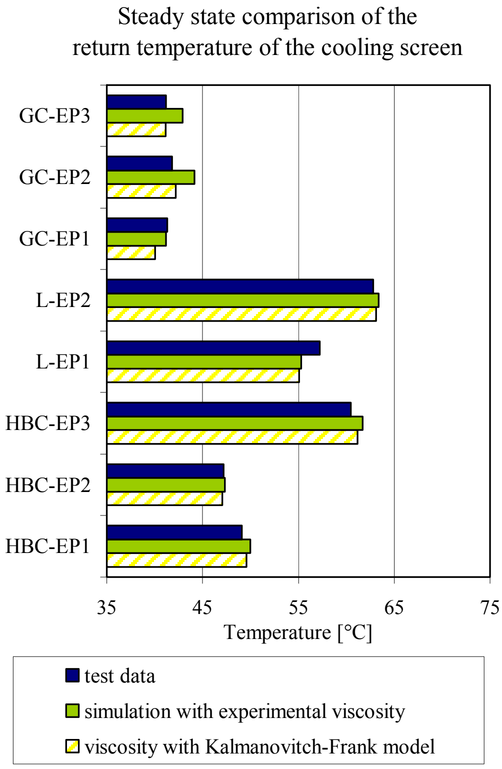


Figure 7: Steady state validation of the model with test data of the Siemens test facility located in Freiberg (In the figure GC means Gas Coal, L belongs to Lignite and HBC to Hard Brown Coal.)

Figure 8 shows the developing of the return temperature of the cooling screen cycle water for a break down of the coal mass flow at time 197 min.

For the regular operation the difference between the simulated and the measured temperature are mostly less than 2 K. As the coal mass flow breaks down and the gasifier operates only with gaseous fuel the difference increases up to 5 K. The cause of this is the calculation of the absorption coefficients for the gas components due to the fact that for the above case the slag layer surface temperature is above the area of validity for these equations. So the value for the absorption coefficient is oversized compared to the emission coefficient. Hence, the heat flow rate from the gas to the slag layer is underestimated.

6 Conclusions

In the article the modelling of the heat flux through the slag coated cooling screen of the SFGT-Gasifier was shown. It could be demonstrated that the developed model reflects the test facility data both steady state and transient with sufficient precision.

The next step will be the scale up of the model to the industrial plant.

References

[1] Klose, E.; Toufar, W.: Grundlagen der Vergasung, 1. Lehrbrief. Lehrbriefe für das Hochschulfernstudiun, 1985

[2] Higman, C.; van der Burgt, M.: Gasification. Gulf Professional Publishing, Amsterdam, 2002

[3] Smith, W.R.; Missen, R.W.: Chemical Reaction Equilibrium Analysis: Theory and Algorithms. John Wiley and Sons, 1982

[4] Brummel, H.-G.; Kakara, E.: Wärmestrahlungsverhalten von Gas-/Feststoffgemischen bei niedrigen, mittleren und hohen Staubbeladungen. In: Wärme- und Stoffübertragung 25 (1990), 129-140

[5] Fleischer, Thomas: Erarbeitung eines Modells zur Berechnung der Wärmeübertragung auf die Kühlschirmwand unter Berücksichtigung der Schlackeeigenschaften. Freiberg, Bergakademie, Fachbereich Maschinen-, Verfahrens- und Energietechnik. Diploma thesis, 2007

[6] Verein Deutscher Ingenieure, VDI-Gesellschaft Verfahrenstechnik und Chemieingenieurwesen (Hrsg.): VDI-Wärmeatlas.

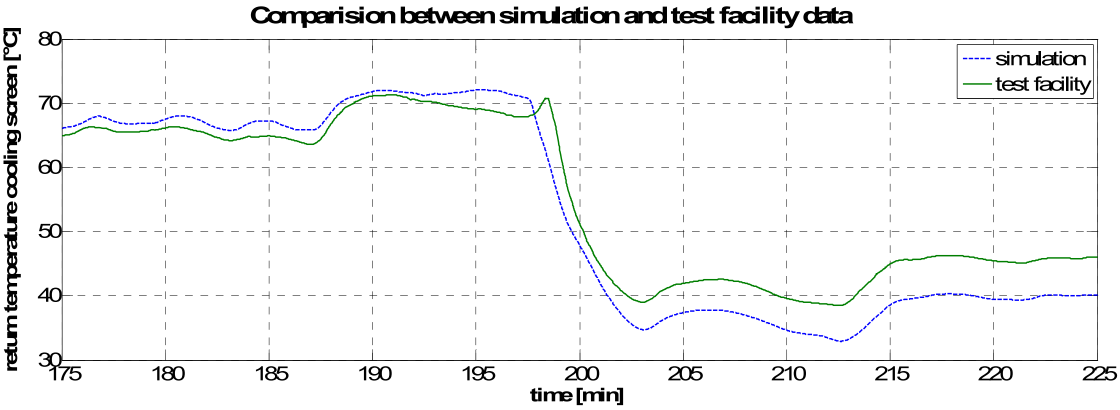


Figure 8: Comparison between simulation and test data for the return temperature of the cooling screen due to the break down of the coal mass flow

Zehnte, bearbeite und erweiterte Aufl. Berlin, Heidelberg: Springer, 2006

[7] Zbogar, A. et al.: Heat transfer in ash deposits: A modelling tool-box. In: Progress in Energy and Combustion Science 31 (2005), S. 371-421

[8] Kumar, V. et al.: Pressure drop and heat transfer study in tube-in-tube helical heat exchanger. In: Chemical Engineering Science 61 (2006), S. 4403-4416

[9] Vargas, S. et al.: Rheological properties of high-temperature melts of coal ashes and other silicates. In: Progress in Energy and Combustion Science 27 (2001), S. 237-429

[10] Hannemann, F. et al.: Application of Siemens Fuel Gasification Technology for different types of coal. 25th Annual Pittsburgh Coal Conference, Pittsburgh, PA, USA, September 29 – October 2, 2008

[11] Rezaei, H.R. et al.: Thermal conductivity of coal ash and slags and models used. In: Fuel 79 (2000), S. 1697-1710

[12] Seggiani, M: Modelling and simulation of time varying slag flow in a Prenflo entrained-flow gasifier. In: Fuel 77 (1998), Nr. 14, S. 1611-1621

Modelling of complex thermal energy supply systems based on the Modelica-Library *FluidFlow*

Manuel Ljubijankic[1] Christoph Nytsch-Geusen[1,2] Steffen Unger[2]
[1]Institute for Architecture and Urban Design, University of Arts Berlin
Hardenbergstraße 33, 10623 Berlin, Germany
nytsch@udk-berlin.de
[2]Fraunhofer Institute for Computer Architecture and Software Technology
Kekuléstr. 7, 12489 Berlin, Germany

Abstract

The new Modelica library *FluidFlow* is being developed for the thermo-hydraulic simulation of complex energy supply systems. This library includes standard hydraulic model classes and specialized components for HVAC-systems and solar thermal systems. Most of these Modelica classes are modelled with equations of the 1D-transient energy transport. The validation of the library takes place both by measuring values from test stations and by comparing with detailed CFD models. A first complex use case of the library represents the simulations-based design of a complex thermal energy supply system of a residential area, as a part of a newly built city in Iran.

Keywords: thermo-hydraulic simulation; validation with CFD; modelling of complex energy supply systems

1 Introduction

In the last years, different Modelica-libraries for the hydraulic and thermo-hydraulic simulation were developed [1, 2]. From our point of view, these libraries are not well suited for the modelling of very complex thermal energy supply systems, because their structure are either too complex within their single components or do not include a lot of the required specialized models. For this reason, the authors decided to develop a new Modelica-library for thermo-hydraulic network simulation, which is called *FluidFlow* [3].

2 Modelica library *FluidFlow*

The present main application field of the *FluidFlow*-library is the modelling of solar thermal systems, HVAC (Heating, Ventilation and Air-Conditioning)-systems and district heating/cooling systems.

2.1 Library structure

The *FluidFlow*-library comprises thermo-hydraulic models and purely hydraulic models. The skeletal structure of the library consists of a set of “ready-to-use” standard hydraulic models, such as pipes, elbows, distributors and pumps. These models are built on variably specialized “partial”-Modelica classes - e.g. for pressure loss calculations, heat transport, model interfaces design - by the intensive use of the object-oriented modelling technique.
In addition to the standard components, the library includes more specialized models from several domains (compare with Figure 1), such as solar thermal technology (collector models), thermal storage technology (storage models) or energy transformation technologies (e.g. models of heat exchangers, absorption chillers and cogeneration plants).

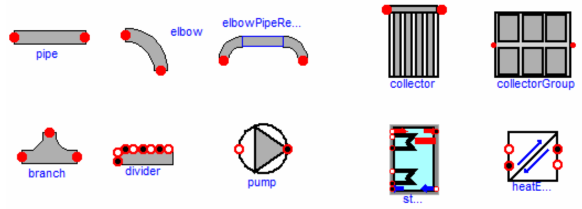


Figure 1 Standard models (left) and specialized models (right) of the thermal-hydraulic library FluidFlow

DOI: 10.3384/ecp09430070

2.2 Physical models

All the hydraulic models of the *FluidFlow*-library are based on a stationary pressure loss-calculation, which depends on the component type, its individual parameters and the present flow conditions. In addition, the thermo-hydraulic component models have transient thermal models both for the 1D-convective energy transport in the flow direction and for the heat transfer to the environment. Some of the models such as the thermal storage also have models for the diffuse and turbulent heat transport within in the fluid. The *FluidFlow*-library also supports reverse flow. So the flow directions within the modelled systems only depend on the external boundary conditions of the corresponding thermo-hydraulic network and the induced pressure values or mass flows of the net-integrated pumps. All the models of the *FluidFlow*-library can be used with or without the *Modelica.Media*-library.

2.3 Validation

The validation of the single thermo-hydraulic components and system models takes place in two different ways. The first method represents the traditional validation with measurement values, used from thermo-hydraulic test stations from the Technical University of Berlin [4]. At the moment the validation of the component models such as thermal water storages with and without internal heat exchangers, external plate heat exchangers, pipes, solar thermal collectors and also of the system model of a solar thermal plant is taking place.

The second method consists of the comparison of the simplified 1D-Modelica models with detailed 3D-models, which are based on Computational Fluid Dynamics (CFD) calculations [5].

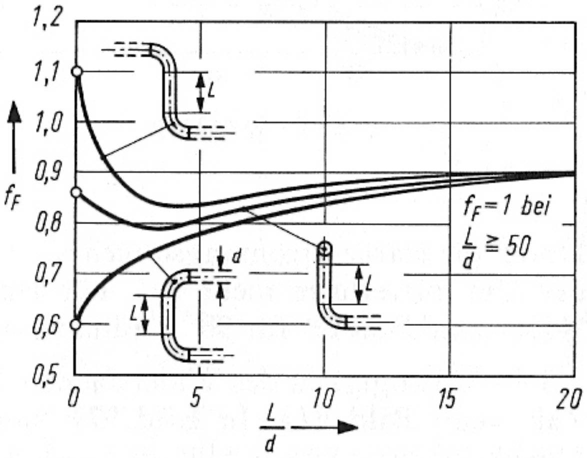


Figure 2 Reduction factor f_F for the total pressure loss for the configuration "elbow – pipe –elbow" in dependency of the length L of the intermediate pipe [6]

One result of this approach is the improvement of the accuracy of the thermo-hydraulic "group-behaviour" of several Modelica component models, which are connected to a system model.

Therefore, we analysed in a first step the pressure loss of a small hydraulic system with three serial connected components – an elbow, a straight pipe and a second elbow with the geometry in Figure 2 down on the left side: if the length *L* of the intermediate pipe is relatively short in comparison to its diameter *d*, then the impact of pressure loss from both elbows on the pressure loss of the pipe is considerable. In this case, the total pressure loss of the three components is smaller than the sum of the single pressure losses of each component (expressed with the reduction factor f_F), because the equations used for the pressure loss calculation in the Modelica models assume for each of the three components an undisturbed flow profile at the inlet and outlet, which here does not exist:

$$\Delta p_{total} = f_F \cdot \left(\Delta p_{elbow1} + \Delta p_{pipe} + \Delta p_{elbow2}\right) \quad (1)$$

But in the case of a relative long intermediate-pipe (values *L/d* ≥ 50), the sum of the single pressure losses approximates the total pressure loss of the small hydraulic system (compare with Figure 2) and f_F becomes to 1.

First, we modelled the described hydraulic system with the CFD-tool ANSYS CFX 11.

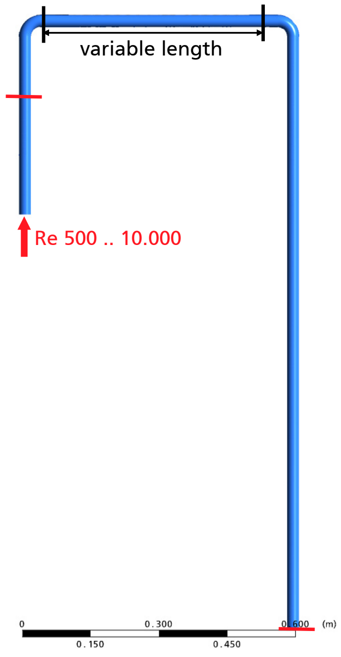


Figure 3 3D-CFD-model of two 90° elbows with an intermediate pipe with variable length

In the CFD-model we add two pipes for an additional inlet and outlet stretch, to have an undisturbed flow profile at the red marked cross section on the left side and only a small influence from the second elbow on flow profile at the red marked outlet cross section on the right side (compare with Figure 3). Then we calculated the total pressure loss between both red marked cross sections. We did also a further CFD-calculation for a straight pipe with the same *L/d*-value. The difference between the total pressure losses of both calculations are induced only by the elbows forcing a flow direction change. We did these calculations for Reynolds-numbers between 500 up to 10,000 and *L/d*-values between 0 up two 50 and deduced a correction function $f_F=f(L/d)$ (compare with Figure 4).

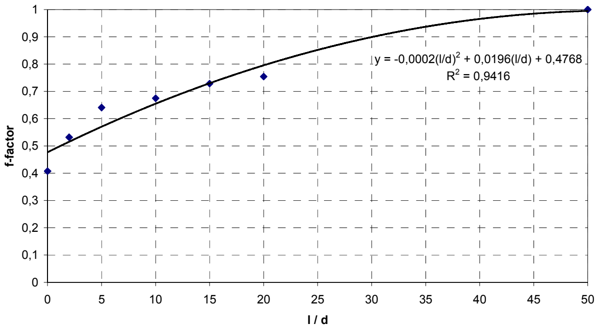


Figure 4 CFD-deduced correction function for the component group "elbow – pipe – elbow" (mean values for Reynolds numbers from 500 to 10,000)

Figure 5, the drawing in the left, shows a "conventional" Modelica configuration of a hydraulic loop, based on single independent components. Figure 5, the drawing in the right, demonstrates the same loop with a merged component, which takes into account the strong hydraulic dependencies between two elbows due to a relatively short intermediate pipe with the help of the correction function.

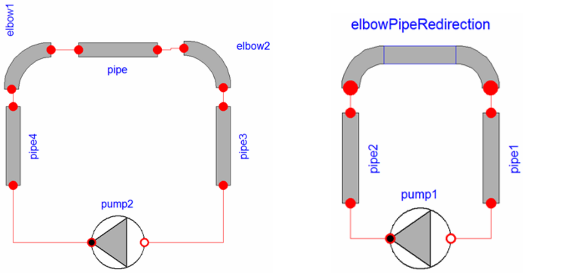


Figure 5 Configuration of a hydraulic loop with single components (left) and a "compound-component" (right)

Figure 6 shows the calculated pressure loss of the hydraulic loop (Reynolds number = 5,000), based on single Modelica component models and with a compound-component, where the calculated pressure loss is modified by the use of the new correction function. For small values of *L/d* the "real" pressure loss lays 5 up to 15 percent lower than a pressure loss calculation without a correction function.

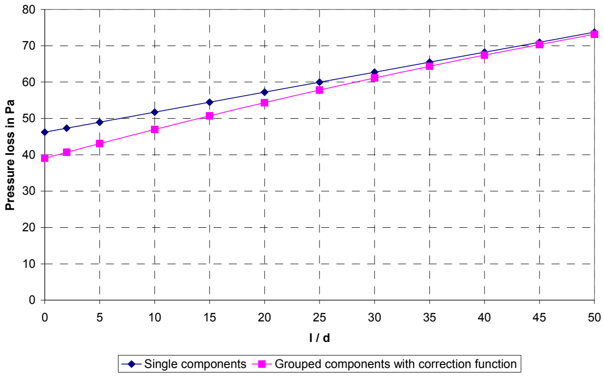


Figure 6 Calculated pressure loss of the hydraulic loop with single components and with a compound-component with the correction function (Reynolds number = 5,000)

3 Use case: modelling of a thermal energy supply system of a district

The newly developed *FluidFlow*-library is being used and evaluated within the research project "Young Cities - Developing Energy-Efficient Urban Fabric in the Tehran-Karaj Region" [7].

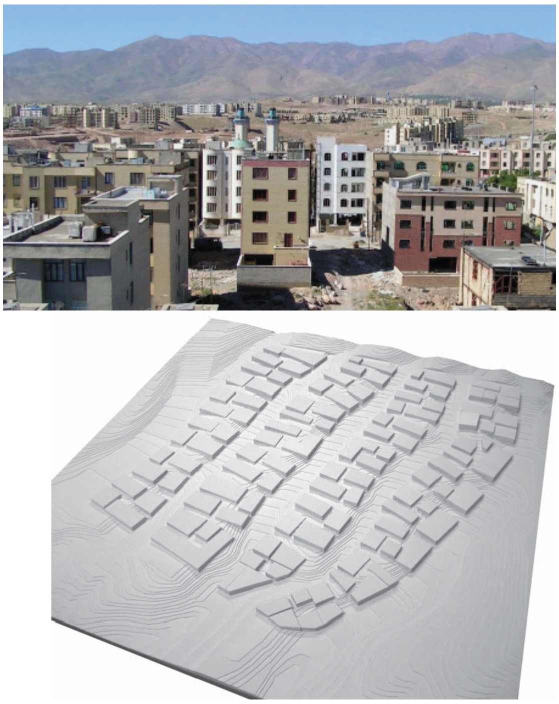


Figure 7 New Town Hashtgerd (Iran) and urban planning model of the 35 ha pilot area for 8,000 inhabitants

Here, the Modelica library is used for the simulation-based design of complex (thermal) energy supply systems for a 35 ha residential district as a part of the newly built Iranian city Hashtgerd with 2,000 accommodation units for 8,000 inhabitants.

For the determination of the approximate size and the boundary conditions for a suitable energy supply system, a first estimation of the heat and cold demand for the 35 ha residential area, based on an early design of the building typologies from the architects and urban planners was performed.

Using the geometry, construction and building materials of a typical three-storey row house of 33 m depth and 7.5 m width, the energy demand for a single building was estimated by using the program CASANOVA [8] with climate data for Karaij. Because energy efficiency is one of the main targets of the "Young Cities"-project, the U-values for the walls, roofs, basement ceilings and windows were chosen to obtain a total thermal energy demand for heating and cooling of 50 percent relative to the limits of the "Code 19"-building Iranian energy standard [9]. The energy demand of the whole 35 ha district was projected as the product of the specific energy demand of the single building and the planned total living area of the district (compare with Figure 8).

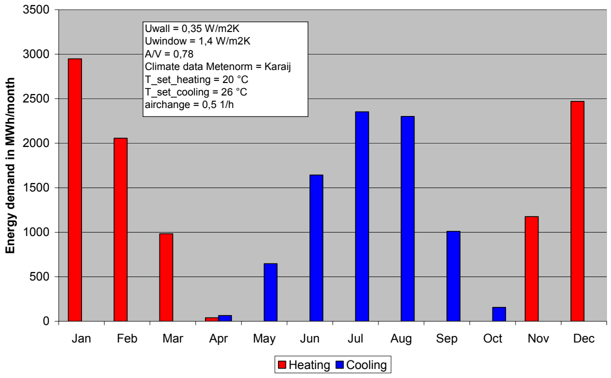


Figure 8 Projected monthly heating and cooling demand for the 35 ha district (50 percent of the Code 19)

To estimate the energy demand, a nominal room temperature of 20°C in heating periods and of 26°C in cooling periods was presumed, in addition to a ventilation rate of 0.5 h^{-1}. During cooling periods sunscreens were introduced, leading to a 50 percent shadowing of the windows. Based on these assumptions the heating period reaches from November to March and the cooling period from May to October. In April the climate in Karaij is balanced well enough that neither cooling nor heating is necessary.

In consideration of these boundary conditions different concepts of the thermal energy supply are being developed. These can have central, semi-central or de-central characteristics. Figure 9 shows an option with central heat supply and decentralised cooling production, based on solar energy. Therefore, this version contains one central thermal distribution network and many separate decentralised absorption chillers and solar thermal collector fields. Because the local solar irradiation of Hashtgerd is approximately 1,900 kWh/m^2a, active solar cooling can be attractive besides passive cooling for building climatisation with a minimum of primary energy.

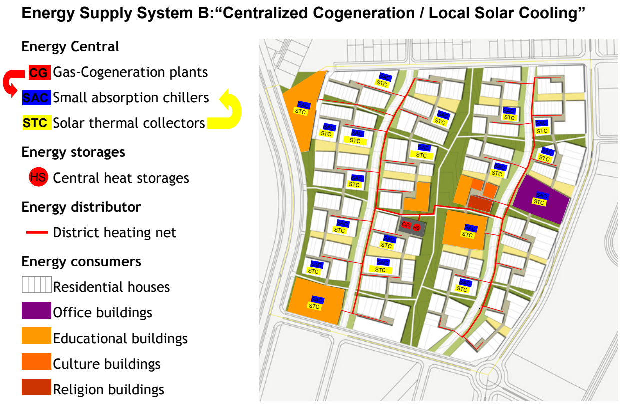


Figure 9 Energy concept of a thermal energy supply system for a 35 ha district in Hashtgerd (Iran)

The application of the *FluidFlow*-library shall be demonstrated by the modelling of the energy system of Figure 9. In this case, only the left part of the energy system is represented in the system model, illustrated in Figure 10.

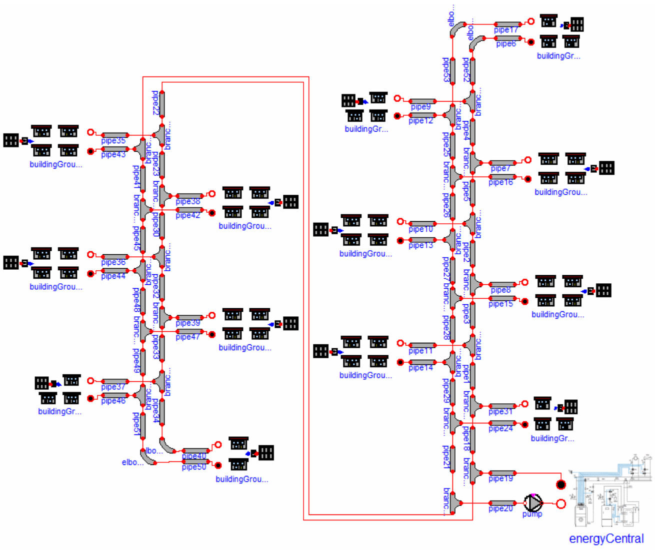


Figure 10 System model of the left part of the thermal energy supply system for the 35 ha district (variant centralised cogeneration/local solar cooling)

The modelled subsystems are the energy central (co-generation plant with a peak boiler and a thermal storage), the district heating net for the heating and warm-water demand and the thermal consumers, modelled as groups of simplified thermal building models, including the decentralised solar cooling systems (compare Figure 10, 11 and 12). The number of equations of the (non symbolic-reduced) system model of the energy supply system amounts to more than 30,000.

Figure 11 shows a sub-model for a building group with two building models. Here, each building model represents a bar of row houses, which are supplied with hot water from the central energy station and with cold water from the decentralised solar cooling system. The back-up thermal energy for the solar cooling system is also provided by the centralized thermal energy supply. Each of the building models has two controllers to adapt the mass flow through the heat exchangers for the respective heat or cooling demand at the moment. The modelling approach for the building models is strongly simplified to reduce the number of equations: The building model takes into account one thermal zone, only separates outer and inner positioned thermal masses and solely calculates the passive solar gains and shading devices for four main orientations [10].

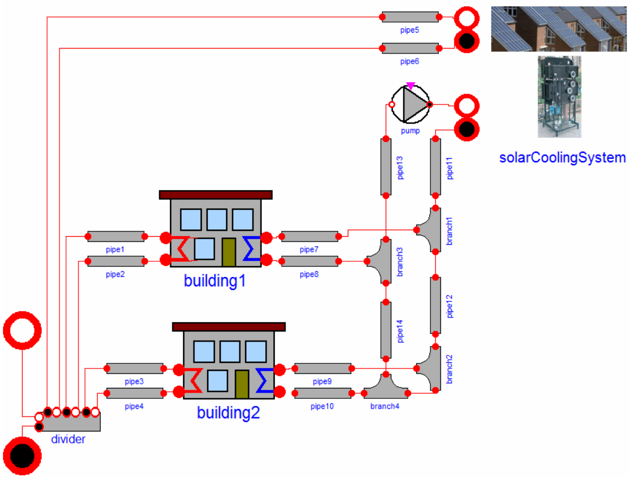


Figure 11 Sub-model of a building group with a decentralised solar cooling system

Figure 12 illustrated the model structure for the decentralised solar cooling system. The most important component of this sub-system model is a small-scale absorption chiller, which can provide some residential houses with cooling energy [11]. At this, the main part of its thermal operating power comes from the thermal storage, which is loaded by the solar collector field. If the absorption chiller needs additional thermal energy from the back-up system, it is transferred by a plate heat exchanger.

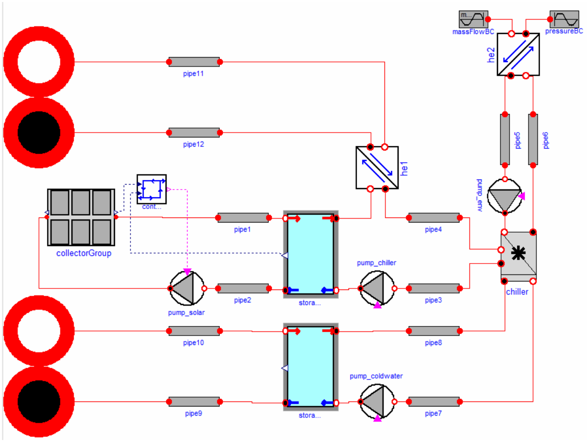


Figure 12 Sub-model of the solar cooling system with an absorption chiller

The thermal waste energy is delivered to the environment by a separate thermal circulation. The produced cold water is stored in further thermal storages, from where the building is provided with cooling energy.

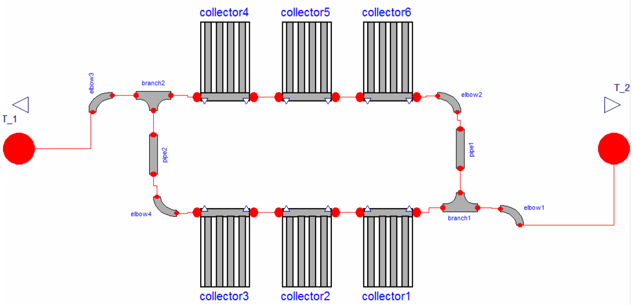


Figure 13 Sub-model of a collector field

Figure 13 shows the sub-model for a collector field. Six solar thermal collectors are serial-parallel connected with the use of the basic components of the *FluidFlow*-library. All the collector fields are integrated in the envelopes of the air-conditioned buildings.

4 Conclusion

The newly developed Modelica library *FluidFlow* is being used the first time to model a very complex thermal energy system. Here, the great challenge consists in an enough detailed modelling for all parts of the system model and a simultaneous limited number of model equations, which enables yearly simulation analysis. So, the next steps of our research activities will aim on the reduction of the complexity of the system model without a too large loss of model accuracy. For this purpose, energy system models with a different level of simplification shall be developed with the help of the *FluidFlow*-library and compared in terms of their accuracy and numerical effort.

References

[1] S. Fabricius, E. Badreddin: Modelica Library for Hybrid Simulation of Mass Flow in Process Plants. QSSFluidFlow. Proceedings 2th International Modelica Conference. DLR, München.18.-19. März 2002.

[2] F. Casella et al.: The Modelica Fluid and Media library for modeling of incompressible and compressible thermofluid pipe networks. Proceedings 5th International Modelica Conference. Arsenal Research, Wien. 4.-5. September 2006.

[3] M. Ljubijankic, C. Nytsch-Geusen: Thermo-hydraulische Simulation solar-thermischer Systeme mit Modelica. In Proceedings: 18. Symposium Thermische Solarenergie, OTTI-Technologiekolleg, Regensburg, 2008.

[4] C. Nytsch, M. Poli, T. Schneider: Messtechnische Untersuchungen an einer solarthermischen Versuchsanlage zur Validierung der solartechnischen Modelle der Simulationsumgebung SMILE. In Proceedings: 10. Symposium Thermische Solarenergie in Staffelstein, OTTI-Technologiekolleg, Regensburg, 2000.

[5] M. Ljubijankic, C. Nytsch-Geusen: Combining different levels of detail in modelling for an improved precision of HVAC plant simulation. In Proceedings: Building Simulation 2009, International Building Performance Simulation Association, Glasgow, 2009.

[6] W. Wagner: Strömung und Druckverlust. 6., bearbeite Auflage, Vogelverlag, 2008

[7] Homepage of the research project “Young Cities”: http://www.youngcities.org

[8] B.M. Kari, R. Fayaz: Evaluation of the Iranian Thermal Building Code, in: Asian Journal of Civil Engineering (Building and Housing) pp.675-684, p.683, Vol. 7, No. 6, 2006.

[9] Homepage CASANOVA: http://nesa1.uni-siegen.de/index.htm?/softlab/casanova_e.htmCode 19

[10] C. Nytsch-Geusen, T. Nouidui: Gebäudesimulation mit adaptiven Modellierungsansätzen. In Proceedings: BAUSIM 2008, IBPSA Germany, Universität Kassel, 2008.

[11] A. Kühn, M. G. Ribigini, F. Ziegler: Dynamisches Betriebsverhalten einer 10 kW Absorptionskälteanlage. KI – Kälte-, Luft-, Klimatechnik 7-8/2006.

Deficiencies of Modelica and its simulation environments for large fluid systems

Kilian Link, Haiko Steuer, Axel Butterlin
Siemens AG, Energy Sector, Fossil Power Generation, Energy Solutions, Erlangen, Germany
{kilian.link, haiko.steuer, axel.butterlin}@siemens.com

Abstract

Modeling of large fluid systems requires in-house (specialized) tools, since applicability of Modelica and existing environments is limited.

Nevertheless Modelica is a very powerful and descriptive modeling language, which is best suited for physical modeling in a heterogeneous environment. Its object oriented approach, the built-in documentation and the availability of commercial and free libraries justifies the decision for Modelica as the preferred modeling language within Siemens Energy.

For an appropriate analysis of transient power plant processes, there often are large fluid systems to be modeled, i.e. there can be several thousand states. For such plant models, we use our in-house tool Dynaplant (DP), which is specialized for large fluid systems. A comparison between DP and Dymola[1] reveals some deficiencies of the Modelica world concerning performance and plant model construction: Especially, successive initialization and sparse matrix solvers are important features in need.

Keywords: Fluid simulation; workflow; performance

1 Introduction

Providing clean and affordable electric power to all human beings is one of the most ambitious challenges in our world. Technological efforts and innovations lead to high effective and environmental gentle methods of power production. Here, transient simulation has become an inevitable tool. Especially matters of unit safety in respect to material stresses of components do require detailed modeling and computational intensive dynamic simulations.

A typical use case which requires a detailed transient simulation of a power plant is a dynamic stability analysis. In the remaining of the introduction the system under consideration and the use case are introduced.

1.1 Plant model

The system under consideration is a special kind of evaporator modeled via several tubes with a total length of some 100 meters. The tubes are filled with water/steam, mostly in the two-phase region. Most of the tubes are heated by a hot flue gas flow through the tube metal wall.

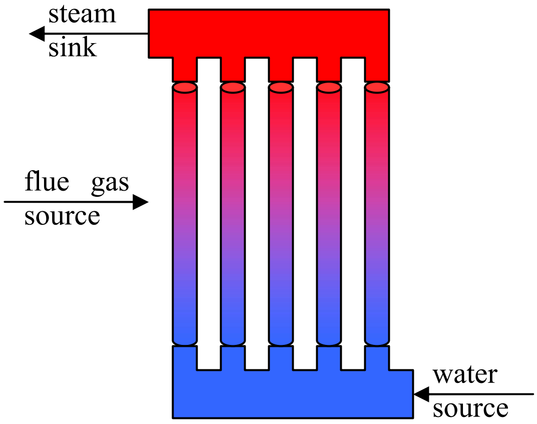


Figure 1 Evaporator part model composed of a splitter, five heated tubes and a mixer.

Usually the plant model is composed of two of the evaporator parts shown in Figure 1. For the purpose of this article, we will focus on one single evaporator part only.

A detailed one-dimensional hydrodynamic tube model is used. The resulting differential-algebraic equation system (DAE) is stiff and non-linear. In addition, there is a narrow spatial discretization along the tubes, such that up to several thousand states have to be considered.

1.2 Use case "dynamic stability analysis"

The purpose of a stability analysis is to avoid spontaneous mass flow fluctuations in the evapora-

DOI: 10.3384/ecp09430034

tor, which could lead to material fatigue [6]. Therefore, a large fluid system has to be built up using component models with detailed geometry parameters and many states.

At first a start point steady state has to be established. Secondly, a dynamic experiment has to be performed starting from this initial steady state with a temporary perturbation. The perturbation is put into the system via a temperature shift of the inlet flue gas. Here, if the system relaxes to the steady state again, it is stable. Otherwise, the evaporator design should be modified.

Our in-house tool DP is specialized for such kind of applications. Its component library is limited but suitable for a dynamic stability analysis. Dymola, on the other hand, is a multi-purpose simulation environment. It is at present the only tool based on Modelica, which supports fluid systems including Modelica.Media and Modelica_Fluid elements.

Both tools will be compared with respect to the reference work flow which covers a use case similar to a stability analysis with a reduced plant model.

2 Reference workflow (using DP)

In this section, we will introduce a typical DP workflow, which is very similar to a dynamic stability analysis. It covers the plant build-up, initialization and a dynamic experiment.

2.1 Plant build-up with successive initialization

The scope of the plant model is a single evaporator part as shown in Figure 1. It is composed of a water source, a splitter for water, several parallel heated tubes, a mixer for steam, a steam sink and a flue gas flow. The flue gas heats the tubes via their metal walls. In each of the parallel tubes, due to the splitter and mixer, the same pressure loss but different heating is applied, such that a certain mass flow distribution will arise.

We will cover two different system sizes, which distinguish only via their spatial discretizations ("small" resp "large" system). In DP, the total plant model results in a DAE system with 101 algebraic and 440 (resp 895) dynamic degrees of freedom for the small (resp large) system system.

The plant model can be edited in DP as follows: Adding of components per drag & drop, editing its parameters and setting start values at the connection points can be carried out via a graphical user interface. The build-up of the plant model takes place using successive initialization:

(A) Starting with just a few components and specifying start values at the connection points, a dynamic simulation with constant boundary conditions can be performed. The inner degrees of freedom of the initial state are computed using interpolations of the connection values. After an appropriate simulation time, the system will be relaxed into a steady state.

(B) The resulting plant model including steady state variables is loaded in DP.

(C) Here, the plant model can be modified. Further aggregates can be added. At new connections, initial state information has to be specified. For new aggregates, the inner state variables are unknown, such that they will be interpolated from these connection values.

(A) In a next simulation, the "old" components start with already computed state variables, and the "new" components start with interpolated values. After this run, there is a new steady state for this larger plant model, in which the "old" states may be modified.

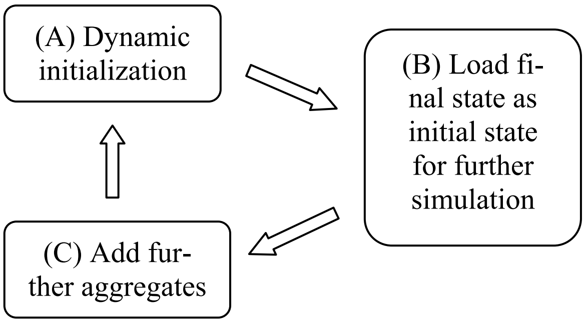

(A) Dynamic initialization
(B) Load final state as initial state for further simulation
(C) Add further aggregates

Figure 2 Successive initialization for plant build-up

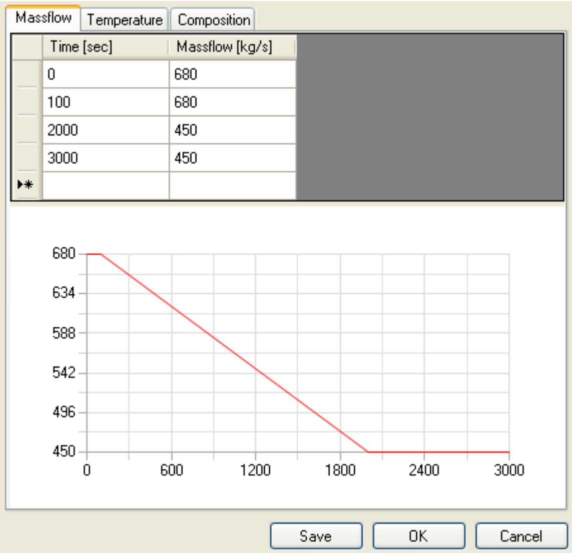
Such, step by step, the plant model can be enlarged in order to successively build up the steady state of the total plant model (see Figure 2).

2.2 Dynamic experiment

Finally, the constructed plant model with a steady state can be used as starting point of a dynamic experiment, which is specified via certain time-dependent boundary conditions.

Depending on the dynamics of the experiment, the DAE system may be more or less difficult to solve. DP uses a fast and stable DAE solver with sparse Jacobian. The partial derivatives are defined in special sub-model functions. The total Jacobian structure is built only once at the beginning of the simulation taking into account potential flow reversals. The water/steam property computations are performed using fast table based functions [5].

The boundary conditions are used to force the evaporator to switch from 100% load case to a part load case. Therefore, the water source changes its mass flow rate and specific enthalpy, the steam sink changes its pressure and the gas source changes its mass flow rate and temperature.



Time [sec]	Massflow [kg/s]
0	680
100	680
2000	450
3000	450

Figure 3 DP parameter dialog of the gas source for specifying dynamic boundary conditions.

In DP, dynamic boundary conditions are set in the boundary components using time table parameters (see Figure 3). The system simulation time is 3000sec.

3 Comparison: DP-Dymola

3.1 Application range

Unlike general purpose tools as Dymola the development of DP is in line with the clearly defined use case: Modeling of one-dimensional hydrodynamics of large fluid systems. For a larger application range two main features are missing,

- Handling of hybrid systems, i.e. discrete states, including advanced event handling.
- Model libraries and the ability for user defined components.

3.2 Modeling and Simulation

The explicit implementation of the Jacobian in DP is time-consuming and error-prone.

Coping with very large systems, as described above, heavy difficulties during computing the initial state may occur, where iterative assembling and initialization of sub models is missing. For the recent case of hard to find steady state solutions, the support for sub model initialization and setting the states to known values is a key feature to gain success. Currently setting fixed start values for parts of the model is a time consuming task in Dymola. This is strongly related to the Modelica specific requirement to define initial values inside of models, while DP allows the definition of initial values in the connection set. In addition identical initial values are not propagated or checked for consistency, hence it is difficult to find out which initial or guess values are in use.

3.3 Performance

In this chapter the performance of DP and Dymola [1], the by far best suited Modelica tool for fluid simulations is compared for the dynamic experiment of the reference work flow. In Dymola, evaluation of parameters as well as the "NoGuard" `userdefs.h` option is used. The Modelica model is slightly simplified, since some details of the DP tube model are not yet implemented in Modelica. It is based on the Modelica_Fluid interfaces [2].

	small system	**large system**
Dymola	691 sec	6780 sec
Dynaplant	46 sec	80 sec

Table 1 CPU times for reference dynamic experiment. The small/large system has about 400 resp. 800 continuous time states.

The main results of the performance comparison are that the Dymola CPU time is very large and critically depends on the system size. This is mainly due to

- Missing high performance water/steam property calculation [5].
- Missing sparse matrix solver.

4 Conclusions

The comparison with Dynaplant reveals features in need for large fluid systems in Modelica simulation environments. Below, they are sorted by priority. The important features in need are tool related. The other items cannot clearly be addressed to the tool vendors alone, since enhancing Modelica will also be necessary. Our intention is however, that the further development of Modelica *and* tools may consider the demands of large fluid system simulations.

4.1 Important features in need

- For generating a steady state successive initialization, i.e. “reload” of old simulation results with component specific states, should be possible.
- A sparse matrix DAE solver is necessary.

4.2 Further improvements

- A standardized solver interface would simplify the usage of external solvers An External Model Interface for Modelica: http://www.modelica.org/events/modelica2008/Proceedings/sessions/session5f.pdf[3], [4].
- Pre-compiled sub-models would reduce the compilation time of large models.
- High performance water/steam property calculation.

4.3 Nice to have features

Both nice-to-have features are related to the setting of guess values used in the initialization routine.

- Redundant specifying of guess values for the ports inside the sub-models may be replaced by specifying values at the connection points.
- Propagation of guess values would further simplify the setting-up of large plant models. This can be done using simple rules for flow variables or by using more sophisticated information from pre-compiled sub-models.

References

[1] Dymola7.1 http://www.dynasim.se/index.htm

[2] Modelica_Fluid: http://www.modelica.org/libraries/Modelica_Fluid

[3] An External Model Interface for Modelica: http://www.modelica.org/events/modelica2008/Proceedings/sessions/session5f.pdf

[4] Modelisar: http://www.itea2.org/public/project_leaflets/MODELISAR_profile_oct-08.pdf

[5] Butterlin, A.; Schiesser, D.; Steuer, H.: Usage of Water & Steam Properties in Computational Intensive Dynamic Simulation, ICPWS XV, September 2008.

[6] Franke, J., Brückner, J.: Dealing with tube cracking at Herdecke and Hamm-Uentrop in Modern Power Systems, October 2008.

Real-Time Simulation of CESA-I
Central Receiver Solar Thermal Power Plant

Javier Bonilla[1] Lidia Roca[1] Luis J. Yebra[1] Sebastián Dormido[2]
[1]CIEMAT- Plataforma Solar de Almería, Ctra. Senes s/n, 04200 Tabernas, Almería, Spain
Centro de Investigaciones Energéticas, MedioAmbientales y Tecnológicas
[2]Universidad Nacional de Educación a Distancia (UNED), Madrid, Spain

Abstract

This paper presents a real-time heliostat field simulator, based on a hybrid model, using Modelica as the modelling language. The development of industrial dynamic simulators, in this work for a central receiver solar thermal power plant: CESA-I from CIEMAT-PSA, is mainly aimed as a tool for the enhancement of advanced control algorithms but it is also useful for training purposes. The developed real-time heliostat field simulator is basically the union of the hybrid heliostat field model and a wrapped model which handles the real-time simulation and communication issues between the heliostat field simulator and HelFiCo (Heliostat Field Control) software which is in charge of manipulating and controlling the heliostat field according to an automatic control strategy. The real-time heliostat field simulator provides a virtual system, with the same response as the real plant.

Keywords: real-time simulation; hybrid modelling; heliostat field; solar thermal power plant

1 Introduction

Design and development of dynamic models for simulation and control system design purposes, is gaining importance in solar thermal industrial processes. An example is the deployment of advanced control systems that optimize the overall performance of solar thermal power plants.

It is a fact nowadays that this task is a priority research line [13] at CIEMAT National Spanish Laboratories (Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas - Research Centre for Energy, Environment and Technology), public organization owned by the Spanish Ministry of Science and Innovation.

During the last years a big effort has been devoted to the development of control systems for solar thermal power plants,making an important part of the experiences directly against the real plant. These real tests have increased the resources needed for development. Nevertheless, some of these systems, are expensive, scarce and present a costly experimentation time. This fact has motivated the development of a dynamic model for the CESA-I heliostat field plant, aimed mainly as a tool for the enhancement of advanced control algorithms. A preliminary research work in this topic was published in [1].

Several softwares have been developed to calculate the solar-flux density distribution on a central receiver system [6] with the aim of optimizing the components of the receiver, studying the optical performance and improving the process. These algorithms include geometry which depends on: the sun vector, the heliostats and tower positions, properties of the components (such as the reflectivity of the mirrors), errors (such as wrong canting), shadows, atmospheric attenuation, etc.

This paper explains the heliostat dynamic as a function of the messages received from a heliostat field control, although a future goal in our working is to include a simplified optical model to calculate the flux distribution caused by a heliostat.

2 Goals

The main goal of this paper is to model and develop a customizable real-time central receiver thermal power plant simulator, which is adapted to CESA-I test-bed facility requirements.

This development is mainly aimed as a tool for the enhancement of advanced control algorithms and also for training purposes. Design, testing and

DOI: 10.3384/ecp09430062

validation of new advanced control strategies in real plants, is expensive, dangerous and requires a long testing time. It is intended to weed out such problems by an efficient, safe and reliable real-time simulator.

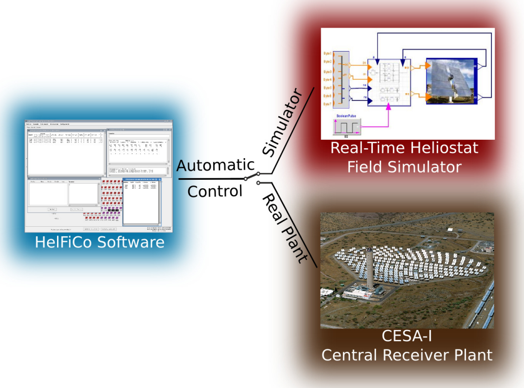


Figure 1: Relationship between HelFiCo software, the real-time heliostat field simulator and CESA-I central receiver plant.

The real-time simulator has to provide a virtual system, with the same behaviour and response as the real plant (see Figure 1). The communication between HelFiCo (Heliostat field control) software and the real-time simulator has to be transparent for the user of the heliostat field control software.

3 Central Receiver Solar Thermal Power Plants

In this section an overview of the basic components and operating procedures for a Central Receiver Solar Thermal Power Plant (CRSTPP) are introduced. Figure 2 shows an explicative diagram of a general CRSTPP.

The operation of this kind of plants is based on the concentration of incoming solar energy using a heliostat field that reflects the incident solar radiation onto a (typically volumetric) receiver (theoretically onto an optical point in the 3-D space). As the sun position changes during the day, each heliostat of the field has to change its position in real time according to the selected aiming point on the receiver, as different aiming points can be selected in order to achieve a uniform temperature distribution on the receiver [5]. The receiver is located at the top of the tower (84m height in CESA-I) and acts as an energy exchanger, receiving solar energy and transferring it to a thermo-hydraulic circuit with air medium (see Figure 2). The system is also composed of an energy storage tank, an air/water-steam heat exchanger (evaporator), blowers and valves. The combined action of the blowers allows feeding either the storage tank or the heat exchanger with hot air. The evaporator is formed of the primary circuit and a secondary one with sub-cooled inlet water and with superheated steam outlet. A measurement of the overall concentrated input radiation, a controlled water pump and an outlet controlled valve define the main boundary conditions for the system.

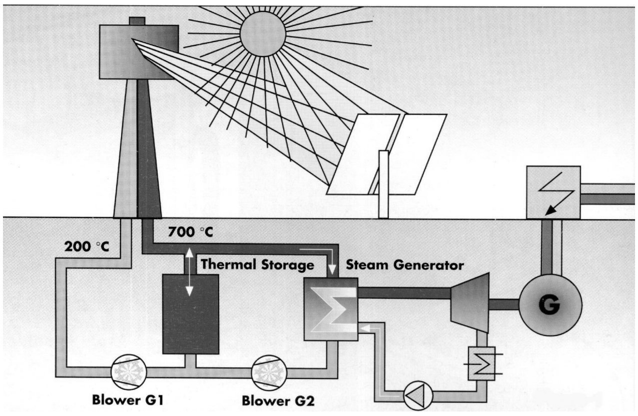


Figure 2: Schematic diagram of a central receiver solar thermal power plant [13].

3.1 CESA-I test-bed facility

The modelling and simulation activities object of this work are focused on the CESA-I facility, a central receiver solar thermal power plant belonging to CIEMAT, and located at the Plataforma Solar de Almería (PSA), South-East of Spain. This test-bed plant can be seen in Figure 3, and it is an experimental prototype for electricity generation among other research projects.

The CESA-I facility collects direct solar radiation by means of a field of 300 heliostats (39.6-m^2-surface) distributed in a 330-x-250-m north field into 16 rows. The heliostats (see Figure 4) have a nominal reflectivity of 92%, the solar tracking error on each axis is 1,2 mrad and the reflected beam image quality is 3 mrad. North of the heliostat field there are two additional test zones for new heliostat prototypes, one located 380 m from the tower and the other 500 m away. The maximum thermal power delivered by the field onto the receiver aperture is 7 MW. At a typical design irradiance of 950 W/m2, a peak flux of 3.3 MW/m2 is obtained. In addition, the 99% of the power is focused on a 4m-diameter circle, 90% in a 2.8-m circle.

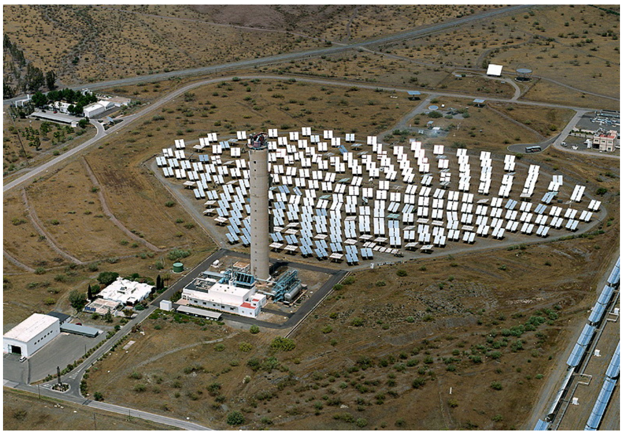


Figure 3: CESA-I facility (CIEMAT-PSA).

The 80-m-high concrete tower has a 100ton load capacity. The tower is complete with a 5-ton-capacity crane at the top and a freight elevator that can handle up to 1000-kg loads. For those tests that require electricity production, the facility has a 1.2 MW two-stage turbine in a Rankine cycle designed to operate at 520 Â°C 100 bar superheated steam.

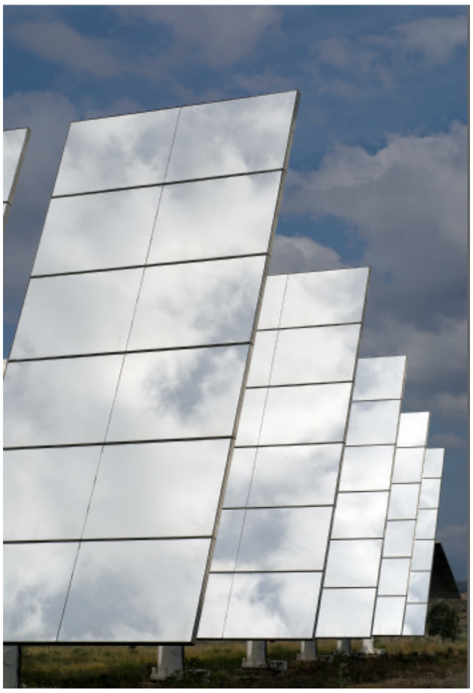


Figure 4: Heliostats from CESA-I facility.

4 Heliostat Field Modelica Model

To take advantage of all the Modelica features, the heliostat field simulator has been developed using this language. Modelica is a standard unified modelling language [8] with many advantages for modelling dynamic systems, because it is both, an object-oriented and acausal language. The object-oriented feature allows developing a set of reusable objects which can be used in future developments and the acausal feature allows describing the behaviour of dynamic systems using the differential equation systems which describe them. Dynamic behaviour and numerical aspects are taken into account in Modelica, because it provides equation sections and event modelling [4].

Moreover, Modelica has a library, *StateGraph*, for modelling hierarchical state machines. The *StateGraph* Modelica library offers features to define conveniently discrete events and reactive systems in Modelica models. Since Modelica is used as an action language, a Modelica translator can guarantee that a State-Graph has deterministic behaviour. StateGraph models can be combined with components of any other Modelica library and can therefore be very easily used to control a continuous plant [9].

A simple hybrid model of a single heliostat has been developed combining the system dynamic continuous variables with operation mode discrete variables. This one-heliostat model may be briefly explained as follows:

- Movement dynamic. The continuous variables of the system are the azimuth, a, and elevation positions, e, which can be obtained through movement equations using the angular velocities as known parameters (ω_a, ω_e).

$$\frac{da}{dt} = \omega_a; \quad \frac{de}{dt} = \omega_e$$

- Operation mode. Not only each single heliostat must achieve the desired reference position defined by the control system software, but also communication failures and timeouts must be taken into account by the heliostat. These characteristics have been included in a state machine using the *StateGraph* library.

  Five main operation modes are defined:

- *Waiting*: The heliostat is waiting for an input message. In this mode, the heliostat may be moving to a position.
- *Request*: The heliostat sends an output message including position information.
- *Reset*: When the heliostat is in reset mode, that updates its positions moving to zero position for azimuth and elevation, moving firstly in the azimuth direction. Once it get to zero-azimuth position, it begins moving in the elevation axis. In this operation mode, the heliostat does not send any output message until the reset time, *Treset*, is reached.
- *Stow*: If the heliostat does not receive an input message during *Timeout* seconds, it moves to stow (zero) position. In this case, the movement is in the two axes at the same time.
- *Control*: The heliostat moves to the position reference specified in the input message.
- *Request* + *Control*: Request and Control modes at the same time.
- *Request* + *Reset*: Request and Reset modes at the same time.

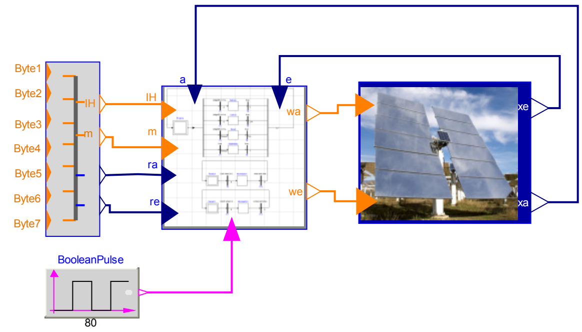


Figure 5: The hybrid one-heliostat model

Figure 5 shows the one-heliostat model which has the following features:

- The input signals are a collection of integer signals which represent the received bytes from the heliostat field control software. Notice that these input signals are the same for all the heliostats which belong to the same heliostat row.
- These bytes are decoded to obtain the heliostat identifier (*IH*) parameter (which identifies univocally each heliostat), the control message (*m*) and the azimuth and elevation references (*ra*, *re*).
- A state machine component (see Figure 6), based on the *StateGraph* library, makes possible to know the angular velocities in both axes (*wa*, *we*) which depend on the heliostat state.

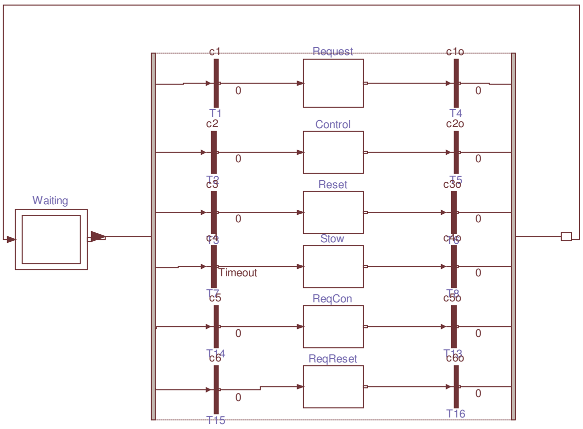


Figure 6: The state machine diagram

- The azimuth and elevation positions (*a*, *e*) are calculated with the movement dynamic using the angular velocities provided by the state machine component.

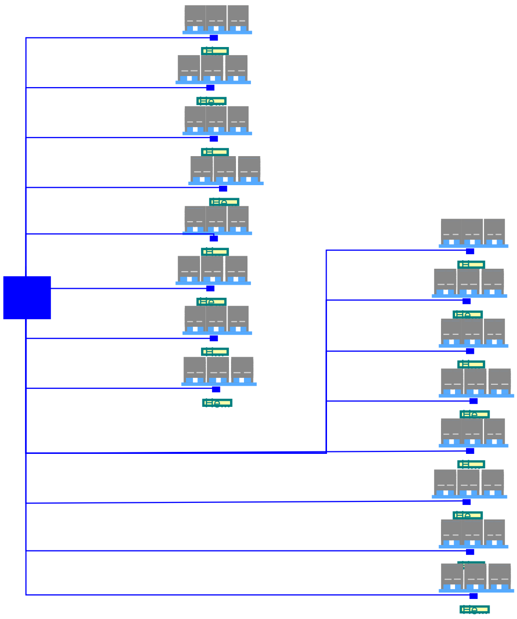


Figure 7: Heliostat field model

A heliostat row model is $NRow_i$ heliostat model instantiations, being $NRow_i$ a parameter that defines the number of heliostats in the i row. Notice that the control system software sends input commands to each one of the 16 rows, therefore the heliostat field simulator is composed of 16-heliostat-row instantiations (see Figure 7).

4.1 Simulation results

In this section a simulation of 16-heliostat row is explained. The inputs messages are shown in Table 1. The heliostat identifiers (IH) are $IH = \{1,2,3,...,15,16\}$, just the heliostat which identifier matches with the one included in the input messages, must process it.

Table 1: Simulation results. Input messages.

Time (s)	IH	Message	ra	re
$t_0 = 0$	10	Request+Reset	100	100
$t_1 = 82$	10	Request+Reset	100	100
$t_2 = 162$	10	Request	0	0
$t_3 = 242$	10	Request	0	0
$t_4 = 322$	10	Request	0	0
$t_5 = 402$	3	Request+Control	100	100
$t_6 = 482$	3	Request+Control	100	100

The output message (see Table 2) specifies the heliostat identifier which has to respond to the request, including its current position, (a, e), and a boolean, *ref*, that turns to 1 when the heliostat achieves the desired reference. At the beginning of the simulation, the heliostat 10 moves to zero position because of the reset message. At time t_2 the heliostat has reached the zero value in azimuth and the output value in this direction is reestablished. The same situation occurs at time t_3 with the elevation direction. The last two messages order the heliostat 3 to move to (100,100) coordinates with a movement request. The output heliostat message at time t_5 shows that its position is (0,0) whereas at time t_6 is (100,100), so the desired reference is reached.

The position of the two heliostats during the simulation are shown in Figure 8 together with the angular velocities in each direction. These angular velocities are parameters that can be changed depending on the heliostat features. For this simulation the velocities were chosen to be ± 5 positions/s. On the other hand, the operation modes for both heliostats are shown in Figure 9.

Table 2: Simulation results. Output messages.

Time	IH	a	e	ref
t_0	10	600	600	0
t_1	10	190	600	0
t_2	10	0	394	0
t_3	10	0	0	0
t_4	10	0	0	0
t_5	3	0	0	0
t_6	3	100	100	1

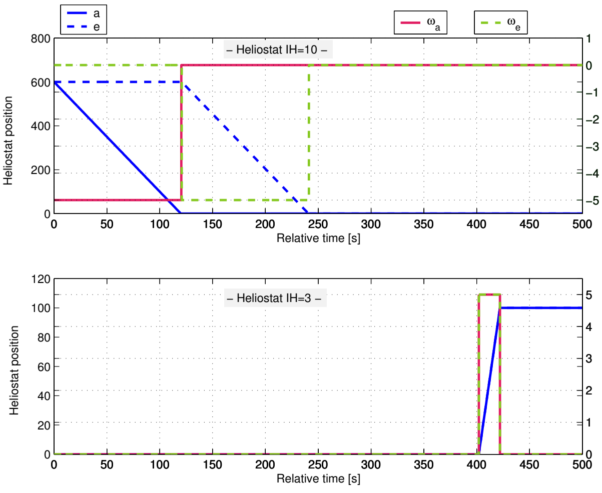


Figure 8: Simulation results. Heliostat positions.

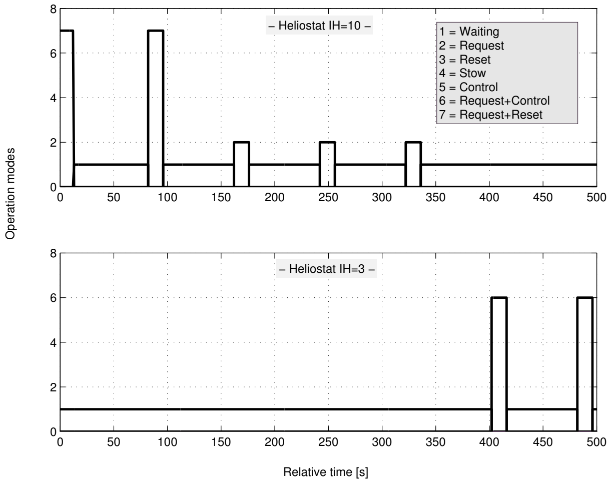


Figure 9: Simulation results. Operation modes.

5 Real-Time Heliostat Field Simulator

The developed real-time heliostat field simulator is mainly the union of the hybrid heliostat field model and a wrapped model which handles the real-time simulation and communication issues. The heliostat field simulator is communicated with HelFiCo (Heliostat Field Control) software, a heliostat field control software [7] which is in charge of manipulating and controlling the heliostat field according to an automatic control strategy.

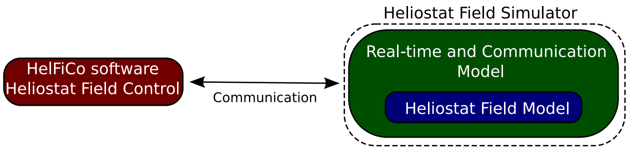


Figure 10: Heliostat field simulator and HelFiCo software block diagram

The figure 10 shows a block diagram which illustrates the concept previously explained and shows the interconnection between the heliostat field simulator and HelFiCo software.

5.1 Real-time and Communication Model

The real-time and communication model wraps the main model, the heliostat field model. This wrapped model provides the following functionality:

- *Real-time simulation:* Synchronize the simulation time dynamically to real-time.
- *Communication issues:* Receive input commands from the heliostat field control software, decode input commands, encode output data from the heliostat field model and send output data to the real-time simulator.

5.1.1 Real-time Model

Real-time demands that the simulation time must be shorter than the simulated time. Since this condition is a fundamental requirement for models to be used in real-time simulations, in some cases, the complexity of the model must be reduced at the expense of loosing quality.

In our case it was not necessary to reduce the complexity of the heliostat field model, just take into account some improvement in the model to optimize the computational efficiency. These improvement will be discussed in the subsection 5.1.2.

Modelica does not have synchronisation support, some Modelica IDEs have this kind of support but it is specific to the IDE tool, such as Dymola [2]. Moreover the synchronisations support in Dymola has the drawback that simulation speed is not limited if simulation time is behind real world time, this can lead to undesirable behaviour of the simulation.

There is an open source real time option for Dymola called JPAARealTime [3]. Since JPAARealTime makes use of Windows libraries it can only be run on Windows operating system. But it was required a multi-platform library for future development in different operating systems. For these reasons a real-time library has been developed for this task. The developed RTSS (Real-Time SimulationS) library allows real-time synchronisation support.

The RTSS library samples the simulation time at certain time instants and stops the current simulation until the simulation time reaches the real time according to the computer's time when the simulation started. This concept is shown in Figure 11.

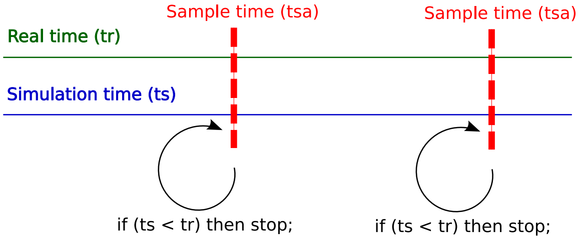


Figure 11: Real-time synchronisation concept

5.1.2 Communication Model

The communication model simulates the behaviour of the communication line between the simulator and HelFiCo (Heliostat Field Control) software, involving receive, send, encode and decode messages. Message decoding is the process of translating the incoming binary message from the heliostat field control software into the corresponding input variables in the Modelica model. On the other hand, message encoding is the reverse process.

The real communication channels between HelFiCo software and CESA-I test-bed facility are RS-485 and RS-232 wires. Each one of the 16 heliostat rows has a shared RS-485 wire, all of them converge to a RS-485 to RS-232 interface converter which is directly connected with the computer where HelFico software is running.

The communication line between both computer programs is simulated by two FIFO (First In, First Out) files in the operative system for each heliostat row in the field, one for sending and the other one for receiving. This FIFO files are share by HelFico software and the real-time simulator, but the operation mode is different in each one. For each heliostat row, the HelFico software open one FIFO file in writing mode (output line) and the other one in reading mode (input line). In the real-time simulator the operation mode of the FIFO files is the opposite, this behaviour is illustrated in Figure 12.

FIFO files were chosen as the inter process communication method for serveral reason. First of all, the FIFO files behaviour is suitable for the developed simulator. The simulator requires just message passing, in order of arrival, as in a queue, and this can be easily obtained using FIFO files, moreover it is not needed synchronization, shared memory or remote procedure calls. Another reason is that the FIFO files behaviour is quite similar to the RS-485 and RS-232 wires behaviour. There are four main operations: open, send, receive and close. It is straightforward to develop an abstract parent class which set the common interface with these four main operations. Then, two subclasses are more specialized versions, inherit attributes and behaviours from the parent class, and introduce their own behaviour. One of these classes implements the FIFO files communication and the other one controls the real communication channels. These two classes allow HelFiCo software to switch between them to communicate properly to the simulator or the real plant. New subclasses can be developed for different kinds of communication, the only requirement is to inherit from the abstract parent class.

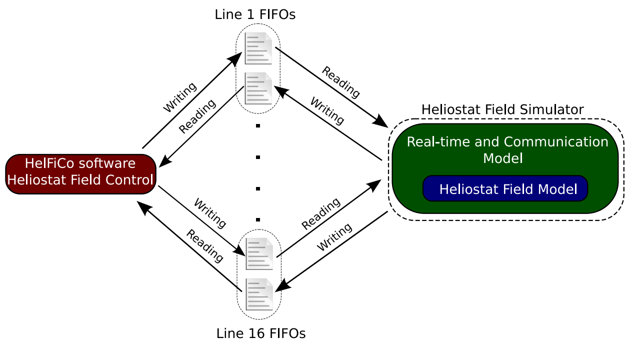


Figure 12: Communication model

5.1.3 Improving the real-time simulator performance

- *Coupling the real-time synchronisation with the data interchange:* One of the improvements consists in setting the same sample time for the real-time synchronisation task and the data interchange between the real-time simulator and HelFiCo software. This can avoid some sample instructions and events, and improve the simulation speed due to the fact that the simulation stops when a sample instruction is processed and therefore, improve the simulator performance.
- *Separated simulation approach:* This approach takes advantage of the multi-core processors (e.g. dual-core and quad-core processors) allowing the simultaneous execution of several processes. For that reason a separated simulation, where the differential equation system is divided in different parts, achieves a simulation speed improvement in a multi-core processor computer.

  This approach can be used because the model is dividable, so that one partial simulator is only responsible for a part of the system. Figure 13 shows a separated simulator where each partial simulator simulates just some heliostat rows of the field, in this case, there are 16 partial field simulators, each one simulates one heliostat row and therefore uses just one pair of FIFO files.

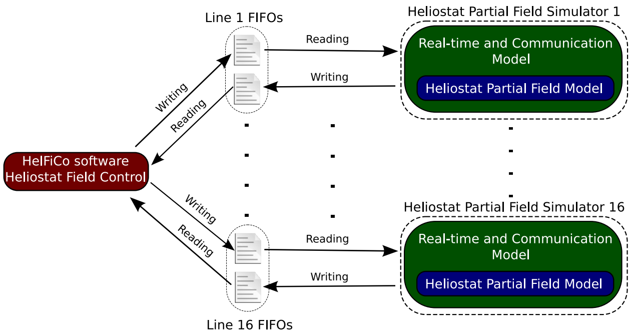

Line 1 FIFOs
Heliostat Partial Field Simulator 1
Reading
Writing
Real-time and Communication Model
Heliostat Partial Field Model
Writing
Reading
HelFiCo software
Heliostat Field Control
Writing
Reading
Heliostat Partial Field Simulator 16
Reading
Writing
Real-time and Communication Model
Heliostat Partial Field Model
Line 16 FIFOs

Figure 13: Separated simulator concept

Table 3 shows the CPU-Time for integration in simulation for different approaches: the traditional approach which is a complete heliostat field simulator (CS), 2 separated simulators (PS-2) and 4 separated simulators (PS-4). The 60-seconds simulation is the same for the 3 approaches and was carried

Table 3: Simulation approaches and results

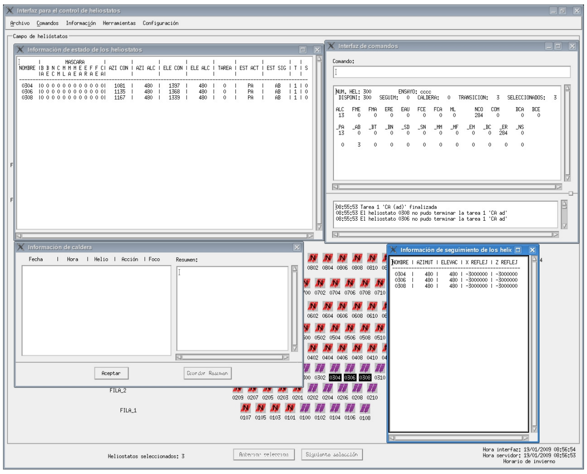
Approach	CS	PS-2	PS-4
Num. Processes	1	2	4
Rows per process	16	8	4
Num. Equations per process	114,904	57,452	28,726
CPU-Time for Integration	37.40 s	21.20 s	18.22 s

out in a computer with a dual-core processor. The speed improvement from CS to PS-2 approach it was expected, because PS-2 approach can take advantage of a dual-core processor. But the PS-4 approach improves the simulation speed even when just a dual-core processor is used and this approach involves more processor context changes. Therefore this simulation speed improvement seems to be because the numerical integration in these simulation tests, do not scale properly with the number of differential equations. The numerical integrator, used in this simulations, was LSODAR [10].

5.2 HelFiCo Software

HelFiCo (Heliostat Field Control) software belongs to Aunergy ThermoSolar S.R.L, a spin-off from CIEMAT (`www.aunergy.com`). This software is in charge of manipulating and controlling each heliostat in the field according to an automatic control strategy. The main features of this software are the following ones:

- It is a generic software allowing different kind of heliostats and even different central receiver solar thermal power plants.
- The software has two different subsystems, an intuitive graphical user interface, see Figure 14, and the heliostat field control system.
- It is a scalable distributed software. This feature allows that the software can be executed in several computers to achieve scalability and robustness independently of the number of heliostats in the field.
- The software allows different communication methods with the heliostat field.
- The software can connect to both the real plant and the simulated plant, and this process is transparent for the HelFiCo software user.
- It is a multi-platform software. The Adaptive Communication Environment (*ACE*) [11] as development framework and *Qt* [12] as user interface framework have been used for the development.



```
       |        MASCARA         |         |         |         |         |       |         |         |   |
NOMBRE |B B N C M M M E E F F C| AZI CON | AZI ALC | ELE CON | ELE ALC | TAREA | EST ACT | EST SIG | T | S
       |A E C M L A E A R A E A|         |         |         |         |       |         |         |   |
------------------------------------------------------------------------------------------------------------
 0304  |0 0 0 0 0 0 0 0 0 0 0 0|  1081   |   480   |  1397   |   480   |   0   |   PA    |   AB    | 1 | 0
 0306  |0 0 0 0 0 0 0 0 0 0 0 0|  1135   |   480   |  1368   |   480   |   0   |   PA    |   AB    | 1 | 0
 0308  |0 0 0 0 0 0 0 0 0 0 0 0|  1167   |   480   |  1339   |   480   |   0   |   PA    |   AB    | 1 | 0
```

```
NUM. HEL: 300          ENSAYO: cccc
 DISPONI: 300   SEGUIM:   0   CALDERA:   0   TRANSICION:   3   SELECCIONADOS:   3

ALC   FME   FMA   ERE   EAU   FCE   FCA   ML       NCO   COM       BCA   BCE
 13     0     0     0     0     0     0    0       284     0         0     0

_PA   _AB   _BT   _BN   _SD   _SN   _MM   _MF   _EM   _BC   _ER   _NS
 13     0     0     0     0     0     0     0     0     0   284     0

  0     3     0     0     0     0     0     0     0     0     0     0
```

```
08:55:53 Tarea 1 'CA (ad)' finalizada
08:55:53 El heliostato 0308 no pudo terminar la tarea 1 'CA ad'
08:55:53 El heliostato 0306 no pudo terminar la tarea 1 'CA ad'
```

```
NOMBRE | AZIMUT | ELEVAC | X REFLEJ | Z REFLEJ
------------------------------------------------
 0304  |   480  |   480  | -3000000 | -3000000
 0306  |   480  |   480  | -3000000 | -3000000
 0308  |   480  |   480  | -3000000 | -3000000
```

```
Heliostatos seleccionados: 3
```

```
Hora interfaz: 19/01/2009 08:56:54
Hora servidor: 19/01/2009 08:56:53
      Horario de invierno
```

Figure 14: HelFiCo graphical user interface

6 Conclusions and Future Work

This paper explains a heliostat field real-time simulator of a central receiver solar thermal power plant (CESA-I from CIEMAT-PSA) developed using Modelica language as the modelling tool. The main goal of the developed model is providing a virtual system, with the same response as the real one, to test the control system software and to reduce the costly real experiments over the real plant. This paper also explains some techniques to improve simulation efficiency.

Future works will include a detailed explanation of dynamic heliostat models. Moreover, the incident solar radiation reflections onto the receiver to obtain the temperature distribution, will be implemented. The combination of the movement dynamic, temperature distribution on the receiver, communication model and real-time simulation will provide an efficient and practical tool to test advanced control systems for aim-point tracking to optimize the overall performance of central receiver solar thermal power plants.

7 Acknowledgment

The authors wish to acknowledge the financial support provided for this work by the CIEMAT, and the Spanish Ministry of Science and Innovation, without which this work could not have been done. The authors want to acknowledge to J. González for his help and contribution with HelFiCo software. The authors also gratefully acknowledge the comments of the reviewers.

References

[1] Bonilla, J., Roca, L., González, J., Yebra, L.J: Modelling and real-time simulation of heliostat fields in Central Receiver Plants. In: 6th Vienna International Conference on Mathematical Modelling, 2009, 2576–2579.

[2] Dynamsim, A.B.: Dymola - Dynamic Modelling Laboratory. Ideon Science Park, SE-223 70 Lund, Sweden. Online: `http://www.dynasim.se`, 2009.

[3] Frey, G.: JPAARealTime V0.1. - An open source real time option for Dymola. Department of Mechatronics Engineering, Saarland University, SaarbrÃ$\frac{1}{4}$cken, Saarland, Germany. Online: `http://www.aut.uni-saarland.de/software/software.html`, 2007.

[4] Fritzon, P.: Principles of Object-Oriented Modelling and Simulation with Modelica 2.1. John Wiley & Sons, IEEE Press, 2004.

[5] García-Martín, F.J., Berenguel, M., Valverde, A., Camacho, E.F.: Heuristic Knowledge-based Heliostat Field Control for the Optimization of the Temperature Distribution in a Volumetric Receiver, In: Solar Energy, vol 66, 1999, 355–369.

[6] Garcia, P., Ferriere, A., Bezian, J.: Codes for solar flux calculation dedicated to central receiver system applications: A comparative review, In: Solar Energy, vol 82, 2008, 189–197.

[7] González, J., Yebra, L. J., Berenguel, M., Valverde, A., Romero, M.: Real-time Distributed control system for heliostat fields (in Spanish). In: XXV Jornadas de Automática, 2004.

[8] Modelica and the Modelica Association. : Modelica Standard Library, Version 2.2.1, 2007. Online: `http://www.modelica.org/libraries/Modelica/releases/2.2.1/`.

[9] Otter, M., Årzén, K-E., Dressler, I.: StateGraph – A Modelica Library of Hierarchichal State Machines. In: Proc. 4th International Modelica Conference, 2005.

[10] Petzold, L. R., Hindmarsh, A. C.: *LSODAR*. Computing and Mathematics Research Division, 1-316 Lawrence Livermore National Laboratory, Livermore CA 94550.

[11] Schmidt, D. C.: The Adaptive Communication Environment (ACE), DOC software. Washington University, University of California, Irvine, and Vanderbilt University. Online: `http://www.cs.wustl.edu/~schmidt/ACE.html`, 2007.

[12] Trolltech:A cross-platform application and UI framework. Online: `http://trolltech.com`, 2008.

[13] Yebra, L.J., Berenguel, M. and Dormido, S. and Romero, M.: Modelling and Simulation of Central Receiver Solar Thermal Power Plants. In: Proc. 2005 European Control Conference Decision and Control CDC-ECC'05. 44th IEEE Conference, 2005, 7410–7415.

Modelica for Embedded Systems

Hilding Elmqvist[1], Martin Otter[2], Dan Henriksson[1], Bernhard Thiele[2], Sven Erik Mattsson[1]
[1] Dassault Systèmes, Lund, Sweden (Dynasim)
[2]German Aerospace Centre (DLR), Institute for Robotics and Mechatronics, Germany
Hilding.Elmqvist@3ds.com, Martin.Otter@DLR.de, Dan.Henriksson@3ds.com,
Bernhard.Thiele@DLR.de, SvenErik.Mattsson@3ds.com

Abstract

New language elements are introduced in Modelica 3.1 to facilitate use Modelica models in embedded systems, e.g., as controllers. Models can be conveniently configured by marking the borders of the respective controller parts and by defining the mapping of the marked parts to target processors and target tasks.

This approach allows to define a "logical" model from which all different "real" controller configurations for Model-, Software-, Hardware-in-the-Loop (MiL, SiL, HiL), rapid prototyping, and production code for multi-processing/multi-tasking are automatically derived by setting configuration options. Furthermore, a new, free library - Modelica_EmbeddedSystems - is presented that provides a convenient user interface to the new language elements. In summary, the power of Modelica in the area of real-time control is improved significantly.

Keywords: Embedded systems, real-time control, multi-tasking, multi-core, multi-rate, model parallelization, Model-in-the-Loop, Software-in-the-Loop, Hardware-in-the-Loop, rapid prototyping.

1 Introduction

Modelica has been used in advanced controller applications for embedded systems for several years, especially when non-linear plant models are part of the control system *(Looye et. al. 2005)*, such as non-linear control systems for aircrafts *(Bauschat et. al. 2001),* for industrial robots *(Thümmel et. al. 2005),* or for power plants *(Franke et. al. 2008)*.

Within the ITEA2 EUROSYSLIB project, a major effort started at end of 2007 to enhance Modelica considerably in the area of model-based control. This effort is also carried on in the ITEA2 MODELISAR project that started during 2008.

Existing methods and tools have been analyzed and different designs have been performed. Typically, controller parts that are to be downloaded to target platforms are defined by the root of a hierarchical structure. However, this standard approach has several inherent limitations and therefore, a novel, new approach was developed where only the borders of control systems are marked and algorithms have been developed to deduce the controller code from this information. Furthermore, in principal any Modelica model is supported, and therefore a controller to be downloaded to a real-time target machine may contain non-linear differential-algebraic equation systems as needed for advanced control systems.

Modelica extensions have been designed and are included in version 3.1 of the Modelica Specification *(Modelica 2009)*. A free library "Modelica_-EmbeddedSystems" has been developed as a convenient user interface to the new language elements. A prototype implementation in Dymola *(Dymola 2009)* was performed to validate the concept. Furthermore, device drivers for Windows game controllers, I/O boards using the Comedi-Interface *(Comedi 2009)* on real-time Linux and CAN-bus have been implemented by DLR. Device drivers for dSPACE (www.dspaceinc.com) hardware and the Lego Mindstorms NXT platform (mindstorms.lego.com) have been implemented by Dynasim, and the concept has been used in a student project *(Akesson et. al. 2009)*.

2 Logical and Technical System Architecture

A model of an embedded system is composed of subsystems which may have local controllers. The subsystems are coupled physically and through con-

DOI: 10.3384/ecp09430096

trol systems (e.g. with buses). The notation from *(Schäuffele and Zurawka 2005)* is used:

Complex controllers, e.g., in vehicles, are first designed with an abstract view which is termed “logical system architecture”. Here all the functional and logical behavior of the control system is defined. In a second step this architecture is mapped to a “technical system architecture” which is the concrete implementation of the control system in several tasks on several micro-controllers inter-connected by communication buses and other communication methods. For complex control systems, as in vehicles with over 60 ECUs interconnected via different bus systems, it must be possible to map from the logical to the technical architecture in a very flexible way. The new Modelica language extensions have been designed to fulfill this demanding requirement.

An already sufficiently complicated, but still rather simple, logical system architecture of a robot is shown in the left part of Figure 1. Every “axis” of the robot has a local control system in addition to the continuous motor and gearbox models. All local controllers are connected to a global controller (at the top of the figure) via a control bus. This system has eight coupled controllers that shall be downloaded to different processors (e.g., all axes controllers on two signal processors and the global controller on another processor and the processors communicate via buses). An example is shown in the right part of Figure 1 where this mapping of the logical to the technical system architecture is sketched. The new method has now the following important properties:

1. The user is not forced to manually assemble the parts belonging to the technical system architecture for download to the ECUs. Instead a Modelica translator performs this automatically from the logical system representation, given information about the mapping to the technical system architecture. Note, with standard methods and tools this is not possible, because the logical system architecture would be destroyed if the user is forced to move all controller parts under a hierarchical structure.
2. The mapping to the technical system architecture can be defined without modifying the logical system architecture. This is performed by a new model that inherits (extends) from the logical system model and where the mapping information is given as modifier, including the selection of hardware drivers. The latter are defined via replaceable external objects.

With these two features it is possible to conveniently configure different use cases, such as:

- Model-in-the-Loop (MiL) simulation
(Plant: variable step size integrators. Controller: ideal, synchronous continuous or discrete controllers)
- Software-In-the-Loop (SiL) simulation
(Plant: variable step size integrators. Controller: non-ideal, asynchronous controllers with modeled latencies).
- Rapid prototyping (real-time)

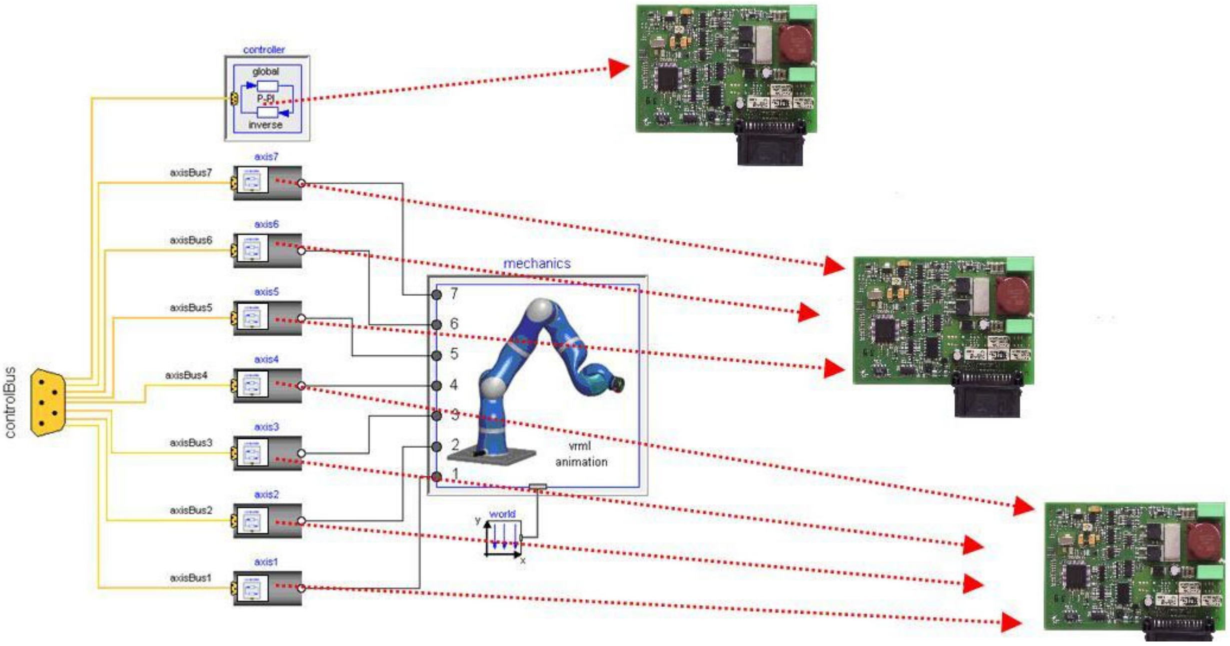


Figure 1: Mapping of logical to technical system architecture for a robot control system.
(Displayed ECUs adapted from http://commons.wikimedia.org/wiki/File:KeylessGoSiemensVDO.jpg)

(Plant: physical prototype. Controller: asynchronous controllers, channel assignment).

- Hardware-In-the-Loop (HIL) simulation (real-time)
(Plant: fixed step size, multi-rate integrators. Controller: embedded in ECUs, multi-tasking, production code, fixed-point representation, channel assignment, bus communication).
- Production code (real-time)
(Plant: Real product. Controller: embedded in ECUs, multi-tasking, production code, fixed-point representation, channel assignment, bus communication).

Dealing with embedded systems in Modelica according to the sketched concept above consists of the following parts:

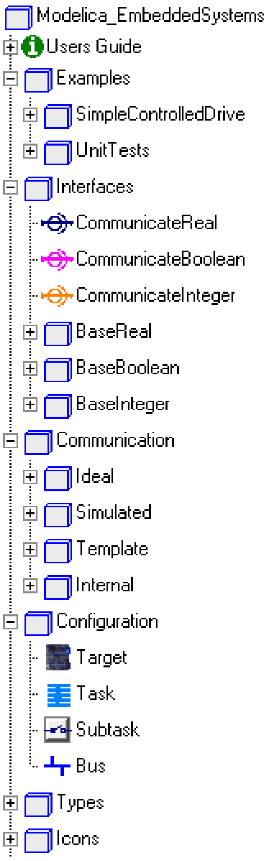
- New Modelica language elements are introduced. This is described in section 4. Basically, a new annotation "mapping" and some new built-in operators are provided. Furthermore, new algorithms are sketched to deduce the controller code from the logical system architecture.
- A new library "Modelica_EmbeddedSystems" is offered that provides a convenient interface to the new language elements. This library is discussed in the next section. It is provided freely and it is planned to be included in the Modelica Standard Library (note: a user may provide its own interface to the new language elements).
- Hardware drivers as Modelica external objects to access hardware from a Modelica model. A few hardware drivers that are available on every PC/notebook are provided in the Modelica_EmbeddedSystems library freely for Windows and for Linux. Other hardware drivers will be provided from third parties (e.g., commercially from tool vendors).

3 Modelica_EmbeddedSystems

The Modelica_Embedded-Systems library is available as an open source library from www.modelica.org/libraries and will be the basis of embedded systems in Modelica. The current status of the library is shown in the screenshot to the right.

The Examples sub-library contains various use cases to demonstrate the usage of the library. Only some of the available examples are currently included in the library.

The Interfaces sub-library contains the basic components to define communication points (i.e., borders of controller parts) and to select the implementation of the actual communication in the embedded system. This can be external I/O, network communication or inter-task communication on the same ECU.



Modelica_EmbeddedSystems
- Users Guide
- Examples
  - SimpleControlledDrive
  - UnitTests
- Interfaces
  - CommunicateReal
  - CommunicateBoolean
  - CommunicateInteger
  - BaseReal
  - BaseBoolean
  - BaseInteger
- Communication
  - Ideal
  - Simulated
  - Template
  - Internal
- Configuration
  - Target
  - Task
  - Subtask
  - Bus
- Types
- Icons

The Communication sub-library contains open source drivers for simulated communication (ideal and with simulated quantization effects taken into account), as well as a simple template for hardware drivers.

The Configuration sub-library contains templates to define the configuration of the embedded target systems (tasks, subtasks, sampling, target processor, etc.).

Types and Icons are utility sub-libraries.

The major goal of the library is to define the splitting of a model in tasks and subtasks and to associate device drivers with input and output signals of the respective parts. The following notation is used:

A "task" identifies a set of equations that are solved together as one entity, so equations are sorted and solved in a "synchronous" way as usual in Modelica. There are no equations that relate variables from different tasks because communication to and from tasks is performed by function calls of Modelica ExternalObjects[1]. Different tasks are executed asynchronously with possible synchronization via the ExternalObjects used for communication and possibly running on different cores or processors. Typically, a Modelica task is mapped to a task of the underlying operating system.

A "subtask" identifies a set of equations inside a task that are executed in the same way within the subtask with regards to sampling and integration method: If a subtask has continuous equations, all these equations are solved with the same integration method. Different subtasks can use different integration methods, e.g., fixed or variable step size methods of different orders. If a subtask is sampled, it is activated at the sampling instants and the equations of the subtask are integrated from the time instant of

[1] A Modelica ExternalObject defines a Modelica interface to C-functions that operate on the same memory and have constructor and destructor functions for this memory.

the last sample instant up to the current sample instant using the defined integration method. If a subtask is running on a real-time system, usually real-time integrators are utilized like explicit fixed-step solvers. The equations of several subtasks in the same task are automatically synchronized via equation sorting.

3.1 Communication Blocks

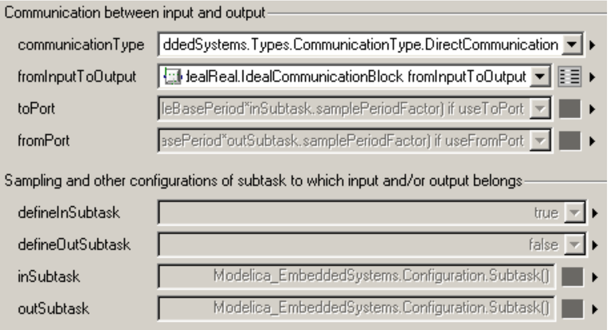
The usage of the Modelica_EmbeddedSystems library will be demonstrated by the very simple use case from Figure 2. The model consists of a reference controller ("ramp"), a feedback controller ("feedback" and "PI") and a plant ("torque", "load" and "speedSensor"). The task of the controller is to control the speed of the load inertia.

The model is split in different partitions by placing "communication blocks" in the signal paths. This is the "logical" model. At this stage it is only defined how the model is split into different parts, but it is not yet defined how to handle these parts (or more precisely, by default all communication blocks just pass their input signal to their output).

A "target" model is derived by inheriting from the model and by applying modifiers on the communication blocks. These modifiers usually reference a "configuration block" (in Figure 2 this is called "comedi") where all details are defined how to map this model to one or more target machines.

The "communication blocks" are the central part of the Modelica_EmbeddedSystems library and provide a graphical user interface between the user and the new Modelica 3.1 language elements. Clicking on one of the communication blocks in Dymola gives the menu shown in Figure 3.

The most important option is the first entry "communicationType". It defines the communication that shall take place between the input and the output of the communication block:


Communication between input and output

communicationType: ddedSystems.Types.CommunicationType.DirectCommunication

fromInputToOutput: dealReal.IdealCommunicationBlock fromInputToOutput

toPort: leBasePeriod*inSubtask.samplePeriodFactor) if useToPort

fromPort: asePeriod*outSubtask.samplePeriodFactor) if useFromPort

Sampling and other configurations of subtask to which input and/or output belongs

defineInSubtask: true

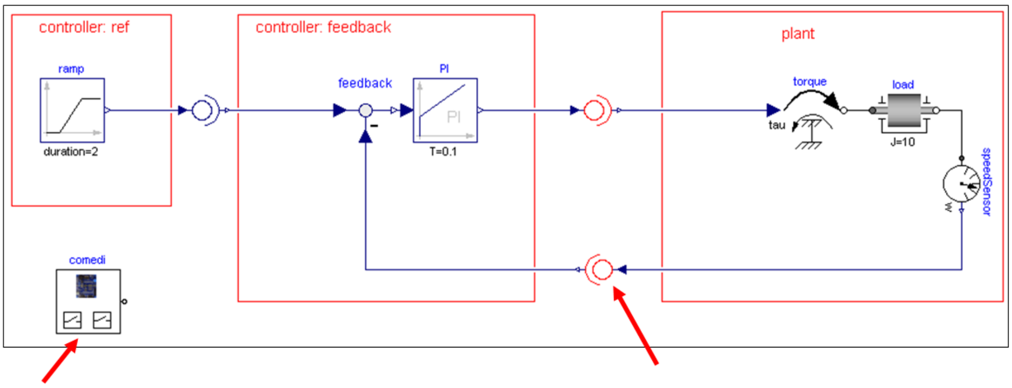
defineOutSubtask: false

inSubtask: Modelica_EmbeddedSystems.Configuration.Subtask()

outSubtask: Modelica_EmbeddedSystems.Configuration.Subtask()

Figure 3: Menu of a communication block.

- Direct communication with Modelica equations (simplest case: y = u). This is mainly used to start and have a meaningful default, and/or to test some controller effects like noise or signal delays.
- Communication between two subtasks. This defines that the input and output signals are in different subtasks. All properties of these subtasks can be configured with the rest of the options, e.g., that the input subtask is periodically sampled with a defined sampling rate.
- Communication between two tasks. This defines that the input and output signals are running in different tasks on the same machine. All properties of the tasks can be configured with the rest of the options, e.g., in which way the communication between the tasks takes place (e.g., via shared memory).
- Communication to a port. This defines that the input signal is sent to an I/O board or to a bus (like the CAN bus). In this case, the communication block has no output signal. All properties of the I/O board can be configured with the rest of



defines configuration of embedded system
(sampling, ECUs, initialization of device drivers, ...)

Defines
- splitting in task/sub-task,
- device drivers,
- references configuration

Figure 2: Simple drive train with two controller parts and three communication blocks.

the options, as well as the task/subtask properties of the equations that generate the signal to be sent to the I/O board.

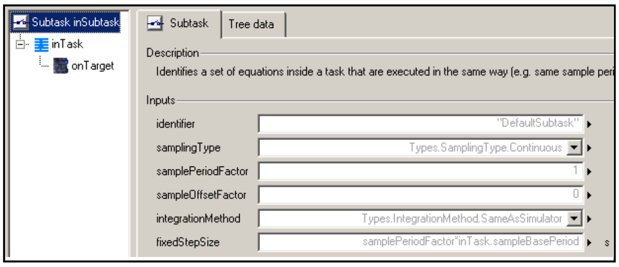
- Communication from a port. This defines that the output signal is received from an I/O board or from a bus. In this case, the communication block has no input signal. All properties of the I/O board can be configured with the rest of the options, as well as the task/subtask properties of the equations that use the received signal.

Depending on the selected option, input fields of the parameter menu are enabled. These are mostly replaceable models using Modelica ExternalObjects. For example, when clicking on "toPort", all currently loaded device drivers to send a signal to an I/O board are listed. Selecting the desired device driver and then clicking on the "table" symbol to the right of the menu, opens the driver specific menu to configure this particular device.

Whenever a user introduces a device driver that is derived by inheritance from one of the partial models defined in Modelica_EmbeddedSystems.Interfaces (like Interfaces.BaseReal.PartialWriteRealToPort), this device driver is automatically included in the corresponding list of the communication block, due to the “choicesAllMatching” annotation defined in this block.

The important point is that all these hardware configuration settings can be made without copying the "logical model" and modifying it, but just inheriting from it and applying modifiers on the communication blocks.

When clicking on "inSubtask", the subtask/task/target properties of the model part can be defined that is connected to the input of the communication block (in a similar way, the properties of the output can be defined with “outSubtask”). In Figure 4 a typical screen shot is shown:



Description

Identifies a set of equations inside a task that are executed in the same way (e.g. same sample peri

Inputs

identifier "DefaultSubtask"

samplingType Types.SamplingType.Continuous

samplePeriodFactor 1

sampleOffsetFactor 0

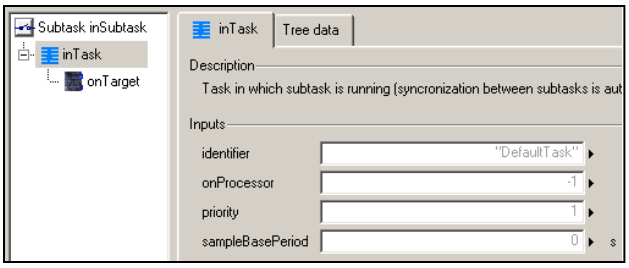
integrationMethod Types.IntegrationMethod.SameAsSimulator

fixedStepSize samplePeriodFactor*inTask.sampleBasePeriod s

Figure 4: Defining subtask properties of the input signal of a communication block.

This is a hierarchical structure where all properties of a mapping annotation can be defined (for details see section 4). In Figure 4, the top-most hierarchical dialog level is shown to define the identifier, the sampling properties and the integration method of the subtask in which the input signal is running. The subtask identifier is a string that must be unique within a task. If the same subtask shall be referenced in different communication blocks, identical subtask identifiers must be given.

The second level of the dialog is shown in Figure 5, to define the task identifier, the task priority, the basic sample period (if the task is periodically sampled) and the core, if the task is running on a multicore machine.



Description

Task in which subtask is running (syncronization between subtasks is aut

Inputs

identifier "DefaultTask"

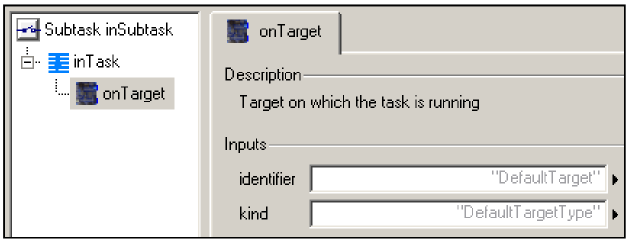
onProcessor -1

priority 1

sampleBasePeriod 0 s

Figure 5: Defining task properties.

Finally, the third level of the dialog (shown in Figure 6) is used to define the identifier of the target on which the task is running and the “kind” of the target. The kind property will identify in a tool-specific way all properties of the target machine that need to be known for the code generation (e.g. if the target has or does not has a floating point unit).



Description

Target on which the task is running

Inputs

identifier "DefaultTarget"

kind "DefaultTargetType"

Figure 6: Defining target properties.

From the information provided in the communication blocks, it is possible to partition all equations of the flattened Modelica model in to the desired pieces. E.g., in the example above, two different C-codes are generated, one for the two controller parts and one for the plant and both are running in different tasks. The reference and the feedback controller are running with different sampling rates in the same task (sub-sampling).

The algorithm to determine the equations that belong to the different parts is sketched in section 4.4. In short, the BLT-algorithm (usually used by Modelica translators to determine the sorting of the equations) must be applied two or three times additionally on the model equations. So, a Modelica tool vendor has the basic algorithm already and must just apply it in some variants.

Note, subtask properties (like sampling period) may be defined at the input and/or at the output of a subtask. If a subtask has several inputs and/or several outputs it is sufficient to define this information only at one location. For example in the use case of Figure 2, the properties of a subtask are defined at the output signal of the respective subtask.

3.2 Configuring Subtasks, Tasks, Targets and Devices.

All information to configure the subtasks, tasks and targets could be given in the hierarchical menus of the communication blocks shown in Figure 4, 5, and 6. However, this has a significant drawback: In the example above, the same target properties have to be defined three times (for all three different subtask definitions) and the task properties of the controllers have to be defined twice (for the "reference" and for the "feedbackController" subtask). Even more critical is the configuration of the device drivers. For example, an IO board is typically initialized once and then channel assignment takes place. It is difficult to define such initialization processes in the communication blocks.

In order to avoid redundant definitions and to have a simple way to initialize the device drivers, it is recommended to define all configuration options at one place once on the top level of the control system. An example is block "comedi" in the lower left part of Figure 2, where the control system is configured for real-time Linux using the comedi device drivers *(Comedi 2009*).

This block consists of record instances to define subtask, task and target properties of all parts of the control system and to initialize the used device drivers. For example, Modelica_EmbeddedSystems.-Configuration.Subtask is defined as:

```
record Subtask
   parameter Task inTask = Task();
   parameter identifier  = "Default";
   ...
end Subtask;
```

The first parameter in the record is "`inTask`" which is an instance of record "`Task`". In order to automatically have a hierarchical menu built up, a default value of "`Task()`" is given, i.e., the record constructor of record "`Task`" is called. When using the Subtask record in a configuration block, the "task" properties are defined in an instance of record "Task" (called "controller" in Figure 7). In the subtask definitions, like "slowSampler" and "fastSampler" in Figure 7, parameter "inTask" is defined as the instance name of the record task ("controller" in Figure 7). By this technique, the configuration is defined in a non-redundant way.

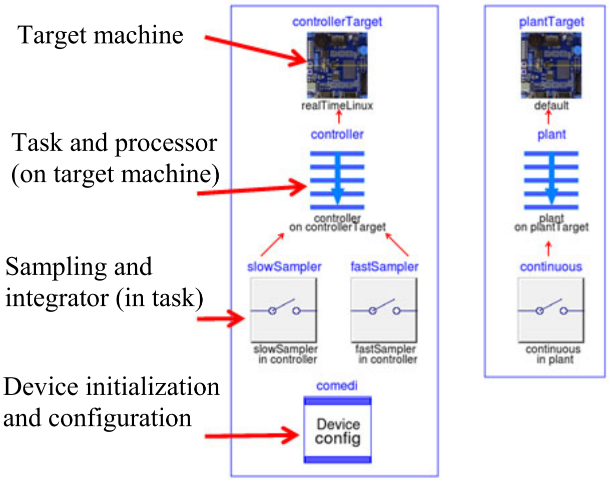

Target machine
Task and processor (on target machine)
Sampling and integrator (in task)
Device initialization and configuration
controllerTarget
realTimeLinux
controller
controller on controllerTarget
slowSampler
fastSampler
slowSampler in controller
fastSampler in controller
comedi
Device config
plantTarget
default
plant
plant on plantTarget
continuous
continuous in plant

Figure 7: Configuration of use case for real-time Linux with the comedi device drivers.

Device drivers are record instances where device-specific configuration options are given and a final parameter is used for the device handle that is defined by a call to the constructor of the respective ExternalObject.

Finally, a hierarchical modifier is used to reference the configuration record instances at the appropriate places in the communication blocks. For example, parameter "inSubtask" in the communication block at the output of the feedback controller in Figure 2 is defined as "comedi.fastSampler". In Dymola this can be conveniently defined by just clicking on the small arrow at the right part of the input field of "inSubTask" and selecting "Insert Component Reference". Dymola presents a list in which the instances (like comedi.fastSampler) are listed that can be utilized in the input field, and the modeler has just to select the appropriate entry.

3.3 Mapping to Target Data Types

Modelica has the four basic data types `Real`, `Integer`, `Boolean` and `String` that are usually mapped by Modelica translators to the C-types "`double`", "`int`", "`int`" and "`char*`", respectively. In many cases a different mapping is desired if the target is an embedded micro-controller. The details how this is defined is not standardized in Modelica 3.1. Standardization will occur when more practical experience is gained. In the meantime, vendor-specific enhancements have to be used.

In the simplest case, the "target.kind" definition in the "Target" record defines the type of the target machine. A vendor may associate different data type mappings for different target kinds. For example, a Modelica "Real" type may then be automatically mapped to a C "float" type.

For cheap microprocessors that do not have a floating point unit, such an automatic type mapping is not sufficient. For this purpose, vendor-specific variable annotations are planned that can be changed via hierarchical modifiers. The purpose is to allow definitions of the following form:

```
block Controller
  Real x(min=20, max=80);
  ...
end Controller;

Controller myController(x annotation(
   mapping(__NameOfVendor(
                 targetType = uint16,
                 min=10, max=100))));
```

The interpretation is that “x” is mapped on the target machine to an unsigned 16-bit integer “xi” with range `[0 .. 65535]` so that `x = 10.0` is mapped to `xi = 0` and `x = 100.0` is mapped to `65535`:

```
xi = round( (x - 10)*65535 / (100-10) )
```

The “targetMapping” min/max values (10 .. 100) might be different from the variable min/max values (20 .. 80) in order to have a margin so that operations on “xi” do not immediately cause overflow.

It is of course not practical to define such a data type mapping on every variable in a controller manually. Here, special tool support is needed. Possible, tool-specific approaches might be:

- All variables from the model part that shall be downloaded are displayed in a hierarchical variable browser and the GUI supports a convenient way to define the mapping quickly for sets of variables (e.g., all selected variables, or all variables of the same type in a particular hierarchy).
- The data type mapping might be defined for a few variables only (e.g., for the input variables). Via the equation-based relationships between variables, this mapping is propagated along the equations. E.g., if “a = b + c” and a data type mapping is defined for “b”, but not for “a” and “c”, then “a” and “c” are mapped in the same way as “b”. This approach is similar to the automatic “unit” propagation in Dymola.

This approach of data type mapping has the big advantage that <u>every Modelica model</u> can be utilized even on cheap microprocessors <u>without any changes</u> to the “logical” model (at least in principal). In contrast, the standard approach in many controller environments is much more restrictive: Every model has to be defined from the beginning in the desired data types of the expected target system. As a result, whenever a controller is designed, it must be re-implemented from scratch for a particular target system.

4 Modelica Language Extensions

In this section the Modelica language extensions and the needed algorithms are sketched to implement the approach discussed above. All the details can be found in the Modelica 3.1 Language Specification *(Modelica 2009, Chapter 16).*

4.1 Defining Subtask Boundaries

Boundaries of subtasks are identified with the following built-in operator (which is part of the built-in package “Subtask”):

```
Subtask.decouple(v);  // same as v
```

A boundary between a subtask A and a subtask B is defined by using this operator in an <u>equation</u> of subtask <u>A</u> with a <u>variable v</u> which is computed in subtask <u>B</u>. The operator returns its argument. Typically, this operator is used as:

```
u = Subtask.decouple(y);
```

where y is an output of subtask B and u is an input of subtask A. The effect is that “u = y” and “u” and “y” are in different subtasks.

4.2 Defining Subtask, Task and Target

The “**mapping**” annotation defines properties of variables. This annotation can only be applied on a declaration of a variable that does not have a **constant** or **parameter** prefix. It is usually applied on input and output variables of a subtask or a task. Example:

```
parameter Modelica.SIunits.Time Ts;
RealInput u annotation(mapping(
 target (identifier = "cluster"),
 task   (identifier = "slowTask",
         sampleBasePeriod = Ts),
 subtask(identifier = "reference",
         samplingType =
         Subtask.SamplingType.Periodic,
         samplePeriodFactor = 4)));
```

The meaning is that variable “u” is in the subtask “reference” which is periodically sampled with a sample period of “`4*Ts`”. Subtask “reference” is within task “slowTask” that has a base sampling period of Ts. Task “slowTask” shall be downloaded to the target machine with the name “cluster”.

The mapping annotation is formally defined by the following hierarchical record definition (as with all annotations, also here vendor-specific extensions can be added):

```
record mapping
  Boolean apply = true;
  Target     target ;
  Task       task   ;
  Subtask    subtask;
end mapping;
```

```
record Target
  String identifier="DefaultTarget";
  String kind = "DefaultTargetType";
end Target;

record Task
  String   identifier = "DefaultTask";
  Integer  onProcessor      = -1;
  Integer  priority         =  1;
  Modelica.SIunits.Period
           sampleBasePeriod =  0;
end Task;

record Subtask
  String identifier= "DefaultSubtask";
  Subtask.SamplingType samplingType =
        Subtask.SamplingType.Continuous
  Integer samplePeriodFactor(min=1)= 1;
  Integer sampleOffsetFactor(min=0)= 0;
  IntegrationMethod integrationMethod
                   = "SameAsSimulator";
 Modelica.SIunits.Period fixedStepSize;
end Subtask;
```

All values supplied to these records can be parameter expressions. If

```
mapping(apply = false,
        target (..),
        task   (..),
        subtask(..));
```

the “target(..), task(..), subtask(..)” definitions are ignored. This is, e.g., used to conveniently define in a parameter menu whether the input and/or the output signal of a communication block defines target/task/subtask properties without complicated Modelica code.

The mapping annotation defines that the respective variable is computed in the task with the identification “task.identifier” and with the task priority “task.priority”, on the target platform (e.g., computer, processor) with the identification “target.identifier” on processor “onProcessor”.

The interpretation of task.identifier, task.onProcessor, task.priority, target.identifier and target.kind is tool-dependent. For example, target.kind may identify a multi-processor or multi-core target machine and task.onProcessor may identify the processor or core on this target. Alternatively, a tool may identify a particular processor or core with target.kind and may ignore task.onProcessor.

The respective task may have one or more subtasks. A task is active when any of its subtasks is active. A subtask is defined with the following properties:

- If samplingType = Subtask.SamplingType.-Continuous, the subtask is a continuous system that is always active.
- If samplingType = Subtask.SamplingType.-Periodic, the subtask is periodically sampled with a sample period of “samplePeriodFactor * task.sampleBasePeriod” and an offset of “sampleOffsetFactor*task.sampleBasePeriod”. So sample period and sample offset are integer multiples of the task.sampleBasePeriod.
- The differential equations in a subtask are integrated according to the “integrationMethod” property. For fixed-step integration methods, a fixed integrator step size of fixedStepSize is used. A tool may adapt the selected fixed step size, e.g., by automatically restricting it to the time from the previous to the actual activation. Usually, fixedStepSize = samplePeriodFactor * task.sampleBasePeriod. In some applications, fixedStepSize might be smaller than one sample period, in order to have several integrator steps in one sample period since otherwise the fixed step size integration method might not be stable.

The mapping annotation influences the simulation result and therefore different simulation results might be obtained if this annotation is removed.

4.3 Inquiring Subtask Properties

In order for purely discrete models to be implemented, there are two operators to inquire properties of the subtask in which the model is running:

- **`Subtask.activated`**`()`:
  Returns true at the activation time instant of the subtask, where this operator is called. At all other time instants when the associated task is exeuted, including initialization, the operator returns false.
- **`Subtask.lastSampleInterval`**`()`:
  Returns the time instant from the activation time instant of a subtask to the previous activation time instant of the same subtask, where this operator is called.

If one of these operators is used, the corresponding subtask is not allowed to have subtask.sampleType = Subtask.SamplingType.Continuous.

In many standard cases these operators are not needed. Typically, a controller block, like a PI block, is implemented in its continuous form. When the subtask is periodically sampled, the Modelica translator automatically derives the discrete form, if an appropriate integration algorithm with fixed step-size is used. For linear control systems it is recommended to use the trapezoidal integration algorithm, since this gives the closest correspondence between the continuous and the sampled form and only small linear equation systems must be solved in the discrete form (since the trapezoidal method is an implicit integration algorithm).

In some cases, a controller has only a discrete representation. A typical example is a finite impulse response (FIR) filter. A mean value FIR filter, would be typically implemented in the following form:

```
block meanValueFilter
  import Modelica.Blocks.Interfaces;
  Interfaces RealInput  u;
  Interfaces.RealOutput y;
equation
  when {initial(), Subtask.activated()}
    then
    y = (u + pre(u)) / 2;
  end when;
initial equation
  pre(u) = u;  // steady state init
end meanValueFilter;
```

4.4 Partitioning a Model in to Parts

Via the **decouple**(..) operator and the **mapping** annotation, certain variables of a model are marked. In this section the algorithm is sketched how to derive all equations that belong to a particular subtask and task, respectively:

This requires the "Block Lower Triangular" (BLT) transformation to be applied several times. The BLT algorithm is, e.g., described in *(Pantelides 1988)*. This is the standard algorithm used in Modelica translators to sort the equations of a model and identify the algebraic loops. In *(Pantelides 1988)* it is shown that a differential-algebraic equation system does not have a unique solution[2], if all "**der**(v)" are replaced by "v" and a unique assignment for all unknown variables is not possible ("v" are treated as unknown variables in this case).

We will use a similar technique below. The motivation is that the "**der**(..)" and "**pre**(..)" operators act as "loop breakers" between equation systems. If "**der**(v1)" is replaced by "v1" and "**pre**(v2)" is replaced by "v2" and all "v1" and "v2" are treated as unknowns, then algebraic loops are present between all equations that need to be "treated" together.

Based on this observation we can now sketch the partitioning algorithm:

<u>1. BLT to determine the (asynchronous) tasks:</u>

If tasks are present, there are function calls to receive signals from another task or from external inputs and to send signals to another task or external outputs. From a Modelica point of view, there is no coupling between variables of different tasks (due to the function calls) and therefore the equations are naturally "cut" in to partitions. These partitions are determined by replacing all **pre**(v1) with v1 and all **der**(v2) by v2 and by performing a BLT transformation.

All BLT blocks that have variables with the same task.identifier annotation belong to the same task. If a BLT block B references one or more variables that are assigned in a BLT block A, that belongs to a task task.identifier, then all equations of B belong to task task.identifier. If a BLT block C references variables that are assigned in B, then all equations of C belong to task task.identifier, and so on. If a BLT block, directly or indirectly, references variables that are assigned in two different tasks, this is an error (wrong mapping annotations). All remaining BLT blocks that do not belong to any task are collected together to a "continuous" default task. This default task is usually running on the host machine or might also be deactivated (not running).

<u>2. BLT to determine the (synchronous) subtasks:</u>

This is achieved by inspecting all equations of every task. For every task, the **decouple**(v) operators are conceptually replaced by zero, so that "v" is no longer part of the equation where **decouple**(v) appeared. As a result, subtasks are decoupled. BLT is performed on all equations of a task by replacing all **pre**(v1) with v1 and all **der**(v2) by v2.

If one or more variables of a BLT-block A have a subtask annotation, the equations belong to this subtask S. If a BLT block B references one or more variables that are assigned in A, all equations of B belong also to S. If a BLT block C references variables that are assigned in B, all equations of C belong also to S, and so on. All remaining BLT blocks that do not belong to any subtask, are collected together to a "continuous" default subtask. It is an error, if a BLT block, directly or indirectly, references variables that are assigned in two different subtasks. If different subtasks have identical samplingType, samplePeriodFactor and sampleOffsetFactor, the subtasks can be merged (the subtasks are sampled at the same time instants but different integration methods are used for the subtasks).

<u>3. BLT to determine the sorted equations in a task:</u>

Standard BLT is performed on the equations of a task (identified in step 1) to determine the execution order of all equations. In this phase, every "**decouple**(v)" operator is replaced by "v". If sampled subtasks are present, the corresponding equations

[2] or more precisely, the system has an infinite index. If on the other hand all variables "v" have a unique assignment, then and only then, the "Pantelides" algorithm to determine the equations to be differentiated will converge.

(identified in step 2) must be guarded by if-clauses and must be only evaluated if the corresponding sampling event occurs. As a result the sorted (synchronous) equations of a task are obtained.

Note, due to the equation sorting, it is guaranteed that a variable reading from an input communication channel is only used after it is read and that a variable is first computed before writing it to an output communication channel.

The description above was made for clarity. It is, however, not the most efficient implementation. For example, it is possible to combine step 1 and 2, by just performing the BLT transformation according to step 2, i.e., in total only two and not three BLTs are needed . The task/subtask annotations are then used in a corresponding way to determine the tasks and subtasks.

5 Conclusions

In this article, a powerful extension to Modelica has been described that is used to define a "logical" model in Modelica and from this *same* model derive various "real" representations as needed for, e.g., Model-in-the-Loop, Software-in-the-Loop, or Hardware-in-the-Loop Simulation, as well as rapid prototyping, or generation of production code. With respect to standard approaches in state-of-the-art control design environments, no copying of model parts takes places and the data type on the target machine is defined as a mapping rule of the "logical" model.

The language elements are not complete and some features are missing. After getting more experience with this new way of describing control systems, more features will be standardized and missing features will be added. Especially, it is planned to include triggered subtasks in the next version.

6 Acknowledgements

Partial financial support of DLR by BMBF (BMBF Förderkennzeichen: 01IS07022F) for this work within the ITEA2 project EUROSYSLIB is highly appreciated (www.itea2.org/public/project_leaflets/EUROSYSLIB_profile_oct-07.pdf).

Dynasim thanks the Swedish funding agency VINNOVA (2008-02291) for partial funding for this work within the ITEA2 project MODELISAR.

Furthermore, we would like to thank our Modelica Association colleagues Ramine Nikoukhah (INRIA), Torsten Blochwitz and Gerd Kurzbach (ITI GmbH) for fruitful discussions.

References

Akesson J., Nordström U., Elmqvist H. (2009): **Dymola and Modelica_EmbeddedSystems in Teaching – Experiences from a Project Course**. In: F. Casella (editor): Proc. of the. 7th Int. Modelica Conference, Como. www.modelica.org/events/modelica2009

Bauschat, M., Mönnich, W., Willemsen, D., and Looye, G. (2001): **Flight testing Robust Autoland Control Laws**. In Proceedings of the AIAA Guidance, Navigation and Control Conference, Montreal CA.

Comedi (2009). Linux Control and Measurement Device Interface. www.comedi.org.

Dymola (2009). **Dymola Version 7.3**. Dassault Systèmes, Lund, Sweden (Dynasim). Homepage: www.dymola.com.

Franke R., Babji B.S., Antoine M., Isaksson A. (2008): Model-based online applications in the ABB Dynamic Optimization framework. In: B. Bachmann (editor): Proc. of the 6th Int. Modelica Conference, Bielefeld. www.modelica.org/events/-modelica2008/Proceedings/sessions/session3b1.pdf

Looye G., Thümmel M., Kurze M., Otter M., Bals J. (2005): **Nonlinear Inverse Models for Control**. In: G. Schmitz (editor): Proc. of the 4th Int. Modelica Conference, Hamburg. www.modelica.org/events/Conference2005/online_proceedings/Session3/Session3c3.pdf

Modelica (2009). **Modelica Language Specification 3.1**. www.modelica.org/documents/ModelicaSpec31.pdf

Pantelides C. (1988): **The consistent initialization of differential-algebraic systems**. SIAM Journal of Scientific and Statistical Computing, pp. 213-231.

Schäuffele J. and T. Zurawka (2005): **Automotive Software Engineering – Principles, Processes, Methods and Tools**. SAE International. ISBN-10 0-7680-1490-5.

Thümmel M., Otter M., Bals J. (2005): **Vibration Control of Elastic Joint Robots by Inverse Dynamics Models**. H. Ulbrich, W. Günthner (editors): IUTAM Symposium on Vibration Control of Nonlinear Mechanisms and Structures, München, ISBN 978-1-4020-4160-0, pp. 343-353.

A New Formalism for Modeling of Reactive and Hybrid Systems

Martin Otter[1], Martin Malmheden[2], Hilding Elmqvist[2], Sven Erik Mattsson[2], Charlotta Johnsson[3]
[1]German Aerospace Centre (DLR), Institute for Robotics and Mechatronics, Germany
[2] Dassault Systèmes, Lund, Sweden (Dynasim)
[3] Department of Automatic Control, Lund University, Sweden
Martin.Otter@dlr.de, Martin.Malmheden@3ds.com, Hilding.Elmqvist@3ds.com,
SvenErik.Mattsson@3ds.com, Charlotta.Johnsson@control.lth.se

Abstract

A new Modelica library is presented that is used to model safe hierarchical state machines in combination with any Modelica model, e.g., controllers, logical blocks, and physical systems described by differential-algebraic equations. It has been designed to simplify usage, improve safety aspects and to harmonize with the design of the new Modelica_EmbeddedSystems library. Furthermore, new blocks are introduced to define actions in a visual way, and not textually. The library is inspired by Statecharts, Sequential Function Charts, Safe State Machines (SSM) and Mode-Automata. It has been designed so that only small extensions to Modelica 3.1 are needed. The algorithms are sketched that are used to guarantee consistent graphs that give a limited number of event iterations. Furthermore, it is shown how a symbolic verifier can be used to guarantee additional properties of state machines.

Keywords: ModeGraph; Statechart, Sequential Function Charts, Mode-Automata, Safe State Machines; NuSMV; reactive systems, hybrid systems.

1 Introduction

In this article the open source Modelica_StateGraph2 library is presented. This is version 2 of the existing Modelica.StateGraph library *(Otter et. al. 2005)*. It is planned to replace Modelica.StateGraph in one of the next releases of the Modelica Standard Library with Modelica_StateGraph2. Besides the basic Step and Transition mechanism, all other parts have been redesigned and significantly improved based on the experience with the experimental ModeGraph library *(Malmheden et. al. 2008)*. Note, below the name "StateGraph" is often used as abbreviation for the full name "Modelica_StateGraph2".

The StateGraph library is inspired by Statecharts *(Harel 1987)*, Sequential Function Charts (SFC), Safe State Machines (SSM) *(André 2003)*, and Mode-Automata *(Maraninchi and Rémond 2002)*. The primary purpose of the library is to provide support for modeling of reactive and of hybrid systems and to verify certain properties of such systems.

Reactive systems react to stimuli from their environment, see, e.g. *(Benveniste et. al. 2003)*. In combination with the Modelica_EmbeddedSystems library *(Elmqvist et. al. 2009)*, the StateGraph library can be used to model such systems and it will be possible to use StateGraph models in production code of embedded systems.

Hybrid systems combine closely continuous-time models and discrete event systems, see, e.g. *(Lynch 2002)*. The StateGraph library is implemented with the Modelica language and therefore every Modelica model, i.e., models consisting of differential, algebraic and discrete equations, as well as functions, can be conveniently and naturally combined with state diagrams constructed with the StateGraph library.

2 Using Modelica_StateGraph2

In this section an overview is given of how to use the library by several small examples.

2.1 StateGraph Elements

A StateGraph graph is constructed by three elements: Step, Transition, and Parallel that will now be discussed in some detail.

Step
A Step is the graphical representation of a state and is said to be either active or not active. A StateGraph

DOI: 10.3384/ecp09430108

model is comprised of one or more steps that may or may not change their states during execution. A StateGraph model must have one initial step. An initial step is defined by setting parameter initialStep at one step to true. The initial step is visualized by a small arrow pointing to this step, see Step s1 in **Figure 1**.

Transition

To define a possible change of states, a Transition is connected to the output of the preceding Step and to the input of the succeeding Step, see, e.g., **Figure 1**, where Transition t1 defines the transition from Step s1 to Step s2. Note: A Transition has exactly one preceding and one succeeding Step. A Transition is said to be enabled if the preceding step is active. An enabled transition is said to be fireable when the Boolean condition defined in the parameter menu of the transition is evaluated to true. This condition is also called "Transition condition" and is displayed in the icon of the Transition. When parameter "use_conditionPort" is set, the Transition condition is alternatively defined by a Boolean signal that is connected to the enabled "conditionPort". A fireable transition will fire immediately. In **Figure 1**, t1 fires when s1 is active and time is greater than one.

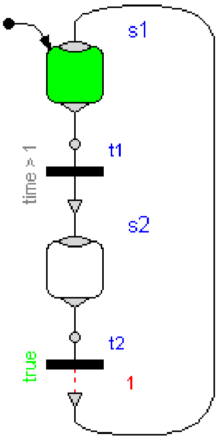


Figure 1: Model with two steps and two transitions

The firing of a transition can optionally also be delayed for a certain period of time. See, e.g., t2 in **Figure 1**, that is delayed for one second before it may fire, given that the condition remains true and the preceding Step remains active during the entire delay time. The evolution of a graph over time can be visualized by diagram animation: Active steps and Boolean variables that are true are marked in green here, see, e.g., **Figure 1**.

Parallel

Subgraphs can be aggregated into superstates by using the Parallel component. This component acts both as a composite step (having just one branch) and as a step that has parallel branches. The Parallel component, often referred to as "p" in the following figures, allows the user to place any StateGraph element inside it, especially Steps, Transitions, and Parallel components.

A Parallel component has always an entry port, see **Figure 2**, and it may have optionally an exit port. All branches in a Parallel Component must start at the entry port and at least one must terminate at the exit port, provided the exit port is enabled via parameter "use_outPort". If a Parallel component shall be entered from the outside via a Transition, parameter "use_inPort" must be set to true, to enable an input port. If a Parallel Component shall be left via a transition to an outside step, parameter "use_outPort" must be set to true, to enable the output and the exit port. A Parallel component may be used as initial step, by setting parameter initialStep to true. This property is again visualized by a small arrow pointing to the Parallel component, see **Figure 2**.

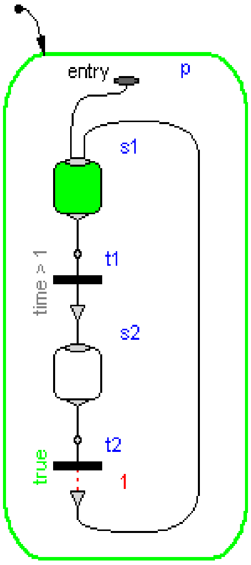


Figure 2: A Parallel component with a small sub-system.

A Parallel component may be suspended and subsequently resumed. In **Figure 3**, Transition T6 fires whenever the input signal u is true, suspending the Parallel component p and the enclosed Steps s2, s3,

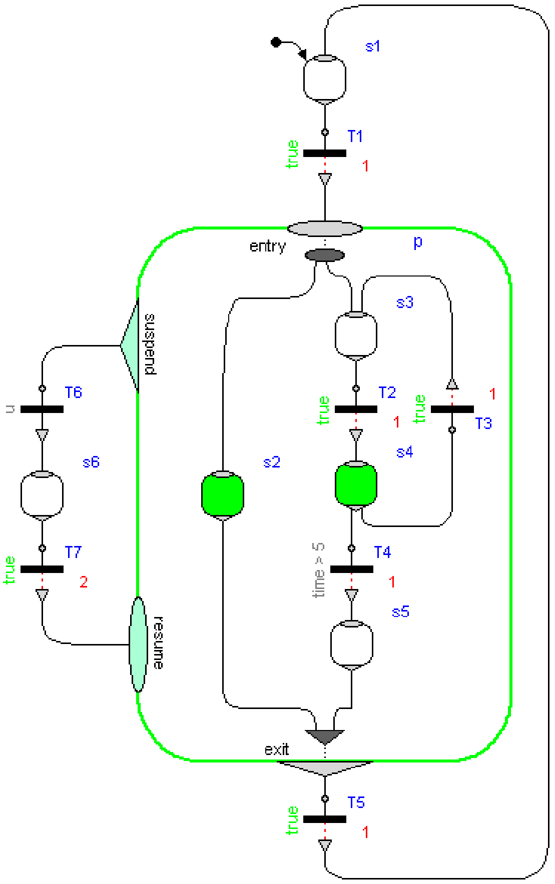


Figure 3: Parallel component with 2 parallel branches that is suspended whenever the input u is true.

s4 and s5 for two seconds. When Transition T7 fires, p is re-activated in the same state as when it was suspended.

As mentioned before, inPorts and outPorts of a Parallel component are optional and can be set by the user. If the parallel component has an inPort, then the entry port constitutes the connection between the Transition connected to the inPort and the first Steps to be activated in the Parallel component. If the Parallel component is configured to have an outPort, an exit port shows up on the bottom of the Parallel component, see **Figure 3**.

The Parallel component allows the entry port to branch out into several parallel paths. These branches are not allowed to interact, see **Figure 3**. When all Steps connected to the exit port are active, the Parallel component is said to be available and may exit when the Transition connected to the outPort fires. In **Figure 3** Transition T5 fires when both Step s2 and s5 have been active together for one second and thereby deactivates the Parallel component p. Note, in Statecharts parallel branches must be synchronized via transition conditions, which is inconvenient. In SFC, all branches are synchronized. In StateGraph, only branches that are connected to the exit port are synchronized, which is more flexible as the SFC approach.

No component contained within the Parallel component may be connected to any other component "outside" of the Parallel component. This rule is used to protect the user from making mistakes that could lead to unexpected results and states of the graph that are not well-defined. Consider for example the graph in **Figure 4Figure** at T=7. Especially, note that the Parallel component p is never properly terminated through either an outPort or a suspend port. If the graph would be allowed to execute, the consequence would be an increasing number of active Steps. Such a situation is reported as an error. The details about the algorithm to accomplish this are given in appendix A2.

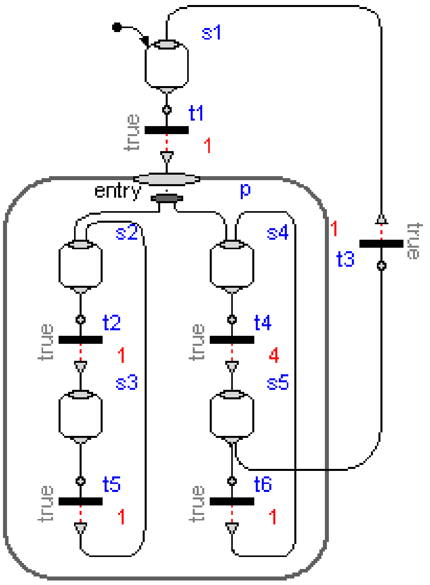


Figure 4: Wrong graph since components in the Parallel component may not be connected to "outside" ones.

In order to graphically organize large graphs in different levels of hierarchy and with encapsulation of variables, StateGraph also contains a component PartialParallel. It is similar to the normal Parallel component but introduces a new hierarchy once the user inherits from it. A number of large subsystems can thus be abstracted into composite steps to improve organization and overview of the subsystems. **Figure 5** shows a component built from a PartialParallel component. As the diagram and the icon layer of the PartialParallel component does not need to be the same size, the user can benefit from collecting large subsystems in smaller closed Parallel components to improve overview and modularization of the full system.

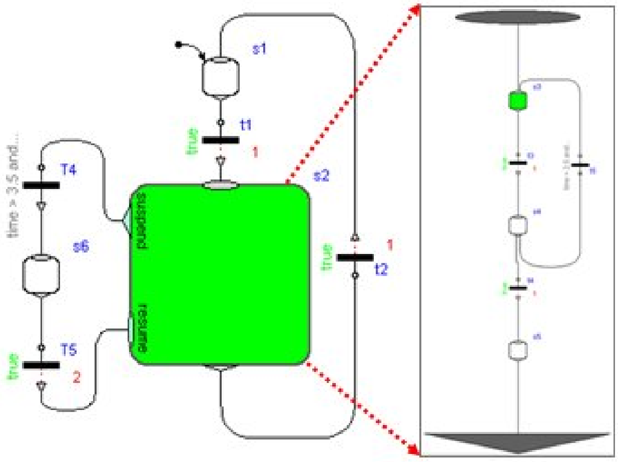
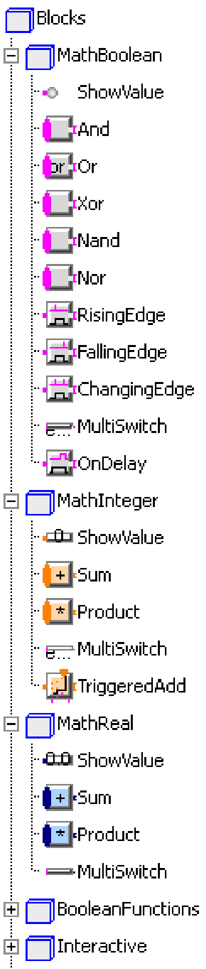


Figure 5: Composite derived from PartialParallel component and its subsystem.

2.2 Graphical Action Blocks

An important practical aspect of state machines is the ability to assign values and expressions to variables depending on the state of the machine. In StateGraph, a number of graphical components, see **Figure 6**, have been added to facilitate usage in a safe and intuitive way. Since these are just input/output blocks and will also be useful in another context, it is planned to add them to the Modelica Standard Library under "Modelica.Blocks". Some of these blocks will be explained in this section.

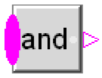
There are a number of standard blocks with common operations/displays of the three basic types (Boolean, Integer, Real) using vector input connectors which enables them to be connected to an arbitrary number of sources. Resizing a vector port and



Blocks
MathBoolean
ShowValue
And
Or
Xor
Nand
Nor
RisingEdge
FallingEdge
ChangingEdge
MultiSwitch
OnDelay
MathInteger
ShowValue
Sum
Product
MultiSwitch
TriggeredAdd
MathReal
ShowValue
Sum
Product
MultiSwitch
BooleanFunctions
Interactive

Figure 6: Blocks to define actions graphically.

connecting to the next free element is performed automatically when connecting to the connector, see Appendix A4. So this is much more convenient than with the Modelica.Blocks.Logical, Modelica.StateGraph or ModeGraph libraries. A vector of input connectors is visualized as an ellipse, see, e.g., the violet connector on the left side of the "**and**" block in the figure to the right where "`y = u[1] and u[2]and ...`".



A MultiSwitch block selects one of n expressions depending on an array of Boolean inputs. The index of the first input in the Boolean array that is true defines the index of the expression in the expression array to be used as the scalar output y. In **Figure 7**, the MultiSwitch component will output the value y = 1 if Step s1 is active, and will output y = 2 if s2 is active as the expression array is defined as {1,2}. If none of the Boolean array inputs is true, the "**else**" value will be used instead that is defined in the parameter menu of the MultiSwitch component and is displayed below the icon. Consider **Figure 7** when Step s3 is active – this will set the output of component "multiSwitch" to the "else" value "3". Alternatively, in the parameter menu of the MultiSwitch component it can be defined to keep its previous value, i.e. y = **pre**(y). If this option would be selected for **Figure 7**, then multiSwitch.y = 2 when Step s3 is active.

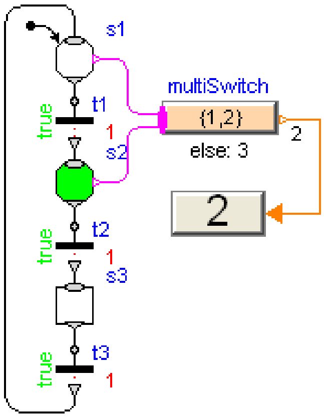


Figure 7: Example of MultiSwitch component for Integer numbers that depends on different steps.

The MultiSwitch block is inspired by "Modes" from Mode Automata *(Maraninchi and Rémond 2002)*: Variable multiSwitch.y has always a unique value, and this value depends on the expressions that are associated with mutually exclusive active steps. The advantages of MultiSwitch are that (1) the definition is performed in a purely graphical way, (2) it can also be used for mutually non-exclusive active steps[1], and (3) it can be implemented in Modelica in a very simple way. The drawback is that the expressions in the MultiSwitch block might no longer be so easy associated with Steps, compared to the alternative where the expressions are defined directly in the respective Steps. This latter approach would, however, require non-trivial extensions to the Modelica language.

The RisingEdge, FallingEdge and ChangingEdge components can be used to generate "pulse" signals depending on the rising, falling or changing values of Boolean signals. An example is shown in **Figure 8** where the Boolean indicator lamp is turned on when Step s2 becomes active and is turned off when Transition t3 fires and Step s3 becomes inactive. Two variants are shown to utilize the "rising" property of a Boolean signal: The Boolean connectors at steps and transitions can be activated via parameters "use_activePort" and "use_firePort", respectively. If s2 becomes active, rising = true and therefore multiSwitch.y = true. If transition t3 fires, t3.firePort=true and therefore multiSwitch.y = false.

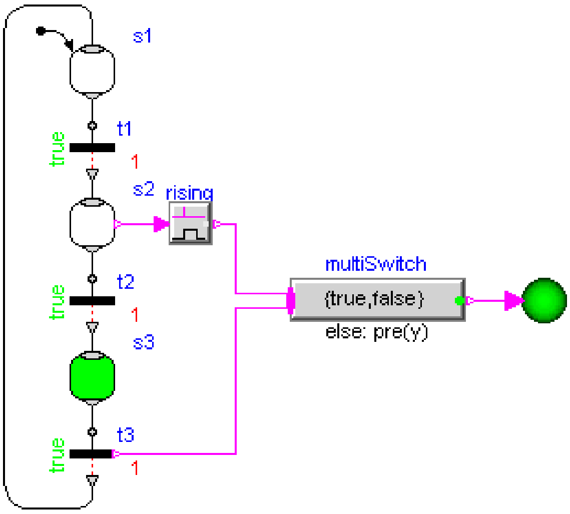


Figure 8: Two variants to control a Boolean indicator by a MultiSwitch component.

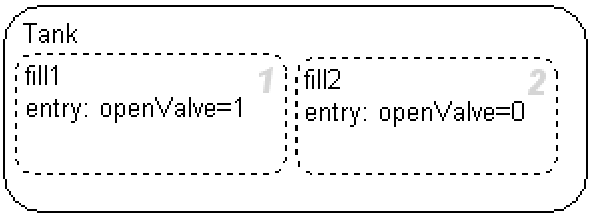
[1] If an MultiSwitch block is connected to steps of different branches of a Parallel component, a priority is present: If several inputs are true, then the one has highest priority that is connected to the lowest index of the vector of input connectors (= connection line "closest" to the icon name).

2.3 Safe StateGraph models

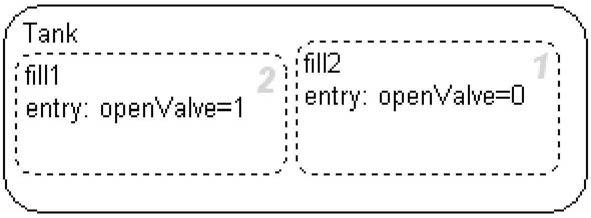
In this section it will be discussed in which sense "StateGraph" models are "safe".

Only valid graph structures are accepted
Contrary to Modelica.StateGraph (version 1 of the library which is distributed with the Modelica Standard Library since 2004), only valid graph structures are accepted for the Modelica_StateGraph2 library. For example, the model of **Figure 4** leads to an error. In order that this was possible, Modelica 3.1 had to be enhanced slightly. Details are given in section A2.

One variable is defined by one equation
In all state machine formalisms problems are present when assignments to the same variables are performed in branches that are executed in parallel. As an example, in the next figure such a situation in Stateflow *(StateFlow 2009)* is shown:



The two substates "fill1" and "fill2" are executed in parallel. In both states the variable "openValve" is set as entry action. The question is whether openValve will have value 0 or 1 after execution of the steps. Stateflow changes this non-deterministic behavior to a formally deterministic one by defining an execution sequence of the states that depends on their graphical position. The grey number on the right of the states shows in which order the states are executed. In the figure above this means that "openValve=0" after leaving the two states. If the second state "fill2" is changed a little bit graphically



"openValve=1" after "fill1" and "fill2" have been executed. This is a critical situation because (a) slight changes in the graphical positioning of states might change the simulation result and (b) if the parallel execution of actions depends on the evaluation order, errors are difficult to detect.

In StateGraph such a situation is not possible. The reason is that StateGraph is implemented in Modelica and a very basic feature of Modelica is that every declared unknown variable must be defined by exactly one equation. This is sometimes called "single assignment rule". It is therefore not possible to assign the same variable twice in a model. The above situation would be described in StateGraph instead with a MultiSwitch action block "openValve" as shown in **Figure 9**. Here, everything is well defined: There are two input connections to the openValve block. If both become true at the same time instant, the connection with the "lowest" index (i.e., the upper signal in the figure) has highest priority. Therefore, openValve gets the value true, once the Parallel component is entered.

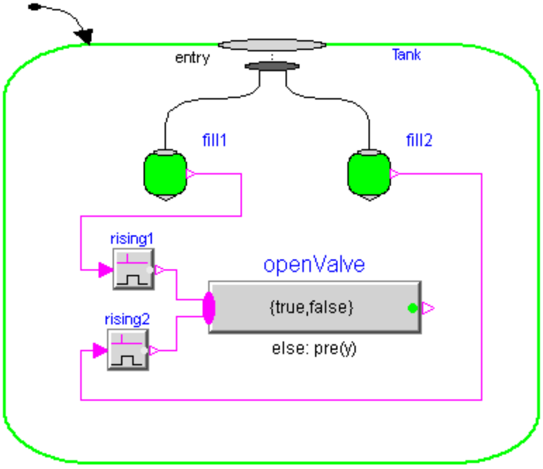


Figure 9: Assignment of variables with forced priority due to Modelicas single assignment rule.

Upper bound on number of model evaluations
At an event instant, an event iteration occurs, due to the Modelica semantics (= whenever a new event occurs, the model is re-evaluated). This means that Transitions keep firing along a connected graph, as long as the firing conditions are true. The question therefore arises, whether infinite event looping is possible? A simple example of this kind is shown in **Figure 10**. Here, all Transition conditions are true and therefore all Transitions fire forever. This is no valid StateGraph model and will result in an error.

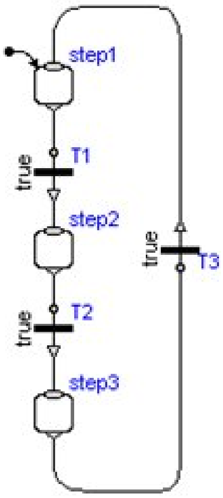


Figure 10: Wrong graph that gives rise to infinite looping.

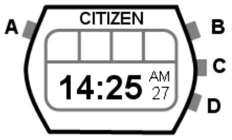
In order to avoid a situation as in **Figure 10**, it is required that a StateGraph model has at least one delayed Transition per loop, see Appendix 0. This means that one of T1, T2, or T3, must be a delayed Transition, otherwise an error occurs. Since event iteration stops at a delayed Transition, infinite event looping cannot occur. This also means that at one time instant every Transition can fire at most once and therefore the number of model evaluations at an event instant is bounded by the number of Transition components.

It is still possible that infinite event looping occurs due to model errors in other parts of the model. For example, if a user introduces an equation of the form “J = pre(J) + 1” outside of a when-clause, event iteration does not stop. Although this situation is not completely satisfactory, it helps already a lot if a tool points out potential problems of a StateGraph model, in case delayed transitions are missing.

3 Application Examples

In this section some involved application examples are shown to demonstrate the usage of the StateGraph library. These and other examples are available in the library under “Examples.Applications”.

3.1 Harels wristwatch



When presenting the Statecharts formalism in *(Harel 1987)*, David Harel identified and described the behavior of his Citizen Quartz Multi-Alarm III wristwatch (see schematic figure to the right) using the new visual formalism as a case study to proof his new formalism to be flexible enough to describe the intricate structure of the wristwatch behavior in a comprehensible and clean way. As the wristwatch example serves as a challenging benchmark for the capabilities of a graphical formalism, it has been included as an application example in the StateGraph library to demonstrate that the library is flexible enough to realize this example in a good way. It also serves as a template for other human interfaces. For example, an automotive cruise control has several switches and some of them have different levels. There are different influences if in cruise mode or not.

The wristwatch display is comprised of a number of different display modes showing the current time (displayed in either 12h or 24h mode), time setting (also in either 12h/24h mode), date/date setting (day, month, day of week, year etc.), alarm setting, chime setting, and a stopwatch display. The stopwatch can be turned on, off, stalled when running to show lap time and reset when stopped. The chime functionality is triggered each time the clock reaches a whole hour that makes the chime beep for two seconds. Furthermore, the wristwatch has two concurrently running alarms that sound when the time hits their respective configured time, display back-light for improved illumination, alarm test functionality and low battery warning.

The wristwatch is operated by four buttons A, B, C and D. Button A switches between the different modes where time and date can be set, alarms and chime can be set and turned on/off and the stopwatch can be run, paused and reset. When in a time-, date-, alarm- or chime-setting mode, button C can be used to flip through between different quantities that the current time/alarm/chime-setting can be incremented with the currently chosen quantity using button D.

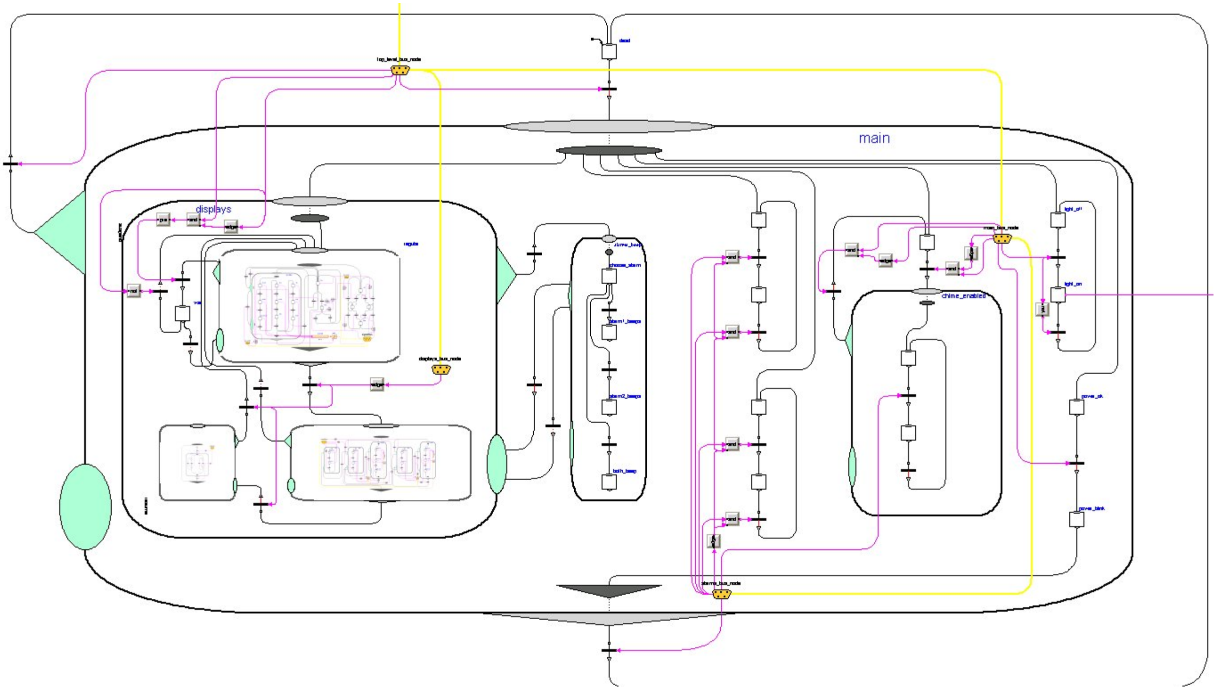


Figure 11: Top level of the StateGraph that defines Harels wristwatch.

When updating the time or an alarm time, button B can be used to immediately return to displaying either time or the current setting of the current alarm.

There are in total six concurrently running subsystems (Main – containing all the display and setting behavior, Alarm 1 Status, Alarm 2 Status, Chime Status, Back-Light and Power Status) that independently of each other react to the user input and the current time. There is also interaction between the Main subsystem and the Alarms/Chime Status to make it possible to concurrently guard the status of each functionality depending on the current time but also provide means to update their setting using only the given four buttons. The Main view of the StateGraph implementation of the wristwatch can be seen in **Figure 11**.

3.2 Controlled tank system

As another application example, the control of a tank system is present in the StateGraph library. This example is based on a similar system from *(Dressler 2004)*, which in turn is based on an example model of Karl Erik Årzén from the JGraphCharts manual. The top level view is shown in **Figure 12**: On the right side a two-tank system is present which is modeled with the Modelica.Fluid library *(Franke et. al. 2009)*: It consists of an infinite reservoir of water, “reservoir”, that flows via two tanks, “tank1, tank2”, to the environment, “ambient”. The flow can be controlled by three valves, “valve1, valve2, valve3”. There are three buttons, “start”, “stop”, “shut”, to control the operation. The actual level of a tank is measured in an ideal way by accessing variables tank1.level and tank2.level. All variables are communicated via an ideal bus “bus” to the tank controller. The basic operation is to fill and empty the two tanks:

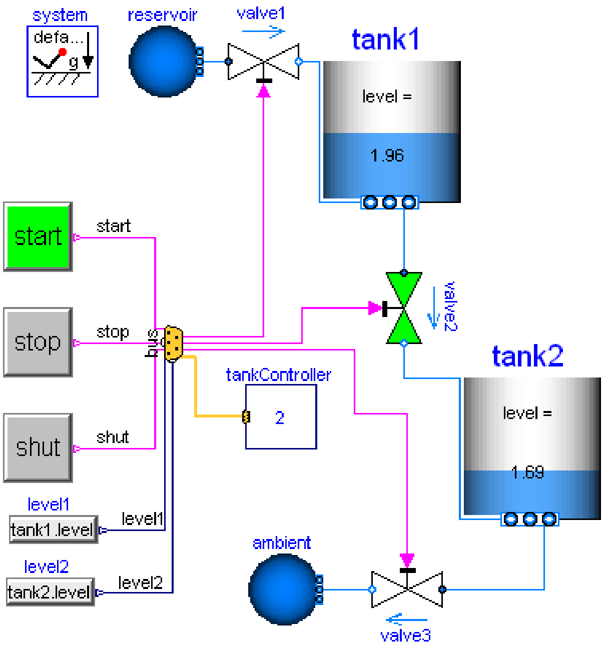


Figure 12: Two tank system controlled by 3 buttons.

1. Valve 1 is opened and tank 1 is filled.
2. When tank 1 reaches its fill level limit, valve 1 is closed.
3. After a waiting time, valve 2 is opened and the fluid flows from tank 1 into tank 2.
4. When tank 1 is empty, valve 2 is closed.
5. After a waiting time, valve 3 is opened and the fluid flows out of tank 2
6. When tank 2 is empty, valve 3 is closed

The above "normal" operation can be influenced by three buttons:

- Button “start” starts the above process. When this button is pressed after a "stop" or "shut" operation, the process operation continues.
- Button “stop” stops the above process by closing all valves immediately. Then, the controller waits for further input (either "start" or "shut").
- Button “shut” is used to shutdown the process, by emptying both tanks at once. When this is achieved, the process goes back to its start configuration. Clicking on "start", restarts the process.

The tank controller is hierarchically modeled with two Parallel components and some logical blocks:

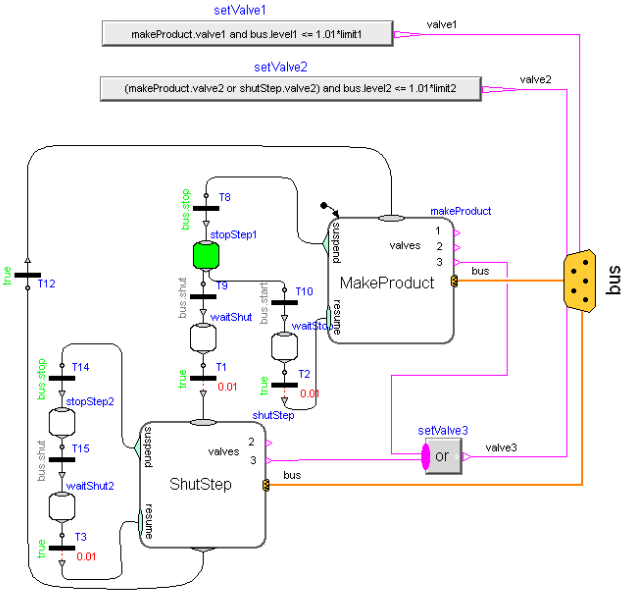


Figure 13: Top level view of tank controller logic.

The “MakeProduct” Parallel component is the initial step and performs the “normal” operation. When the “stop” button is pressed, the suspend transition T8 fires, the “MakeProduct” step is suspended and the graph goes in to step “stopStep1”. Note, the transition condition of T8 is “bus.stop”, i.e., this transition fires when variable stop from the bus is true. When "start" is pressed again, the "MakeProduct" step is resumed at the place where it was suspended. When "shut" is pressed, the Parallel component "ShutStep" is entered to shut down the tank system. Here it is still possible to press the "stop" button and then again continue with "shut".

4 Formal definition of StateGraph

In section 2.1 an informal introduction to the StateGraph formalism was given. In this section, a precise mathematical description of StateGraph models will be presented. The formal definition describes the structure of a StateGraph model and its interpretation algorithm (= semantics).

4.1 Structure of a StateGraph model

A StateGraph model, Γ, is described by a 4-tuple:

$$\Gamma = < V_c, G, T, g_I >$$

where

- V_c is a set of Boolean expressions. Boolean expressions are used as conditions of transitions. They are either external inputs or the outputs of Modelica models. A Modelica model consists of a set of differential, algebraic and discrete equations, see *(Modelica 2009, Appendix C)*.
- G is the set of Generalized Steps, $G = \{g_1, g_2, ...\}$. A Generalized Step $g_i \in G$ can be active or not active, signaled by the Boolean Active(g_i). A Generalized Step $g_i \in G$ is described by the 5-tuple
  $< I, R, O, S, \Gamma_s >$
  where
  I is a vector of in (entry) ports $I = [i_1,i_2,...]$,
  R is a vector of resume ports $R = [r_1,r_2,...]$,
  O is a vector of out (exit) ports $O = [o_1,o_2,...]$,
  S is a vector of suspend ports $S = [s_1,s_2,...]$,
  Γ_s is a set of sub-graphs $\Gamma_s = \{\gamma_1,\gamma_2,...\}$
  A Generalized Step g_i that has only in and out ports, $< I, O >$, is also called Step.
  A Generalized Step g_i where R, S or Γ_s is not an empty set, is also called Parallel Step.
  A sub-graph $\gamma_i \in \Gamma_s$ is described by a 5-tuple $< V_c, G, T, g_I, g_E >$ where V_c, G are a set of Boolean expressions and a set of Generalized steps as described above, T is the set of Transitions as described below, $g_I \in G$ is the initial generalized step that is first activated when the sub-graph γ_i is "normally" activated and $g_E \in \{\emptyset, G\}$ is the optional exit generalized step that is the last active step, before the sub-graph γ_i is de-activated.
- T is the set of transitions, $T = \{t_1, t_2, t_3, \ldots\}$. A transition $t_i \in T$ is defined by the 4-tuple
  $t_i = < p_{IR}(t_i), p_{OS}(t_i), \text{Condition}(t_i), \text{Delay}(t_i) >$
  where
  $p_{IR}(t_i)$ is a connected port of an in or resume vector of a succeeding generalized step $g_i \in G$.
  $p_{OS}(t_i)$ is a connected port of an out or suspend vector of a preceding generalized step $g_i \in G$.
  Condition(t_i) $\in V_c$ is the fire condition associated with t_i
  Delay(t_i) $\in \{\emptyset, R^+\}$ is the optional delay time associated with t_i. If present, the delay time is a positive real number, Delay(t_i) > 0.
  There is the restriction, that every "loop" must have at least one transition t_i with Delay(t_i) > 0 in order to avoid infinite transition looping.
- g_I is the initial generalized step, $g_I \in G$.

4.2 Interpretation Algorithm

The dynamic behavior of a StateGraph $\Gamma = < V_c, G, T, g_I >$ is given by the interpretation algorithm presented below:

(1) The initial step g_I is activated. If the initial step has sub-graphs $\gamma_i \in \Gamma_s$, then all initial steps g_I of these sub-graphs are activated as well. If an initial step $g_i \in G_I$ of a sub-graph has again sub-graphs, then all initial steps of these sub-graphs are recursively activated.

(2) Active(g_i) of all Generalized Steps, including all recursive sub-graphs, is set to true, if g_i is active. Otherwise it is set to false. Models are solved using Active(g_i) as inputs.

(3) The condition expressions of all transitions in T and in all recursive sub-graphs are evaluated (either from external inputs or from outputs of models).

(4) All Transitions are determined where (a) the Transition condition is true and (b) the preceding Generalized Step is active and (c) all exit steps g_E of all sub-graphs $\gamma_i \in \Gamma_s$ of the preceding Generalized Step are active, as well as of all exit steps of sub-graphs of exit steps recursively. For every such Generalized Step, at most one Transition can fire. For Transitions having the same preceding Generalized Step, the one connected to the out port or if both are connected to the same port vector, the one with the smallest vector index of the out or the suspend port respectively is marked as "fires".

(5) All Transitions that are marked as "fires" in (4) are firing, i.e., the respective preceding Genera-

lized Step of a Transition is deactivated and the succeeding Generalized Step of the Transition is activated. If a transition has a non-zero delay-time, it fires after the delay time, provided all conditions of (4) remain true during the delay time.

Deactivating a Generalized Step that has sub-graphs $\gamma_i \in \Gamma_s$ means, that all Generalized Steps in these sub-graphs and their recursive sub-graphs are deactivated as well.

Activating a Generalized Step that has sub-graphs $\gamma_i \in \Gamma_s$ means that either (a) all initial steps g_I of these sub-graphs and their recursive sub-graphs are activated, or (b) the Generalized Steps are activated that have been active when this step was deactivated the last time. Case (b) is used, if the last deactivation of this step was performed via a transition of a suspend port. Otherwise case (a) is used.

Goto (2).

4.3 Example

The StateGraph given in **Figure 3** can be presented by the 4-tuple Γ.

$$\Gamma = < V_c, G, T, g_I >$$

where

$V_c = \{true, u\}$,
$G = \{s1, s6, p\}$
$s1 = < i[1], o[1], \varnothing, \varnothing, \varnothing >$
$s6 = < i[1], o[1], \varnothing, \varnothing, \varnothing>$
$p = < i[1], o[1], s[1], r[1], \{\gamma_1, \gamma_2\} >$
$T = \{T1, T5, T6, T7\}$
$T1 = < s1.o[1], p.i[1], true, Delay(T1)=1 >$
$T5 = < p.o[1], s1.i[1], true, Delay(T5)=1 >$
$T6 = < p.s[1], s6.i[1], u, \varnothing >$
$T7 = < s6.o[1], p.r[1], true, Delay(T7)=2 >$
$g_I = s1$

and a sub-graph can be represented by the 5-tuple $< V_c, G, T, g_I, g_E >$:

Sub-graph γ_1:
$V_c = \{\varnothing\}$,
$G = \{s2\}$
$s2 = < i[1], o[1], \varnothing, \varnothing, \varnothing >$
$T = \{\varnothing\}$
$g_I, = s2$
$g_E = s2$

Sub-graph γ_2:
$V_c = \{true, time > 5\}$,
$G = \{s3, s4, s5\}$
$s3 = < i[2], o[1], \varnothing, \varnothing, \varnothing >$
$s4 = < i[1], o[2], \varnothing, \varnothing, \varnothing >$
$s5 = < i[1], o[1], \varnothing, \varnothing, \varnothing >$
$T = \{T2, T3, T4\}$
$T2 = < s3.o[1], s4.i[1], true, Delay(T2)=1 >$
$T3 = < s4.o[2], s3.i[2], true, Delay(T3)=1 >$
$T4 = < s4.o[1], s5.i[1], true, Delay(T4)=1 >$
$g_I, = s3$
$g_E = s5$

5 Verification of StateGraph models

Even if a state machine is checked to be structurally correct, its behavior might be faulty and dangerous. A typical example is if the behavior would deadlock, i.e., that no further transitions can be performed. Such behavior is related to the action and transition logic, not only to the topology of the StateGraph itself. Dymola *(Dymola 2009)* has been experimentally extended to extract all Boolean equations in order to facilitate model checking with external tools. The language used is SMV (Symbolic Model Verification) and the tool used is NuSMV *(NuSMV 2009)*.

Consider the example in **Figure 14**. It has four independent StateGraph models, two are modeling some processes which compete on using two resources. The allocations of the resources are done in opposite order which means that there is a risk of deadlock. Detecting such potential problems is in general hard. Dymola produces code in SMV as shown below:

```
freeA_inport_fire := release1A_fire |
release2A_fire;
next(pre_freeA_newActive) :=
  freeA_inport_fire | freeA_active &  !
  freeA_outport_fire;
```

Relations are converted to unknown inputs. When-clauses are converted to if (case) according to Modelica specification. Condition for non-deadlock is expressed using temporal logic according to the Computational Tree Logic syntax, e.g.:

```
_Dymola_SMV(
"CTLSPEC AG (! pre_freeA_newActive ->
 EF pre_freeA_newActive)");
```

The String argument to the special built-in function _Dymola_SMV means "For **A**ll states such that **not** pre_freeA_newActive (resource A not free) there **E**xists eventually in the **F**uture a state when pre_freeA_newActive (resource A free)"

NuSMV uses a BDD (Binary Decision Diagram) algorithm to verify the specification (NuSMV command check_ctlspec). If the specification is not always true, NuSMV presents a sequence of input events that will show the failure, i.e., in this case deadlock. Such a deadlocked situation is shown in Figure 14 with active Steps marked green.

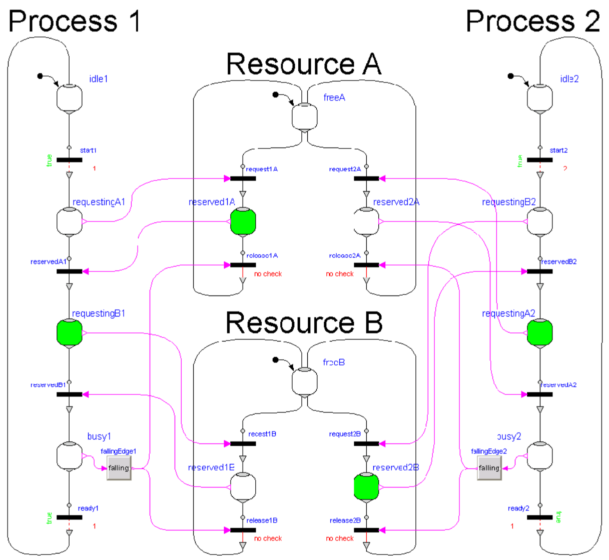


Figure 14: Two processes trying to acquire two resources ending up in a deadlock.

6 Conclusion

A new library Modelica_StateGraph2 was presented to model safe hierarchical state machines in combination with any Modelica model, e.g., controllers, logical blocks, functions and physical systems described by differential-algebraic equations. The library is designed to model the logic of reactive systems and to describe hybrid systems. The library is freely available from www.modelica.org/libraries, it is distributed in Dymola 7.3, and it is planned to include it in one of the next versions of the Modelica Standard Library. The work on the library will continue especially to take advantage of the features of the Modelica_EmbeddedSystems library *(Elmqvist et. al. 2009)*:

7 Acknowledgements

Partial financial support of DLR by BMBF (BMBF Förderkennzeichen: 01IS07022F) for this work within the ITEA project EUROSYSLIB (www.itea2.org/public/project_leaflets/EUROSYSLIB_profile_oct-07.pdf) is highly appreciated.The authors also would like to thank Daniel Weil from Dassault Systèmes for fruitful discussions.

References

André, C. (2003): **Semantics of S.S.M (Safe State Machine)**. I3S Laboratory – UMR 6070 University of Nice-Sophia Antipolis / CNRS. www.i3s.unice.fr/~map/WEBSPORTS/Documents/2003a2005/SSMsemantics.pdf

Bauschat, M., Mönnich, W., Willemsen, D., and Looye, G. (2001): **Flight testing Robust Autoland Control Laws**. In Proceedings of the AIAA Guidance, Navigation and Control Conference, Montreal CA.

Benveniste A., Caspi P., Edwards S.A., Halbwachs N., Le Guernic P., and Simone R. (2003): **The Synchronous Languages Twelve Years Later**. Proc. of the IEEE, Vol., 91, No. 1. Download: www.irisa.fr/distribcom/benveniste/pub/synch_ProcIEEE_2002.pdf

Dressler I. (2004): **Code Generation From JGrafchart to Modelica**. Master thesis. Supervisor: Karl-Erik Arzen, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden. www.control.lth.se/documents/2004/5726.pdf

Dymola (2009). **Dymola Version 7.3**. Dassault Systèmes, Lund, Sweden (Dynasim). www.dymola.com/.

Elmqvist H., Otter. M., Henriksson D., Thiele B., Mattssson, S.E. (2009): **Modelica for Embedded Systems**. In Proc. of Modelica'2009 Conference, Como, Italy. www.modelica.org/events/modelica2009

Franke R., Casella F., Otter M., Proelss K., Sieleman M., Wetter M. (2009): Standardization of thermo-fluid modeling in Modelica.Fluid 1.0. In Proc. of Modelica'2009 Conference, Como, Italy. www.modelica.org/events/modelica2009

Harel, D. (1987): **Statecharts: A Visual Formalism for Complex Systems**. Science of Computer Programming 8, 231-274. Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel. www.inf.ed.ac.uk/teaching/courses/seoc1/-2005_2006/resources/statecharts.pdf

Lynch N., Segala R., and Vaandrager F. (2002): **Hybrid I/O Automata**. MIT Laboratory for Computer Science, techreport, MIT-LCS-TR-827b. Download: theory.lcs.mit.edu/tds/papers/Lynch/HIOA-final.ps

Malmheden M., Elmqvist H., Mattsson S.E., Henriksson D., and Otter M. (2008): **ModeGraph - A Modelica Library for Embedded Control Based on Mode-Automata**. B. Bachmann (editor), in Proc. of Modelica'2008 conference, Bielefeld, Germany. www.modelica.org/events/modelica2008/Proceedings/sessions/session3a3.pdf

Maraninchi, F. and Rémond, Y. (2002): **Mode-Automata: a New Domain-Specific Construct for the Development of Safe Critical Systems.** wwwverimag.imag.fr/~maraninx/SCP2002.html

Modelica (2009). **Modelica Language Specification 3.1**. www.modelica.org/documents/ModelicaSpec31.pdf

Mosterman P.J., Otter M., and Elmqvist H. (1998): **Modeling Petri Nets as Local Constraint Equations for Hybrid Systems Using Modelica**. In Proceedings of SCS Summer Simulation Conference, pp. 314-319, Reno, Nevada, July. www.modelica.org/publications/papers/scsc98fp.pdf

NuSMV (2009): A symbolic model checker. nusmv.irst.itc.it.

Otter, M., Årzén, K.-E., Dressler, I. (2005): **StateGraph - A Modelica Library for Hierarchical State Machines**. Proceedings of the 4th International Modelica Conference. TU-Hamburg-Harburg, Germany. www.modelica.org/events/Conference2005/online_proceedings/Session7/Session7b2.pdf

Stateflow (2009): www.mathworks.com/products/stateflow

Appendix

A1 Mapping StateGraph to Modelica

In this section it is sketched how a StateGraph model is mapped to Modelica. This section is based on the implementation technique used in (*Mosterman et. al. 1998*, *Malmheden et. al. 2008*, *Otter et. al. 2005*): Steps, Transitions, and Parallel components are mapped to Boolean equations. These equations are handled as any other Modelica equations, e.g., for the code generation the equations are sorted and therefore the evaluation sequence of a StateGraph model and/or of a hybrid system is automatically determined. Therefore, defining how the StateGraph elements are mapped to Boolean equations defines automatically also the semantics of hybrid systems built by StateGraph and other Modelica models. The mapping algorithm starts with a sketch of the used interfaces between the elements:

A Step component has a vector of connectors called “Step_in” in order to connect from transitions to a step, and a vector of connectors called “Step_out to connect from a Step to Transitions.

A Transition component has a (scalar) connector called “Trans_in” to connect from a Step to a Transition and a (scalar) connector called “Trans_out” to connect from a Transition to a Step.

Only unary connections are allowed, i.e., exactly one connection must be made between one element of a vector of connectors and a scalar connector. The connector classes use pair-wise the same variables, but with different causalities (with exception of “node”), as shown in the next table:

connector Step_out	**connector** Trans_in	
output	**input**	Boolean available
input	**output**	Boolean fire
output	**input**	Boolean checkLoop
		Node node
connector Trans_out	**connector** Step_in	
output	**input**	Boolean fire
output	**input**	Boolean checkLoop
input	**output**	Boolean checkUnary
		Node node

```
record Node
   Boolean suspend;
   Boolean resume;
   function equalityConstraint
      input  Node node1;
      input  Node node2;
      output Real residue[0];
   algorithm
   end equalityConstraint;
end Node;
```

The meaning is the following: When an element of the “Step_out” vector at a Step is connected to the “Trans_in” connector of a Transition, then the signals “available, checkLoop” are computed in the Step and are communicated to the Transition. On the other hand, the signal “fire” is computed in the Transition and communicated to the Step. The meaning of “node” is explained in section A2.

When input/output prefixes are used in a Modelica connector, then block diagram semantics applies for a connector (e.g., only one signal can be connected to an input). Since connectors “Step_out” and “Trans_in” have both input and output variables, only unary connections are possible, as desired. The basic form of “Trans_out” and “Step_in” has either only “output” or “input” variables and therefore unary connections are not guaranteed. For this reason, the dummy variable “checkUnary” is introduced with opposite input/output prefixes. Now, only unary connections are here possible too[2].

A Transition is basically defined by the following equations, depending on the options that have been selected in the parameter menu:

Equations of a Transition component
Immediate transition: `fire = condition and trans_in.available;`
Delayed transition: `enableFire = condition and`

[2] The alternative to use an assert with cardinality is not possible, because the resume connector is conditional and then it cannot be referenced in an assert.

```
                trans_in.available;
when enableFire then
   t_next = time + waitTime;
end when;
fire = enableFire and time >= t_next;
```

Propagation of signals (in both cases):

```
trans_in.fire  = fire;
trans_out.fire = fire;
```

Basically, the equations state that variable fire = true, if (1) the fire condition "condition" is true and (2) if the preceding step is active (trans_in.available =true). For a delayed transition, additionally a time delay is introduced. The "fire" variable is then reported to the preceding and the succeeding steps.

A Step is basically defined by the following equations:

Equations of a Step component

Set active flag:

```
newActive =
  if node.resume then oldActive
  else anyTrue(step_in.fire) or (active
       and not anyTrue(step_out.fire))
       and not node.suspend;
active = pre(newActive);
when node.suspend then
  oldActive = active;
end when;
```

Set available flag:

```
for i in 1:size(step_out,1) loop
  step_out[i].available = if i == 1
    then active and not node.suspend
    else step_out[i-1].available and not
         step_out[i-1].fire and not
         node.suspend;
end for;
```

The function `anyTrue(..)` returns true, if at least one element of the input vector is true. In a Step, the next value of "active" is computed (called: "newActive"). It is assigned in the next event iteration to the actual value, "active", via "active=**pre**(newActive)". The equations state, that the Step becomes active in the next iteration when one of the transitions connected to the step_in connectors fire. The Step remains active if it was active and no transition connected to one of the step_out connectors fire.

If the Step is used inside one or more Parallel components, the state of the nearest enclosing Parallel component is propagated via the record "node". Details are given in section A2. At this stage it is sufficient to know that if node.suspend = true, then an enclosing Parallel component was suspended and if node.resume = true, then an enclosing Parallel component was entered via the resume port. If a Parallel component is suspended, the current value of "active" is saved in "oldActive", and "newActive" is set to false. If a Parallel component is resumed, "newActive" is set to the saved value of "oldActive".

The "active" flag of a Step is reported to the transitions connected to this Step in the following way: If a step has only one outgoing transition:

```
step_out[1].available =
    active and not node.suspend
```

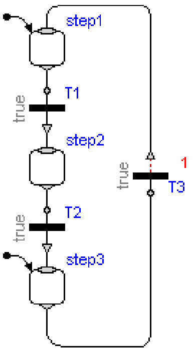
Therefore, the "available" flag propagated to the Transition is set to true, if the step is active and if an enclosing Parallel component is not suspended.

If a Step has several outgoing transitions, two or more might fire at the same time instant. The transition that is connected to the lowest index of the step_out connector vector is defined to have highest priority. For example, if a Step has two outgoing transitions, then the "available" flag of step_out[1] is set as previously. The "available" flag of step_out[2] is only set to true, if the transition that is connected to step_out[1] does not fire and no enclosing Parallel component is suspended.

The equations for a Parallel component are handled similarly to a Step. For space reasons, they are not listed here.

A2 Guaranteeing graph properties and propagation of suspend/resume flags

In the previous section A1 it is sketched how the basic elements are defined by Boolean equations and how only 1:1 connections can be made. Still some properties of a StateGraph are not yet guaranteed. For example, two initial steps might be defined in a simple StateGraph model (see Figure to the right). This gives perfectly legal Modelica code, but the simulation would be wrong. We will now discuss how the basic graph properties are guaranteed and how the suspend/resume information of Parallel components is propagated:



Record "node" in the connectors, see definition in section A1, is an "overdetermined record" due to function "equalityConstraint()", see *(Modelica 2009, section 9.4)*. The idea is the following: The overdetermined record R in a connector has more variables than permitted by a "balanced model". When two connectors c1 and c2 are connected, then the desired connection equations are c1.R = c2.R. If a loop of connected components is present, this might give too many equations (= more equations as unknowns). If this is the case, exactly for one connection set in a loop the equations "0 = R.equalityConstraints(c1.R, c2.R)” have to be used instead of the desired equations “c1.R = c2.R”. For example, a transformation matrix has 9 redundant elements describing 3 independent variables. In this case, the equalityConstraint(...) function has to return the 3 constraint equations between the 9 redundant variables.

In order that a translator can select which connection equations to use, built-in operators are provided to construct an undirected dependency graph of the connectors. For example, if a component has two connectors ca and cb, a definition of the form:

```
Connections.branch(ca.R, cb.R);
```

must be present in the component. This definition states that cb.R is equal to ca.R in this component. One connector must be defined as root of the graph. As a result, a set of undirected graphs is constructed. The translator has to arbitrarily cut a graph at connection sets, so that a spanning tree is constructed. In the “tree”, connection equations of the form c1.R = c2.R are used. For all connectors that have been removed to arrive at a “tree”, the connection equations 0=R.equalityConstraint (c1.R, c2.R) are used.

In the StateGraph library, suspend and resume flags are stored in an overdetermined record “Node“. The Node.equalityConstraints(..) function returns a vector with size zero. Therefore, no equations are generated for connections that have been removed to arrive at a “tree”. When the root of a graph is appropriately selected, then the suspend/resume flags are just propagated to all components in this graph, even if loops are present (since the loops are cut, and no connection equations for node variables are introduced at these cuts).

The operators available in Modelica 3.1 are not sufficient and two additional ones had to be introduced: “**Connections.uniqueRoot**(R, message)” states that “R” is a unique root of the graph. If this operator is used, the corresponding graph must have exactly one such definition. The second argument “message” shall be reported in the error message, if more than one root is defined.

The usage of “uniqueRoot(..)” and of “branch(..)” are sketched in **Figure 15**: Roots are defined at the initial step (root1) and at the entry port of every branch of a Parallel component (root2, root3). Then “branches” are defined along the corresponding state machine structure. If any such connection graph has more then one root, the StateGraph graph is wrong. E.g., if two initial steps would be defined, or if a branch of a Parallel component would branch out into the “outer” loop, the connection graph would have two roots which would trigger an error.

With the new built-in operator “I = **Connections.uniqueRootIndices**(Ra, Rb, message)”, information about the connection structure of a Parallel component can be obtained: Ra is a vector of roots and Rb is a vector of other overdetermined records. The function returns an Integer vector “I”. I[i], i=1:size(Rb,1), defines that there is a path from root I[i] to record Rb[i]: Ra[I[i]] → Rb[i]. It is an error if such a path does not exist. The remaining elements of vector I are the indices of Ra that do not have a path to an element of Rb. Due to the construction, the function returns an error, if there are no paths to all exit ports. So, every branch that ends at an exit port, must start at an entry port of the same Parallel component.

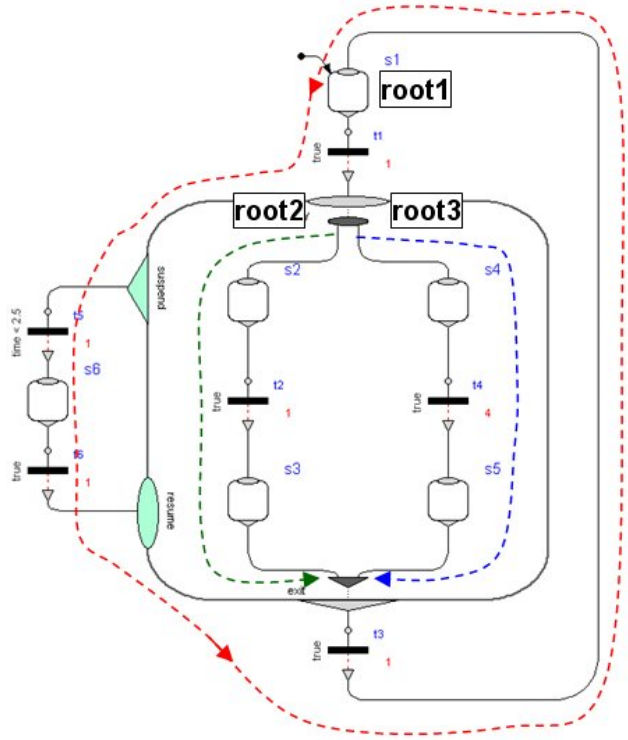


Figure 15: 3 virtual connection graphs to verify StateGraph properties and to propagate resume/suspend flags.

Typical usage of this function:

```
  EntryPort entry  [nEntry];
  ExitPort  exit   [nExit];
  Integer   indices[nEntry];
equation
  Connections.uniqueRoot(entry, "...");
  indices = Connections.uniqueRootIndices
                     (entry, exit, "...");
```

Example: The function returns the following values for the graph in Figure 15:

```
nEntry=2, nExit=2,
indices[1] = 1, indices[2] = 2
```

The meaning is that there is a path from entry[indices[1]] (connected to Step s2) to exit[1] (connected to Step s3), and a path from entry[indices[2]] (connected to Step s4) to exit[2] (connected to Step s5).

A3 Avoiding infinite transition loops

The basic semantics of a StateGraph graph is that at one time instant, during event iteration, all transitions fire, until none of the transitions can fire anymore. In order that no infinite looping can occur, there must be at least one delayed transition in "every loop", since at a delayed transition the looping stops at the current time instant.

In order to verify this property, the Boolean flag "checkLoop" is propagated through the connection structure, see connectors in section A1. At delayed transitions and at steps that do not have an input transition, this flag is initialized. If there is no delayed transition in a loop, an algebraic system of Boolean unknowns occurs. Since this system of equations cannot be solved, an error is triggered. In the connectors, "checkLoop is defined with the new annotation "BooleanLoopMessage = string". If the corresponding variable appears in an algebraic loop with Boolean unknowns, the BooleanLoopMessage is included in the error message, in order to get meaningful error reporting.

A4 Automatic connection to next free index

When connecting a Step with a transition, the dimension of the vector of connectors Step.outPort has to be increased by one, say to dimension N, and then the connection has to be performed from Step.outPort[N] to the scalar transition input port. Performing this manually is very inconvenient and error prone. For this reason, in Modelica 3.1 *(Modelica 2009, section 17.6)* the new annotation "connectorSizing" was introduced, that is used for all vector connections in the StateGraph library.

Example:

```
model Step
   parameter Integer nIn=0 annotation(
            Dialog(ConnectorSizing=true));
   StepIn inPort[nIn];
   ...
end Step;
```

When this model is used and a connection is made to vector "inPort", then the tool increments the dimension nIn by one and performs the connection to this new index. Therefore, performing connections between Steps and Transitions is convenient for a user and only requires dragging a line between the corresponding connectors.

Modelica as a design tool for hardware-in-the-loop simulation

Marco Bonvini[a],Filippo Donida[b], Alberto Leva
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy
{leva,donida}@elet.polimi.it
[a]Graduate student at the Dipartimento di Elettronica e Informazione
[b]PhD student at the Dipartimento di Elettronica e Informazione

August 21, 2009

Abstract

This paper focuses on the automatic generation of microcontroller code for hardware-in-the-loop simulation using Modelica models. In this work a test is presented and commented in which Modelica is used to specify a control system, the inline integration code is obtained from the Modelica model and executed part on a PC, and part on a microcontroller board. The presented application, albeit created basically for educational purposes, covers quite different *scenarii*, therefore evidencing the usefulness of Modelica in the addressed context, and providing as a consequence some future research directions.

The contribution aspect of this work is twofold: on one side the *entire* cross-compilator software chain is built within the same framework; on the other hand, all the involved software tools are open-source (mainly GPL) licensed, making the application extremely modular and extensible. Furthermore this work will be included in the next release of the open source Modelica environment SimForge [3], thus enriching its Modelica back-ends support.

1 Introduction

This paper presents some experimental results relative to the usage of Modelica as modelling and specification language for hardware-in-the-loop simulation aims at control design.

The topic is of high interest both from the methodological and the application-oriented points of view. For the latter, it is evident that the usage of a single tool for the simulation of a control system irrespective of the code being run on the same computer as the model or on the final control architecture is of great help for the designer. Similarly, also having the process simulator running on dedicated hardware can help, particularly if the control strategy is the main object of the simulation studies. Also, having the possibility of deploying process simulation code on dedicated hardware could be of interest for testing control strategies directly on the field, where a PC may not be available.

From the methodological point of view, the envisaged activity apparently requires to obtain optimised simulation code with inline integration, starting from potentially complex object oriented models, in an user-transparent and reliable manner.

The importance having an integrated environment when testing the control performance in a closed loop simulated environment using an integrated environment was already well known in the 1996 [6], when the Control Aided Control System Design tools were used to off-line control design and HIL simulate the ABS systems. More in deep, in the automotive field the HIL Simulation is a quite established technique, both for engine control system test and design [8] and suspension control [2, 7]. Other works focus on the hydraulic servo position [9], machine tools and manufacturing sys-

DOI: 10.3384/ecp09430087

tems [13] and spacecraft [11] control problem respectively. A quite detailed review on the benefits the HIL simulation can introduce for the design process is given in [5]. Moreover, as in [12], where a magnetic levitation device control is presented, the HIL simulation technique can also be used for educational scopes. In accordance with that work, this paper has strong educational intent and focuses on the possibility to use a complete (both software and hardware) GPL environment to control designing through the HIL simulation. The goal is to simulate a closed loop SCARA process. The model of the robot is obtained through the Lagrangian equations and formalized as a system of Differential Algebraic Equations, and the position controller is realised either as a continuous time system and as a digital algorithm[1]. The architecture is composed of a PC and a microcontroller board, communicating through the USB port. Different experiments — not reported here for space reasons — have been conducted where the two CPUs alternatively play the role of the process and of the control, the python language (and more precisely the python-visual library) being used on the PC as a 3d visualisation system.

The first part of the paper, section 2, briefly presents the Modelica model of the complete controlled system.

Section 3 describes the employed microcontroller board (namely the *Arduino*, [1] and its development systems, in connection with the Modelica environment. Convenient references are also provided to the interested reader in full detail.

Then, in section 4 it discussed how the various components of that system can be automatically turned into inline integration algorithms, so as to cover all the possible combination of microcontroller and PC in the simulation of the overall system.

Subsequently, in section 5, the configurations of the performed test is described, evidencing for each configuration which particular aspect of the research claims stated in the introduction is being investigated: for example, a test in which the microcontroller runs the simulator and the PC the control system is useful to analyse the latter together with the communications, while a test where the PC simulates the robot and the micro the control helps estimating the feasibility of a given algorithm on the target architecture, and so on. The test results are finally presented, ending with some configuration on the subsequent research activity.

2 The SCARA: model and control in Modelica

The SCARA robot we considered in this work in a two rotational degrees of freedom mechanical planar chain, actuated via two ideal electrical servos. The model of the robot is now introduced, starting from some considerations about the links masses.

Making the hypothesis of the mass of each link concentrated in a single point, the links inertias could be expressed as:

$$I_i = \frac{1}{12} m_i {a_i}^2 \tag{1}$$

with $_i = 1,2$ being the subscript for identifying the link, m the link mass, a the position of the center of mass with respect to the beginning of the link measured along the link itself.

From the Lagrangian equations, with some algebra, it is trivial to express the system of differential equations of the robot in the form $\tau = f\left(\theta, \dot{\theta}, \ddot{\theta}\right)$:

$$\tau = B(\theta)\theta + C\left(\theta, \dot{\theta}\right)\dot{\theta} + g(\theta) \tag{2}$$

where the inertia matrix $B(\theta)$ is:

$$B(\theta) = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} \tag{3}$$

with the $B_{i,j}$ being:

$$\begin{aligned} B_{1,1} &= m_1{l_1}^2 + I_1 + m_2{a_1}^2 + m_2{l_2}^2 + \\ &\quad + 2m_2a_1l_2\cos(\theta_2) + I_2 \\ B_{1,2} &= m_2{l_2}^2 + m_2a_1L_2\cos(\theta_2) + I_2 \\ B_{2,1} &= m_2{l_2}^2 + m_2a_1l_2\cos(\theta_2) + I_2 \\ B_{2,2} &= m_2{l_2}^2 + I_2 \end{aligned} \tag{4}$$

the matrix corresponding to the centripetal, Coriolis, and viscous friction forces, $C\left(\theta, \dot{\theta}\right)$, is:

$$C\left(\theta, \dot{\theta}\right) = \begin{bmatrix} C_{1,1} & C_{1,2}C_{2,1} & C_{2,2} \end{bmatrix} \tag{5}$$

[1] Only the discrete time version of the controller is treated in this paper.

being the $C_{i,j}$ elements:

$$\begin{aligned} C_{1,1} &= -2m_2 a_1 l_2 \sin(\theta_2)\dot{\theta_2} \\ C_{1,2} &= -m_2 a_1 l_2 \sin\left(\dot{\theta_2}\right)\dot{\theta_2} \\ C_{2,1} &= m_2 a_1 l_2 \sin(\theta_2)\dot{\theta_1} \\ C_{2,2} &= 0 \end{aligned} \tag{6}$$

while the matrix for the effects of the gravitational filed, $g(\theta)$ is:

$$g(\theta) = 9.81 \begin{bmatrix} g_{1,1} \\ g_{2,1} \end{bmatrix} \tag{7}$$

with:

$$\begin{aligned} g_{1,1} &= (m_1 l_2 + m_2 a_1)\cos(\theta_1) + m_2 l_2 \cos(\theta_1 + \theta_2) \\ g_{2,1} &= m_2 l_2 \cos(\theta_1 + \theta_2) \end{aligned} \tag{8}$$

Considering also that the plane containing the robot joint space is orthogonal to the gravity direction, we can simplify the equation 2, thus obtaining:

$$\tau = B(\theta)\theta + C\left(\theta, \dot{\theta}\right)\dot{\theta} \tag{9}$$

Finally the “Lagrangian” Modelica model of the SCARA can be straightforward obtained from the previous DAE system implementing the equation 9 and specifying the torques (τ) as inputs while the angular positions (θ) as outputs.

A *more* object-oriented and `Modelica.Mechanics` based model for representing the robot had also been developed and mainly used for testing the correctness of the “Lagrangian” model. Unfortunately, at the time this experiment has been conducted, it has not been possible to use the OO Modelica model of the SCARA with the OpenModelica compiler, because of the current limitation in supporting the whole Modelica Standard Library.

The control model is a vectorial discrete-time Proportional Derivative regulator implemented as a Modelica algorithm with saturation on the control signal.

Moreover, to complete the control schema architecture, it has been necessary to implement a model for computing the robot inverse kinematic. The code is reported in the following.

```
model InverseKinematic
  import MBI = Modelica.Blocks.Interfaces;
  parameter Real L[2];//Lenght of links 1 and 2
  MBI.RealInput xy[2];//x & y positions
  MBI.RealOutput a[2];//joint space angles
algorithm
  a:=InverseKinematic(xy, L1, L2);
end InverseKinematic;

function InverseKinematic
  input Real xy[2];
  input Real L[2];
  output Real a[2];
protected
  Real x,y,c2,s2,c1,s1,a1,a2;
algorithm
   x  := xy[1];
   y  := xy[2];
   c2 := (x*x+y*y-L[1]*L[1]-L[2]*L[2])/(2*L[1]*L[2]);
   s2 := sqrt(1-c2*c2);
   a2 := atan2(s2,c2);
   c1 := ((L[1]+L[2]*c2)*x+L[2]*s2*y)/(x*x+y*y);
   s1 := ((L[1]+L[2]*c2)*y-L[2]*s2*x)/(x*x+y*y);
   a1 := atan2(s1,c1);
   a  := {a1,a2};
end InverseKinematic;
```

3 The Arduino microcontroller

Arduino is an open-source microcontroller electronic platform, featuring 14 digital pins that can be used both as input or output and 9 analog input pins. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). In figure 1 the ArduinoDuemilanove microcontroller is shown.

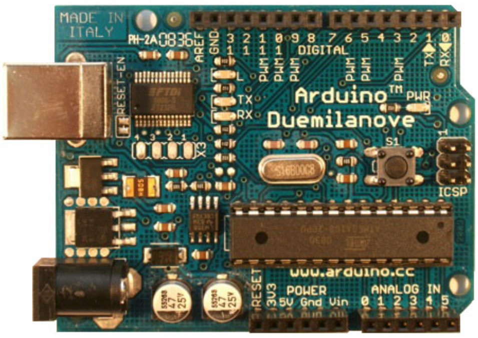


Figure 1: The Aurduino Duemilanove microcontroller.

Arduino executables can be a stand-alone processes or can communicate with other processes, i.e. running processes on a PC. Usually the structure of an Arduino program is quite standardized and is organized in four main parts:

- the *Preamble* section: collects all the includes and definitions statements,
- the *Variables and functions definitions* section. According to the data types of the Arduino language, all the variables used within the program must be declared here and, if necessary, instantiated. In addition the user-defined functions headers must be listed in this section.
- the *Setup* function setup(): this is called when your program starts, initializing the user-defined variables, pin modes, etc. The setup function runs only once, after each powerup or reset of the Arduino board.
- the *Loop* function loop(): this function contains the code to be sequentially executed, iteratively until resetting the Arduino.

With the intention to better explain the Arduino coding procedure, in the following a simple program for that board has been reported.

```
int buttonPin = 3;

// setup initializes serial and the button pin
void setup(){
  beginSerial(9600);
  pinMode(buttonPin, INPUT);
}

// loop checks the button pin each time,
// and will send serial if it is pressed
void loop(){
  if (digitalRead(buttonPin) == HIGH)
    serialWrite('H');
  else
    serialWrite('L');

  delay(1000);
}
```

As one could easily evince from the code lines reported above, the program is very simple, it just makes the Arduino sending a ‘H’ or ‘L’ character according to the state of a button (pressed or released).

4 Code generation

In this section a cross-compiler software to automatically translate the Modelica code into Arduino microcontroller code is presented. The code generation procedure, reported in figure 2 involves four different software layers: the OpenModelica compiler (in fucsia), the Java environment (light green), the Arduino environment (yellow) and the Maxima [10] symbolic manipulator (grey). In this section a description of the whole procedure is reported, for more details please refer to the SimForge documentation.

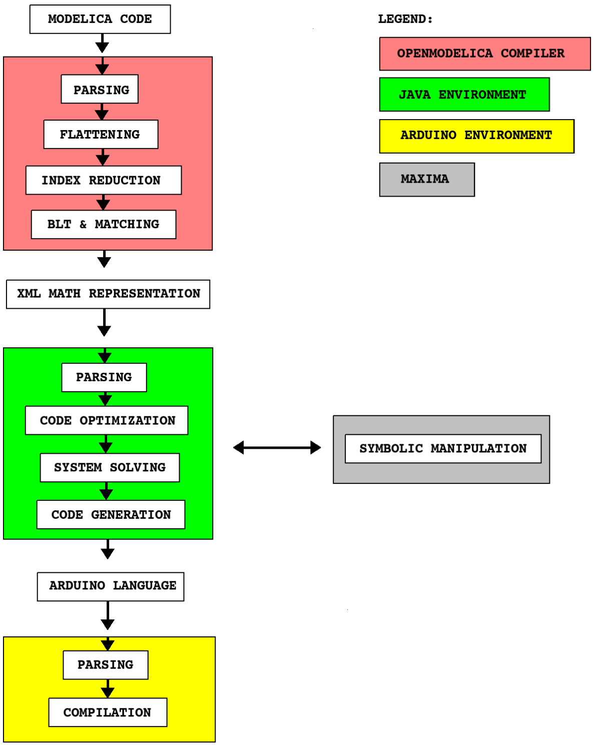


Figure 2: The steps for the Aurduino code generation from Modelica model.

The first step of the procedure is performed by the OpenModelica compiler that reads the Modelica file of the model, parses the Modelica source and creates the corresponding Abstract Syntax Tree (AST). After doing that, the tree is traversed for flattening the model, the index reduction algorithm is performed (if necessary) and the Tarjan algorithm is operated, thus obtaining the BLT structure and the variables-equations matching. At this stage the model is represented as an index-one system of differential algebraic equations and can be dumped out from the OpenModelica compiler through the XML dump module. This module of the compiler has been implemented with the intention to provide a standardized way for representing such a system of differential algebraic equations. To make the XML source syntactically constrained, an *ad hoc* XML Schema [4] has been created, requiring that it contains the variables lists (unknown, known and external), the equations lists (equations, algorithms, zero crossings, simplified equations, . . .) and, optionally, information for solving the system, i.e. the BLT structure and/or the matching algorithm output.

To make an example, if considering the following Modelica model:

```
  model test_equation
    Real x(start = 1);
    parameter Real a = -1;
  equation
    der(x) = a*x;
  end test_equation;
```

a XML-equivalent representation could be[2]:

```
<?xml version="1.0" encoding="UTF-8"?>
<dae xmlns:p1="http://www.w3.org/1998/Math/MathML"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation=...
           ..."http://home.dei.polimi.it/donida...
           .../Projects/AutoEdit/Images/DAE.xsd">

<variables dimension="2">

 <orderedVariables dimension="1">
  <variablesList>
   <variable id="1" name="x" variability=...
   ..."continuousState" direction="none" ...
   ...type="Real" index="-1" origName="x" ...
   ...fixed="true" flow="NonConnector" ...
   ...stream="NonStreamConnector">
    <classesNames>
    <element> test_equation </element>
    </classesNames>
    <attributesValues>
      <initialValue string="1.0"> </initialValue>
    </attributesValues>
    </variable>
  </variablesList>
 </orderedVariables>

 <knownVariables dimension="1">
  <variablesList>
   <variable id="1" name="a" variability=...
   ..."parameter" direction="none" ...
   ...type="Real" index="-1" origName="a" ...
   ...fixed="true" flow="NonConnector" ...
   ...stream="NonStreamConnector">
    <bindValueExpression>
      <bindExpression string="-1"> </bindExpression>
    </bindValueExpression>
    <classesNames>
    <element> test_equation </element>
    </classesNames>
    </variable>
  </variablesList>
 </knownVariables>
</variables>

<equations dimension="1">
    <residualEquation id="1">der(x) - a * x = 0
    </residualEquation>
</equations>

<additionalInfo>
    <solvingInfo>
    <matchingAlgorithm>
      <solvedIn variableId="1" equationId="1" />
    </matchingAlgorithm>
    <bltRepresentation>
      <bltBlock id="1">
        <involvedEquation equationId="1" />
      </bltBlock>
    </bltRepresentation>
    </solvingInfo>
</additionalInfo>

</dae>
```

The XML representation reported in 4 is quite intuitive: at the top the header specifies where to find all the related schemes, then the `dae` tag contains all the variables, the equations and the additional information. The variables list is usually[3] split into two separated list: the ordered variables and the known variables lists respectively, the former containing all the state (also dummy states) and algebraic variables, while the latter listing all the parameters and constants. The second section is the equations section, and then, at the bottom of the file, the additional information are located, showing the matching algorithm output as well as the BLT representation of the system.

This representation is extremely useful since offers a standard machine-readable exchange format for the DAE system.

In next macro-step of the cross-compilation procedure, the XML file is processed through an *ad hoc* implemented Java routine. This software layer has two main goals:

- implement some xml functionalities to handle with the XML representation of the DAE system and

- provide some basic symbolic manipulation capabilities through the Java-Maxima interface.

[2] Given a Modelica model, the XML representation is not unique, since some parameters can optionally be specified when dumping the model (i.e. if add the solving information, if dump the equations using the residual form, etc.), thus changing the content.

[3] In some cases also the external variables and/or the external classes lists could be present. We invite the interested reader to refer to [4].

The Java module execution chain is basically made up of four sub-steps:

- First the DAE XML file is parsed and all the classes corresponding to the mathematical entities (variables, equations, matching output, BLT blocks, states selection,...) are instantiated within the Java environment,
- then the system of equations is re-formulated for the code generation, trying to use the matching algorithm output to solve the equations system. It can happen that the greater block of the BLT matrix is not a scalar. If this happen there are two possibilities: the Maxima interface can handle with the problem, thus solving the system for the variables specified from the matching algorithm output, otherwise (when Maxima can find more than a solution or none at all) the system must be included within the real code, together with a Newton procedure to iteratively found the solution, at each time step. As the reader can easily imagine, the latter scenario is not really desirable, since there is no *a priori* guarantee for the convergence of the Newton algorithm within the given time step.
- Thirdly the inlining procedure is applied to all the state equations, according to the specific integration algorithm (forward Euler, backward Euler, trapezium) and the obtained discrete-time system is solved with respect to the new discrete states variables.
- Finally the file required to compile the model of the controller within the Arduino environment is generated.

5 The SCARA: HIL

In this section the HIL experiment is described. As already introduced in section 1, the Arduino microcontroller runs the control algorithm, while the PC creates the position set points, simulates the robot model and visualizes the SCARA robot through the Python script. More precisely, an algorithm on the PC generates the position set points that are sent to the Arduino through the USB port. The Arduino microcontroller, translates the position set points into angle set points, using the inverse kinematic block and then, according to the PD gains, the torques to apply to each joint. The control signals are finally sent back to the PC, that simulates and visualizes the SCARA movement.

In this work, the objective has been to make the robot drawing a star. To do that, the position set points generation function samples 12 points on a couple of concentric circles (6 equidistant points over the inner and the other 6 equidistant points over the outer circle respectively), thus obtaining a star path. After doing that, the set point trajectory is thicken specifying the number of sub-samples between a point and the next on the star path.

In figure 3 the Scilab visualisation window is reported, showing how the controlled robot follows the target trajectory.

The experiment refers to a simulated SCARA having the first link with a length of 1 [m] and a weight of 2 [Kg], while the second 0.6 [m] and 1 [Kg]. The discretization technique has been used is the Forward Euler method, with a fixed step of 0.1 [s]. A time of 2[s] has been chosen for moving from a point to the next of the set point trajectory.

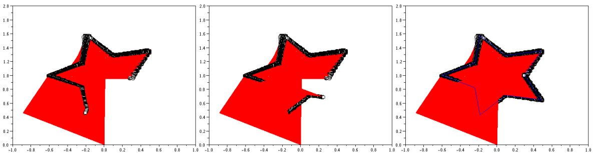


Figure 3: Graphical representation of the star drawing process of the closed loop system.

The Process Values (blue) and Set Points (red) of both the angles at top and Control Signals (green) at bottom for the star experiment are reported in figure 4 (the first angle is reported on the right column while the second angle on the left column).

Even if the two signals (PV and SP) seem to be quite overlapped, they are not identical, as it is possible to view from the figure 5, where a particular is shown.

In addition to that, a Python script has been implemented to 3d visualize the robot closed loop behavior. Figure 6 refers to a test case in which the simulation is done on the microcontroller, while the PC runs the control algorithm and the visualising machinery. In detail, the figure shows the animation window obtained with the python-visual

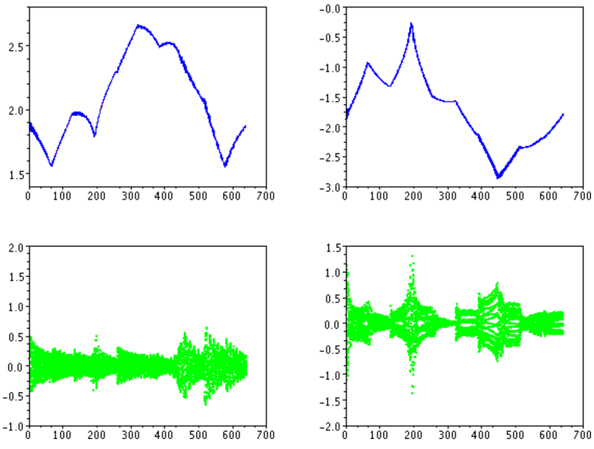


Figure 4: Process Values (blue) and Set Points (red) of both the angles at top and Control Signals (green) at bottom (the first angle on the right column while the second angle on the left column) for the star experiment.

library for the online visualisation of the SCARA simulation.

All the Modelica, C (for the microcontroller), and python code will be made available as free software to the scientific community.

6 Conclusions

A closed-loop discrete-time SCARA process has been simulated with the HIL technique, with the Arduino microcontroller running the PD control, while the PC generating the trajectory, simulating the model of the robot and 3d visualising its movement.

Even if this work has mainly educational intentions, clearly shows the possibility of using an integrated GPL-licensed framework for automatically produce the HIL code from the Modelica language, for control aims. The openness of the presented framework is of great importance, specially for maintaining the modularity of the project, thus ensuring the scalability and the extensibility.

Future works will probably focus in two main directions: on one hand the possibility of specifying more complex control strategies will be inspected, on the other hand the eventuality of using a couple of Arduino microcontrollers for HIL simulation will be envisaged.

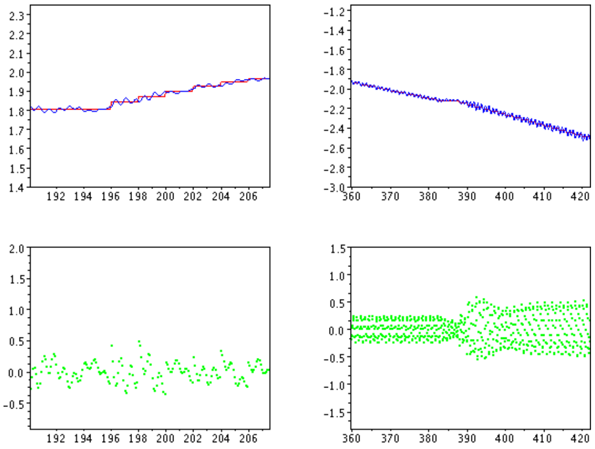


Figure 5: A particular of picture 4.

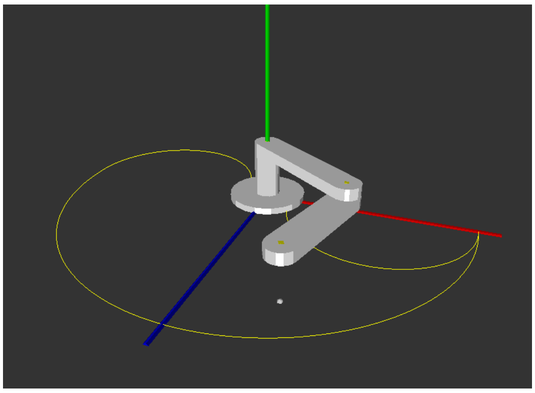


Figure 6: An example of python-based online animation of the simulation results obtained by the microcontroller, and fed to the visualising PC through the USB interface.

References

[1] The arduino home page. `http://www.arduino.cc`.

[2] S. B. Choi, Y. T. Choi, and D. W. Park. A Sliding Mode Control of a Full-Car Electrorheological Suspension System Via Hardware in-the-Loop Simulation. *Journal of Dnamic Systems, Measurements and Control*, 122:114–121, March 2000.

[3] Politecnico di Milano sede di Cremona. Simforge: a graphical modelica environ-

ment. https://trac.wd.dei.polimi.it/simforge.

[4] Filippo Donida. Xml schema for dae representation with the openmodelica compiler, 09 2009. http://home.dei.polimi.it/donida/Projects/AutoEdit/Images/DAE.xsd.

[5] R. Ernst. Codesign of embedded systems: status and trends. *Design & Test of Computers, IEEE*, 15(2):45–54, Apr-Jun 1998.

[6] H. Hanselmann. Hardware-in-the-Loop Simulation Testing and its Integration into a CACSD Toolset. In *IEEE International Symposium on Computer Aided Control System Design*, pages 152–156. IEEE, September 1996.

[7] K. S. Hong, H. C. Sohn, and J. K. hedrick. Modified Skyhook Control of Semi-Active Suspensions: A New Model, Gain Scheduling, and Hardware-in-the-loop Tuning. *Journal of Dnamic Systems, Measurements and Control*, 124:158–167, March 2002.

[8] R. Isermann, J. Schaffnit, and S. Sinsel. Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, 7:643–653, August 1999.

[9] M. Linjama, T. Virvalo, J. Gustafsson, J. Lintula, V. Aaltonen, and M. Kivikoski. Hardware-in-the-loop environment for servo system controller design, tuning and testing. *Microprocessors and microsystems*, 24:13–21, December 2000.

[10] MIT. Maxima, a computer algebra system, 1960. http://maxima.sourceforge.net/.

[11] A. Ptak and K. Foundy. Real-time spacecraft simulation and hardware-in-the-loop testing. In *Real-Time Technology and Applications Symposium, 1998. Proceedings. Fourth IEEE*, pages 230–236, Jun 1998.

[12] P. S. Shiakolas and D. Piyabongkarn. Development of a Real-Time Digital Control System With a Hardware-in-the-Loop Magnetic Levitation Device for Reinforcement of Controls Education. *IEEE Transaction on Control Education*, 46(1):79–87, February 2003.

[13] G. Stoeppler, T. Menzel, and S. Douglas. Hardware-in-the-loop simulation of machine tools and manufacturing systems. *Computing & Control Engineering Journal*, 16(1):10–15, Feb.-March 2005.

Real-Time Simulation of Modelica-based Models

Torsten Blochwitz Thomas Beutlich
ITI GmbH
Webergasse 1, 01067 Dresden, Germany
{blochwitz,beutlich}@iti.de

Abstract

This paper shows the various steps a simulation tool has to perform to create a real-time-capable model from a Modelica model. Reduction techniques are often necessary for complex models to meet the real-time requirements. For non-linear models with discontinuities no automatic methods of model reduction are known. The analysis methods supporting developers in identifying critical model parts are explained by means of an illustrating example model.

Keywords: real-time simulation; hardware-in-the-loop; model reduction

1 Introduction

The method of physical modeling is more and more establishing itself in the engineering departments of OEMs and component suppliers. The engineers do no longer formulate the model equations by hand but compile their models using sophisticated model libraries. Thus, detailed models are built up in comparatively short time. These models simulate the dynamic behavior of the system in detail. E.g., the vibrational behavior of drive trains or hydraulic systems is explored.

During software development of Electronic Control Units (ECU) offline (non real-time), system simulations are performed using Model-in-the-Loop (MIL) techniques. In this development stage the detailed simulation models from the system design can still be used. During the test phase of the ECU, Hardware-in-the-loop (HIL) techniques are used requiring the simulation models to run in real-time.

Costs and resources can be saved if the plant models built up during system design can be reused for real-time simulation [1, 2]. The SimulationX® [3] high-level system simulation tool supports the engineer in reusing and reducing the simulation models. The prospects and limitations of such model reuse and reduction are shown.

2 Real-Time Requirements

In the general case physical models can be represented by a DAE (differential algebraic equation) system of the form

$$0 = f(x,\dot{x},z,u,p,t) \tag{1}$$

$$y = g(x,\dot{x},z,u,p,t) \tag{2}$$

with

x	...	Continuous states variables
z	...	Discrete states variables
u	...	Inputs
y	...	Outputs
p	...	Parameters
t	...	Time.

Appropriate implicit DAE solvers can directly solve the DAE system in offline simulation.

The explicit ODE (ordinary differential equation) form

$$\dot{x} = f(x,z,u,p,t) \tag{3}$$

is numerically easier to solve than the DAE form.

Real-time capable models need to be solved within a predictable execution time per time step. The model execution time has to be less than the step size.

Implicit solvers needed for DAE calculations work by iterative methods. The execution time depends on the number of executed iterations. A common workaround is to limit the number of iterations. However, this limitation might lead to numerical inaccuracies. Additionally the Jacobian matrix needs to be updated from time to time. Hence the execution time

DOI: 10.3384/ecp09430119

of iterative methods is not predictable making them inapplicable for HIL simulation.

Explicit solvers do meet this requirement but can only be used for solving ODE systems. Therefore the physical model needs to be translated to the explicit ODE (3) form.

Efficient offline solvers are characterized by step size adaptations. E.g. the step size is decreased for a robust calculation of high-frequency oscillations. However, real-time capable solvers require a constant step size.

Variable step size solvers are also used to precisely detect the discontinuities and events. This cannot be guaranteed under real-time conditions; hence a robust formulation of discontinuities and events is required to prevent improper model behavior after an event.

The maximal model step size for stable calculation of a differential equation using a given solver integration algorithm depends on the natural frequencies, time constants and non-linearities. In other words if real-time is required the dynamics and non-linearities need to be limited, too.

Finally the model complexity is limited by the computing power of the target hardware. The model execution time must not exceed the available calculation time.

Summing up, the real-time requirements are

- Explicit ODE form of the system,
- Limited dynamics and non-linearities,
- Robust treatment of discontinuities,
- Limited model complexity.

3 Model Generation for HIL Simulation

The steps shown in Fig. 1 are necessary to get from a physical model to a HIL model.

In the first step the user defines the interfaces of the HIL model, i.e. the model inputs, outputs and parameters. The SimulationX Modelica compiler translates the model to explicit ODE form. The translated model is then written as C code to file.

The SimulationX Code Export Wizard guides the user step by step through the workflow. The model independent code parts (i.e. the solver code and the target specific code) are generated for selected real-time targets. For other HIL environments based on Simulink® and the MATLAB® Real-Time Workshop® these code parts are generated afterwards during the Real-Time Workshop code generation.

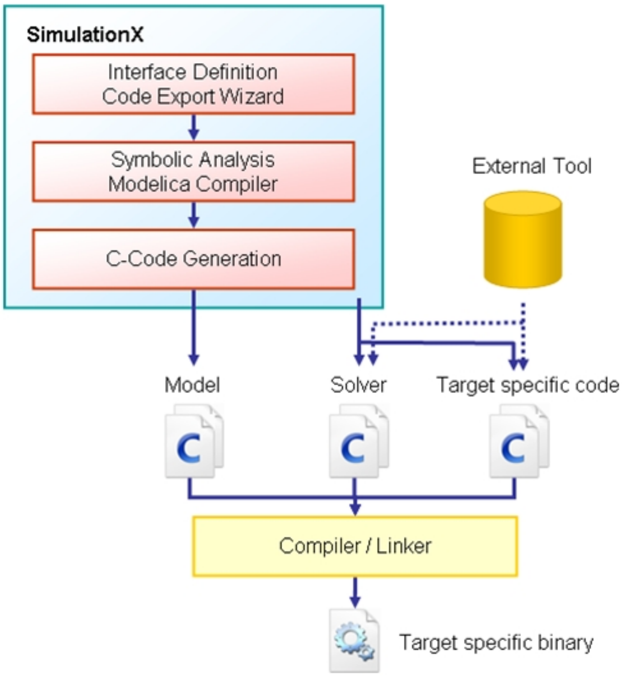


Figure 1: Workflow of HIL model generation

4 SimulationX Guidance

During the HIL model generation the simulation tool can influence the compliance with the real-time requirements. If such supporting measures are not sufficient model reduction techniques need to be taken into account. As before SimulationX supports the user in model reduction, too.

4.1 Symbolic Preprocessing

Using a modeling description language like Modelica requires symbolic preprocessing of the algorithms/equations of the entire dynamic system resulting in a simplified system of equations prepared for numerical integration.

The SimulationX Modelica compiler can either create the DAE or explicit ODE form of the system of equations. The translated model can be calculated within the simulation tool or be exported as C code (explicit ODE form only).

At the time of the symbolic preprocessing all model equations are known and can be optimized (even for offline simulation). The general optimization techniques involve

- Simplification of complex expressions,
- Constants are only assigned once,
- Elimination of dead branches of conditional alternatives,
- One-time calculation of repeatedly used expressions,
- Expansion of vectors and matrices,
- Loop unrolling.

For real-time simulation the optimization can even be continued. Since the user defines the necessary inputs, outputs and parameters of the model all other model parts that do not contribute to the calculation of the outputs can be cancelled. E.g. for the mechanical spring-damper in Fig. 2 the change of potential energy and the power loss are dispensable results as displayed in Fig. 3.

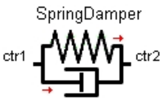
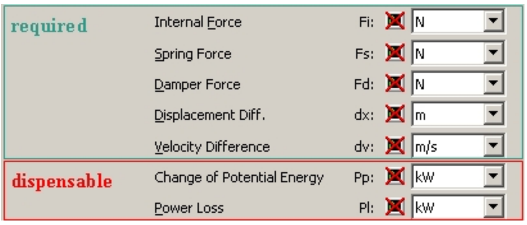


Figure 2: SimulationX Spring-Damper library element



required	Internal Force	Fi:	☒	N
	Spring Force	Fs:	☒	N
	Damper Force	Fd:	☒	N
	Displacement Diff.	dx:	☒	m
	Velocity Difference	dv:	☒	m/s
dispensable	Change of Potential Energy	Pp:	☒	kW
	Power Loss	Pl:	☒	kW

Figure 3: Required and dispensable results of the mechanical spring-damper

If the physical model contains implicit relationships (algebraic loops) the symbolic preprocessing tries to solve them when translating the model to explicit ODE form. Non-solvable relationships are transformed to local blocks of equations that additionally need to be solved along with the calculation of the RHS (right hand side) of the explicit ODE. Linear and non-linear systems are detected and separately solved. The non-linear implicit systems are solved by iterative methods that actually are inconsistent with the real-time requirements. However, a fast calculation is guaranteed

- by a small dimension (2...10) of the non-linear implicit blocks,
- as a symbolic Jacobian matrix is provided that results in superlinear convergence,
- as well-chosen start values for the iteration are given. (Assuming a low rate of change of the unknown variables the results from the previous time step can be used as start values for the current iteration.)

If performance problems are still an issue the user is informed of the blocks of implicit equations and the unknown variables. This information finally allows specific model changes.

Additional steps (such as index reduction and minimum dynamic state selection) might be necessary for higher index DAE systems.

4.2 Solver

In complex systems the execution time of the model mainly depends on the calculation of the model. By the introduction of a modified stability region it was shown that the well-established Euler Forward solver is the most efficient solver for complex models and most suitable for HIL applications [4]. Additionally the Euler Forward solver has the lowest numerical error on discontinuities.

Complex numerical solvers require multiple calculations of the model per time step. The stability region increases with multiple calculations of the model, i.e. the model step size can be increased as well. However, the increased model step size does not compensate the increased calculation time due multiple calculations of the model. A stabilized fixed step size solver was developed that performs better for special model classes. The distinction between the model sample rate and the integration step size allows oversampling leading to excellent results as proven by experience from numerous applications.

4.3 C Code Generation

The result of the code generation is target independent C code with defined interfaces [5].

If there are loops within auxiliary functions (e.g. characteristic curves with non-equidistant nodes, delay buffers with variable dead time) efficient search algorithms are applied. It is also ensured that no dynamic memory is allocated or freed during the model runtime.

Own implementations are provided for suboptimally implemented functions in the runtime libraries. The SimulationX Code Export Wizard can interface the following HIL environments

- Targets based on Simulink and the Real-Time Workshop are addressed by Simulink C coded S-Functions [6].
- DS1006 Processor Board [7] from dSPACE,
- SCALE-RT [8] from CosateQ,
- NI VeriStand [9] from National Instruments.

The architecture of the target specific code is different for each of the targets. In case of the dSPACE target the complete application code consisting of the model code, the solver code and the simulation engine code needs to be generated. The I/O function calls for selected dSPACE I/O boards are realized by external C function calls used within custom library elements as demonstrated in Fig. 4.

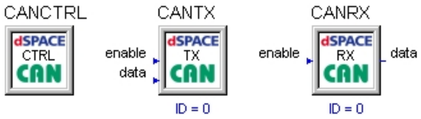


Figure 4: Custom CAN library elements for the DS1006 Processor Board target

SCALE-RT provides a simulation framework that is addressed by the SimulationX model. Using distinct custom I/O library elements the handling of the I/O function calls is similar to the dSPACE target.

The NI VeriStand target provides an extended model simulation framework. Aside from the model code only the solver code with a matching interface has to be generated. No custom I/O library elements need to be modeled as all I/O hardware access is handled outside the physical model. The NI VeriStand System Explorer accomplishes the mapping between the physical I/O channels and the model inputs and outputs after the code compilation.

5 Model Reduction

Whereas automatic model reduction techniques are neither available nor known a formal model reduction approach is described in [10]. The reduction steps closely depend on the user know-how. The following features and analysis methods of SimulationX support the user by the demanding model reduction task.

5.1 Switchable Complexity

Most complex library elements feature switchable complexity. E.g. the gear drive in Fig. 5 has to be elastically modeled for Noise - Vibration - Harshness (NVH) analyses.

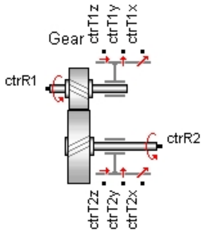
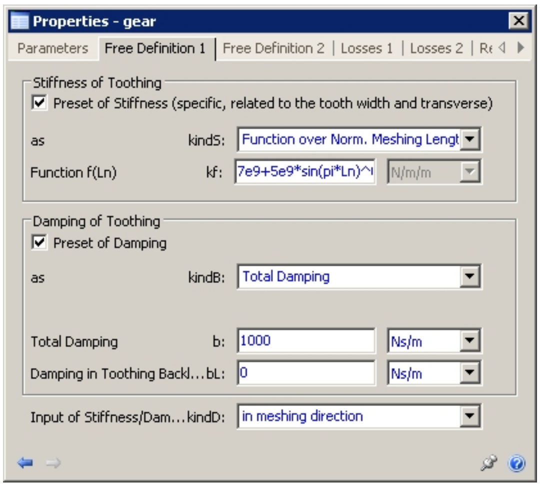


Figure 5: SimulationX Gear library element

Fig. 6 shows the complex parameterization of the stiffness and the damping of the toothing.



Stiffness of Toothing
☑ Preset of Stiffness (specific, related to the tooth width and transverse)
as kindS: Function over Norm. Meshing Lengt
Function f(Ln) kf: 7e9+5e9*sin(pi*Ln)^ N/m/m

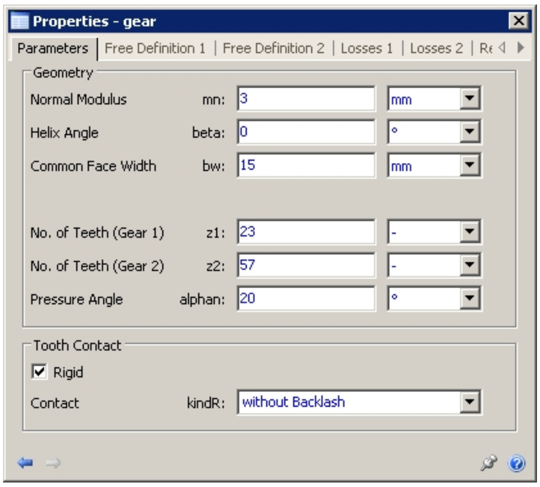
Damping of Toothing
☑ Preset of Damping
as kindB: Total Damping
Total Damping b: 1000 Ns/m
Damping in Toothing Backl...bL: 0 Ns/m

Input of Stiffness/Dam...kindD: in meshing direction

Figure 6: Elastic gear modeling with non-linear stiffness, damping and backlash

On the other hand the gear toothing is considered as rigid for HIL applications. Due to its complexity and high dynamics the vibration behavior is no longer part of the real-time simulation. If the gear parameter *rigid* is selected the gear works as ideal rigid transmission with reduced dynamics, dimension and complexity as displayed in Fig. 7.

All deactivated parameters are disabled and there values are saved.


Properties - gear
Parameters | Free Definition 1 | Free Definition 2 | Losses 1 | Losses 2 | Re

Geometry

Normal Modulus	mn:	3	mm
Helix Angle	beta:	0	°
Common Face Width	bw:	15	mm
No. of Teeth (Gear 1)	z1:	23	-
No. of Teeth (Gear 2)	z2:	57	-
Pressure Angle	alphan:	20	°

Tooth Contact

☑ Rigid

Contact kindR: without Backlash

Figure 7: Rigid gear modeling without backlash

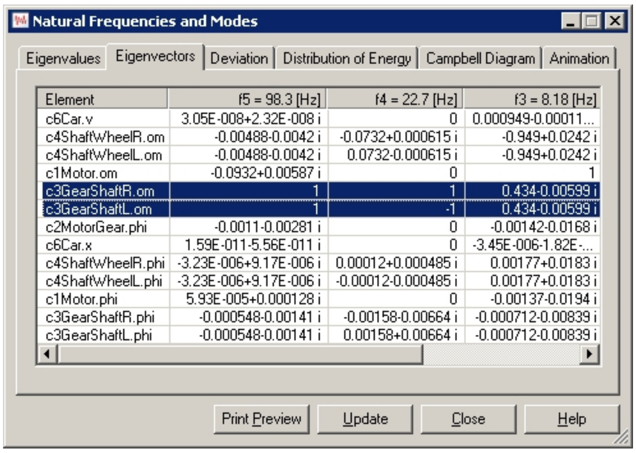
5.2 Model Analysis

Certain components need to be alternatively modeled in order to reduce the eigenvalue spectrum. E.g. mechanical elastic components need to be regarded as rigid components or hydraulic throttles must be neglected. It always is a non-trivial task to identify those model components.

A simple automotive drive train (Fig. 8) is used as illustrating example.

5.2.1 Analysis of Natural Frequencies and Mode Shapes

The analysis of the natural frequencies and mode shapes calculates the eigenvalues and eigenvectors at the current working point. The eigenvectors provide information on the influence of the state variables on the respective mode shape. Fig. 9 displays the eigenvectors corresponding to the highest three natural frequencies of a drive train. Thus the critical state variables can easily be identified.

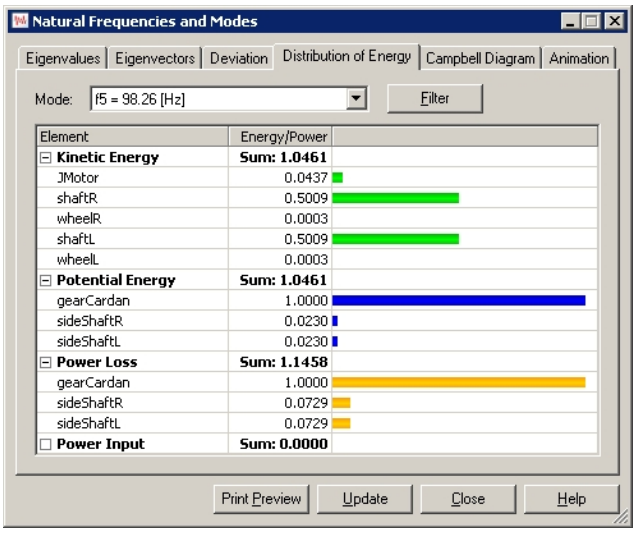

Natural Frequencies and Modes
Eigenvalues | Eigenvectors | Deviation | Distribution of Energy | Campbell Diagram | Animation
Print Preview
Update
Close
Help

Element	f5 = 98.3 [Hz]	f4 = 22.7 [Hz]	f3 = 8.18 [Hz]
c6Car.v	3.05E-008+2.32E-008 i	0	0.000949-0.00011...
c4ShaftWheelR.om	-0.00488-0.0042 i	-0.0732+0.000615 i	-0.949+0.0242 i
c4ShaftWheelL.om	-0.00488-0.0042 i	0.0732-0.000615 i	-0.949+0.0242 i
c1Motor.om	-0.0932+0.00587 i	0	1
c3GearShaftR.om	1	1	0.434-0.00599 i
c3GearShaftL.om	1	-1	0.434-0.00599 i
c2MotorGear.phi	-0.0011-0.00281 i	0	-0.00142-0.0168 i
c6Car.x	1.59E-011-5.56E-011 i	0	-3.45E-006-1.82E-...
c4ShaftWheelR.phi	-3.23E-006+9.17E-006 i	0.00012+0.000485 i	0.00177+0.0183 i
c4ShaftWheelL.phi	-3.23E-006+9.17E-006 i	-0.00012-0.000485 i	0.00177+0.0183 i
c1Motor.phi	5.93E-005+0.000128 i	0	-0.00137-0.0194 i
c3GearShaftR.phi	-0.000548-0.00141 i	-0.00158-0.00664 i	-0.000712-0.00839 i
c3GearShaftL.phi	-0.000548-0.00141 i	0.00158+0.00664 i	-0.000712-0.00839 i

Figure 9: Eigenvectors of a drive train

5.2.2 Distribution of Energy

Especially for mechanical systems the energy analysis as shown in Fig. 10 graphically displays the component effects on the respective mode shape.


Natural Frequencies and Modes
Eigenvalues | Eigenvectors | Deviation | Distribution of Energy | Campbell Diagram | Animation
Filter
Print Preview
Update
Close
Help

Mode: f5 = 98.26 [Hz]

Element	Energy/Power
⊟ **Kinetic Energy**	**Sum: 1.0461**
JMotor	0.0437
shaftR	0.5009
wheelR	0.0003
shaftL	0.5009
wheelL	0.0003
⊟ **Potential Energy**	**Sum: 1.0461**
gearCardan	1.0000
sideShaftR	0.0230
sideShaftL	0.0230
⊟ **Power Loss**	**Sum: 1.1458**
gearCardan	1.0000
sideShaftR	0.0729
sideShaftL	0.0729
□ **Power Input**	**Sum: 0.0000**

Figure 10: Energy distribution for a selected mode

Thus the components with the highest influence on the critical eigenvalues can be identified.

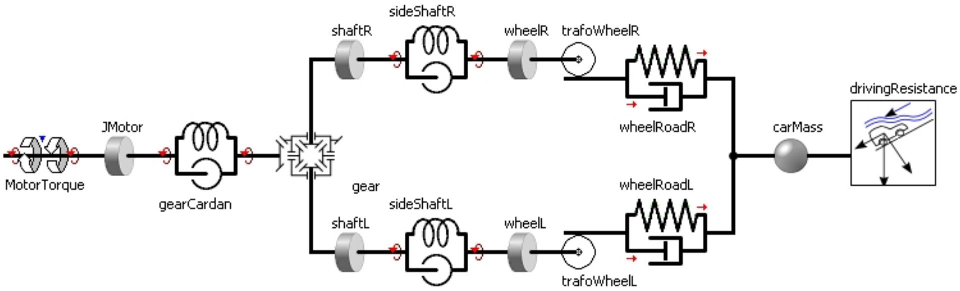
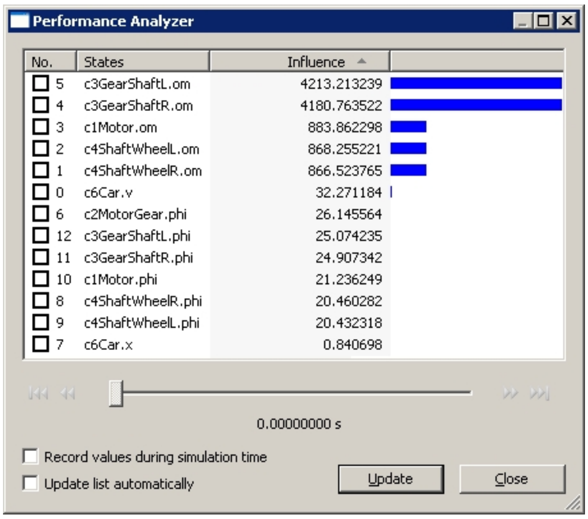


Figure 8: SimulationX model of a simple drive train

5.2.3 Performance Analysis

Both the analysis of the natural frequencies and the energy analysis operate at the current working point of the linearized system. Often a conclusion over the complete simulation period is required. The Performance analysis of Fig. 11 records an error criterion for each state variable during the offline simulation by summing up all local error estimates. Therefore it can be applied to identify critical model parts, e.g. stiff components or strong non-linearities.



No.	States	Influence
☐ 5	c3GearShaftL.om	4213.213239
☐ 4	c3GearShaftR.om	4180.763522
☐ 3	c1Motor.om	883.862298
☐ 2	c4ShaftWheelL.om	868.255221
☐ 1	c4ShaftWheelR.om	866.523765
☐ 0	c6Car.v	32.271184
☐ 6	c2MotorGear.phi	26.145564
☐ 12	c3GearShaftL.phi	25.074235
☐ 11	c3GearShaftR.phi	24.907342
☐ 10	c1Motor.phi	21.236249
☐ 8	c4ShaftWheelR.phi	20.460282
☐ 9	c4ShaftWheelL.phi	20.432318
☐ 7	c6Car.x	0.840698

0.00000000 s

☐ Record values during simulation time

☐ Update list automatically

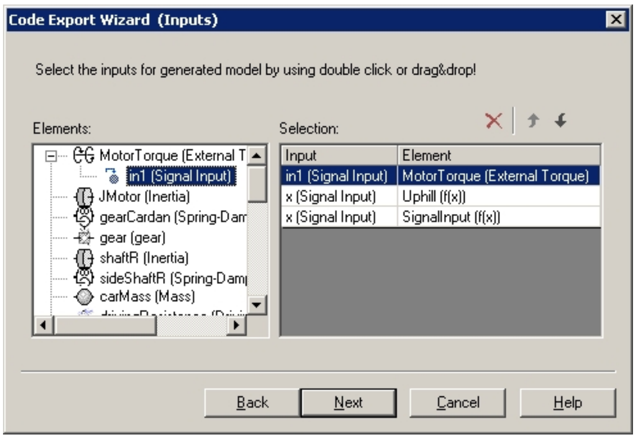
Figure 11: Performance Analyzer

6 Modelica 3.1 Language Extensions

The new Modelica language 3.1 specification [11] introduces language extensions that ease the mapping of models to execution environments. These language extensions are useful for the generation of HIL models, too.

The SimulationX Code Export Wizard is used to create a HIL model for a chosen HIL environment. As shown in section 3 the HIL target is selected here and inputs, outputs and parameters are defined (Fig. 12).

The Code Export Wizard also manages the subsequent steps (code generation, compilation and upload to the real-time target). This proceeding is very convenient if the complete model is mapped to one HIL platform. If the real-time model consists of several parts, or a model has to be split to run on several processor cores this approach becomes a little bit cumbersome. The user has to break up such models, copy each part to separate submodels and generate C code for each of them.



Select the inputs for generated model by using double click or drag&drop!

Elements:

- MotorTorque (External T
  - in1 (Signal Input)
- JMotor (Inertia)
- gearCardan (Spring-Dam
- gear (gear)
- shaftR (Inertia)
- sideShaftR (Spring-Damp
- carMass (Mass)

Selection:

Input	Element
in1 (Signal Input)	MotorTorque (External Torque)
x (Signal Input)	Uphill (f(x))
x (Signal Input)	SignalInput (f(x))

Figure 12: SimulationX Code Export Wizard with inputs page, model tree view and some inputs selected

Using the new Modelica 3.1 language extensions

- `decouple()` operator,
- `mapping` annotation,
- the task/subtask definition

a model can be separated in place and exported at once.

These new Modelica features look very promising for HIL targets that support the option to run models in several parallel tasks.

7 Conclusions

A simulation tool can already provide fundamental real-time support by

- Symbolic preprocessing,
- Efficient model code generation,
- Appropriate solvers.

For complex physical models a reduction is mostly additionally required and supported by SimulationX by

- Switchable model complexity,
- Analysis of natural frequencies and mode shapes,
- Distribution of energy,
- Performance analysis for state variables.

The consideration of the potential real-time capability of physical models already during the modeling stage results in better performance since similar models for offline and real-time simulation can shorten the model reduction steps. An appropriate model structuring (e.g. replaceable types) can also ease the model reduction and lead to higher process reliability.

ITI systematically deals with the real-time challenge. E.g. the TEMO project [12] is a joint research project by TLK Thermo GmbH, Braunschweig University of Technology, Visteon Deutschland GmbH, Daimler AG and ITI GmbH for preparation of real-time capable model components in heat conduction and thermal-fluid applications.

References

[1] Kurz, S., Wittler, G.: Hardware-in-the-Loop-Simulation: Eine Technologie im Wandel der Zeit. In: Proceedings of the 7th Haus-der-Technik-Tagung “HIL Simulation”, München, Germany, 27-28 February 2007.

[2] Blochwitz, T., Uhlig, A.: Modellgenerierung für HIL-Simulationen auf der Basis physikalischer Ansätze. In: Proceedings of the 8th Haus-der-Technik-Tagung “HIL Simulation”, Kassel, Germany, 16-17 September 2008.

[3] SimulationX: http://www.simulationx.com

[4] Richter, S.: Untersuchung zur Echtzeitsimulation von Modellen aus ITI SimulationX. Dresden, Germany: Master thesis, Dresden University of Technology, Faculty of Electrical Engineering and Information Technology, Institute of Automation, 2006.

[5] Blochwitz, T., Kurzbach, G., Neidhold, T.: An External Model Interface for Modelica. In: Proceedings of the 6th Modelica Conference 2008, Bielefeld, Germany, Modelica Association, 3-4 March 2008.

[6] Simulink: Writing S-Functions. The MathWorks, Inc., Natick, USA, March 2009.

[7] dSPACE DS1006 Processor Board: http://www.dspace.de

[8] SCALE-RT: http://www.scale-rt.com

[9] NI VeriStand: http://www.ni.com/veristand

[10] Rodionow, P., Grützner, S., Schreiber, U.: Erstellung, Reduktion und Validierung von Simulationsmodellen am Beispiel eines kompletten Kfz-Antriebsstranges. In: Proceedings of the 1st SimPEP Kongress, Veitshöchheim, Germany, 14-15 June 2007.

[11] Modelica Association: Modelica, A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version 3.1, 27 May 2009.

[12] TEMO – Thermische Echtzeitfähige Modelle: http://www.pt-it.pt-dlr.de/_media/Infoblatt_TEMO.pdf

Modelica Library for Building Heating, Ventilation and Air-Conditioning Systems

Michael Wetter
Simulation Research Group, Building Technologies Department,
Environmental Energy Technologies Division, Lawrence Berkeley National Laboratory,
Berkeley, CA 94720, USA

Abstract

This paper presents a freely available Modelica library for building heating, ventilation and air conditioning systems. The library is based on the `Modelica.Fluid` library. It has been developed to support research and development of integrated building energy and control systems. The primary applications are controls design, energy analysis and model-based operation.

The library contains dynamic and steady-state component models that are applicable for analyzing fast transients when designing control algorithms and for conducting annual simulations when assessing energy performance. For most models, dimensional analysis is used to compute the performance for operating points that differ from nominal conditions. This allows parameterizing models in the absence of detailed geometrical information which is often impractical to obtain during the conceptual design phase of building systems.

In the first part of this paper, the library architecture and the main classes are described. In the second part, an example is presented in which we implemented a model of a hydronic heating system with thermostatic radiator valves and thermal energy storage.

Keywords: building energy systems, heating, ventilation, air-conditioning, controls

1 Introduction

Buildings account for a large fraction of carbon dioxide emission and energy consumption. For example, in the United States, buildings account for 38% of total carbon dioxide emissions, 70% of electricity consumption and 50% of natural gas consumption, while less than 2% of the building sector's energy consumption is from renewable energy [5]. Several government bodies and professional societies have set the goal to mandate Net Zero Energy Buildings (ZEB) in the next 15 to 20 years. Such buildings should produce as much energy as they consume on an annual average. The challenges inherent in designing and operating high performance buildings and ZEBs demand a number of breakthroughs, both in technology, including software and information technology, and in the fundamental knowledge of optimizing whole building performance through integration and component operation [4]. To accelerate innovation towards ZEB, we started the development of a freely available open-source Modelica library for building energy and control systems that is available from `http://simulationresearch.lbl.gov`. For the early applications, we are particularly interested in enabling:

1. Rapid prototyping of new building components and systems.

2. Development of advanced control systems.

3. Reuse of models during operation for energy-minimizing controls, fault detection and diagnostics.

The current development is focused on the development of models for building heating, ventilation and air-conditioning equipment and their control systems, as opposed to the building envelope. However, the library contains an interface that allows coupling Modelica with the EnergyPlus whole building energy simulation program [3] for co-simulation. This allows the use of the detailed, extensively validated EnergyPlus program for modeling the heat transfer through the envelope and the daylight illuminance in rooms, while using Modelica for rapid prototyping and analysis of innovative energy and control systems. The coupling is done through the Building Controls Virtual Test Bed (BCVTB) that is currently under development at the Lawrence Berkeley National Laboratory [13]. The BCVTB is a middleware that is

DOI: 10.3384/ecp09430042

based on Ptolemy II [1, 6]. Ptolemy II is an open-source software framework to study modeling, simulation, and design of concurrent, real-time, embedded systems, with focus on the assembly of concurrent components and the use of heterogeneous mixtures of models of computation that govern the interactions between components. Ptolemy II allows modeling, analysis and simulation of systems that communicate and interact in a variety of ways such as synchronous or asynchronous, buffered or unbuffered. The BCVTB adds functionalities to Ptolemy II that allow coupling Modelica, EnergyPlus, MATLAB and Simulink to Ptolemy II for data exchange during the simulation. Interfaces to actual building control systems will be added in the future to enable use of models during the operation of the building.

The here described `Buildings` library is based on the `Modelica.Fluid` library 1.0 that uses the new concept of stream variables [8]. The `Modelica.Fluid` library provides a set of component models for one-dimensional thermo-fluid flow in networks of pipes. It demonstrates how to implement fluid flow component models that may have flow friction, heat and mass transfer. The library demonstrates how to deal with difficult design issues such as connector design, handling of flow reversal and initialization of states in a computationally efficient way. While many classes of this library can be used for our application domain, we provide in the `Buildings` library classes that extend and augment models from `Modelica.Fluid` where applicable, using the same modeling approach as `Modelica.Fluid`. Our library implements classes that are specifically needed for energy and control analysis at the whole building system level, as opposed to the development of individual equipment such as a refrigeration engine. Since our applications typically involve annual simulations, we generally do not model two-phase flow and refrigerant distribution in vapor compression cycles such as in [11, 10], although our library can be coupled to more detailed models.

When designing building energy systems, decisions that significantly affect building performance are typically done in the early design stage prior to specific equipment selection and prior to sizing the duct and piping networks. To enable assessing the performance of building energy systems when such detailed information is not yet available, many models in the `Buildings` library are implemented using dimensional analysis. Using dimensional analysis allows computation of a component's performance over a range of operating conditions based on a user-specified nominal operating point and its nominal performance, which is then scaled to different operating points based on laws of physics. Equations (4) and (7) are examples of such models.

Other developments for building energy system simulations in Modelica include the `ATPlus` library [9, 7]. We chose however to base our library on `Modelica.Fluid` as it allows modeling fluids with multiple compositions and trace substances. The first is important for humidity control in buildings while the second is needed to assess indoor air quality, for example, in variable air volume flow systems.

In Section 2, we will discuss the architecture of the `Buildings` library. Section 2.1.3 discusses the main partial models, and Sections 2.1.1 to 2.1.11 discuss the main packages and their models. In Section 3, we present an illustrative example in which we modeled a hydronic heating system. For other examples that include controls design, we refer to [12].

2 Library for building energy and control systems

Version 0.6.0 of the Buildings library consists of 73 models and blocks, and 26 functions that are public and non-partial.

2.1 Packages of the `Buildings` Library

The `Buildings` library is organized into the packages shown in Fig. 1. Components in these packages augment components from the Modelica Standard Library and from the `Modelica.Fluid` library. Most packages contain a package called `Examples`, which contains example applications that illustrate the typical use of components in the parent directories and that are used to conduct unit tests.

2.1.1 Package `Controls`

The package `Controls` contains blocks that are typically needed to implement controllers of building energy systems. For example, the package `Controls.Continuous` contains a block of a composite controller where a hysteresis block can switch equipment on/off, a timer allows for the locking out of equipment for a minimum time, and a PID controller computes the actuator signal when the controller is in the on state. Such a controller can for example be used to control a modulating boiler. The package `Controls.SetPoints` includes blocks for gain scheduling and

```
Buildings.Controls.Continuous
                  .Discrete
                  .SetPoints
Buildings.Fluids.Actuators.Dampers
                          .Motors
                          .Valves
                .Boilers
                .Chillers
                .Delays
                .FixedResistances
                .HeatExchangers
                .HeatExchangers.CoolingTowers
                               .Radiators
                .Interfaces
                .MassExchangers
                .MixingVolumes
                .Movers
                .Sensors
                .Sources
                .Storage
                .Utilities
         .HeatTransfer
         .Media
         .Utilities.Diagnostics
                   .IO
                   .Math
                   .Psychrometrics
                   .Reports
```

Figure 1: Package structure of the `Buildings` library. Only the major packages are shown.

time schedules. For example, there is a gain scheduler that can be used to compute in a hydronic heating system the set point of the supply water temperature as a function of the outside temperature. There is also a block for an occupancy schedule that has, as one of its outputs, the time until the next occupancy. This output can, for example, be used to start ventilating the building prior to occupancy to remove volatile organic compounds that may have accumulated when the ventilation was switched off.

2.1.2 Package `Fluid`

The `Fluid` package contains component models for thermo-fluid flow systems. The level of modeling detail is comparable with the models of the `Modelica.Fluid` library, and most models in `Buildings.Fluid` extend models from `Modelica.Fluid` to form components that are typically needed when modeling building energy systems.

The pressure drop calculation of most resistance models in our library is implemented in the partial model `BaseClasses.FlowModels.BasicFlowModel`. This model computes the relation

$$\dot{m} = \operatorname{sign}(\Delta p)\, k \sqrt{|\Delta p|}, \qquad (1)$$

where the flow coefficient k is assigned by models that extend `PartialResistance`. The model is used to model pressure drop in valves, pipes and mechanical equipment. The implementation is realized using the function `Modelica.Fluid.Utilities.regRoot2`, which regularizes the equation near the origin. Our implementation uses mass flow rate instead of volume flow rate. This has been done to avoid the influence of density, and hence temperature, on the pressure drop calculation. In pressure drop calculations for piping and duct networks in buildings, the uncertainty in the pressure drop calculation is typically larger than the error introduced when assuming a constant density. Note that our implementation still allows the use of a fluid model with variable density in order to model, for example, pressure differences due to a stack effect which can be important for high rise buildings or for naturally ventilated buildings.

At low flow rate, equation (1) is regularized to model laminar flow and to avoid numerical problems as its derivative is unbounded. For undisturbed flow in a pipe, the flow transition between laminar and turbulent typically occurs for Reynolds numbers in the range of 1500 to 4000, but turbulence may occur at much smaller Reynolds numbers due to flow mixers, diverters or bends. Also, in early design of buildings, the piping or ducting diameters are typically unknown. Thus, in our models, a user can specify at what fraction of the nominal flow rate equation (1) is regularized. Alternatively, the transition region can be specified by entering parameter values for the hydraulic diameter and the critical Reynolds number where turbulence occurs. If a user requires more detailed flow friction models, then models from `Modelica.Fluid` can be used in conjunction with the `Buildings` library.

Next, we will discuss the package `Fluids.Interfaces` which provides partial classes that are used by most models in the `Fluids` package. After this discussion, we will present the other packages.

2.1.3 Package `Fluids.Interfaces`

Similarly to `Modelica.Fluid.Interfaces`, there is a package `Buildings.Fluids.Interfaces` that defines the partial models for components that exchange heat or mass with one or two fluid streams. Such partial models are implemented for components with two and with four fluid ports. For models with two fluid ports, i.e., models that have one fluid stream, the top-level model is `PartialStaticTwoPortInterface`, which extends `Modelica.Fluid.Interfaces.PartialTwoPort` to add equations for the states at the ports and variables for the mass flow rate and pressure drop. This partial model is extended by two models: `PartialStaticTwoPortHeatMassTransfer` and `PartialDynamicTwoPortTransformer`. The first adds equations for the enthalpy, species, trace substances and pressure drop between its ports. It also introduces the variables `Q_flow` and `mXi_flow[Medium.nXi]` that need to be assigned by models that extend this partial model. These variables can for example be used to implement a steady-state model for a heater and humidifier. The second partial model, `PartialDynamicTwoPortTransformer`, adds a pressure drop element and an instantaneously mixed volume to the base class. This partial model is used to implement dynamic components that add heat or species to the fluid. Clearly, it would have been convenient to use `PartialDynamicTwoPortTransformer` also for steady-state models by configuring the volume model in such a way that energy and mass balance are steady-state. Such a configuration has been tested, but it led to a larger equation system, it required using the computationally less efficient function `actualStream()` to compute the enthalpy flow rate at the ports of the volume, and it led in test models to divisions by zero for zero mass flow rate. For all of the above models, there are also similar versions with four fluid ports that are used as partial models for implementing models that exchange heat or mass between two fluid streams.

2.1.4 Package `Fluids.Actuators`

The package `Fluids.Actuators` contains models of valves and air dampers, as well as of motors that can be used in conjunction with the actuators. Actuator models are based on a flow coefficient ϕ, defined as the flow rate at the current actuator position $y \in [0, 1]$ divided by the flow rate at $y = 1$, i.e.,

$$\phi = \frac{C_v(y)}{C_v(y=1)}. \qquad (2)$$

This flow ratio is proportional to k in (1). For the two-way valve model with linear opening characteristics, `TwoWayLinear`, the flow ratio is $\phi = l + y(1-l)$, where $0 \leq l \ll 1$ is the valve leakage. Fig. 2 shows ϕ as a function of the valve opening y for a linear valve, an equal percentage valve and a quick opening valve. For better display, untypical values of $l = 0.05$ and a rangeability for the equal percentage valve of 10 have been selected. These two-way valve models are also used to construct three way valve models with linear characteristics, or with a combination of equal percentage characteristics in the main flow path and linear characteristics in the bypass, to allow the modeling of valves that are typically used in hydronic heating systems.

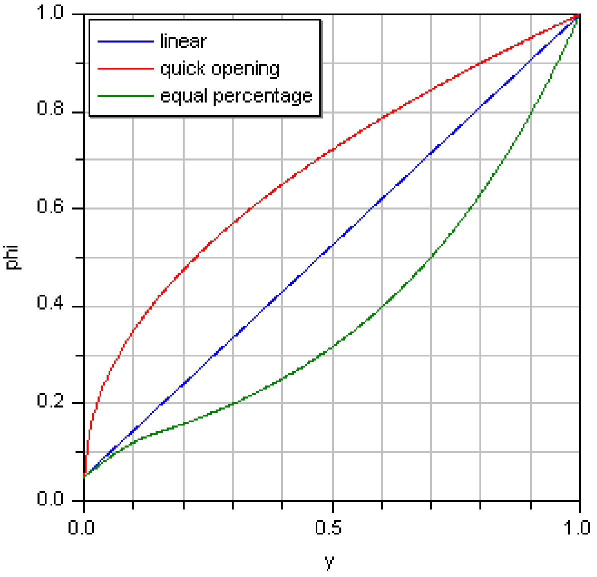


Figure 2: Flow characteristic $\phi(y)$ for valves.

Besides valves, there are also air damper models with exponential opening characteristics and models for terminal boxes of variable air volume flow systems.

There is also a model of a motor with hysteresis and finite actuation speed that can be used with the valve or the damper models. If the current actuator position y is below (or above) the input signal u by an amount bigger than a hysteresis δ, then the position y is increased (decreased) using a finite speed until it reaches u.

2.1.5 Package `Fluids.Boilers`

The package `Fluids.Boilers` contains a boiler model that can either be configured as a dynamic model or as a steady-state model. The heat transferred from the combustion to the the water, $\dot{Q}$, is computed as

$$\dot{Q} = \dot{Q}_0 \, \frac{\eta(y,T)}{\eta(1,T_0)} \, y, \qquad (3)$$

where $\dot{Q}_0$ is the nominal heating capacity, $y \in [0,1]$ is a control signal (typically, $y \in 0 \cup [0.3,\ 1]$ for a modulating boiler) and $\eta \colon [0,\ 1] \times \Re \to \Re$ is the furnace efficiency based on the load fraction y and the furnace temperature T. The nominal heating capacity $\dot{Q}_0$ and its associated temperature T_0 are parameters. The model allows users to select different functional forms for $\eta(\cdot,\cdot)$. The heat $\dot{Q}$ is added to a volume from `Modelica.Fluid` (to model the fluid's heat balance) which is connected to a solid heat storage element from the Standard Modelica Library (to model heat stored in the metal). The model also exposes a heat port of a heat conductor to allow modeling heat losses to the ambient environment.

2.1.6 Package `Fluids.Chillers`

This package contains a model of a chiller whose coefficient of performance (COP) is proportional to the Carnot efficiency. Parameters are either the Carnot effectiveness $\eta_{c,0}$ at the nominal conditions, or the COP and the evaporator and condenser temperatures at the nominal conditions, COP_0, $T_{e,0}$ and $T_{c,0}$. In the latter case, the Carnot effectiveness is computed as

$$\eta_{c,0} = \text{COP}_0 \, \frac{T_{c,0} - T_{e,0}}{T_{e,0}}. \tag{4a}$$

A user can specify what temperatures should be used as the evaporator (or condenser) temperature. The available options are the temperatures of the fluid volume, of `port_a`, of `port_b`, or of the average temperature of `port_a` and `port_b`. The chiller COP is computed as the product

$$\text{COP} = \eta_{c,0} \, \frac{T_e}{T_c - T_e} \, \eta_{pl}(y), \tag{4b}$$

where $\eta_{pl}(\cdot)$ is a polynomial in the control signal y that can be used to take into account a change in COP at part load conditions. The electrical power consumption of the compressor is $P = P_0\, y$, where P_0 is a parameter for the nominal compressor power. The heat extracted from the evaporator is $\dot{Q}_e = \text{COP}\ P$, and the heat transferred to the condenser is $\dot{Q}_c = \dot{Q}_e + P$. The condenser and evaporator are modeled using volumes from `Modelica.Fluid` and can therefore be modeled dynamic or at steady-state.

2.1.7 Package `Fluids.HeatExchangers`

This package contains steady-state and dynamic heat exchanger models, some of which compute condensation of water vapor that may occur at a cooling coil. Simple models include a model with prescribed heat input into the medium, i.e., the transferred heat is $\dot{Q} = \dot{Q}_0\, u$, where the maximum heat transfer $\dot{Q}_0$ is a parameter and $u \in [0,\ 1]$ is an input signal.

There is also a constant effectiveness heat exchanger in which

$$\dot{Q} = \varepsilon \min(|\dot{C}_1|, |\dot{C}_2|)\, \Delta T_{in}, \tag{5}$$

where $\varepsilon \in (0,1)$ is a parameter, $\dot{C}_1$ is the heat capacity flow of the stream 1, and ΔT_{in} is the temperature difference of the two inlet temperatures.

If a more detailed dynamic model is required, then finite volume models of a dynamic cooling coil, optionally with water vapor condensation, can be used. In these models, each pipe is discretized along its flow path. Pipes can be arranged in parallel to form a register. There are headers to redirect the flow between registers, or between pipes inside a register. The registers are exposed to the air stream that flows from one register to another. The most basic element of the coil models consists of two fluid volumes, one for the air stream and one for the water stream, that are connected to a thermal storage model for the metal pipe (see Fig. 3).

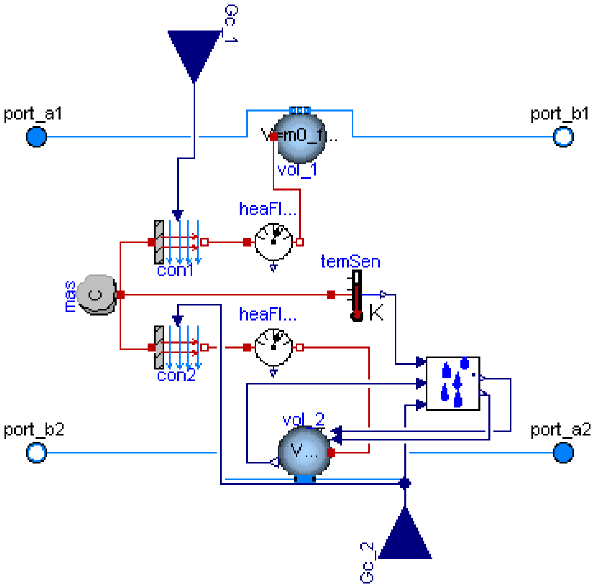


Figure 3: Basic element for a dynamic cooling coil with condensation. Models with index 1 and 2 refer to the water and air side, respectively.

The sensible convective heat transfer coefficient can be constant, or it can be a function of mass flow rate and temperature. The convective mass transfer coefficient is calculated based on similarity laws between heat and mass transfer. Using the Lewis number, which is defined as the ratio between the heat and mass diffusion coefficients, the ratio between convection heat transfer coefficient h (in $\mathrm{W}/(\mathrm{m}^2\,\mathrm{K})$) and mass transfer coefficient h_m (in m/s) is obtained as

$$\frac{h}{h_m} = \rho\, c_p\, Le^{(1-n)}, \tag{6a}$$

where ρ is the mass density, c_p is the specific heat capacity of the bulk medium and n is a coefficient from the boundary layer analysis, which is typically $n = 1/3$. From this equation, the water vapor mass flow rate $\dot{m}_w$ (in kg/s) is obtained as

$$\dot{m}_w = \frac{G_c}{c_p\, Le^{(1-n)}}\,(X_s - X_\infty), \tag{6b}$$

where $G_c = hA$ is the sensible heat conductivity (in W/K) and X_s and X_∞ are the water vapor mass fractions in the boundary layer and in the bulk of the medium. In this model, X_s is the saturation water vapor mass fraction corresponding to the surface temperature of the pipe.

The package `Fluids.HeatExchangers.CoolingTowers` contains models of cooling towers. The air inlet temperature is obtained from an input signal which can be set to the dry bulb or wet bulb temperature. There is a simple model in which the water outlet temperature minus the air inlet temperature is constant. There is also a more detailed model in which the water outlet temperature is computed based on the performance curve of a York cooling tower. This model can operate either with forced flow when the fan is operating or in free convection mode.

The package `Fluids.HeatExchangers.Radiators` contains a radiator model for hydronic space heating systems. The model can be configured as a steady-state or a dynamic model, and it can be discretized into finite volumes. If $n \geq 1$ denotes the number of discretization volumes, then the convective and radiative heat transfer is

$$\dot{Q}_c = (1-f_r)\,\dot{Q}_0\,\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\Delta T_{c,i}}{\Delta T_0}\right)^n, \tag{7a}$$

$$\dot{Q}_r = f_r\,\dot{Q}_0\,\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\Delta T_{r,i}}{\Delta T_0}\right)^n, \tag{7b}$$

where f_r is the fraction of radiative heat transfer, $\dot{Q}_0$ and ΔT_0 are the nominal heating capacity and temperature difference, and $\Delta T_{c,i}$ and $\Delta T_{r,i}$ are the convective and radiative temperature differences of the i-th volume. The parameters f_r, $\dot{Q}_0$ and ΔT_0 are typically available from product catalogs.

2.1.8 Package `Fluids.MassExchangers`

This package contains two models for mass exchangers. The model `HumidifierPrescribed` adds water to an air stream in the amount of $\dot{m}_w = y\,\dot{m}_{w,0}$, where $\dot{m}_{w,0}$ is a parameter and $y \in [0,\ 1]$ is an input.

The model `ConstantEffectiveness` transfers heat and moisture in the amount of

$$\dot{Q} = \varepsilon_s\,\min(|\dot{C}_1|, |\dot{C}_2|)\,\Delta T_{in}, \tag{8a}$$

$$\dot{m} = \varepsilon_l\,\min(|\dot{m}_1|, |\dot{m}_2|)\,\Delta X_{in}, \tag{8b}$$

where ε_s and ε_l are the sensible and latent effectiveness, $\dot{C}_1$ is the heat capacity flow rate of medium one, ΔT_{in} is the temperature difference over the inlet streams, $\dot{m}_1$ is the mass flow rate of stream one and ΔX_{in} is the difference in water vapor mass fraction across the two inlet streams.

2.1.9 Package `Fluids.Storage`

This package contains models of thermal energy storage tanks.

The model `Stratified` uses a user-specified number of fluid volumes that are connected in series to model a stratified storage tank. Heat conduction is modeled between the volumes through the fluid, and between the volumes and the ambient environment through the tank enclosure. A heat port that is connected to the fluid volume can be used to add a temperature sensor or to inject heat into the fluid when modeling a heat exchanger. The tank also contains a model that emulates buoyancy if there is a temperature inversion in the tank.

The model `StratifiedEnhanced` extends the above model to add equations that reduce the numerical dissipation that is introduced when fluid flows from one volume to another. Such numerical dissipation is typical for upwind discretization schemes. To reduce the numerical dissipation, we implemented a model that is based on the one described by Wischhusen [14].

2.1.10 Package `Media`

This package contains media models that can be used in addition to the models from `Modelica.Media`. Some of the media models in this package are based on simplified state equations and property equations that lead generally to a faster and more robust simulation compared to the models of `Modelica.Media`. For example, there are models that are similar to `Modelica.Media.Air.MoistAir`, but our implementations in `Buildings.Media.PerfectGases` are based on the

Table 1: Benchmark tests for medium models.

system model	Medium model	init.			simulation			computing time in [s]
		l	nl	J	l	nl	J	
1	`Modelica.Media.Air.MoistAir`	0	25	0	0	1	26	20
1	`Buildings.Media.PerfectGases.MoistAir`	0	25	0	0	1	26	13.5
1	`Buildings.Media.GasesPTDecoupled.MoistAir`	0	7	0	0	1	26	8.6
2	`Modelica.Media.Air.SimpleAir`	0	31	0	0	1	0	*
2	`Buildings.Media.GasesPTDecoupled.SimpleAir`	0	21	0	0	0	26	0.55

* Failed to solve initialization problem.

thermally perfect medium assumption [2], i.e., the specific heat capacity is constant and internal energy and enthalpy are functions of temperature only.

There is also a package `Buildings.Media.GasesPTDecoupled` that contains medium models that do *not* follow the ideal gas law. Instead, the medium model is based on the equation $\rho/\rho_{\text{stp}} = p/p_{\text{stp}}$, where p denotes pressure and ρ denotes mass density, and $p_{\text{stp}} = 101325\,\text{Pa}$ and $\rho_{\text{stp}} = 1.2\,\text{kg/m}^3$ for air. Hence, in an isobar heat exchange, density remains constant, which generally leads to smaller equation systems. Table 1 shows benchmark problems conducted with Dymola 7.1 using models that are part of the example models of the `Buildings` library. The system model 1 is a dynamic cooling coil with feedback control and valve motor with hysteresis (model `WetCoilDiscretizedPControl`). The system model 2 is an open-loop simulation of a variable air volume flow system that serves 6 rooms (model `MITScalable`). In Tab. 1, the first three columns are the statistics for the initialization problem, the next three columns are for the time integration, and the last column is the computing time. The columns with header "l" or "nl" denote the largest dimension of a linear or nonlinear system of equations, and the columns with header "J" denote the number of numerical Jacobians. For the system model 1, decoupling pressure and temperature reduced computing time by more than a factor of two. For the system model 2, Dymola was only able to solve the initialization problem with the simplified medium model.

2.1.11 Package `Utilities`

The package `Utilities.Diagnostics` contains blocks with assert statements that can for example be used to check that setpoints are within expected ranges, which can be helpful to automate some of the debugging during the development of large models.

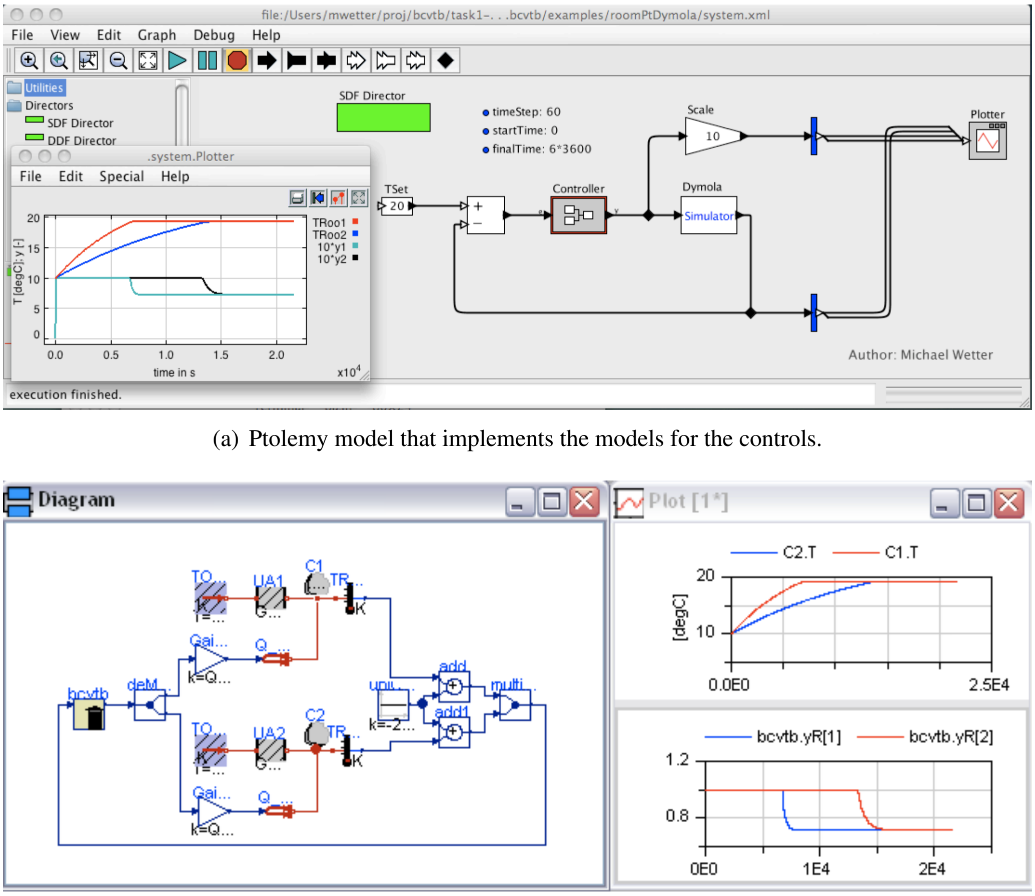
The package `Utilities.IO` contains blocks for input and output. Its subpackage `Utilities.IO.BCVTB` contains a block that allows linking Modelica to the Building Controls Virtual Test Bed (BCVTB). At the start of the simulation, this block establishes a socket connection using the Berkeley Software Distribution socket (BSD socket) implementation, and then exchanges data with the BCVTB during the simulation. The BCVTB interface allows

1. co-simulation between Modelica and the EnergyPlus whole building energy simulation program, MATLAB and/or Simulink,
2. the use of Ptolemy II to link heterogeneous models of computation with Modelica models, and
3. linking Modelica with building control systems through the controls interface that will be added to the BCVTB.

Fig. 4 shows a simple model that was used for testing the BCVTB interface. Fig. 4(a) shows the implementation in Ptolemy II, which models the controls. The block labeled "Dymola" sends control signals to Dymola and receives measured temperatures from Dymola. Fig. 4(b) shows the implementation of the heat transfer model in Dymola. The model on the very left sends the measured temperatures to Ptolemy II and receives the control signal from Ptolemy II. The data exchange is done through BSD sockets and hence can be across a network.

The package `Utilities.Math` contains functions and blocks that augment the mathematical functions of the Modelica Standard Library. Examples are once continuously differentiable approximations to the functions $y = \min(x_1, x_2)$, $y = \exp(|x|)$ and $y = |x|^n$, for $n > 0$, that are used by various models in the `Buildings` library.

There is also a package `Utilities.Psychrometrics` with psychrometric functions and a package `Utilities.Reports` with blocks for reporting output values to files.



(a) Ptolemy model that implements the models for the controls.

(b) Modelica model that implements the models for the heat transfer.

Figure 4: Illustration of the BCVTB interface for co-simulation using Ptolemy II and Modelica.

3 Applications

We will now present an example application of a hydronic heating system which is part of our library (see model `Buildings.Examples.HydronicHeating`). Applications in which the library has been used for controls design can be found in [12].

Fig. 5 shows the Modelica model of the hydronic heating system. On the lower left quadrant of the figure is the heating plant, with a dynamic model of a boiler and a circulation pump. The boiler loop is connected to a stratified thermal energy storage tank. Based on the temperatures of the top and bottom segment of the tank, a finite state machine switches the boiler on and off. The vertical lines in the middle of the figure are the water supply and return pipes to the rooms. The supply water temperature set point is scheduled based on the outside air temperature. An equal-percentage three-way valve mixes the water. The distribution pump has a controller that adjusts the pump revolution in order to maintain a constant pressure difference across the pump. On the upper right quadrant are models of two rooms. Each room has a thermostatic radiator valve with equal-percentage opening characteristics. The radiator models are dynamic and exchange radiative heat with the room enclosures and convective heat with the room air. The room air is modelled using a thermal capacity model and a thermal conductor (to model ventilation heat losses) from the Modelica Standard Library, and a finite difference model for transient heat conduction through the walls. There is also an occupancy schedule for each room, which is used to model heat gains from occupants, lights and equipment.

The total system model contained 353 simulation components that led to a differential algebraic equation system with 2278 scalar equations and 29 state variables. A simulation of two days took about 12 seconds on a desktop computer.

Fig. 6 shows the time trajectories for the second day of simulation. The top figure shows the room temperatures. The middle figure shows the valve positions (red and blue are radiator valves and black is the three-way valve). The bottom figure shows the supply

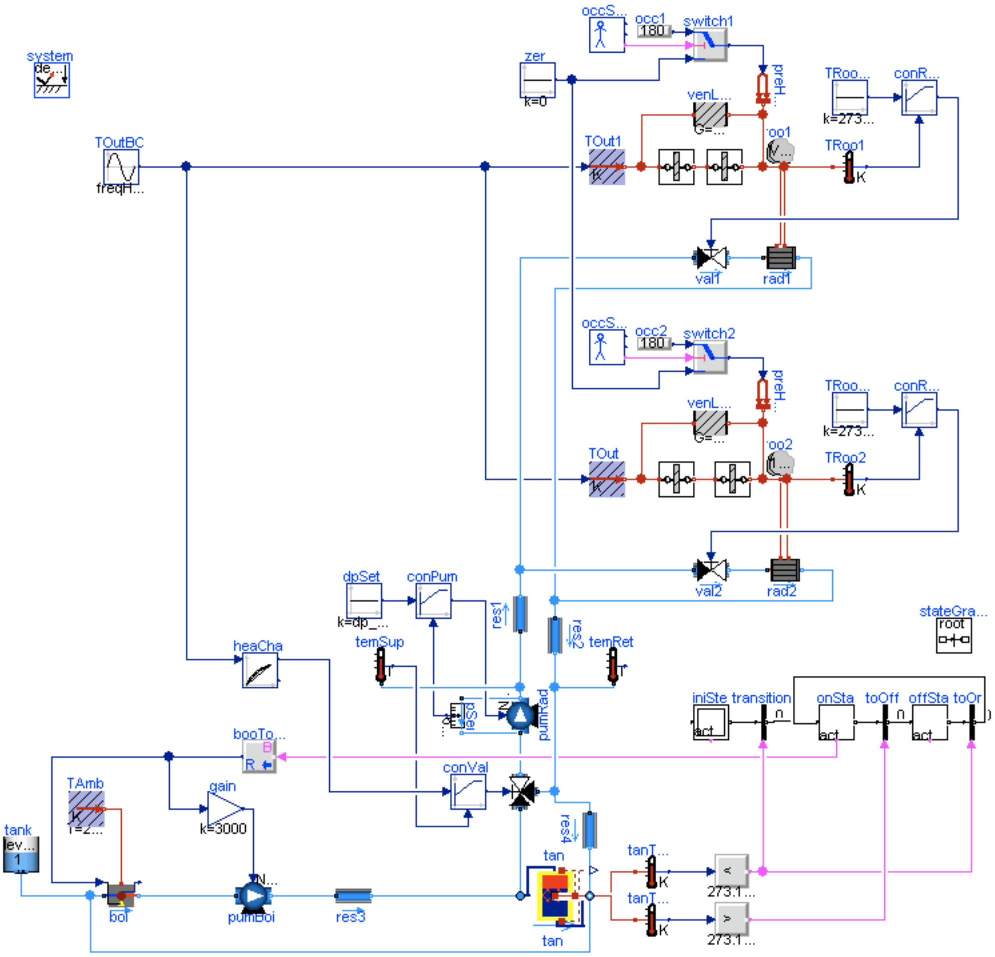


Figure 5: Model of the hydronic heating system with thermal energy storage.

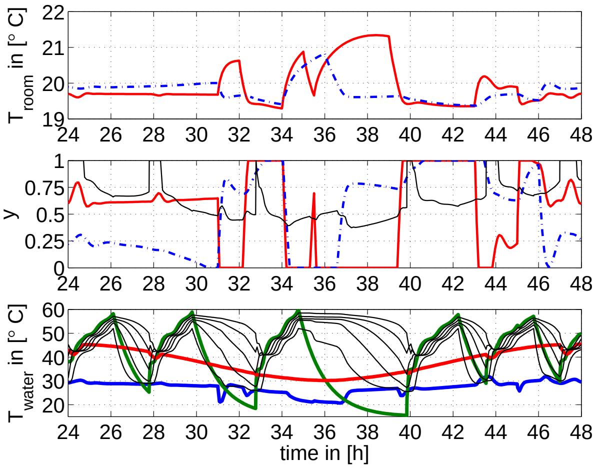


Figure 6: Time trajectories for the second day of simulation of the hydronic system model.

water temperature after the three-way valve (red), the return water temperature from the two rooms (blue), the boiler temperature (green) and the temperatures of the five volumes used to discretize the thermal storage tank (black).

The room temperature set point is well maintained. The peaks in room air temperature are due to internal heat gains and cause the thermostatic radiator valves to close as expected.

4 Conclusions

The `Modelica.Fluid` package provided a rich set of basic models that we could either use directly or adapt to implement a library for building energy systems. The flexibility of Modelica has been shown to enable analysis of building energy and control systems [12] that are outside the capability of traditional building energy simulation programs.

However, numerically solving for initial conditions and time trajectories of such systems can sometimes pose difficulties for solvers. We believe that further advances in model formulation and in solution algorithms are needed to make equation-based modeling of building energy and control systems available to a larger audience who is not trained in addressing numerical problems. In the meantime, however, Modelica is a valuable tool for researchers who can debug numerically challenging problems, as it allows flexible system modeling and the addition of new models to existing libraries with less effort compared to causal programming languages.

5 Acknowledgments

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

References

[1] Christopher Brooks, Edward A. Lee, Xiaojun Liu, Steve Neuendorffer, Yang Zhao, and Haiyang Zheng. Ptolemy II – heterogeneous concurrent modeling and design in Java. Technical Report No. UCB/EECS-2007-7, University of California at Berkeley, Berkeley, CA, January 2007.

[2] William B. Brower. *A primer in fluid mechanics: dynamics of flows in one space dimension.* CRC Press, Boca Raton, FL, USA, 1999.

[3] Drury B. Crawley, Linda K. Lawrie, Curtis O. Pedersen, Richard J. Liesen, Daniel E. Fisher, Richard K. Strand, Russell D. Taylor, Frederick C. Winkelmann, W. F. Buhl, A. E. Erdem, and Y. J. Huang. EnergyPlus, a new-generation building energy simulation program. In *Proc. of Renewable and Advanced Energy Systems for the 21st Century*, Lahaina, Maui, Hawaii, April 1999.

[4] Building Technologies Program – planned program activities for 2008-2012. Technical report, U.S. Department of Energy, Energy Efficiency and Renewable Energy, 2008.

[5] Buildings Energy Data Book. Technical Report DOE/EE-0325, U.S. Department of Energy, Washington, D.C., September 2008.

[6] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, J. Ludwig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity – the Ptolemy approach. *Proc. IEEE*, 91(1):127–144, January 2003.

[7] Felix Felgner, Rolf Merz, and Lothar Litz. Modular modelling of thermal building behaviour using modelica. *Mathematical and Computer Modelling of Dynamical Systems*, 12(1):35–49, 2006.

[8] Rüdiger Franke, Francesco Casella, Martin Otter, Katrin Proelss, Michael Sielemann, and Michael Wetter. Standardization of thermo-fluid modeling in Modelica.Fluid. In Francesco Casella, editor, *Proc. of the 7-th International Modelica Conference*, Como, Italy, September 2009. Modelica Association.

[9] Rolf Mathias Merz. *Objektorientierte Modellierung thermischen Gebäudeverhaltens.* PhD thesis, Universität Kaiserslautern, September 2002.

[10] Christoph C. Richter. *Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic Systems.* PhD thesis, Technischen Universität Carolo-Wilhelmina zu Braunschweig, Germany, January 2008.

[11] Hubertus Tummescheit, Jonas Eborn, and Katrin Prölss. AirConditioning - a Modelica library for dynamic simulation of AC systems. In Gerhard Schmitz, editor, *Proceedings of the 4th Modelica conference*, pages 185–192, Hamburg, Germany, March 2005. Modelica Association and Hamburg University of Technology.

[12] Michael Wetter. Modelica-based modeling and simulation to support research and development in building energy and control systems. *Journal of Building Performance Simulation*, 2(2):143–161, June 2009.

[13] Michael Wetter and Philip Haves. A modular building controls virtual test bed for the integration of heterogeneous systems. In *Proc. of SimBuild*, Berkeley, CA, August 2008. IBPSA-USA.

[14] Stefan Wischhusen. An enhanced discretization method for storage tank models within energy systems. In Christian Kral and Anton Haumer, editors, *Proc. of the 5-th International Modelica Conference*, volume 1, pages 243–248, Vienna, Austria, September 2006. Modelica Association and Arsenal Research.

HumanComfort Modelica-Library
Thermal Comfort in Buildings and Mobile Applications

Boris Michaelsen   Joerg Eiden
XRG Simulation GmbH
Harburger Schlossstr. 6-12, 21079 Hamburg, Germany
{michaelsen,eiden}@xrg-simulation.de

1 Abstract

The `HumanComfort` library provides basic models to predict the thermal comfort of occupants within an air-conditioned space in mobile or stationary applications. The library is modularly structured to allow a flexible use in combination with air conditioning systems or complex zone simulations. The `HumanComfort` library provides models and functions to establish a multi zone model to analyse the interaction between the inertia of the zone and a HVAC (heating ventilation and air conditioning) system relating to the thermal comfort of the occupants. The validation of the building simulation including a thermal comfort analysis was done by a comparative validation test with EnergyPlus using DesingBuilder [1].

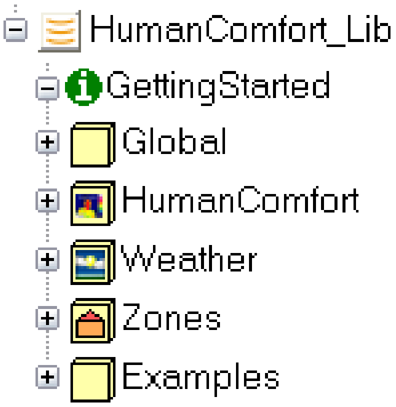
Keywords: human comfort, thermal comfort, PMV, PPD, GTO, multi zone model

2 Introduction

Energy systems are often optimized with regard to economical rules, on the other hand humans feel comfortable within certain limits defined by thermal and personal factors. Studies showed that the change of the thermal sensations can be defined by means of characteristic numbers and standardized mathematical methods. The `HumanComfort` library is currently developed within the European research project EuroSysLib-D, providing basic models to predict the human comfort in form of mathematical criteria and also graphical visualizations. The `HumanComfort` library uses an integrated approach to simulate a zone (building or cabin model) and an air-conditioning simultaneously. This approach is crucial for the optimal dimensioning of an air conditioning system by taking the thermal comfort into account.

3 Library Structure

The main structure of the `HumanComfort` library is shown in fig. 1.



- HumanComfort_Lib
  - GettingStarted
  - Global
  - HumanComfort
  - Weather
  - Zones
  - Examples

Figure 1: `HumanComfort` library structure

The **GettingStarted** package includes a brief introduction of the library concept and shows the most important steps to handle the library.

The **Global** package encloses common models and functions, icons, interfaces, and special types.

The **HumanComfort** package contains basic functions and models to predict the thermal comfort. It also contains a package for the model animation of the characteristic numbers.

The **Weather** package encloses functions and models to provide annual weather data for the zone model.

The **Zone** package provides models for stationary and mobile applications. Mobile applications will be covered by models for aircraft and automotive cabins. The mobile zones are designed as detailed as necessary to analyse the human comfort. It is not the aim of the mobile zones to cover up all effects in detail.

The **Examples** package includes examples for typical applications and verification models.

4 Human Comfort

Thermal comfort is defined as that condition of mind that expresses satisfaction with the thermal environment. Dissatisfaction may be caused by warm or cool discomfort of the body or may be caused by an unwanted cooling or heating [2]. Thermal comfort standards are based on collected data from laboratory and field studies which provide the necessary statistical data to define the conditions that a specified percentage of occupants will find thermally comfortable.

The implemented thermal prediction models consider statistic and adaptive comfort models, based on the following standards:

- DIN EN ISO 7730 [2]
- ASHRAE Standard 55-2004 [3]
- ISSO 74 [4]

To predict the thermal comfort, it is necessary to consider the thermal factors: temperature, radiant temperature, humidity, and air velocity as well as the personal factors: activity (metabolic rate) and clothing insulation.
The relation between this factors are provided by the standards as mathematical formulated characteristic numbers.

Using the characteristic numbers Predicted Mean Vote (PMV) and Predicted Percentage of Dissatisfied (PPD) the thermal comfort can be calculated.

The local thermal discomfort will be considered by the Percentage of Discomfort (PD) and can be divided into the following sections:

- Radiant temperature asymmetry
- Draft
- Vertical air temperature difference
- Cool or warm wall
- Cool or warm ceiling
- Cool or warm floors

Adaptive thermal comfort consider the fact that occupants will tolerate a wider range of temperatures and internal conditions when more control is allowed over their internal environment. The implementation of the ISSO 74 (Dutch Thermal Comfort Guideline) expands the `HumanComfort` library with additional characteristic numbers and diagrams regarding the adaptive thermal comfort.

The operative indoor temperature, the Adaptive Temperature Limits (ATG) and the annual characteristic numbers Weighted Temperature Exceeding Hours (GTO), TO25°C and TO28°C (Operative Temperature Limits) cover up this adaptive thermal comfort.

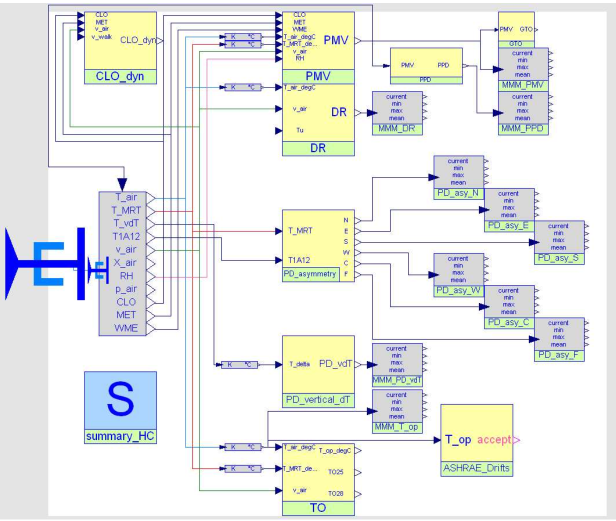


Figure 2: Diagram layer of the `HumanComfort` model; including the functions and models for the PMV, PPD, and PD's calculation, and the HumanComfort adapter

The characteristic numbers are pooled in one model named `HumanComfort` (see fig. 2). The therefore established `HumanComfort` adapter enables an easy interface for the end-user. This allows a complete thermal comfort analysis by simply connecting the `HumanComfort` model to this adapter.

4.1 Characteristic Numbers

Calculation methods and the resulting characteristic numbers are provided in several international standards ([2], [3], [4]). The characteristic numbers are implemented as functions and models for a flexible use. The most important characteristic numbers are described in detail in the following sections.

4.1.1 Predicted Mean Vote (PMV)

The PMV is an index that predicts the mean value of the votes of a large group of persons on the sevenpoint thermal sensation scale [3]. It uses heat balance principles to relate the thermal and personal factors for thermal comfort to the average response of people on the sensation scale. The ideal range is between -0.5 and +0.5 (neutral) (see fig. 3).

PMV	Comfort
+3	hot
+2	warm
+1	slightly warm
0	neutral
-1	slightly cool
-2	cool
-3	cold

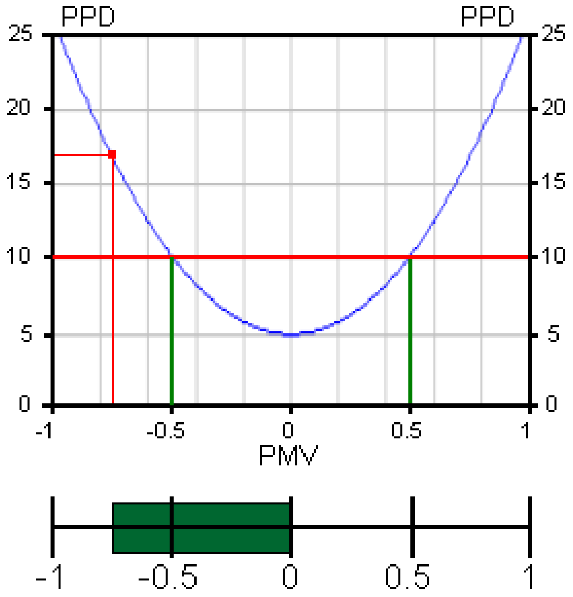


Figure 3: PMV and PPD visualization, screen shot of the Dymola model animation window

PMV	=	Predicted Mean Vote	$[-]$
PPD	=	Predicted Percentage of Dissatisfied	$[\%]$

Thermal factors:

T_a	=	ambient temperature	$[K]$
T_r	=	mean radiant temperature	$[K]$
v_{air}	=	relative air velocity	$[m/s]$
p_a	=	water vapour pressure	$[Pa]$
RH	=	relative humidity	$[\%]$

Personal factors:

CLO	=	thermal isolation of the clothing	$[clo = 0.155 \cdot m^2 \cdot K/W]$
M	=	metabolic rate	$[met = 58.15 W/m^2]$
W	=	external work	$[W/m^2]$

Internal values of PMV function:

I_{cl}	=	thermal resistance of the clothing	$[m^2 \cdot K/W]$
h_c	=	convective heat transfer coefficient	$[W/m^2 \cdot K]$
f_{cl}	=	ratio of clothed body	$[-]$
HL_x	=	heat loss factors	$[-]$
TS	=	thermal sensation transformation coefficient	$[-]$

The first step to determine PMV is the calculation of the thermal resistance of the clothing I_{cl} and the ratio of clothed body f_{cl} (see eq. 1 and 2).

$$I_{cl} = 0.155 \cdot CLO \tag{1}$$

$$f_{cl} = \begin{cases} 1.05 + 0.645 \cdot I_{cl} & \text{for} \quad I_{cl} > 0.078 \\ 1 + 1.29 \cdot I_{cl} & \text{for} \quad I_{cl} \leq 0.078 \end{cases} \tag{2}$$

It is necessary to iterate, due to $h_c = f(T_{cl})$ and $T_{cl} = f(h_c, T_{cl})$, see eq. 3 and 4. The allowed failure of the iteration process is 10^{-9}.

$$h_c = \begin{cases} 2.38 \cdot |T_{cl} - T_a|^{\frac{1}{4}} & \text{for} \quad 2.38 \cdot |T_{cl} - T_a|^{\frac{1}{4}} > 12.1 \cdot \sqrt{v_{air}} \\ 12.1 \cdot \sqrt{v_{air}} & \text{for} \quad 2.38 \cdot |T_{cl} - T_a|^{\frac{1}{4}} \leq 12.1 \cdot \sqrt{v_{air}} \end{cases} \tag{3}$$

$$\begin{aligned} T_{cl} \quad = \quad & 35.7 - 0.028 \cdot (M - W) \\ & - I_{cl} \cdot [3.96 \cdot 10^{-8} \cdot f_{cl} \cdot (T_{cl}^4 - T_r^4) + f_{cl} \cdot h_c \cdot (T_{cl} - T_a)] \end{aligned} \tag{4}$$

The user has two options to describe the input of the humidity. The first one is to use the water vapour pressure p_a. The second option is to use the relative humidity RH. The following equation describes the relation between p_a and RH.

$$p_{a,internal} = \begin{cases} RH \cdot 10 \cdot e^{16.6536 - 4030.183/(t_a + 235)} & \text{for} \quad p_a = 0 \\ p_a & \text{for} \quad p_a > 0 \end{cases} \tag{5}$$

The calculations of all heat losses and the thermal sensation transformation coefficient are the last steps to determine the PMV (see eq. 6 to 12).

$$TS = [0.303 \cdot e^{-0.036 \cdot M} + 0.028] \tag{6}$$

$$HL_{1(skin)} = 3.05 \cdot 0.001 \cdot (5733 - 6.99 \cdot (M - W) - P_a) \tag{7}$$

$$HL_{2(sweat)} = \begin{cases} 0.42 \cdot (M - W - 58.15) & \text{for} \quad M - W > 58.15 \\ 0 & \text{for} \quad M - W \leq 58.15 \end{cases} \tag{8}$$

$$HL_{3(latent\_respiration)} = 1.7 \cdot 0.00001 \cdot M \cdot (5867 - p_a) \tag{9}$$

$$HL_{4(dry\_respiration)} = 0.0014 \cdot M \cdot (34 - t_a) \tag{10}$$

$$HL_{5(radiation)} = 3.96 \cdot 0.00000001 \cdot f_{cl} \cdot (T_{cl}^4 - T_r^4) \tag{11}$$

$$HL_{6(convection)} = f_{cl} \cdot h_c \cdot (t_{cl} - t_a) \tag{12}$$

The final step is the calculation of the PMV.

$$PMV = TS \cdot (M - W - HL_1 - HL_2 - HL_3 - HL_4 - HL_5 - HL_6) \tag{13}$$

The PMV is the basis of the PPD and the GTO.

4.1.2 Predicted Percentage of Dissatisfied (PPD)

The PPD is an index that establishes a quantitative prediction of the percentage of thermally dissatisfied people determined from PMV (see eq. 14 and fig. 4) [3].

$$PPD = 100 - 95 \cdot e^{-0.03353 \cdot PMV^4 - 0.2179 \cdot PMV^2} \tag{14}$$

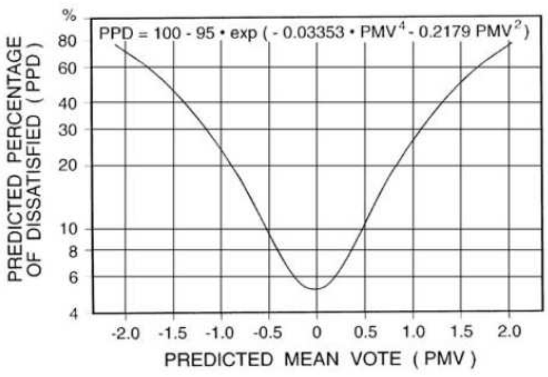


Figure 4: Correlation between PMV and PPD

4.1.3 Weighted Temperature Exceeding Hours (GTO)

The GTO-method which is based on the analytical `PMV` model is an annual characteristic number.

In the GTO-method the hours during which the calculated or actual PMV exceeds the PMV-limit of +0.5 are weighted proportional to the PPD. With this, a temperature that results in a PPD of 20% during 1 hour will be weighted twice as severe as a temperature that results in 10% dissatisfied occupants [4]. When the PMV has a value of 0.5 (PPD=10) the weighting factor is 1.0, see tab. 1:

PMV	PPD	Weighting factor
0	5	0.0
0.5	10	1.0
0.7	15	1.5
1.0	26	2.6

Table 1: Correlation of PMV, PPD, and the GTO weighting factor [4]

The GTO will be calculated every 3600 seconds, by using the actual PMV value as the input. This is implemented by using the Modelica sample function, see source code 1. The governing equations for the GTO are described in eq. 15- 17.

$$PPD_{hour} = 100 - 95 \cdot e^{-0.03353 \cdot PMV_{hour}^4 - 0.2179 \cdot PMV_{hour}^2} \qquad (15)$$

$$GTO_{hour} = \begin{cases} \frac{PPD_{hour}}{10} & for \quad PMV \geq 0.5 \\ 0 & \text{for} \quad PMV < 0.5 \end{cases} \qquad (16)$$

$$GTO = \sum_{hour=1}^{8760} GTO_{hour} \qquad (17)$$

PMV_{hour}	=	Predicted Mean Vote of current hour	[–]
PPD_{hour}	=	Predicted Percent. of Dissatisfied of curr. hour	[%]
GTO_{hour}	=	Weighted temperature of current hour	[–]
GTO	=	Weighted temperature Exceeding Hours	[–]

The following source code shows the implementation of the GTO-Method in Modelica.

```
 hour_step = floor(time/3600);
 hour = time/3600;
 day = hour/24;

when initial() then
  startTime = time;
end when;

der(PMV_int) = PMV;
PMV_mean = noEvent(if time − startTime > 0 then
  PMV_int/(time − startTime) else PMV);

when sample(0, 3600) then
  PMV_int_step = PMV_int;
  PMV_int_step_pre = pre(PMV_int_step);
  PMV_mean_h = noEvent(if time − startTime > 0 then
    (PMV_int_step − PMV_int_step_pre)/(3600) else PMV_in);
  if abs(PMV_in) >= 0.5 then
    GTO = pre(GTO) + PPD(PMV_in)/10;
  else
    GTO = pre(GTO);
  end if;
end when;
```

Source Code 1: Modelica code of the GTO `model`

The in [4] given annual limit of 150 will be considered in the analyses of the GTO. Fig. 5 describes the correlation between the hourly values of PMV, PPD and GTO. In this theoretical case the PMV increase from -3 to +3 in a time period of 60h. The grey area in the graph shows a constant GTO value within the PMV limit from -0.5 to +0.5.

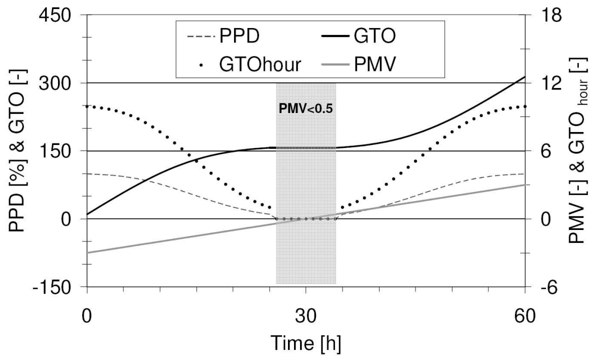


Figure 5: Correlation of the hourly values of PMV, PPD, and GTO by an increasing PMV from -3 to +3

4.2 Other Characteristic Numbers

The following characteristic numbers are defined in the standards and are implemented in the `HumanComfort` library, but will not be described in this paper in detail.

- DR - Draught Rating [2]
- PD Due to Radiant Asymmetry [2]
  - PD Due to Cool Wall
  - PD Due to Warm Wall

  - PD Due to Cool Ceiling
  - PD Due to Warm Ceiling
- PD Due to Cool or Warm Floors [2]
- PD Due to Vertical Air Temperature Difference [2]
- ATG - Adaptive Temperature Limits [4]
- TO - Operative Temperature Limits [4]
- DAR - Limits on Temperature Drifts and Ramps [3]

5 Zones

The zones are separated into models for mobile and stationary applications. The stationary applications are single rooms or small buildings. The mobile applications will be automotive or aircraft cabins. Each application can be combined with the weather or HumanComfort module (see fig. 6).

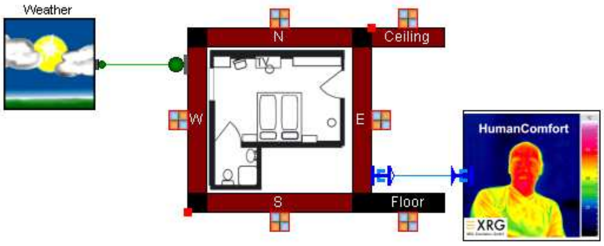


Figure 6: One room model with a connected weather and HumanComfort model

The main features of the zone models are:

Internal heat radiation between visible surfaces. The RadiationExchangeApproximated model considers the heat exchange due to heat radiation of surfaces to each other. The external solar radiation through the windows and the internal heater radiation will be considered as well. Therefore the heat emissions of all surfaces, defined by their areas, temperatures and long wave emissions coefficients will be collected in a virtual heat pool. The solar radiation, defined by the short wave emission coefficient, and the heater radiation are also collected in the virtual heat pool. The absorbed heat flows of the surfaces are defined by their areas, temperatures and absorption coefficients. The schematic of the virtual heat pool is shown in fig. 7.

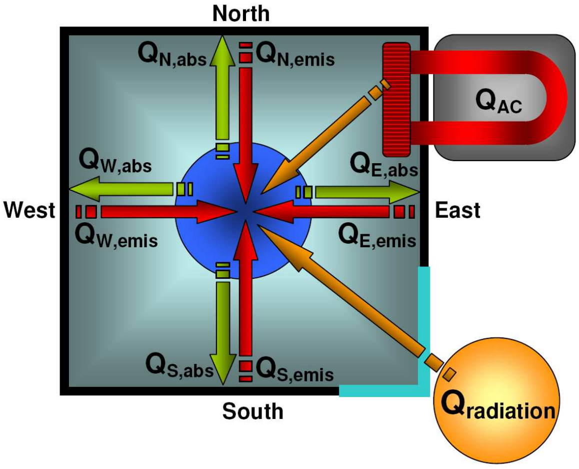


Figure 7: Schematic of the virtual heat pool implemented in the RadiationExchangeApproximated model

The solar radiation reflection and back reflection on all surfaces will affect the final amount of the solar radiation into the virtual heat pool. The mean absorption rate of a zone is calculated with the absorption ratio of all surfaces and a reflection parameter. The reflection parameter defines the mean number of internal reflections of all sun rays, before they leave the zone. The source code 2 presents the implementation of the solar reflection calculation.

```
 Q_total = sum(Q_out[i] for i in 1:12)
           + G_total*abs_mean + Q_radiator;
 SolarGain=G_total*abs_total;
abs_mean=sum(A_abs[i] for i in 1:12)/A_total;
emission_mean=sum(A_emi[i] for i in 1:12)/A_total;

 factor[1]=1;
 absorpion_factor[1]=abs_mean*factor[1];

for i in 2:1:10 loop
  factor[i]=min(max(reflections-i+1,0),1);
  abs_factor[i]=(1-sum(abs_factor[x] for x in 1:i-1))
                 * abs_mean*factor[i];
end for;

 abs_total=sum(abs_factor);

for i in 1:1:12 loop
  if A[i]<eps then
    Q_out[i] =  0
  else
    Q_out[i] =(emission[i]*sigma*A[i]*T[i]^4);
  end if;
  Q_in[i] = Q_total/A_total*A[i]*abs[i]/abs_mean;
  A_abs[i]=A[i]*abs[i];
  A_emi[i]=A[i]*emission[i];
end for;
```

Source Code 2: Modelica code of the solar refection.

This virtual heat pool method is an approximation of the exact calculation of the heat radiation exchange. To prove the HumanComfort approach, the model was validated against the EnergyPlus program, which uses the Hottel Grey Interchange Method. This method is an exact formulation of radiant heat transfer within an enclosure of grey diffuse surfaces [5].

The validation shows that the surface and mean radiant temperatures, calculated with the HumanComfort library, are within acceptable limits.

With regard to an annual simulation of a complex building, the approximated radiation model of the HumanComfort library provides a fast and sufficient method to determine the heat radiation exchange.

Control volume with energy and mass balance. The HumanComfort library provides two control volumes. A ControlVolumeAir model for the indoor air with heat and flow ports, using the Modelica_Media moist air media model, and the ControlVolumeWater model for water flows in heating cycles, fitted to improve incompressible flows.

Multi-layer wall model. The wall models considers the heat conduction and convection at the in- and outside. The internal and external radiation absorption as well as the internal heat radiation will be considered. Every layer has a homogeneous mass to store thermal energy. It is also possible to define the orientation of the wall to the sun. The wall model input is a record which includes a parameter matrix. The parameters define the conductivity, thermal capacity, density, surface area, and thickness for every layer. The HumanComfort library contains severals pre-designed wall records, ready to use. In addition, it is possible to connect a heating system (e.g. floor heating) to a single layer in the wall model via heat flow connectors.

Air exchange due to pressure and density differences between adjacent rooms with doors or other openings. The WallFlowDensityPressure model defines the air exchange between two rooms. The mass flow rate depends on pressure and density differences and the size of the opening (see fig. 8). The opening size can be varied during the simulation to consider realistic office building behavior.

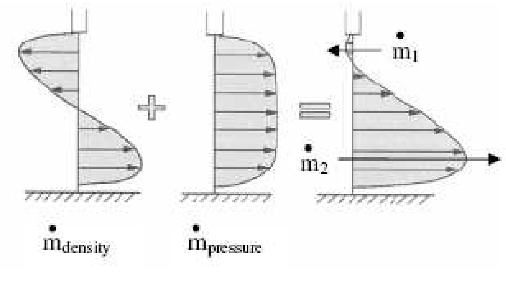


Figure 8: Influence of pressure and density for air exchange between two rooms [6]

The governing equation for the resulting mass flow rate are described as follows:

$$\dot m_{density} = \frac{W}{3}\cdot H^{\frac{3}{2}}\cdot\mu_0\cdot\sqrt{g\cdot\frac{\Delta\rho}{\rho_m}}\cdot No_{doors} \tag{18}$$

$$\dot m_{pressure} = \sqrt{\frac{\Delta p\cdot 2}{\rho_m\cdot\zeta}}\cdot\frac{W\cdot H\cdot\rho_m\cdot No_{doors}}{2} \tag{19}$$

$$\dot m_1 = \dot m_{pressure} - \dot m_{density} \tag{20}$$

$$\dot m_2 = \dot m_{pressure} + \dot m_{density} \tag{21}$$

H	=	height	$[m]$
W	=	weight	$[m]$
g	=	gravity	$[m/s^2]$
Δp	=	pressure difference	$[Pa]$
$\Delta\rho$	=	density difference	$[kg/m^3]$
ρ_m	=	mean density	$[kg/m^3]$
No_{doors}	=	number of doors	$[-]$
μ_0	=	coefficient of airflow	$[-]$
ζ	=	coefficient of friction	$[-]$

Air exchange between zone and ambient. Air exchange of a zone model and the ambient occurs due to variable indoor air pressure effected by internal temperature changes. This air exchange is inverse formulated, by setting the pressure difference to zero. The corresponding mass flow will be calculated by the model. This results in an efficient code that significantly reduces the computation time of an annual simulation.

Dynamic calculation of heat transfer coefficients for indoor and outdoor surfaces. Three calculation methods are implemented for the convective heat transfer. The first method describes the heat transfer coefficient as a constant parameter. The second method calculates the heat transfer coefficient depending on the temperature difference and the surface roughness. The third one defines the heat transfer coefficient depending on the temperature difference, the surface roughness, the wind speed, and the wind direction. The correlation of the second and third method are described in [1].

Window models with pre-designed window property records. The Window model considers two effects, the reflection of solar radiation and the thermal conductivity. The total solar gain through windows is considered by parameters based on the fraction of solar heat gain that passes through compared to either the incident solar radiation or the transmission of a reference glazing type. They are given as a decimal value in the range 0-1. The HumanComfort library use the Solar Heat Gain Coefficient (SHGC) or the G-Value.

Internal time table based heat source due to equipment and light sources. The influence of the variable operating time of the equipment is an important factor for the load and energy analyses calculations. The operating time can be set for every hour of a day. The following equation describes the implantation:

$$\dot{Q}_{equipment} = \dot{Q}_{heat\_source} \cdot f_{time\_table\_equ} \tag{22}$$

$\dot{Q}_{heat\_source}$	=	heat flow due to equipment	$[W]$
$\dot{Q}_{equipment}$	=	total heat flow due to equipment	$[W]$
$f_{time\_table\_equ}$	=	ratio of active equipment	$[-]$

Internal time table based heat- and water vapour sources due to occupants. The heat and vapour emission depends on the air temperature, the activity level, and the presence of the occupants. The InternalLoads model generates heat flow and humidity flow into the room. Eq. 23- 25 describe the correlation defined in [1].

$$MET = met \cdot A_{skin} \tag{23}$$

$$\begin{aligned}\dot{Q}_{met\_rate} = \; & 6.461927 + 0.946892 \cdot MET + 0.0000255737 \cdot MET^2 \\ & +7.139322 \cdot T_{air} - 0.0627909 \cdot T_{air} \cdot MET \\ & +0.0000589172 \cdot T_{air} \cdot MET^2 - 0.19855 \cdot T_{air}^2 \\ & +0.000940018 \cdot T_{air}^2 \cdot MET \\ & -0.00000149532 \cdot MET^2 \cdot T_{air}^2 \end{aligned} \tag{24}$$

$$\dot{Q}_{occupant} = \dot{Q}_{met\_rate} \cdot occ \cdot f_{time\_table\_occ} \tag{25}$$

met	=	spec. metabolic rate due to skin surface	$[W/m^2]$
MET	=	metabolic rate	$[W]$
A_{skin}	=	skin surface	$[m^2]$
$\dot{Q}_{met\_rate}$	=	metabolic rate	$[W]$
T_{air}	=	indoor air temperature	$[°C]$
$\dot{Q}_{occupant}$	=	total heat prod. acc. to occupants	$[W]$
occ	=	number of occupants	$[-]$
$f_{time\_table\_occ}$	=	ratio of present occupants	$[-]$

[7] defines the vapour emission of the occupants depending on the air temperature and the activity as a matrix, see tab. 2.

Air temperature	$°C$	18	20	22	23	24	25
$\dot{m}_{vapor}$ (120 W)	g/h	35	35	40	50	60	60
$\dot{m}_{vapor}$ (190 W)	g/h	95	110	125	135	140	145
$\dot{m}_{vapor}$ (270 W)	g/h	165	185	215	225	230	240

Table 2: Vapor emission of occupants depending on temperature and activity level defined by [7]

Simplified HVAC components. The `HumanComfort` library includes simplified HVAC components defined by [8](e.g. pump and radiator), which are compatible with the Modelica_Fluid library. The radiator model consists of a control volume, a pressure drop model, and a heat transfer model. The input of the pressure drop model can be either a zeta value or a K_v value. Two different heat transfer models are available and described in the following:

1. The heat flow is calculated by a heat transfer coefficient, the heat exchange surface, and the inbound water and air temperature difference. The model is discretized to increase the precission.

$$\dot{Q} = k \cdot A \cdot (T_{water} - T_{air}) \tag{26}$$

$\dot{Q}$	=	radiator heat flow to the room	$[W]$
k	=	heat transfer coefficient	$[W/m^2 \cdot K]$
A	=	heat transfer area	$[m^2]$
T_{water}	=	water temperature	$[K]$
T_{air}	=	air temperature	$[K]$

2. The `radiator` heat flow is calculated by an approximating function, which uses a nominal characteristic of existent radiators. The input values for nominal conditions at the operating point are defined by radiator manufacturers. These values are available by using pre-designed records for severals radiator types.

$$\dot{m}_n = \frac{\dot{Q}_n}{(T_{in,n} - T_{out,n}) \cdot cp_n} \tag{27}$$

$$\Delta T_{m,n} = \frac{T_{in,n} + T_{out,n}}{2} - T_{air,n} \tag{28}$$

$$\Delta T_{max} = max\left((T_{in} - T_{air}) - \Delta T_{out,min}\right), 0) \tag{29}$$

$$\Delta T = min\left(\frac{L \cdot (T_{in,n} - T_{out,n}) \cdot (T_{in} - T_{air})}{T_{in,n} - T_{air,n}} \cdot \frac{\dot{m}_n}{\dot{m}}, \Delta T_{max}\right) \tag{30}$$

$$\Delta T_m = max\left(\frac{T_{in} + (T_{in} - \Delta T)}{2} - T_{air}, 0\right) \tag{31}$$

$$f_1 = \left(\frac{\Delta T_m}{\Delta T_{m,n}}\right)^{n_{exp}} \tag{32}$$

$$f_5 = (\frac{p_0}{p_{air}})^{0.75} \tag{33}$$

$$\dot{Q} = \dot{Q}_n \cdot \frac{L \cdot f_1}{f_2 \cdot f_3 \cdot f_4 \cdot f_5} \tag{34}$$

$\dot{Q}_n$	=	nominal heat flow	$[W]$
$\dot{m}_n$	=	nominal mass flow	$[kg/s]$
cp_n	=	nominal thermal capacity	$[J/kg \cdot K]$
$T_{in,n}$	=	nominal inbound water temperature	$[°C]$
$T_{out,n}$	=	nominal outbound water temperature	$[°C]$
$T_{air,n}$	=	nominal air temperature	$[°C]$
$\dot{Q}$	=	heat flow	$[W]$
$\dot{m}$	=	mass flow	$[kg/s]$
T_{in}	=	inbound water temperature	$[°C]$
T_{out}	=	outbound water temperature	$[°C]$
T_{air}	=	air temperature	$[°C]$
$\Delta T_{m,n}$	=	mean nominal temperature difference	$[°C]$
ΔT_{max}	=	maximal temperature difference	$[°C]$
$\Delta T_{out,min}$	=	minimal outlet temperature difference	$[°C]$
L	=	radiator length	$[m]$
n_{exp}	=	radiator exponent	$[-]$
f_1	=	excess temperature factor	$[-]$
f_2	=	temperature splay factor	$[-]$
f_3	=	correction factor - type of connection	$[-]$
f_4	=	correction factor - type of cover	$[-]$
f_5	=	correction factor - atmospheric pressure	$[-]$

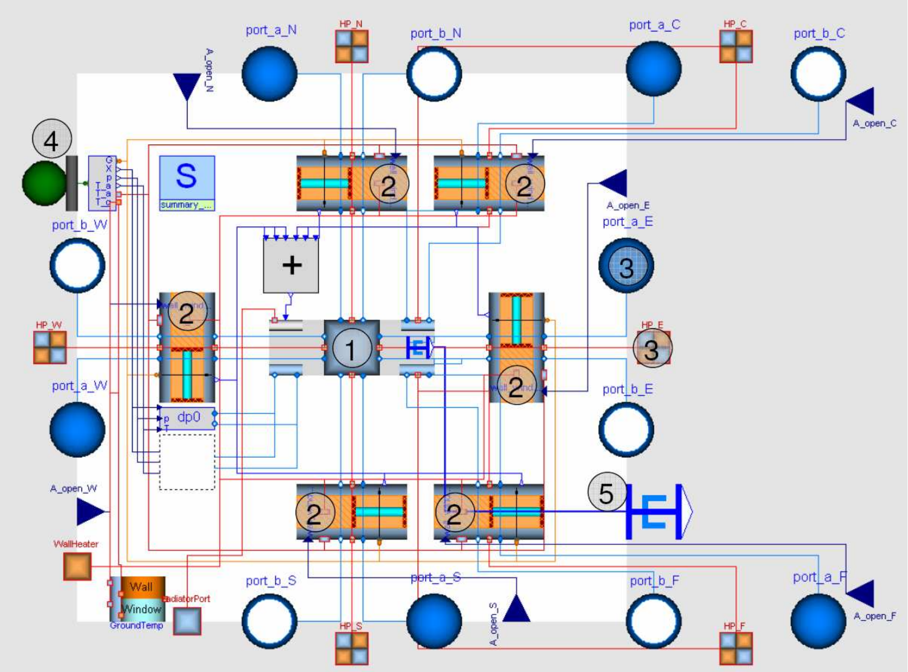


Figure 9: Diagram layer of the room model

Mean radiant temperature. Three different functions are implemented to calculate the mean radiant temperature of a cubical room with six sides.

1. The `mean_T_MRT` function sets the mean surface temperature to the mean radiant temperature [1].

$$T_{MRT} = \frac{\sum_{i=1}^{x} T_i}{x} \tag{35}$$

2. The `mean_T_to_A_MRT` function calculates surface weighted mean radiation temperatures [1].

$$T_{MRT} = \frac{\sum_{i=1}^{x} A_i \cdot T_i}{A_{Total}} \tag{36}$$

T_{MRT}	=	Mean Radiant Temperature	[°C]
T	=	Temperature	[°C]
A	=	Area	[°C]

3. The `deltaT_max_MRT` function calculates the mean radiation temperature depending on the mean surface temperature and the temperature that has the highest difference to the mean value $T_{max,dif}$. The `HumanComfort` library use $T_{max,dif}$ as an extreme value, because normally the surface that is next to the occupant would be used as the maximal value. But the model has no geometric data of the position of the occupant so the maximal possible value will be used.

$$T_{MRT} = \frac{T_{max,dif} + T_{mean}}{2} \tag{37}$$

Fig. 9 shows a main model of the zone package, the room model. It consists, amongst others, an air control volume (1), a radiation exchange model (1), internal heat and vapour sources (1), six multi-layer wall models (2), several heat and fluid connectors (3), a weather connector (4), and a human comfort adapter (5).

6 Weather

The weather package provides severals ambient conditions by reading an external text file.

- Ambient and temperature, pressure and humidity
- Global, direct and diffuse radiation to a tiled surface
- Azimuth (north based) and Zenith
- Sun position and radiation, depending on the location

7 Verification and Validation

An essential part of the library development is the verification and validation. The verification process of the `HumanComfort` models and functions was done in accordance with the implemented standards.

A comparative validation test was done by comparing the `HumanComfort` library with the EnergyPlus/DesignBuilder program [1], which is validated according to ANSI/ASHRAE Standard 140-2004.

Therefore a two room test model was established (see fig. 10).

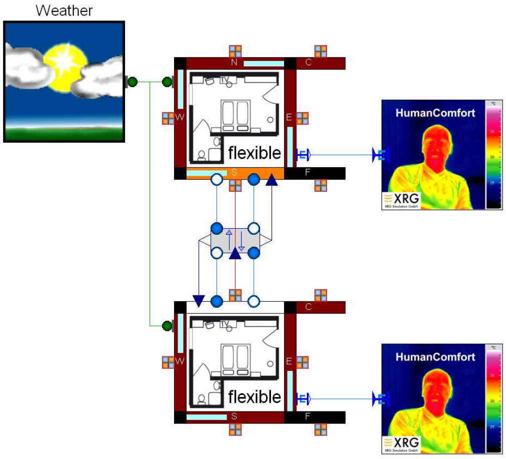


Figure 10: Graphical representation of the `HumanComfort` test model for the validation

The rooms have an opening in the adjacent wall to consider an air exchange. No heating system was implemented to generate higher temperature differences between the rooms. The rooms have one occupant with a normal metabolic rate for office activities (1met). The north room has two windows, one orientated to the west and one to the east. The south room has one window orientated to the south. The ambient condition are defined by the weather data, which references to the city of Hamburg. The graphical representation of the two room test model is shown in fig. 11.

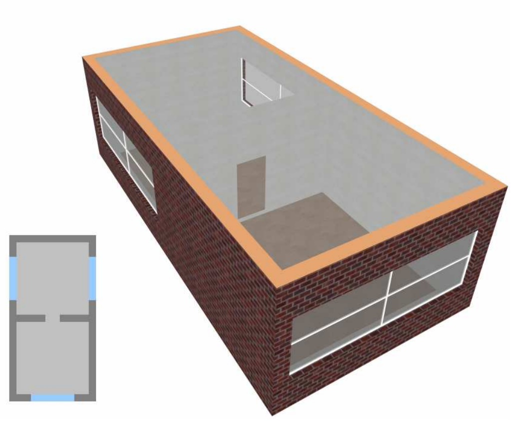


Figure 11: Graphical representation of the DesignBuilder test model for the validation

As the operative, mean radiant, and the indoor air temperature were important factors for thermal comfort, their simulation results will be discussed. Fig. 12 shows the simulation results of the north room operative and outside temperature over a time period of one year.

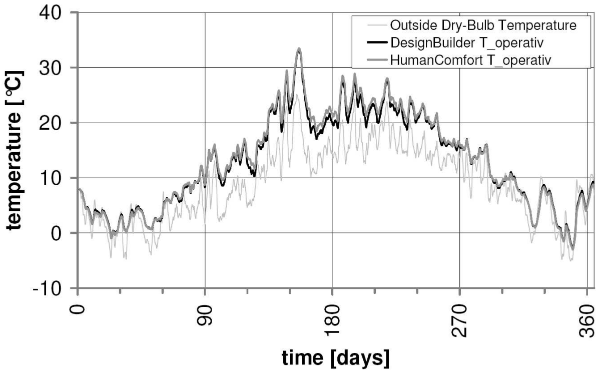


Figure 12: Simulation results of the operative and outside temperature from the `HumanComfort` library and DesignBuilder of the north room

Tab. 3 shows the mean values of the different temperatures, the deviation to each other and the mean of the absolute deviations.

mean value	HC	DB	dev.	abs. dev.
unit	°C	°C	°C	°C
indoor temperature	13.058	12.633	0.425	0.625
operative temperature	13.215	12.816	0.399	0.611
mean radiant temperature	13.372	13.000	0.372	0.620

Table 3: Selected temperatures of the north room

The maximum deviations occurs in the summer during rapidly decreasing outside temperatures. The mean deviation of the operative temperature amounts to 0.399°C with an absolute deviation of 0.611°C. The mean deviation of the indoor air temperature amounts to 0.425°C with a maximal deviation of 2.4°C at 11th of June.

Fig. 13 depicts the relative humidity of the north room during the course of the year. The occurring deviations can be explained by the fact that the relative humidity depends on the temperature.

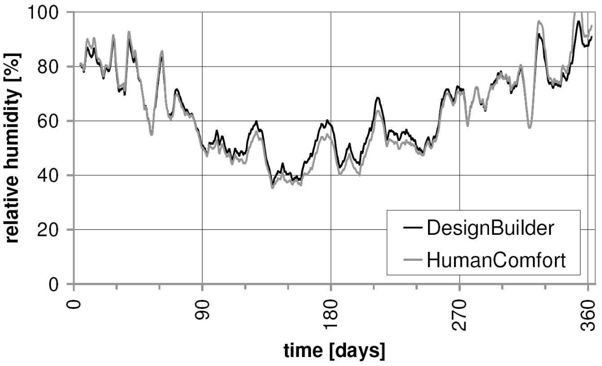


Figure 13: Simulation results of the relative humidity

The resulting PMV of the north room is shown in fig. 14. Due to the fact that the PMV depends on the temperature and relative humidity the deviations behave analogous.

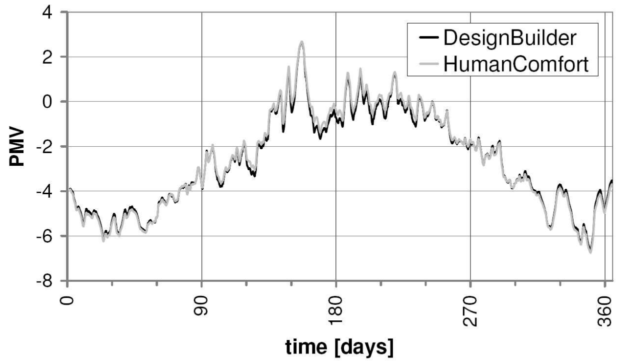


Figure 14: Simulation results of the PMV from HumanComfort library and DesignBuilder of the north room

Tab. 4 displaces the mean values of the PMV, the deviation to each other and the mean of the absolute deviations.

mean values	HC	DB	deviation	absolute dev.
PMV	-2.62	-2.67	-0.047	0.151

Table 4: Mean values and their deviations of the PMV from the north room

The comparative validation test reveals that the results predicted by the HumanComfort library are within the range of EnergyPlus/DesignBuilder.

8 Conclusion

The HumanComfort library enables the prediction of thermal comfort in zone models. All relevant characteristic numbers, described in the standards, are implemented. The validations shows that the stationary zone model is in the range with another accurate energy simulation program on the market. The library development is still going on. The complete implementation of the mobile application will be finished until the end of the EuroSysLib-D project.

Acknowledgement

The HumanComfort library is developed within the European research project EuroSysLib-D funded by German Federal Ministry of Education and Research (promotional reference 01IS07022B). The project has been started in October 2007 and will end in March 2010. The authors bear the sole responsibility for the content of this publication.

SPONSORED BY THE

Federal Ministry of Education and Research

References

[1] US Department of Energy. *EnergyPlus - Engineering Reference*, June 2007.

[2] DIN EN ISO 7730. Ergonomics of the thermal environment - Analytical determination and interpretation of thermal comfort using calculation of the PMV and PPD indices and local thermal comfort criteria. Beuth Verlag GmbH, 2005.

[3] ASHRAE Standard 55-2004. Thermal Environmental Conditions for Human Occupancy, January 2004.

[4] A.C. van der Linden A.C. Boerstra W. Plokker A.K. Raue, S.R. Kurvers. Dutch Thermal Comfort Guidelines from Weighted Temperature Exceeding Hours Towards Adaptive Temperature Limit, 2004.

[5] Strand Pedersen. *Modeling Radiant Systems in an Integrated Heat Balance Based Energy Simulation Program*. ASHRAE Transactions, 2002.

[6] Frank Meyer zur Heide. Energieeinsparung und Komfortgewinn. CCI Print, 2005.

[7] Schramek Recknagel, Sprenger. *Taschenbuch für Heizung und Klimatechnik*. Oldenbourg Industrieverlag, 2005.

[8] Bernd Glück. *Strahlungsheizung - Theorie und Praxis*. C.F. Müller Verlag Heidelberg, 1982.

Integrated Thermal Management Simulation: Evaluating the Effect of Underhood Recirculation Flows on AC-System Performance

Zhu Wang[1], Kristian Tuszynski[2], Hubertus Tummescheit[2], Ales Alajbegovic[1]

1) Exa Corporation
Livonia, MI 48152, USA

2) Modelon AB
Ideon Science Park, Lund, Sweden

ales@exa.com

kristian.tuszynski@modelon.se

Abstract

Presented is a model for the simulation of the interaction between the airflow and the AC-system. Demonstrated is 1) a successful coupling of flow solver (PowerFLOW 4.1) with the Modelica-based Dymola system tool and the AirConditioning Library, making use of the previously validated underhood-environment, and 2) the importance of a careful design of the underhood flow for the AC-system performance. The validity of the developed simulation capability is tested by successful comparison with the available experimental data for the condenser at the given operating conditions. Shown is the potential for the analysis of the flow details and structures affecting the condenser performance like airflow recirculation.

Keywords: *HVAC simulation; underhood flow; simulator coupling*

1 Introduction

Vehicle can be seen as a system composed of multiple sub-systems. One such subsystem is the AC-system which is used to maintain passenger comfort. A key component of the AC-system is the condenser. The condenser is used to condensate the refrigerant by extracting the heat from the refrigerant to the airflow. For the operation of the AC-system is very important that the refrigerant condensation is completed at the outlet from the condenser. It is possible that this does not occur in cases of adverse airflow conditions in the proximity of the condenser. For example, airflow recirculation can cause local temperature peaks and consequent reduction of the condensation rate.

The ability to predict such behavior and if possible avoid it is of great importance for the design of vehicle cooling airflow performance. Srinivasan *et al.* [1] presented an omni-tree meshing technology for rapid mesh generation for Navier-Stokes solvers. The cooling airflow simulations using Navier-Stokes solvers coupled with heat exchanger calculations for passenger cars were presented by [1], [2], [3], and for trucks by [4], [5], [6].

Fortunato *et al.* [7] used the Lattice-Boltzmann Equation (LBE) solver for the cold flow simulation over the entire car including the underhood and a Navier-Stokes solver for the underhood flow. The velocity field on the entrance surfaces into the underhood area calculated by the LBE solver was used as the inlet boundary condition to the Navier-Stokes solver calculation. Fully coupled simulations between the LBE solver and a system simulation tool for the heat exchanger were presented by Alajbegovic *et al.* [9], [10]. A detailed validation of the cooling package using coupling with the AMESim system simulation tool was shown in [11]. The simulation capability presented here used the same experimental data for the validation of the developed methodology.

2 Simulation Methodology

The presented simulation methodology is based on coupled simulations between the flow and system solvers. Therefore, the following three major simulation components are:

1. Flow modeling with the Lattice-Boltzmann Equation solver
2. Dymola AC model

DOI: 10.3384/ecp09430098

3. Automatic coupling between the flow solver and Dymola

2.1 FLOW MODELING

The flow simulation is performed using the Lattice-Boltzmann Equation (LBE) based solver. The Lattice-Boltzmann solvers are numerically very efficient, accurate and robust. The numerical efficiency allows handling of lattices with very large element (or voxel) counts. It is quite normal to have 100 million voxels for a full vehicle analysis. This is required to resolve an as large as possible span of turbulence scales and in this way increase the accuracy of the predictions.

Properties of the Boltzmann equation allow for an improved treatment of fluid interactions with the wall surface. Surface elements (or surfels), are designed as active elements that interact with the neighboring lattice elements. The combination of both large lattices and dynamic surface treatment allow accurate representation of surfaces without the need for geometry simplification.

Earlier use of Lattice-Boltzmann equation in fluid flow simulations was done by Frisch, Hasslacher, & Pomeneau [12]. After that, significant efforts were made to develop Lattice-Boltzmann flow solver [13], [14], [15]. Turbulence effects are modeled using a modified k-ε model based on the original RNG formulation [16], [17]. This LBE based description of turbulent fluctuation carries flow history and stream information, and contains high order terms to account for the nonlinearity of the Reynolds stress [18]. This is contrasted with typical Navier-Stokes solvers, which tend to use the conventional linear eddy viscosity based on the Reynolds stress closure models. Turbulence and temperature equations are solved on the same lattice using a modified Lax-Wendroff-like explicit time marching finite difference scheme.

Simulations presented in this work were performed using the flow solver described in the following references [19], [20], [21], [22], [23].

2.2 DYMOLA MODELING

Dymola (Dynamic Modeling Laboratory) is a multi-engineering software package suitable for modeling of a large variety of physical and engineering systems, such as mechanical, electrical, hydraulic, chemical, thermodynamic, control, etc., see [24]. Based on the open modeling language Modelica [25], [26], Dymola supports hierarchical model composition, libraries of truly reusable components, domain-specific connectors taking care of mass- and energy conservation on the system level and non-causal connections. A typical representation of the graphic user interface is shown in Figure 1.

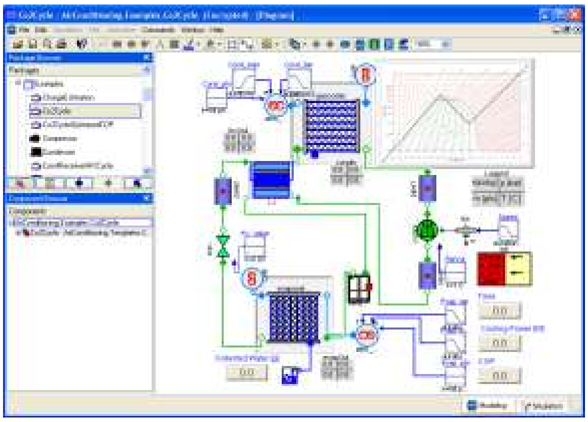


Figure 1. Dymola Graphic User Interface

Model libraries are available in many engineering domains. The heat exchanger models used in this paper are from the AirConditioning Library developped by Modelon AB (see [27], [28]). The Library has been validated in many industrial projects (e.g. [28]) and before commercialization in many years of research use at DaimlerChrysler and TU Hamburg-Harburg. The AirConditioning library was selected as the preferred AC- systems tool by Audi, BMW, Daimler and Volkswagen and their suppliers in 2004. The condenser model used in this paper is built based on the latest AirConditioning Library 1.7 (ACL). An example of the user interface for a horizontal flow microchannel flat tube condenser from ACL is shown in Figure 2.

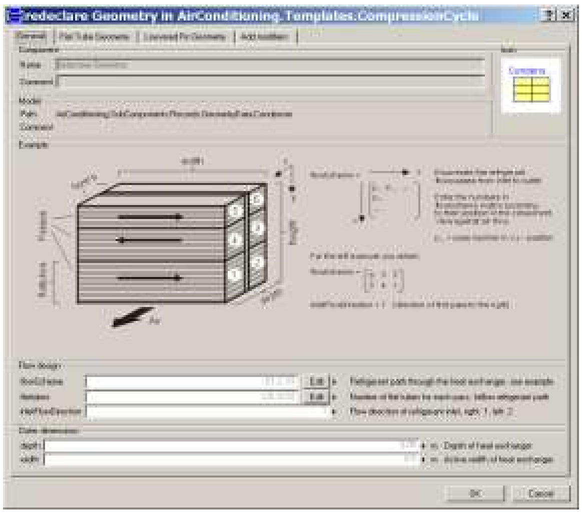


Figure 2. Dymola condenser model from the AC library

2.3 COUPLING PROCESS

The PowerFLOW and Dymola two-way coupling is implemented via an automated process using scripts. The coupling process involves the extraction and exchange of flow data distributions on each of the heat exchanger inlet and outlet surfaces. PowerFLOW provides the cooling air velocity and temperature field, and Dymola calculates the distribution of the heat source and provides the heat flux distribution back to PowerFLOW. The coupling process is also described in Figure 3.

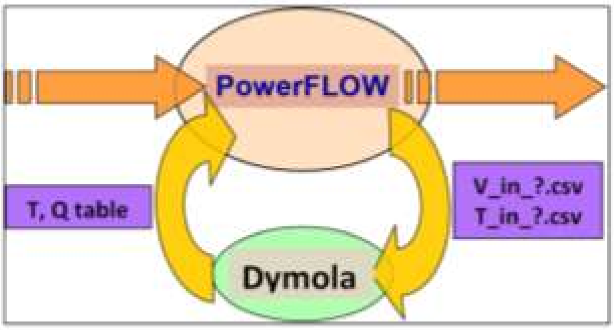


Figure 3. General two-way coupling scheme

In further detail, the coupling process between the flow solver and Dymola is realized by the table reread functionality of PowerFLOW shown in Figure 4, and by polling for new data on the Dymola side. PowerFLOW treats the heat exchangers as porous media with heat release per volume, whereas in the Dymola and the ACL, the interaction is through discretized surfaces represented as connector variables.

Before flow simulation starts, the Dymola program “dymosim” is executed through a batch process. At each coupling step during the simulation, a coupling script is executed. The coupling script reads the description of each porous medium in the *pmspec* file, which contains the support information for the data extraction performed by the program *exatool.* Once the data are extracted (*.csv) for each heat exchanger, “dymosim” will read-in the *.csv* files and then generate the heat rejection table file for each heat exchanger. The flow solver calculates the flow field again once all heat exchangers table files are available. The flow solver divides these two-dimensional heat fluxes data by the thickness of the porous media to get the distribution of the volumetric heat. Once the coupling is completed, the flow solver automatically rereads the new tables. This process continues until convergence is reached for each time step.

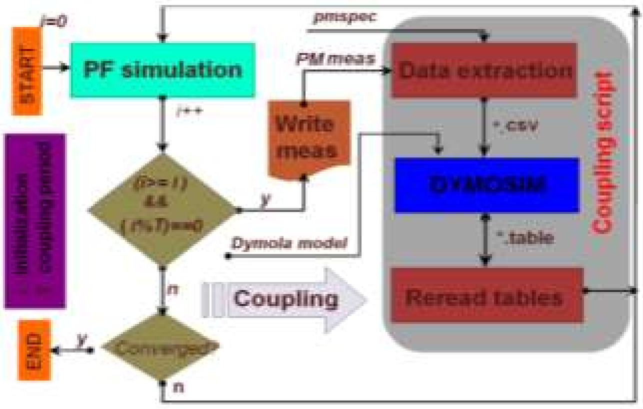


Figure 4. Flow chart of the coupling process

The output from the Dymola heat exchanger models does not require special treatment due to the uniform grid discretization. The velocity and temperature field at the inlet plane of the heat exchanger are mapped to Dymola grid uniformly.

2.4 Validation

The vehicle used for the validation of the coupling between the flow solver and the Modelica based simulation of the HVAC system was the Renault Scenic II, Figure 5. Shown are the vehicle, its underhood geometry and cooling package. Flow predictions provide air velocity and temperature distributions in the entire domain including the engine compartment and vehicle exterior. The details of the geometry preparation, case setup and boundary conditions can be referred to in [11].

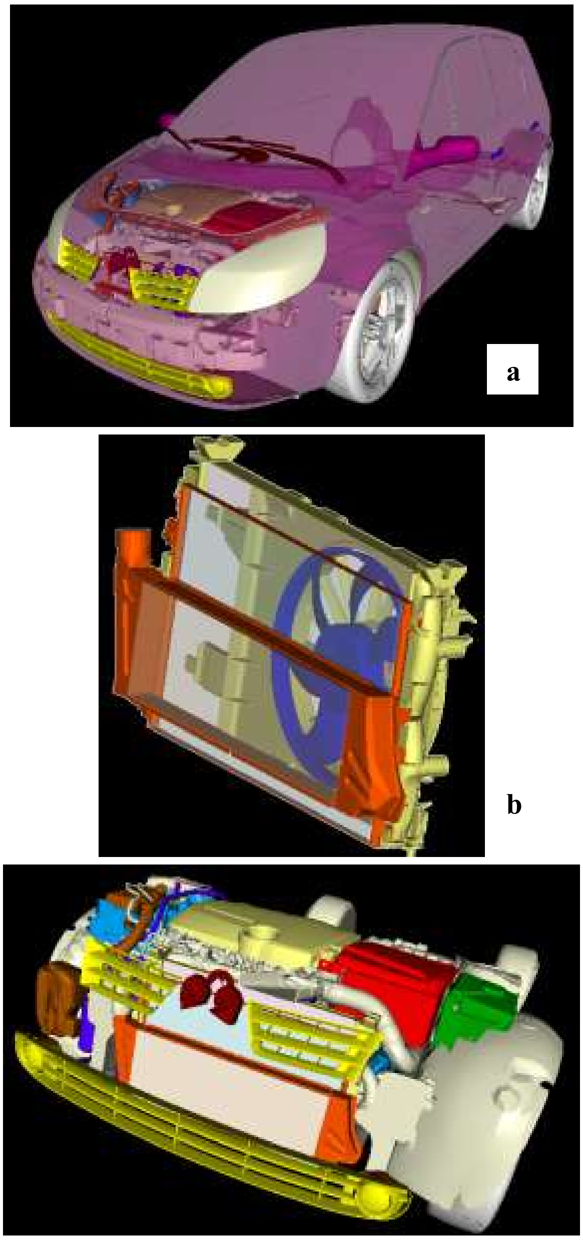


Figure 5. a.) Vehicle geometry, b.) Cooling package details and c.) Underhood geometry

Vehicle placement in the thermal wind tunnel and geometry with the nozzle profile is shown in Figure 6. Validation was focused on evaluating the condenser performance. The predicted values were compared to the measurements.

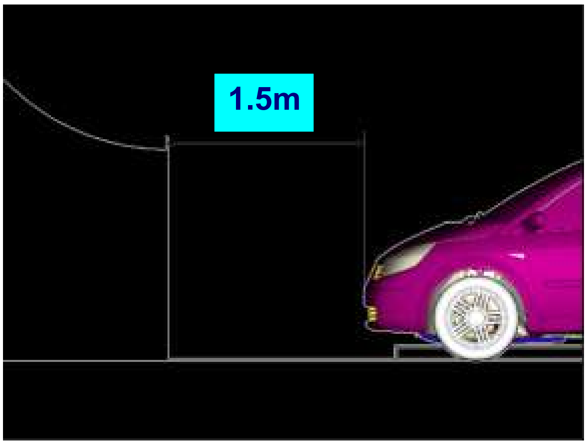


Figure 6. Vehicle in Thermal Wind Tunnel

2.5 TESTING CONDITIONS

Vehicle cooling package performance was measured in the thermal wind tunnel at a velocity of 40 km/h. The distance between the nozzle exit and the bumper of the vehicle was 1.5 m. In order to prevent flow separations in the front region of the underbody, a boundary layer suction system was used extracting 0.78 m^3/s of air from the suction section of the wind tunnel and returning the same amount of volume flow back to the wind tunnel downstream of the vehicle as indicated in Figure 7.

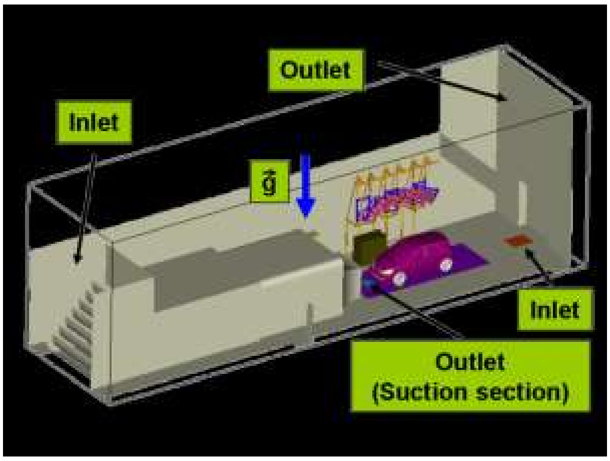


Figure 7. Overview of thermal wind tunnel

WIND	Air Velocity	km/h	*39.7*
	Air Temperature	°C	*44.8*
FAN	Voltage	V	13.0
	Intensity	A	32.5
CONDENSER	Inlet Pressure	bar	*25.0*
	Outlet Pressure	bar	23.6
	Inlet Temperature	°C	*106.3*
	Outlet Temperature	°C	67.46
	Volumetric Flow Rate	l/h	*136.8*
	Heat Rejection	kW	6.4

Table 1. Simulation conditions

Heat exchanger conditions used during the experiments are summarized in Table 1. The numbers in denoted with italic font are used as input conditions to the simulations.

2.6 CONDENSER MODEL

As mentioned in the previous section, the condenser model is based on Dymola air conditioning library 1.7 and it could be one component of the HVAC system within a complete circuit loop. The condenser model utilizes the details of the condenser geometry, while the charge air cooler and radiator uses the Log Mean Temperature Difference (LMTD) approach to calculate the outlet air and outlet refrigerant temperatures based on the inlet temperatures, mass flow rates and heat transfer coefficients. The condenser consists of four passes.

The Modelica schematic of AC-system is shown in Figure 8.

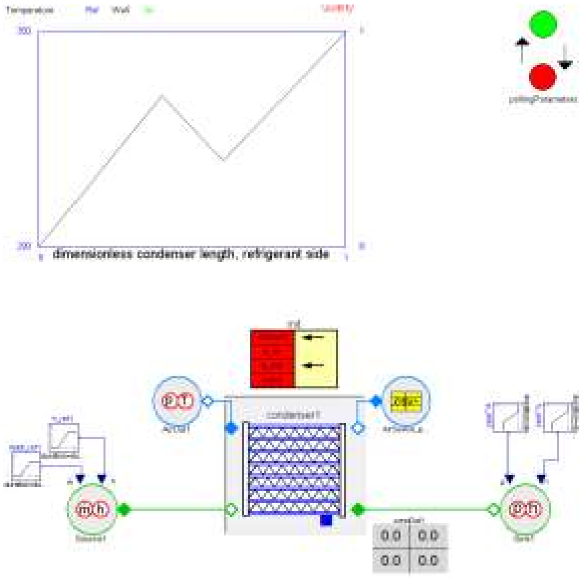


Figure 8. Dymola condenser schematic

3 RESULTS AND DISCUSSION

The results show good agreement between the simulations and available experimental data. The temperature field in front of the condenser is shown in Figure 9, and behind it in Figure 10. Figure 11 shows the velocity and temperature distribution at the fan middle plane. It can be observed the heating of the air by the condenser through the coupling between the flow solver and Dymola.

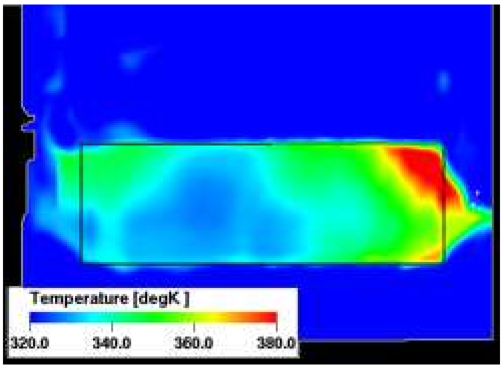


Figure 9. Temperature distribution in front of the condenser

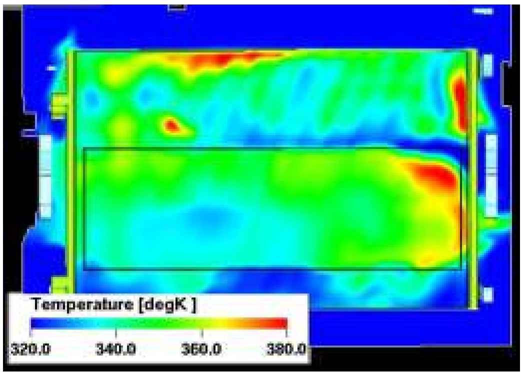


Figure 10. Temperature distribution behind condenser

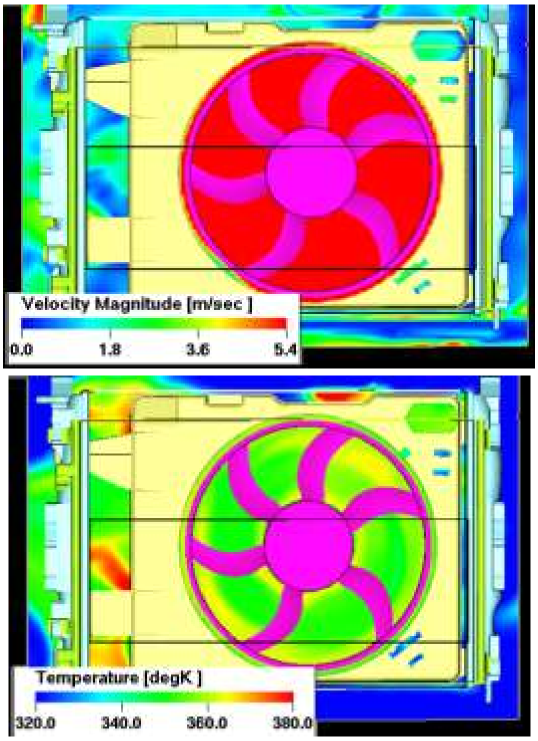


Figure 11. Velocity and temperature distribution at the fan middle plane

The details of the predicted flow velocity field within the underhood region are shown in Figure 12. Shown is the velocity field magnitude on several critical vertical planes. These locations are top left grille, fan center, center vertical plane and top right grille. High velocity at grille inlets and fan area are identified. Due to the complicated geometry, the flow within the underhood region is very complex. Recirculation, separation and local acceleration can be observed to occur almost everywhere.

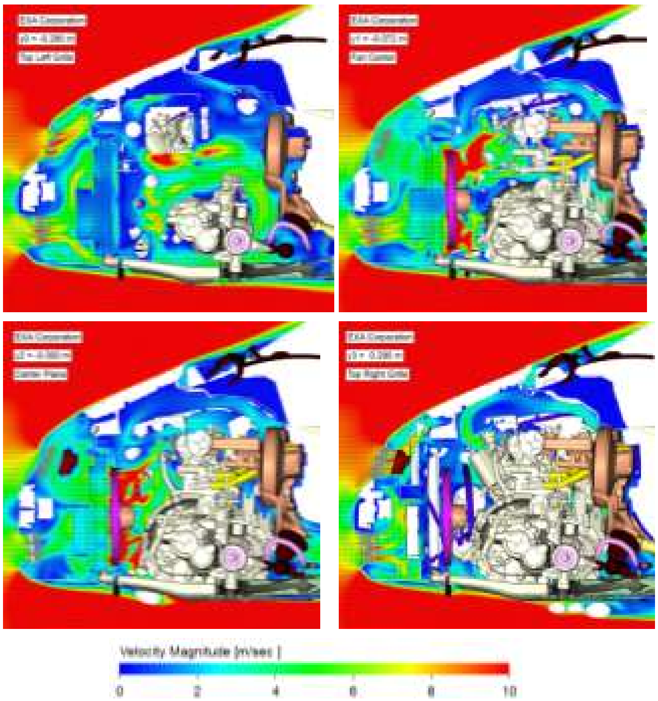


Figure 12. Velocity on y-plane in the underhood region

The details of the temperature field in the underhood region are shown in Figure 13. Shown are both streamlines and temperature field for the same vertical planes as in Figure 12. The aerodynamic and thermal behavior of cooling air can be easily captured, which can be used for the optimization of the underhood components.

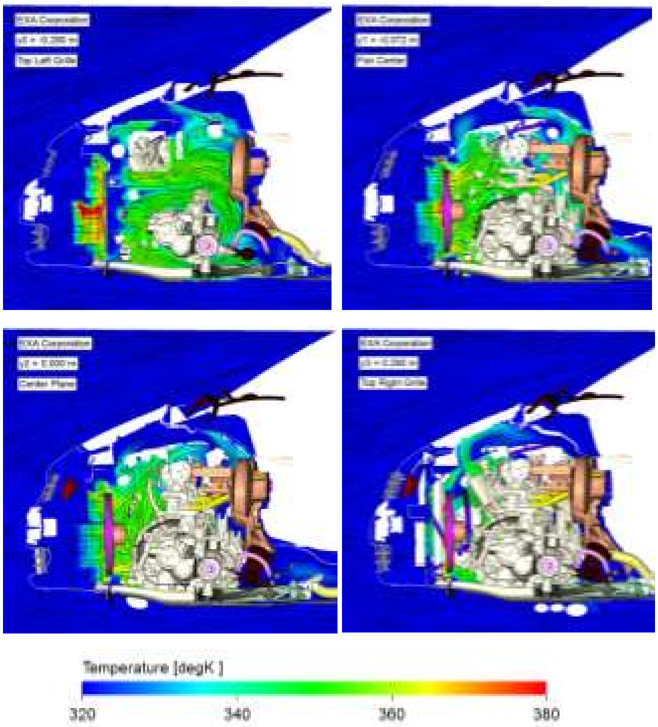


Figure 13. Temperature field on the y-plane in the underhood region

Figure 14 and Figure 15 show velocity and temperature distributions separately on several horizontal planes. These locations are bottom grille, fan center and top grilles.

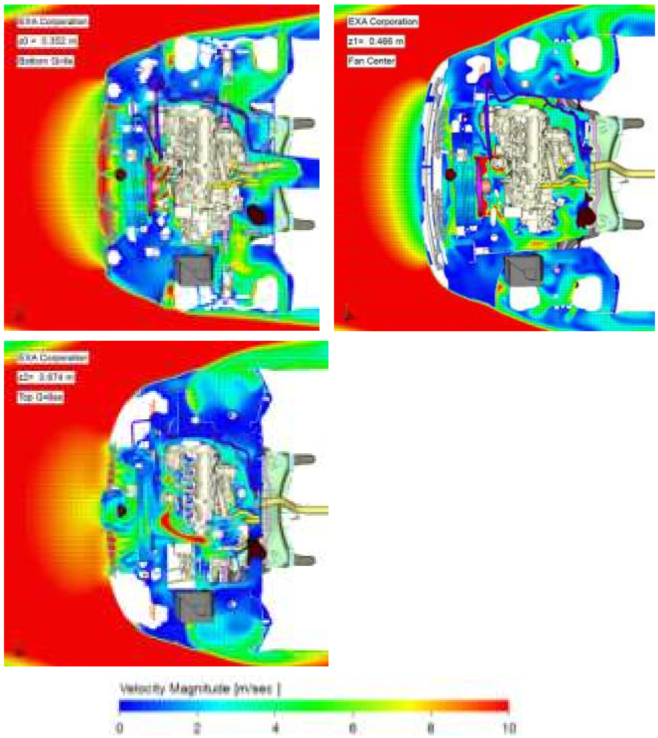


Figure 14. Velocity on the z-plane in the underhood region

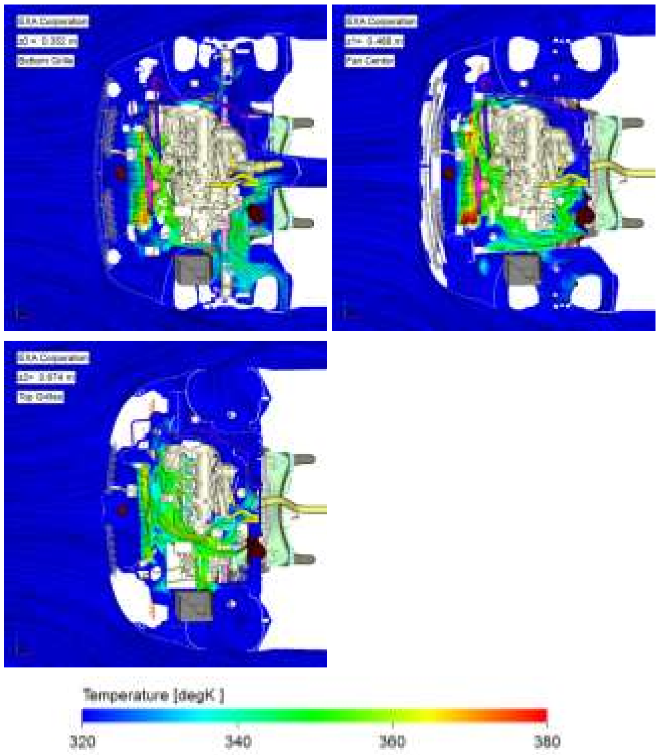


Figure 15. Temperature on the z-plane in the underhood region

The obtained results demonstrate the overall capability of the presented simulation approach where both the flow field parameters and the performance of the cooling package are evaluated.

Figure 16 shows the Dymola output for the cooling mass flow rate and the heat rejection for the condenser.

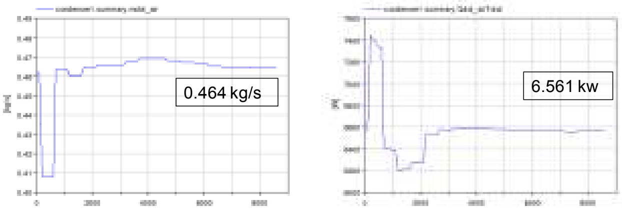


Figure 16. Dymola output: Cooling air mass flow rate and heat rejection for the condenser

Table 2 presents simulation results compared to the measured values. The predicted heat rejection for the condenser is very close to the measured value.

HEAT REJECTION [kW]	Condenser
EXPERIMENT	6.40
SIMULATION	6.56

Table 2. Heat rejection comparison

The averaged temperatures at the monitor locations use the fluid measurements close to 2 cm window at the corresponding locations. Figure 17 shows these locations.

The averaged fluid temperatures at the monitoring locations before and after the condenser are summarized in Table 3. Excellent correlation with the experimental data is obtained for P2R, P3FLT, P3FRT and P3FLB.

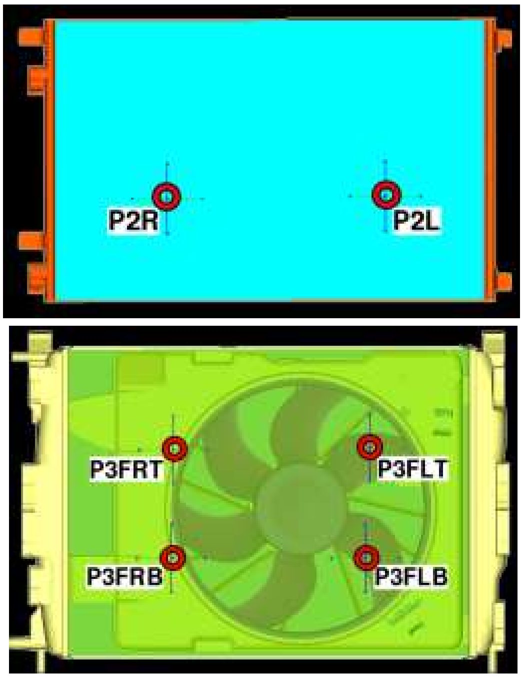


Figure 17. Locations and descriptions of the thermocouples before and after the condenser (front view)

TEMP [°C]	P2L	P2R	P3FLT	P3FRT	P3FLB	P3FRB
EXP.	78.0	54.5	58.0	71.7	67.3	72.4
PRE.	65.8	56.4	62.1	74.0	71.2	61.2

Table 3. Comparison between experimental and averaged predicted fluid temperature

3.1 Study of Recirculation Airflow Effects

To understand the interaction of the AC-System with the recirculating flow, a true coupling of a system tool and the flow solver is necessary. For the given heat exchanger case, the system tool actually discretizes the air flow across the heat exchanger face in both directions and takes inhomogeneous flows and temperatures into account. For the closed thermodynamic cycle, the hot recirculated air raises the pressure, which in turn leads to a higher temperature after the AC condenser. This positive feedback loop can either trip the high-pressure cut-off switch for the AC-system, or leads at best to severe performance degradation of the AC-circuit. The AC-system is potentially run far away from standard conditions, close to the critical pressure in the condenser, until the compressor is switched off or regulated to minimum flow.

Figure 18 and Figure 19 shows the streamline colored by the velocity magnitude around the cooling packages. The cooling airflow in front of condenser is highly non-uniform. The air streams coming through the top grilles and bottom grilles mix with each other. Recirculation of cooling airflow can be observed right before top and bottom of the condenser. With the help of the fan nozzle, no obvious recirculating hot air downstream of the fan flows back into upstream of heat exchangers.

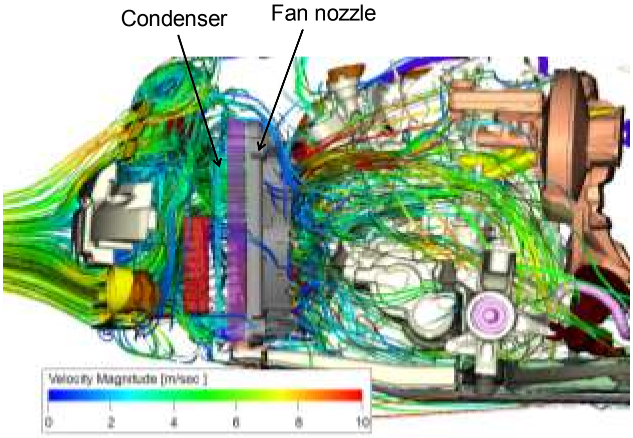


Figure 18. Streamline around heat exchangers (side view)

In Figure 19, a low velocity zone is identified at the top right corner of the condenser. This is due to the blockage of the fan nozzle at the right side. This low cooling air velocity zone causes limited local back flow and local high temperature, as can be seen clearly in Figure 20.

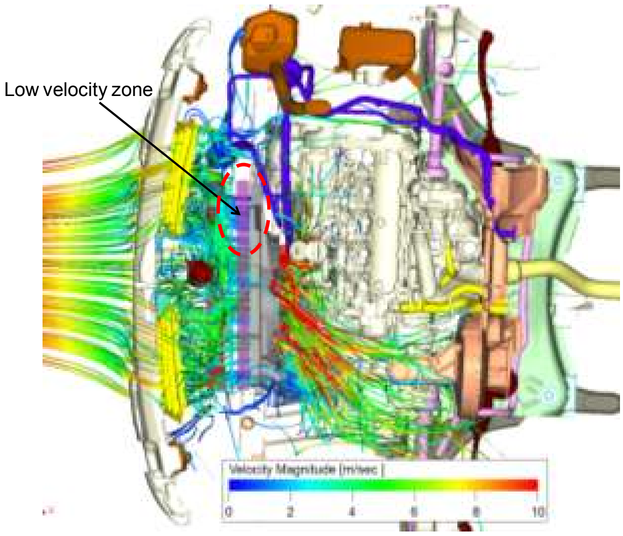


Figure 19. Streamline around heat exchangers (top view)

With the help of a post-processing tool, the approach presented in this paper can easily help analyze the flow structure and thermal behavior of underhood cooling air. Small details like recirculation, separation and blockage can be quickly observed. This can help quick optimization of the underhood components.

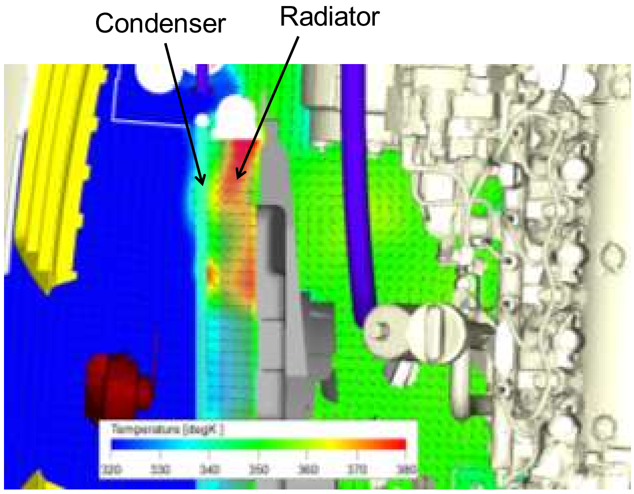


Figure 20. Local high temperature and back flow

4 Conclusions

Presented was the simulation methodology for the AC-system that consists of the flow solver PowerFLOW coupled with the Modelica based system simulation tool Dymola. Shown were the details of the coupling procedure that enables two way coupling between the three-dimensional flow effects and the condenser. The results demonstrate that the tightly coupled interaction of the underhood air flow with the AC-system can predict the measured heat rejection while allowing detailed analysis of the flow and temperature fields. The developed simulation methodology can be used for the studies of the vehicle air cooling performance and its optimization.

References

[1] K. Srinivasan, Z.J. Wang, W. Yuan, R. Sun, “Vehicle thermal management simulation using a rapid omni-tree based adaptive Cartesian mesh generation methodology,” HT-FED2004-56748, 204 ASME Heat Transfer/Fluids Engineering Summer Conference, July 11-15, Charlotte, North Carolina, USA.

[2] B. Uhl, F. Brotz, J. Fauser, U. Krueger, “Development of engine cooling systems by coupling CFD simulation and heat exchanger analysis programs,” SAE 2001-01-1695.

[3] G. Seider, F. Bet, T. Heid, U. Hess, T. Klein, and J. Sauer, “A numerical simulation strategy for complex automotive cooling systems,“ SAE 2001-01-1722.

[4] H. Knaus, C. Ottosson, F. Brotz, W. Kuehnel, ”Cooling module performance investigation by means of underhood simulation,“ SAE 2005-01-2013.

[5] T.P. Nobel, S.K. Jain, “A multidimensional approach to truck underhood thermal management,” SAE 2001-02-2785.

[6] C.L.R.Siqueira, P. Vatavuk, M. Jokuszies, M.R. Lima, “Numerical simulation of a truck underhood flow,” SAE 2002-01-3453.

[7] E.A. Costa, “CFD approach on underhood thermal management of passenger cars and trucks,” SAE 2003-01-3577.

[8] F. Fortunato, F. Damiano, L. Di Matteo, P.Oliva, “Underhood cooling simulation for development of new vehicles,” SAE 2005-01-2046.

[9] A. Alajbegovic, R. Sengupta, W. Jansen: "Cooling Airflow Simulation for Passenger Cars using Detailed Underhood Geometry," SAE 2006-01-3478, SAE Conference, Chicago, October 2006

[10] B. Xu, A. Konstantinov, J. Amodeo, W. Jansen, A. Alajbegovic, "Simulation of Cooling Airflow under Different Driving Conditions," SAE 2007-01-0766, SAE World Congress, Detroit, April 2007

[11] S. Brémont, G. Servera, E. Fares, J. Abanto, A. Alajbegovic: "Experimental Investigation and Numerical Validation of Cooling Airflows of a Realistic Vehicle," 6th FKFS Conference, Stuttgart, Germany, October 2007

[12] U. Frisch, B. Hasslacher, and Y. Pomeneau, “Lattice gas automata for the Navier-Stokes equation,” *Physical Review Letters*, 56:1505-1508, 1986.

[13] S. Chen and G. D. Doolen, “Lattice Boltzmann method for fluid flows”, *Annual Review of Fluid Mechanics*, 30:329-364, 1998.

[14] S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*, Series Numerical Mathematics and Scientific Computation, Clarendon Press, Oxford, 2001.

[15] D. d'Humieres, P. Lallemand and Y. H. Quian, "Lattice BGK models for Navier-Stokes equations," *Europhysics Letters*, 17(6):479-484, 1992.

[16] V. Yakhot, and S.A., Orszag, "Renormalization Group Analysis of Turbulence. I. Basic Theory" *J. Sci. Comput.*, 1(2), 3-51, 1986.

[17] V. Yakhot, V., S.A. Orszag, S. Thangam, T. Gatski, and C. Speziale, "Development of turbulence models for shear flows by a double expansion technique," *Phys. Fluids A*, 4 (7), 1510-1520, 1992.

[18] H. Chen, S.A. Orszag, I. Staroselsky, and S. Succi, "Expanded Analogy between Boltzmann Kinetic Theory of Fluid and Turbulence", *J. Fluid Mech.*, 519: 307-314, 2004.

[19] H. Chen, "H-theorem and generalized semi-detailed balance conditions for lattice gas systems," *J. Stat. Phys.* 81:347-359, 1995.

[20] H. Chen and C. Teixeira, "H-Theorem and origins of instability in thermal lattice Boltzmann models," *Comp. Phys. Commun*ication, 129:21-31, 2000.

[21] H. Chen and R. Zhang, "Lattice Boltzmann method for simulations of liquid-vapor thermal flows," *Phys. Rev.* E67(6): Art. no. 066711 Part 2, 2003.

[22] C. M. Teixeira, "Incorporating turbulence models into the lattice-Boltzmann method," *Int. J. Modern Physics C*, 9(8):1159-1175, 1998.

[23] PowerFLOW User's Guide, Release 4.1, Exa Corporation, Boston, Massachusetts, 2007.

[24] Dymola User's Guide, Release 7.2, Dynasim AB, Lund, Sweden, 2009.

[25] www.Modelica.org, accessed August 2009

[26] S.E., Mattsson, H. Elmqvist, M. Otter, "Physical system modeling with Modelica," *Control Engineering Practice*, **6**, 501-510, 1998.

[27] D. Limperich, M. Braun, G., Schmitz, and K. Prölß, "System Simulation of Automotive Refrigeration Cycles," 4th International Modelica Conference, Hamburg, 2005.

[28] H. Tummescheit and D. Limperich, "The AirConditioning library for simulation of advanced vehicle A/C systems," VTMS8: Vehicle Thermal Management Systems Conference & Exhibition, Nottingham, 2007.

Investigating the Multibody Dynamics of the Complete Powertrain System

Alessandro Picarelli   Mike Dempsey
Claytex Services Ltd.
Edmund House, Rugby Road, Leamington Spa, CV32 6EL, UK
alessandro.picarelli@claytex.com   mike.dempsey@claytex.com

Abstract

The specifications and integration of two new Modelica libraries is presented: The Powertrain Dynamics (PTDynamics) and the Engines libraries. The libraries enable the simulation and modelling of powertrain systems including their fluid dynamic, pollutant emission, mechanical and thermal performances in one simulation environment (Dymola), utilising the object orientated modelling language Modelica.

Two variants of the Engines library are presented: a Mean Value variant (MVEL) and a Crank Angle Resolved variant (CAREL).

Both the PTDynamics and Engines libraries make use of a new approach to modelling the mechanics that captures the full MultiBody effects of the Powertrain system without the computational cost of using the standard Modelica MultiBody library.

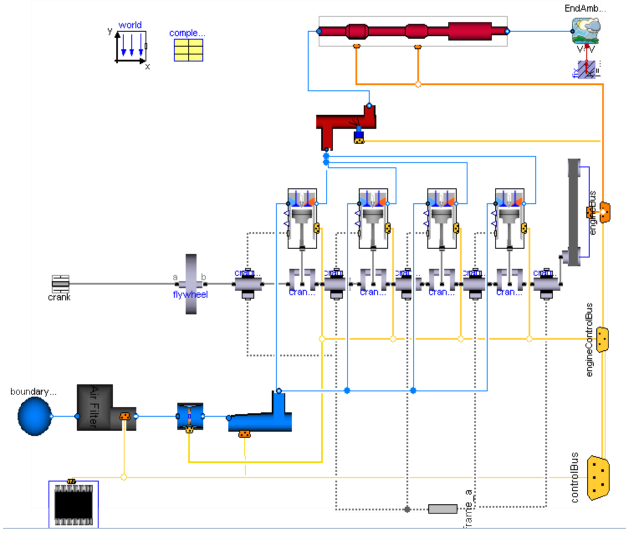
1 Introduction

Development of powertrain systems and components is an ongoing and relentless activity in the automotive industry. The ability to reduce the number of expensive prototypes, engineering costs and development time is an attractive feature of CAE simulation tools.

Simulation time performance is an important feature particularly in real-time applications such as for SIL (Software-in-the-loop) and HIL (Hardware-in-the-loop) experiments. For this particular reason, the libraries presented in this paper have been designed with efficiency in mind and show reduced model complexity when compared to analogous system models built with the Modelica standard library components whilst retaining equal or improved levels of accuracy.

2 Engine Library

The Engine library is capable of modelling both Spark Ignition (SI) and Compression Ignition engines and is split into two variants with different levels of fidelity. Both levels of the Engines library have been designed to work with common engine architecture templates. This enables quick model setup and ensures a consistent layout for a variety of engine architectures.



Example of a multi-cylinder MVEM layout

The mechanical components are modelled using a new Rotational3D library described in section 3 and the Fluids models are based on the new Modelica Fluids library [1].

2.1 Mean Value Engine Library (MVEL)

This level of the library is capable of predicting cycle-averaged values for engine torque, thermal effects and emissions. The methods for predicting the engine torque and emissions are map based and/or neural network based.

DOI: 10.3384/ecp09430085

The library is of particular use for investigating different control algorithms and their effect on the engine transient response. This variant of the engine library is also suited to driveability analysis where the transient torque output of the engine is fed through the transmission and reacted into the transmission and engine mounts. Furthermore, engine models using this library are capable of running in real time making it suitable for SIL and HIL testing of control systems.

Mass flow rate through the engine cylinders is computed by means of [2]:

$$\dot{m}_{ap}(n, p_i) = \frac{V_d}{120RT}(s_i p_i + y_i)\frac{n}{1000}$$

Where:

m is the mass of fluid within the cylinder, V_d is the volumetric displacement of each cylinder
R is the gas constant
T is the fluid temperature (K)
p_i is the intake manifold pressure (bar)
n is the engine speed in *rpm*.

The above equation relates the mass flow rate through each cylinder to the engine speed and intake manifold pressure. The equation can be scaled to account for variations in engine displacement provided the engine technology and valve timing remains similar.

Derivation of the engine specific constants s_i and y_i must be arranged prior to use. These are obtained by a linear fit of a rearranged version of the mass flow rate function (shown below) vs. intake manifold pressure.

$$\dot{m}_{ap} * \frac{120RT*1000}{V_d * n} = (s_i p_i + y_i)$$

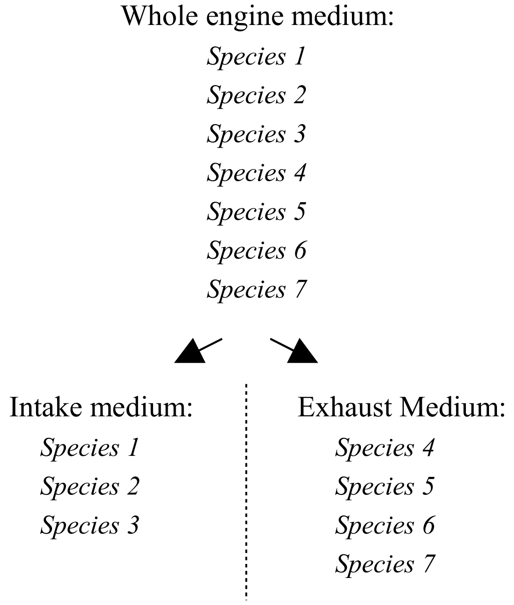
The linear fit yields an equation in the form of y = mx + c, where the gradient m will be equal to s_i and the intercept c will be equal to y_i:

If the engine makes use of variable induction mechanisms (variable valve timing, variable length intake runners, etc.) the user might wish to determine s_i and y_i for various engine speed intervals, to improve the accuracy of the mass flow rate. A map of these values can then be input into the model.

Once s_i and y_i have been determined, the computed mass flow rate is used in a mass flow rate source and sink, each representing the flow past the intake valves and the exhaust valves respectively.

The intake and exhaust system mediums are based on Modelica.Media medium models and are modelled as separate fluids, each containing the appropriate species for that part of the engine.

The main reason for utilising two specific medium models (intake and exhaust), as opposed to a single medium model encompassing all intake and exhaust species, is down to CPU time reduction.



Whole engine medium:
Species 1
Species 2
Species 3
Species 4
Species 5
Species 6
Species 7

Intake medium:
Species 1
Species 2
Species 3

Exhaust Medium:
Species 4
Species 5
Species 6
Species 7

If we were to use the whole engine medium model, we would have to:

- set the exhaust specific species mass fractions to zero on the intake side
- set the intake specific species mass fractions to zero on the exhaust side

Due to computational errors the mass fractions of these mediums might end up not being exactly equal to zero. By using two separate, simpler, medium models the mass fraction balancing becomes more robust and gains in CPU time are achieved.

2.2 Crank Angle Resolved Engine Library (CAREL)

This variant of the engine library is capable of predicting crank angle resolved values for torque, thermal effects and friction with more detailed intake, exhaust and combustion models. Typical applications of this variant of the library would be for investigating/modelling:

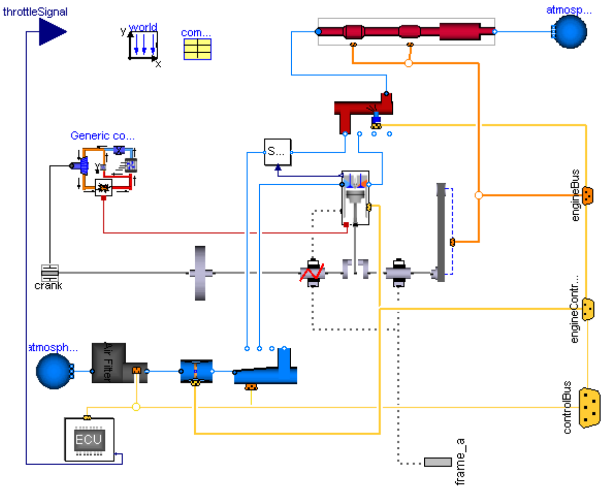
- Mount forces
- Excitation of driveline with full cyclic torque
- Cranking (start-up) and engine warm up
- Detailed friction modelling

Methods for predicting the engine torque are based on the widely used Wiebe function. Where more detailed combustion or heat release models are required, bespoke models can be "plugged in" provided their interfaces are compatible with the Engine Library architecture.

2.3 Surrogate models and real time simulation

To minimise CPU time for the simulations and achieve real-time simulation, in addition to a conventional multi-cylinder architecture, the Engines library adopts an option presented in [3]. A single-cylinder model is parametised and a duplicating model replicates the variables of interest, namely the flows, temperatures and torque generated. With appropriate connections made from the variable duplicating model to the relevant components, a multi cylinder engine model can be simulated with minimised CPU time and negligible loss of accuracy (<2%). This solution has been successfully implemented in both variants of the Engines library

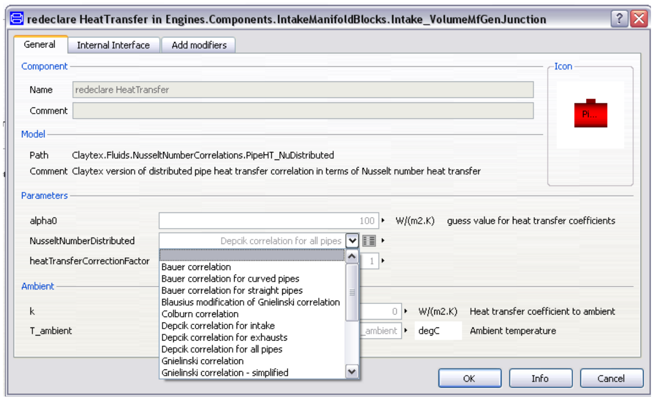
The surrogate model method also allows a reduction in architecture diagram complexity and a quick and effective way to vary the number of cylinders in the engine.



Example of surrogate MVEM layout

2.4 Thermal effects

Pipe wall and fluid thermal effects are taken care of using the Modelica Fluid dynamic pipe models (Modelica.Fluid.Pipes.DynamicPipe) [1]. In addition to the heat transfer models within Modelica.Fluid, the Engines library includes a further heat transfer model where existing or bespoke Nusselt Number correlations can be "plugged in" with a drop-down list of options [4].


redeclare HeatTransfer in Engines.Components.IntakeManifoldBlocks.Intake_VolumeMfGenJunction

Component

Name: redeclare HeatTransfer

Comment:

Icon

Model

Path: Claytex.Fluids.NusseltNumberCorrelations.PipeHT_NuDistributed

Comment: Claytex version of distributed pipe heat transfer correlation in terms of Nusselt number heat transfer

Parameters

alpha0: 100 W/(m2.K) guess value for heat transfer coefficients

NusseltNumberDistributed: Depcik correlation for all pipes

heatTransferCorrectionFactor: 1

Bauer correlation
Bauer correlation for curved pipes
Bauer correlation for straight pipes
Blausius modification of Gnielinski correlation
Colburn correlation
Depcik correlation for intake
Depcik correlation for exhausts
Depcik correlation for all pipes
Gnielinski correlation
Gnielinski correlation - simplified

Ambient

k: 0 W/(m2.K) Heat transfer coefficient to ambient

T_ambient: _ambient degC Ambient temperature

Heat transfer GUI with Nusselt number correlation drop-down list

These correlations make use of the Re (Reynolds number), Pr (Prandtl number), the medium temperature and the medium pressure and are used in the equations below to calculate the heat transfer between the fluid and the pipe wall.

A heat transfer correction factor has been introduced for correlation purposes.

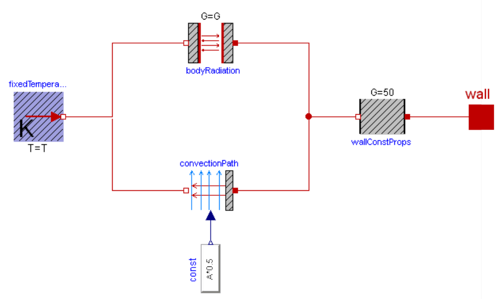
```
Q_flows = {alphas[i]*surfaceAreas[i]*(heatPorts[i].T -
        Ts[i])*nParallel for i in 1:n};
alphas = lambdas*heatTransferCorrectionFactor .* Nus ./
        dimensions;
```

where for the descretised pipe:

- Q_flows is the vector of heat flows
- alphas is the vector of heat transfer coefficients
- surfaceAreas is the vector of heat transfer surface areas
- heatPorts.T and Ts are the vectors of the heat port temperatures and fluid temperatures
- nParallel is the number of parallel pipes
- lambdas is the vector of thermal conductivities
- Nus is the vector of Nusselt numbers
- Dimensions is the vector of pipe diameters

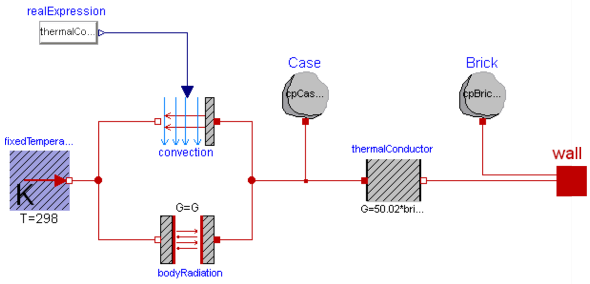
Pipe Wall Heat Dissipation Models

Three options for pipe wall heat transfer model are available within the Engines library models. The first is a fixed temperature model where the user fixes the pipe wall temperatures for steady state tests. The second option models the thermal energy dissipation to ambient (including convection and radiation effects). This might be of particular interest when performing transient engine tests where the temperatures of the pipe walls throughout the test may vary significantly.



Heat transfer-to-air model

The third is a bespoke model for catalytic converter heat transfer. This model accounts for the thermal capacities of the catalyst brick and the casing of the catalytic converter. It can be used to model the catalyst light off in a transient test.



Catalyst heat dissipation model

All models are based on Modelica.Thermal library components.

2.5 Engine Friction

Bearing models with friction take into account the crank shaft, camshaft support and valvetrain mechanism bearing friction. Established friction models have been implemented [5] [6] [7] [8] which describe the friction torque resulting from hydrodynamic and rolling contact bearings. A more detailed description of the bearing models can be found in section 3.

Piston Assembly

Piston assembly friction is cyclic and related to the cylinder pressure, piston speed and piston ring geometry. Piston skirt - cylinder liner friction is also modelled and relates to piston speed, geometry and the resultant lateral forces. Boundary and Hydrodynamic friction types are modelled [7] [8].

Valve train

Cam friction is also cyclic and calculated with reference to the cyclic vertical and horizontal loading, cam geometry and material.

The sliding friction of the cams has been modelled as a multi-stage solution [5]. The type of lubrication existing at the cam sliding surface is determined by means of the equation below where λ is the film thickness parameter, H is the minimum film thickness for hydrodynamic lubrication, Rx is the effective radius of the sliding pair of surfaces, and σ is the measured composite surface roughness of the two surfaces [5].

$$\lambda = \frac{H * Rx}{\sigma}$$

$\lambda > 1$ denotes a hydrodynamic lubrication regime
$\lambda \cong 0$ denotes a boundary lubrication regime and
$0 > \lambda > 1$ denotes a mixed lubrication regime

Both viscous and boundary components of the sliding surface friction are then calculated and summed to yield the total sliding surface friction.

$$F_{tot} = F_b + F_v$$

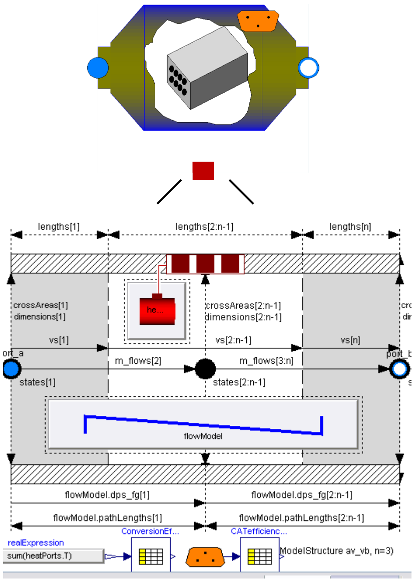
2.6 Aftertreatment

Catalysis

3-way catalytic converter models make use of the species tracking within the exhaust medium to model the catalysis of the pollutant emissions.

The catalysis is modelled phenomenologically using efficiency maps. The catalysis efficiency is dependant on the air fuel ratio and the brick temperatures. Each of the pollutant emissions is reduced accordingly and adjustments are made to the mass fractions of other species within the medium model [9] [10] [11].

The heat release is calculated according to the number of moles of each pollutant emission that have been converted. In addition, a Secondary Air Injection and Diesel particulate filter have been developed.



3 –way Catalytic converter model

Engine Control Unit

A generic engine management system is provided for basic control of the throttle, variable valve timing, emissions control systems (such as Exhaust Gas Recirculation and Secondary Air Injection), pressure charging, spark timing and injection timing and duration. All the required engine sensor signals are available to the ECU via the control bus to ensure interchangeability with a bespoke control system. A template using this control bus is available for such systems to be developed.

3 Powertrain Dynamics Library

The PTDynamics library is a new library for modelling rotating MultiBody systems. The components are designed to be a more efficient way to model rotational mechanics capturing the full MultiBody effects of the rotation. The development was driven from the fact the 1D Rotational library is too simple and the MultiBody library too inefficient for modelling transmission and driveline system dynamics.

PTDynamics includes shafts, bearings, gear mesh models, flexible joints and complex assemblies such as epicyclic and differential models along with the associated mounting systems.

3.1 Rotational3D Approach

Both the Engines and PTDynamics library make use of a new Rotaional3D library that uses the standard Modelica connector called FlangeWithBearing [12]. Within the scope of this library it is assumed that the bearingFrame is always included and it is used to capture the MultiBody dynamics of the rotating system. Within the connector, the flange connector is used to capture the rotation angle of the body and the torque being applied around the axis of rotation. The bearingFrame is used to track the position, orientation, forces and other torques being applied to the body. The rotation of the body is assumed to always be about the local x-axis of the bearingFrame.

To capture the full MultiBody dynamics of the body the rotation angle of the flange connector and the position and orientation of the bearingFrame connector have to be combined. The angular velocity of the flange connector and the bearingFrame connector are resolved in to a virtual orientation frame. The virtual orientation frame follows the bearingFrame orientation and rotates with the flange connector.

$$\omega_{body} = \left\{ \begin{array}{c} \omega_{frame}[1] + \omega_{flange} \\ \omega_{frame}[2] * \cos(\varphi_{flange}) + \omega_{frame}[3] * \sin(\varphi_{flange}) \\ \omega_{frame}[2] * \sin(\varphi_{flange}) + \omega_{frame}[3] * \cos(\varphi_{flange}) \end{array} \right\}$$

Where φ_{flange}, ω_{flange} mean the angle and angular velocity of the flange connector, ω_{frame} means the angular velocity of the bearingFrame connector and ω_{body} is the overall angular velocity of the body resolved in to the virtual orientation frame.

From this the acceleration of the centre of mass can be determined and thus the inertial effects can be calculated to determine the forces acting on the bearingFrame due to the rotation on the body.

The torque acting in the flange (τ_{flange}) is the torque required to accelerate the rotation of the body and is dependent only on the inertia of the body around the principle axis of rotation. The reaction torque in to the bearingFrame is the difference between the torque in the flange connector and total torque acting on the body:

$$\tau_{flange} = I_{xx} * \alpha_{flange}$$

$$\tau_{body} = I * \alpha_{body} + \left(\omega_{body} \times (I * \omega_{body})\right)$$

$$\tau_{frame} = \tau_{body,frame} - \{\tau_{flange}, 0, 0\}$$

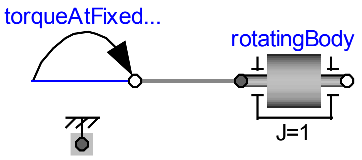
Where α_{flange} is the angular acceleration of the flange connector, I is the inertia matrix, α_{body} is the angular acceleration of the body and τ_{body} is the torque acting on the body resolved into the virtual orientation frame, $\tau_{\text{body,frame}}$ is this torque resolved in to the orientation of the bearingFrame and τ_{frame} is the torque in the bearingFrame.

Benchmarking:

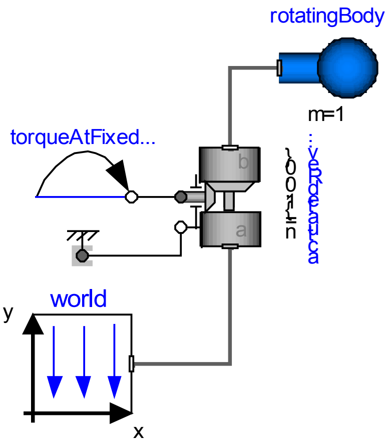
To understand the benefits of this approach and to validate the method, a series of benchmark cases were developed to compare the Modelica 1D Rotational and MultiBody libraries with the new Rotational3D model. The simplest test case is shown below where a torque actuator is used to accelerate an inertia.

Using Dymola these test cases are reduced to the system of equations shown below. The 1D Rotational model is of course the most simple system possible but it ignores many important effects. The Rotational3D and MultiBody models both capture exactly the same effects and predict the same motion of the body and reaction forces and torques in to the world object.

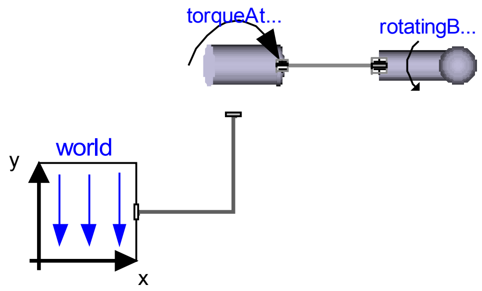
The advantage of the Rotational3D approach is that the linear set of equations seen in the MultiBody example is eliminated and the number of time varying variables is reduced to 15 from 27.



```
Continuous time states: 2 scalars
Time-varying variables: 2 scalars
Sizes of linear systems of equations: { }
Sizes after manipulation of the linear systems: { }
Sizes of nonlinear systems of equations: { }
Sizes after manipulation of the nonlinear systems: { }
```



```
Continuous time states: 2 scalars
Time-varying variables: 27 scalars
Sizes of linear systems of equations: {3}
Sizes after manipulation of the linear systems: {0}
Sizes of nonlinear systems of equations: { }
Sizes after manipulation of the nonlinear systems: { }
```



```
Continuous time states: 2 scalars
Time-varying variables: 15 scalars
Sizes of linear systems of equations: { }
Sizes after manipulation of the linear systems: { }
Sizes of nonlinear systems of equations: { }
Sizes after manipulation of the nonlinear systems: { }
```

3.2 Bearings

Bearings can be modelled as an ideal bearing which pins the shaft in position or with compliance that allows the shaft to move within the bearing. A number of friction models are available ranging from plain bearings to hydrodynamic lubrication so that different types of bearing can be modelled. Within each type of friction model we can define particular characteristics pertinent to that friction type.

In all bearings a full hydrodynamic model is available which makes use of the Sommerfeld number [7] [8] and bearing clearances to determine the lubrication regime. The model also accounts for the non-hydrodynamic type of lubrication under critical speeds and loads.

Rolling element bearing friction is also modelled with a selection of predefined friction coefficients that depend on the type of bearing and geometry of the rolling elements (single or double row ball, roller, taper roller).

Seal friction is taken into account using the normal force generated by the seal on the shaft and the defined coefficient of friction [13].

The rolling element bearing friction torque is calculated using the following formula:

$$M_r = \left(F * f * \frac{d}{2} \right) + \left(F_s * f_s * \frac{d}{2} \right)$$

M_r = Friction torque (Nm)

F = Radial (or axial load) (N)

F_s = Seal radial (or axial load) (N)

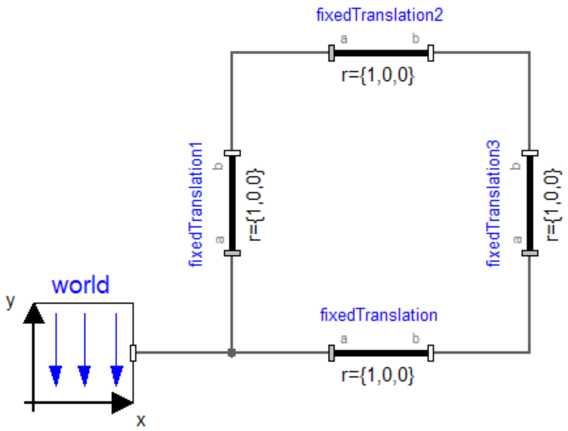
f = coefficient of friction of rolling bearing

f_s = coefficient of friction of seal

d = Diameter of the bore of the bearing (Shaft diameter) (m)

D = Outside diameter of the bearing (m)

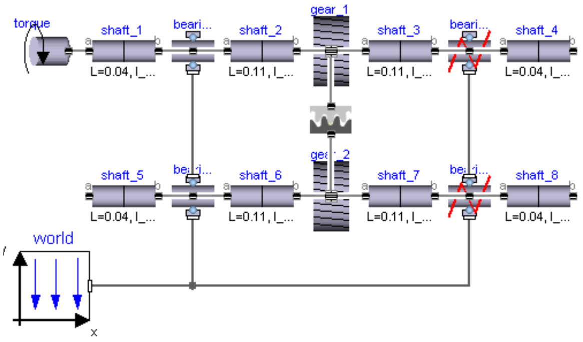
Careful attention has to be paid by the modeller to avoid mechanical loops as these can be easily introduced. A mechanical loop is one where a position could be calculated via two or more paths and this cannot be handled automatically in Modelica. A simple case is illustrated below where the fixedTranslations form a loop.



Using the shaft and bearing components within the PTDynamics library it would be very easy to create a mechanical loop. In the simplest case this would consist of a shaft with a support bearing at each end which would form a loop. To overcome this problem the bearing components include a flag **breakMechanicalLoop** and the modeller then has to follow the simple rule that only 1 bearing supporting a shaft can have this flag set to false, all the other bearings must have this flag set to true. The bearing icon is changed to reflect the value of this flag to make it easy for the modeller to verify this rule.

3.3 Gears

Within the PTDynamics library the fact that two gears are meshing is defined by adding a gear mesh component between the two gear bodies. This gear mesh model then calculates the forces and torques acting between the gears based on their relative positions and geometry.

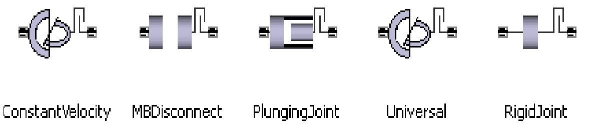


The mesh models account for the pressure angle and helix angle to calculate the radial and axial forces acting on the shafts. The mesh models also account for the rotation induced in the shafts due to their axial movement and the sliding of the gear teeth against one another in non-spur gears. A range of different mesh models will be available to account for mesh stiffness and backlash within the gear pair.

Specialised mesh models are also defined for use in epicyclic and differential gears that allow the forces and torques acting on each gear to be calculated.

3.4 Joints

A range of different shaft couplings are available allowing articulation of the shafts. The Joints can all include torsional compliance effects such as backlash and account for the cyclic speed and torque effects present in joints such as the Hookes joint (Universal joint). Also available are plunging joints with friction and constant velocity joints.



To simplify models and eliminate joint articulation two special joints are provided, a rigid joint and one called a MBDisconnect joint. The rigid joint eliminates all degrees of freedom in the joint. The MBDisconnect joint provides a complete break in the MultiBody system.

4 Coupling the Engines and Power-Train Dynamics Libraries

Being able to replicate engine torque pulsations through the driveline for torsional vibration analysis is a key part of driveline design. The designer could just input a table based torque signal obtained by means of a logged test but the ability to dial in a prescribed throttle profile and being able to vary it for different tests, thus being able to generate the corresponding torque is a desirable feature.

Using the VehicleInterfaces library [12] a complete vehicle model has been developed using the Engine library and the PTDynamics library to model the Transmission, Driveline and Chassis systems.

A rear-wheel drive vehicle has been developed and used to perform driveability tests whilst exploring the mount reaction forces and driveline vibration.

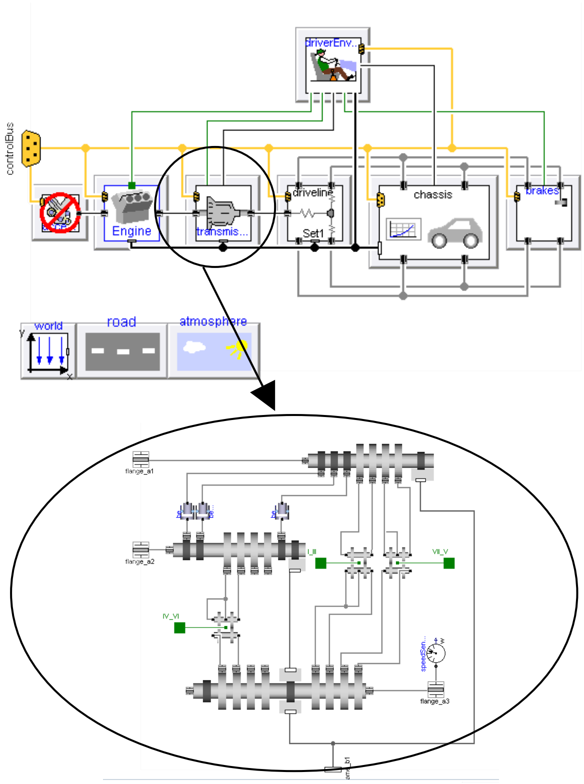


Diagram of the 6-speed dual clutch gearbox used in the vehicle model.

5 Driveability study

In this example, a 1.8L gasoline inline 4-cylinder engine is coupled to a manual transmission, 1100kg kerb weight, rear wheel drive vehicle and made to perform a typical tip-in/tip-out test whilst travelling in a straight line. The resulting driveline vibrations are displayed. The differential movement on its mounts is also shown during the tests

5.1 The Engine

The engine has been modelled using both variants of the library. For use during the tip-in/tip-out test the MVEL variant of the engine is utilised. The CAREL variant is used for correlation purposes and in other tests using the same vehicle.

The engine is a 4 cylinder inline, spark ignition, direct injection engine with a total volumetric displacement of 1800cc. It's naturally aspirated with 4 valves per cylinder and direct acting camshafts. The engine is mounted in the vehicle with 3 non-linear mounts.

5.2 The transmission

The vehicle modelled uses a 6-speed dual clutch transmission rigidly mounted to the engine. The model includes backlash, synchro parallel gear mesh models and compliant bearing models with relevant parasitic loss (friction) and efficiency models.

5.3 The driveline

An open differential with 4-point mounting system has been used. The PTDynamics Differential Gear and Bevel Gear mesh models have been used for both differential assembly and the pinion-wheel mesh. Parasitic losses have been implemented within the bearings and the power-dependant losses are modelled as efficiency terms within the mesh models. All main shafts and joints are compliant with backlash applied to specific joints within the driveline model.

5.4 The Chassis

A Pacejka Magic Formula tyre model was used, utilising the SAE J2452 rolling resistance model. The vehicle body used a 3 degree of freedom model capturing pitch and bounce in addition to the longitudinal motion of the vehicle.

5.5 Correlation

Vehicle coast downs were performed for the vehicle in each gear to correlate the aerodynamic, rolling resistance and driveline losses.

5.6 Tip-in tests

The terms tip-in and tip-out are used within driveability tests to describe a positive and negative step change in throttle input. In the context of this paper, the tip-in will refer to a 70% step throttle opening. Tip-out will refer to a full throttle pedal lift off.

Tip-in tests were performed using open-loop control of the driver throttle pedal. The same test can be repeated in each of the gears in the transmission and for varying amounts of throttle opening applied.

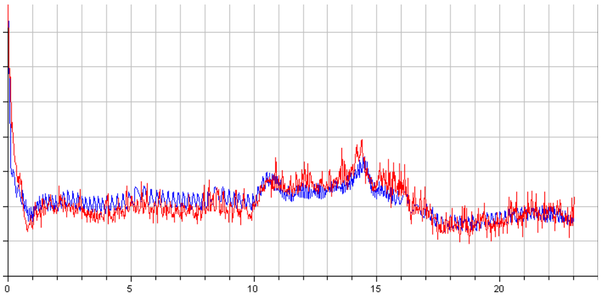
The vehicle is left to settle to a predefined speed at which a tip in event is triggered. At a predefined engine speed we then trigger the tip-out event.

The oscillatory phenomena at the investigated events will be captured for driveline vibration analysis.

6 Results

6.1 Engine: MVEM and CAREM vs Test Data

Simulation data from a fully correlated engine model generated in a widely used engine simulation package was used to correlate both a MVEM (Mean Value Engine Model) and a CAREM (Crank Angle Resolved Engine Model).

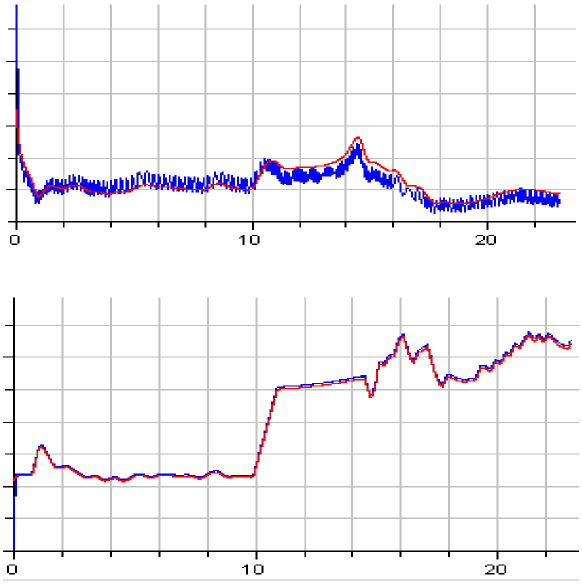


CAREM results (blue) vs. correlated engine model (red)

Real Time with MVEM

Simulation of the first 23s of the ECE15 cycle are completed within 11s of CPU time. The model includes the catalysis of the exhaust gasses and the modelling of the 3-way catalytic converter brick light-off.

MVEM Vs. CAREM



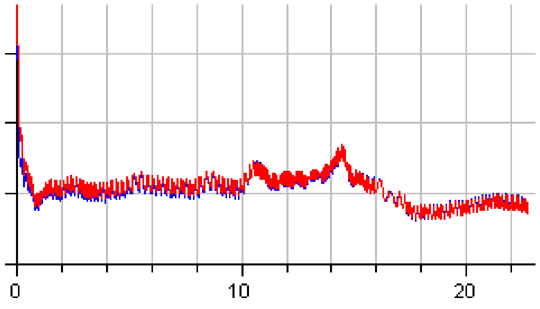
Plot of plenum pressure and throttle body mass flow rate for a CAREM (blue) and MVEM (red).

The results from the MVEM and CAREM were compared to each other to ascertain similarity within the two models.

The plenum pressure error comparing a MVEM and CAREM representation of the same engine is within +/- 5%. The throttle body mass flow rate is within +/- 1%. The discrepancy during the first 10 seconds of the plenum pressure plot can be attributed to slight a miscalibration of the MVEM mass flow rate function at low throttle openings and engine speeds.

6.2 Multi Cylinder vs. Surrogate

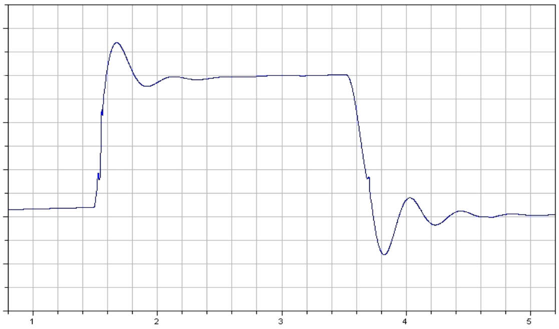
The advantages of surrogate models lie in model simplification, which translates to a reduction in CPU time. A test was performed to demonstrate negligible loss in accuracy when using the surrogate cylinder engine model to replace a multi-cylinder engine model.
Whilst the error was contained within +/- 2%, the benefit in running a surrogate CAREM model over a multi-cylinder CAREM is a 4.5 times reduction in CPU time.



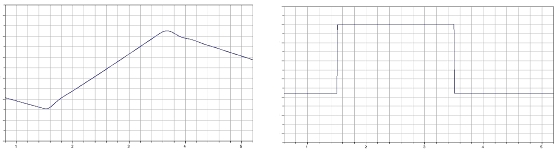
Plot of plenum pressure for a CAREM multi-cylinder representation (blue) vs. a surrogate representation of the same 4-cylinder SI engine (red).

6.3 Tip-in tip-out tests

Results from a second gear tip-in tip-out test are shown below.

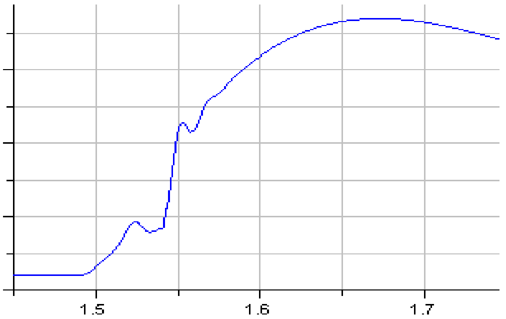


Vehicle longitudinal acceleration



Vehicle longitudinal velocity and engine throttle position

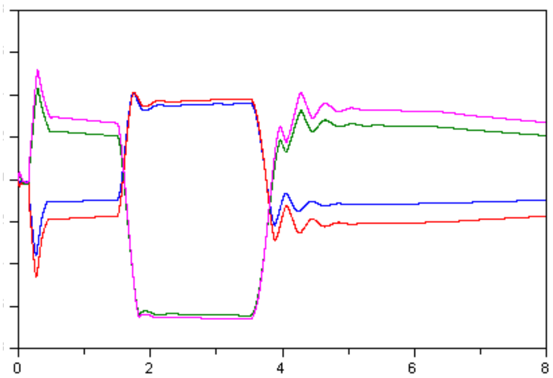
The vehicle longitudinal acceleration presents damped oscillations typical of this type of manoeuvre at and after the tip-in and tip-out events. The backlash in the driveline is particularly visible between the tip in/out events and the first peak in the vehicle acceleration oscillations as shown below.



Vehicle longitudinal acceleration showing discontinuities due to backlash regions being crossed.

The mount displacements for the differential were in agreement with expectations for this type of manoeuvre.

The relative roll of the differential is visible shortly after the tip-in and tip-out points (1.5s and 3.5s). A positive pitch angle for the differential assembly is demonstrated during the tip-in acceleration event (front mounts have moved upwards and rear mounts have moved downwards) whilst a negative one is shown for deceleration event.



Vertical displacement of the 4 differential mounts during the tip-in/tip-out test. The blue and red lines represent the front mounts and the green and magenta lines represent the rear mounts.

7 Conclusions

Two new libraries have been developed for modelling Engines and Powertrain system dynamics. These both utilise a new approach to modelling the mechanics that capture the full MultiBody effects in a more efficient manner than the standard Modelica MultiBody library.

Using these two libraries a complete vehicle model has been built to study a range of different behaviours. Results for the engine model operation on its own are presented along with results from a tip-in/tip-out manoeuvre.

The developed vehicle model can therefore be used for driveline vibration analysis as well as performance and vehicle dynamics tests. By plotting the mode shapes we can identify the sources of vibration and adjust the source components accordingly to counteract undesired phenomena.

References

1. Franke, R. et al. *“Standardization of thermo-fluid modeling in Modelica_Fluid 1.0”.* Modelica Conference 2009.
2. Hendricks et al. *“Modelling of the Intake Manifold Filling Dynamics”* SAE 960037 1996.
3. John J. Batteh Charles E. Newman. *“Detailed Simulation of Turbocharged Engines with Modelica”* Modelica Conference, 2008.
4. Finol C A and Robinson K. *“Thermal modeling of modern engines: a review of empirical correlations to estimate the in-cylinder heat transfer coefficient”.* Department of Mechanical Engineering, University of Bath, UK.
5. Yang et al. *“A Valve Train Friction and Lubrication Analysis Model and Its Application in a Cam/Tappet Wear Study”* SAE 962030 1996.
6. R. C. Coy. *“Practical applications of lubrication models in engines”* Tribology International vol. 31 No. 10.
7. Heywood J.B. *“Internal Combustion Engine Fundamentals”* McGraw Hill.
8. Stone, R. *“Introduction to Internal Combustion Engines”* SAE International, 1999
9. Masoudi. M. *“Pressure Drop of Segmented Diesel Particulate Filters”.* SAE 2005-01-0971 2005
10. Kladopoulou et al. *“A study Describing the Performance of Diesel Particulate Filters During Loading and Regeneration – A lumped Parameter Model for Control Applications”.* SAE 2003-01-0842 2003
11. Silva et al. *“Evaluation of SI engine exhaust gas emission upstream and downstream of the*

catalytic converter". Mechanical Engineering Department, Technical University of Lisbon, Portugal. 2006

12. Dempsey, M. et al. *"Coordinated automotive libraries for vehicle system modeling"*. Modelica Conference 2006.
13. www.roymech.co.uk
14. Kandylas I. P. and Stamatelos A.M. *"Engine exhaust system design based on heat transfer computation"* Energy Conversion and Management 40 (1999).
15. Batteh J. J. and Kenny P. J. *"Modelling the Dynamics of Vehicle Fuel Systems"*. Modelica Conference 2006.

Detailed Loss Modelling of Vehicle Gearboxes

Clemens Schlegel
Schlegel Simulation GmbH
Freising, Germany
cs@schlegel-simulation.de

Andreas Hösl
BMW Group
Munich, Germany
andreas.ha.hoesl@bmw.de

Sergej Diel
University of Applied Sciences Landshut
Landshut, Germany
sergej.diel@fh-landshut.de

Abstract

Drag torques of gearboxes are an important part of the overall losses in today's vehicle drive trains. From measurements it is well known that overall drag torques of vehicle gearboxes vary significantly over the range of operating points and speeds, depending on the interaction of the losses of the single gearbox elements like bearings, gearings, etc. Because today's vehicle emission regulations are becoming stricter and stricter "drag torque design" of gearboxes will be even more important in the future. Prediction of losses helps to save cost (e.g. drag torque measurements), speeds up the development and allows to assess many concepts in short time.

We collected detailed semi-analytical drag models for the common gearbox components from literature and from manufacturer information and implemented them in a Modelica library. This library contains models for radial shaft seals, rotary unions, synchronizers, multi-disc clutches, helical gearings, planetary gearings, various kinds of bearings, lubrication systems and lubricant characteristics. Using this library drag torques of any vehicle gearbox may be computed for any operating condition (engaged gear, speed, torque, temperature). Simulation results for a 7 speed double clutch transmission show good correlation with measurements.

Keywords: Automotive; Gearboxes; Drag torques

1 Introduction

Simulation of gearboxes is used widely for e.g. gear ratio design, for investigating noise, vibration and harshness, for assessment and tuning of control strategies and similar design aspects. Gearbox components are mostly modelled ideal, without losses, because drag torques are considered as negligible for the mentioned cases. If detailed investigations with respect to fuel consumption and emissions are needed, mostly loss maps based on measurements are used, an explicit computation of gearbox losses is done rarely. Since in an early design stage measurements are not available, the loss maps have to be estimated or derived from former designs.

A lot of papers and studies on loss computation of gearbox components are available, for a survey see e.g. [1]. For some components like torque dependent losses of gears very reliable loss descriptions exist, for other components like multi-disc clutches and most of the further torque independent losses only rough estimates are available in literature.

Losses which are independent from the gearbox input torque are a major part of the overall losses in driving cycle computations (like NEDC) which are in turn an important part of any driveline assessment. Even the most advanced codes for gearbox loss computations which are generally available [2] do not or only in part take into account torque independent losses what results in reduced reliability of the computations. Losses of synchronizers, multi-disc clutches and rotary unions are rarely taken into account.

A further problem in this context is that loss computations for bearings need the forces and torques acting on the bearings as input, but no general approach to compute them for all kinds of bearings is available, because for the highly nonlinear stiffness relations of bearings no analytical descriptions are known.

2 Losses of gearbox components

2.1 General considerations

Naunheimer [3] gives a rough survey of typical overall efficiencies of vehicle gearboxes under full load:

DOI: 10.3384/ecp09430059

Manual gearbox	Automatic gearbox	Mechanical CVT	Hydrostatic CVT
92 - 97 %	90 – 95 %	87 – 93 %	80 – 86 %

The losses of manual and automatic gearboxes are made up of drag torques due to the following effects in each single gearbox component (P_V is the power loss):

- Gears: Mesh friction, swash and squeeze in splash lubrication, oil impulse in spray lubrication (P_{VZ})
- Bearings: Rolling and sliding friction, lubrication losses and losses in seals (P_{VL})
- Radial shaft seals: Friction between the sealing lip and the rotating shaft (P_{VD})
- Rotary unions: Friction between the lateral surfaces depending on oil pressure and shear flow in the pressure chamber (P_{VDDF})
- Synchronization: Fluid drag between synchronizer and friction cone (P_{VS})
- Clutch: Fluid drag in wet multi-disc clutches (P_{VK})
- Oil pump: Torque consumption depending on system pressure, temperature and oil viscosity (P_{VNA})

The total power loss is the sum of the power losses of the single elements and can be expressed as:

$$P_V = P_{VZ} + P_{VL} + P_{VD} + P_{VDDF} + P_{VS} + P_{VK} + P_{VNA} \tag{2.1}$$

Losses of gears and bearings may be split up into a part depending on the transmitted torque and a torque independent part. Torque dependent losses occur when two surfaces which are under pressure move relatively to each other, for example in gears or bearings. These losses are depending on the force, sliding speed and the friction coefficient (which is, in turn, depending on the lubrication) in the contact area.

Torque independent losses arise even if no torque is transmitted but the shafts rotate. These kinds of losses occur for example in seals and rotary unions.

Losses in gearings and bearings dominate the overall losses of manual gearboxes. In contrast, the losses of wet multi-disc clutches in automatic gearboxes can account for even more than 50 % of the total power loss.

The computed overall loss of transmitted power is used to set up an efficiency map, which may be used in other simulation models for fuel consumption investigations. In the following chapters all the listed effects influencing the total power loss are described in more detail. Since the losses are modelled physically as “torques”, the equations are given with the same definition.

2.2 Gears

The total loss torque M_{VZ} in gears can be decomposed into the torque dependent part $M_{VZ,la}$ and the torque independent part $M_{VZ,lu}$:

$$M_{VZ} = M_{VZ,la} + M_{VZ,lu} \tag{2.2}$$

2.2.1 Torque dependent losses

Ohlendorf [4] introduced the first theoretical approach for the calculation of torque dependent losses in gears in 1958. The power loss in [4] is calculated on the assumption of a constant coefficient of friction μ_m, and a constant normal force:

$$M_{VZ,la} = M_{an}\mu_m H_V \tag{2.3}$$

M_{an} is the acting torque and H_V is the gear loss factor which takes into account the geometry of the teeth. According to Schlenk [5], μ_m can be expressed as:

$$\mu_m = 0{,}048\left(\frac{F_{bt}/b}{v_{\Sigma C}\rho_{redC}}\right)^{0,2} \eta^{-0,05} R_a^{0,25} X_L \tag{2.4}$$

F_{bt} is the tangential force at the base circle, b the tooth width, $v_{\Sigma C}$ the sum speed at the operating pitch circle, ρ_{redC} the reduced radius of curvature at the pitch point, η the dynamic oil viscosity, R_a the arithmetic mean roughness, and X_L a factor for the oil type.

The gear loss factor H_v was introduced by Wimmer [6]. He only depends on the gear geometry:

$$H_V = \pi \frac{i+1}{z_A\, i \cos\beta_b}(\alpha_0 + \alpha_1|\varepsilon_A| + \alpha_2|\varepsilon_B| + \alpha_3|\varepsilon_A|\varepsilon_A + \alpha_4|\varepsilon_B|\varepsilon_B) \tag{2.5}$$

i is the gear ratio, z_A is the number of teeth of the pinion, β_b is the base helix angle, ε_A and ε_B are the contact ratios of acting and driven wheel and α_1 to α_4 are coefficients depending on the values of ε_A and ε_B. The parameters ρ_{redC} and $v_{\sum C}$ are:

$$\rho_{redC} = \frac{1}{2} d_w \sin(\alpha_{wt}) \frac{i}{(i+1)} \frac{1}{\cos\beta_b} \tag{2.6}$$

$$v_{\sum C} = 2 v_t \sin(\alpha_{wt}) \tag{2.7}$$

with d_w the pitch diameter, α_{wt} the service pressure angle and v_t the pitch line speed.

2.2.2 Torque independent losses

Two kinds of lubrications are commonly used in gearboxes: Splash lubrication and spray lubrication. For splash lubricated gears the loss torque can be written as:

$$M_{VZ,lu} = M_{VZ,Q} + M_{VZ,Pl} + M_{VZ,V} \tag{2.8}$$

In the case of spray lubrication the loss torque is defined as:

$$M_{VZ,lu} = M_{VZ,Q} + M_{VZ,I} + M_{VZ,V} \tag{2.9}$$

The squeeze loss $M_{VZ,Q}$ is caused by the displacement of the oil in the contact area between the teeth. $M_{VZ,Pl}$ is the resistance of the gear wheel when rotating in an oil bath. The ventilation loss $M_{VZ,V}$ is the air drag due to air or oil mist. The oil stream in spray lubrication causes the impulse loss torque $M_{VZ,I}$, since it acts on the rotating wheel.

The losses of splash lubrication are in general higher compared with spray lubrication. The losses of spray lubrication will be described in more detail, because the gearbox investigated is lubricated by oil spray.

Mauz [7] gives a definition for $M_{VZ,Q}$:

$$M_{VZ,Q} = 4{,}12\, C_1 \rho \dot{Q}_e^{0,75} r_w\, v_t^{0,25} b^{0,25} m_n^{0,25} \cdot \left(\frac{\nu}{\nu_0}\right)^{0,25} \left(\frac{h_Z}{h_{Z0}}\right)^{0,5} \tag{2.10}$$

Here ρ is the oil density, $\dot{Q}_e$ is the oil volume flow, r_w is the pitch radius, and m_n is the normal module. The kinematic oil viscosity ν and the tooth height h_z are normalized by the reference values ν_0 and h_{z0}. Equation (2.10) is only valid for the cases *BO* and *BU* (see fig. 1). For the cases *AO* and *AU* the loss torque $M_{VZ,Q}$ is zero. The factor C_1 is a scaling factor for gravity effects, it is 1 for *BO* and 0.9 for *BU*.

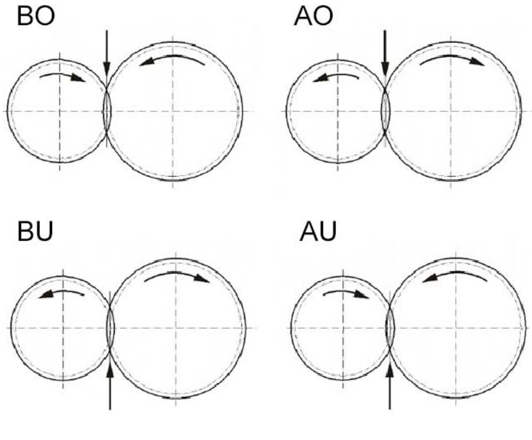


Figure 1: Different cases of spray lubrication [7]

In the cases BO and BU the spray jet hits the approaching contact area from top or from bottom, respectively. In the cases AO and AU the oil stream meets the regressing contact area. In these cases, there is almost no oil remaining on the teeth because of centrifugal forces. Therefore the loss is very small and can be neglected.

The impulse loss torque $M_{VZ,I}$ usually provides the main part of the torque independent losses in spray lubricated gears. Ariura [8] defines $M_{VZ,I}$ as:

$$M_{VZ,I} = C_2\, \rho \dot{Q}_e\, r_w (|v_t| - v_s) \quad \text{for BO, BU} \tag{2.11}$$

$$M_{VZ,I} = C_2\, \rho \dot{Q}_e\, r_w (|v_t| + v_s) \quad \text{for AO, AU} \tag{2.12}$$

v_s is the velocity of the oil stream. The coefficient C_2 takes into account gravitation effects. For *BO* and *AO* the coefficient C_2 is 1, for *BU* it is 0,9 and for *AU* it is 0,85.

According to Maurer [9], the ventilation loss torque $M_{VZ,V}$ is composed of the loss of the gear wheel itself (M_{VZ,V_Z}) and the loss in the contact area between the teeth (M_{VZ,V_E}):

$$M_{VZ,V\,Z} = 1{,}37 \cdot 10^{-9}\, v_t^{1,9}\, d_w^{1,6}\, b^{0,52}\, m_t^{0,69}\, F_{Wand} \tag{2.13}$$

$$M_{VZ,V\,E} = 1{,}17 \cdot 10^{-6}\, v_t^{1,95}\, i^{0,73}\, b^{1,37}\, F_{Wand} \tag{2.14}$$

m_t is the transverse module and F_{Wand} describes the influence of the gearbox housing.

2.3 Bearings

While the forces acting on the mesh or on statically determined bearings can easily be computed from force and torque equilibrium using the transmitted torque and geometry parameters (see fig. 2), the force calculation for statically over-determined bearings requires some preprocessing.

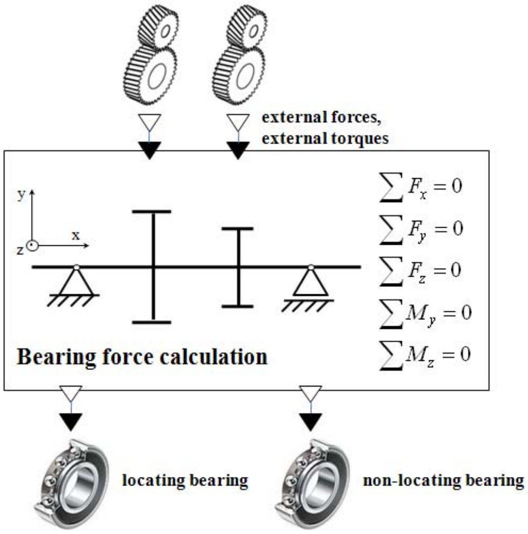


Figure 2: Model for the calculation of bearing forces of statically determined shafts

For the latter case, which is not uncommon for vehicle gearboxes, and for more detailed investigations computation of the forces and torques acting on a bearing requires detailed knowledge of the stiffness properties which are highly nonlinear for the rolling bearings commonly used.

Since manufacturers do not publish detailed stiffness data, we used a separate bearing calculation software provided by a bearing manufacturer [10] to compute for each over-determined shaft a map of bearing forces and torques depending on the transmitted torque and the gear engaged. The same procedure may be used for adjusted bearings.

Because several gearings may act on the same shaft, all single gearing forces and torques acting on a couple or triple of bearings must be collected and routed accordingly to the bearing drag computation model component.

In vehicle gearboxes, mostly rolling contact bearings are used. The respective loss computations will be described in the following sections in more detail.

Figure 3 shows six different types of losses in a rolling contact bearing: Rolling and sliding friction between rolling element and bearing rings (1 and 2), sliding friction between cage bar and bearing rings (3), sliding friction between cage and rolling element front surface (4), sliding friction between rolling element and outer ring (5), and friction between cage and rolling element (6) [11].

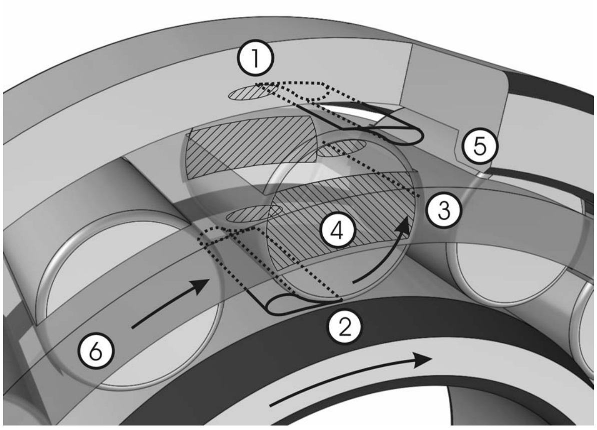


Figure 3: Different types of losses in a rolling contact bearing.

The first detailed approach to drag torque calculation of rolling contact bearings was published by Palmgren [12]. He proposed to split the drag into a torque dependent and a torque independent part. The investigations by Harris [13] and INA/FAG [14] are based on [12] and have been used for a long time. The relatively new approach by SKF [15] allows more detailed investigations because the losses are assigned to the places where they occur in the bearing.

All mentioned methods have been implemented in the library. Because of the importance of the SKF-method it will be described shortly:

$$M_{VL} = \phi_{ish}\,\phi_{rs}\,M_{rr} + M_{sl} + M_{seal} + M_{drag} \qquad (2.15)$$

In this equation the rolling frictional torque M_{rr} and the sliding frictional torque M_{sl} represent the torque dependent losses. M_{seal} is the frictional torque of bearing seals and M_{drag} is the frictional torque due to churning, splashing etc. M_{seal} and M_{drag} represent the torque independent losses.

The rolling and the sliding friction torques M_{rr} and M_{sl} are given by:

$$M_{rr} = G_{rr}(\nu\, n)^{0,6} \qquad (2.16)$$

$$M_{sl} = G_{sl}\,\mu_{sl} \qquad (2.17)$$

The variables G_{rr} and G_{sl} depend on the bearing type, the bearing mean diameter, the radial load F_r and the axial load F_a , n is the rotational speed, ν the kinematic viscosity of the lubricant and μ_{sl} is the sliding friction coefficient.

ϕ_{ish} describes the influence of the lubricating film thickness on the rolling friction. The factor ϕ_{rs} considers the lubricant displacement in the contact zone due to overrolling which results in a lower rolling friction torque.

The frictional torque in bearing seals is:

$$M_{seal} = K_{S1} d_s^{\beta} + K_{S2} \tag{2.18}$$

The constants K_{S1} and K_{S2} depend on the bearing and the seal type, d_s is the sealing counterpart diameter and β an exponent which depends on the bearing and seal type.

The definition of M_{drag} for ball bearings is:

$$M_{drag} = V_m K_{ball} d_m^5 n^2 \tag{2.19}$$

and for roller bearings:

$$M_{drag} = 10 V_m K_{roll} B d_m^4 n^2 \tag{2.20}$$

V_m is a variable depending on the oil level, K_{ball} and K_{roll} are bearing type related constants, B is the bearing inner ring width and $d_m=(D+d)/2$ is the mean of the bearing outer diameter D and the bore diameter d.

The SKF-method is not applicable to needle bearings. In this case other methods for example [13] or [14] may be used.

2.4 Radial shaft seals

The loss torque of radial shaft seals is a result of the friction between the sealing lip and the rotating shaft. For the library presented, the approaches by Linke [16] and by Kettler [17] have been used because of their simplicity compared with other approaches. The definition by Linke is:

$$M_{VD} = \left[145 - 1{,}6\vartheta + 350 \cdot \log(\log(\nu_{40^\circ} + 0{,}8))\right] \cdot 10^{-7} d_w^2 \frac{30}{\pi} \tag{2.21}$$

d_w is the shaft diameter and ϑ is the temperature. Kettler gives a different formulation for the computation of the friction torque:

$$M_{VD} = 7{,}9163 \cdot 10^{-6} F_{D,\rho} d_w^2 \frac{30}{\pi} \tag{2.22}$$

The factor $F_{D,\rho}$ represents the effect of the temperature dependent viscosity change.

2.5 Rotary unions

In gearboxes rotary unions are used to permit the flow of oil to activate clutches or to transport the lubricant through the shafts from a stationary inlet to a moving part such as a rotating shaft or from one rotating shaft to another. The oil is kept within the rotary union using a (mostly rectangular) seal ring. Figure 4 shows the structure of a rotary union.

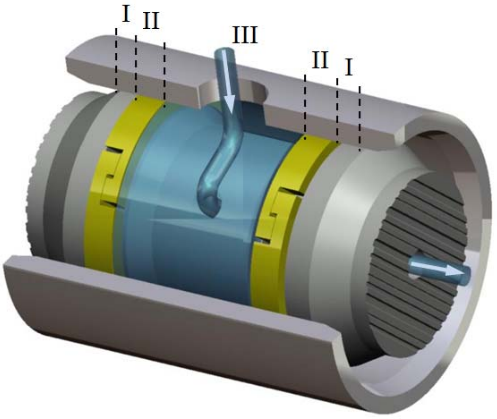


Figure 4: Structure of a rotary union

Gronitzki [18] recently investigated the losses in rotary unions in detail. He describes the loss torque as the sum of three main loss sources:

$$M_{VDDF} = 2M_I + 2M_{II} + M_{III} \tag{2.23}$$

M_{II} is the friction between the seal ring and the groove (see area II in figure 4). M_{III} (area III) is the loss torque due to the shear flow between the inner and the outer shaft in a pressurized chamber. The loss torque M_I in area I as a result of the oil leakage is small compared to the losses in areas II and III and can therefore be neglected. For further details see [18].

2.6 Synchronizers and clutches

The analytical description of synchronizer losses is subject of ongoing research [19]. The results of this research are possibly also applicable to multi-disc clutches. For our implementation we used tabulated drag data for synchronizers and clutches based on measurements.

2.7 Oil pump

Since no reliable models are available in literature we used measured, tabulated drag data for the oil pump. A direct calculation would be rather complex.

3 Library implementation

For the actual implementation of our library Modelica and the tool Dymola [20] have been used in order to easily fit into existing gearbox and drive train models at BMW. Because of the intended use for arbitrary gearbox topologies a signal oriented approach would be not feasible.

The used loss descriptions are either based on the respective geometrical and physical properties, on parameterized semi empirical descriptions, or on tabulated measurement data. All parameters are organized in hierarchical data structures and stored using a data sheet library approach. Since a gearbox may comprise separated subassemblies, for each lossy gearbox component a different lubricant may be chosen and a dedicated temperature may be parameterized.

Each component (e.g. a bearing) contains one (torque independent) or two (torque independent and torque dependent) loss components. These loss components may be specific to a certain component subtype (e.g. a roller bearing) and contain a certain type of loss computation (e.g. according to reference A or B). Each loss component may comprise one or more functions for the computation of the actual drag torque part (e.g. splash loss and ventilation loss). Finally, all loss torques are summed up and routed to the Mechanics.Rotational connectors of the respective component.

Since all formulas in literature are given only for a certain range of operating conditions (e.g. between 1000 rpm and 2500 rpm speed, below 2000 rpm, above 2000 rpm), special care has to be taken to allow usage of the models for zero speed and zero transmitted torque. For practical reasons, the components may optionally be described by a percentage loss factor (e.g. gears) or as ideal, without any losses. Since we are not interested in transient effects, stick-slip phenomena have not been modelled. A smooth friction characteristic with limited derivative at zero is used instead.

The strict separation of torque independent and torque dependent losses is useful because in many cases only torque independent measurements are available. The torque dependent losses can be calculated separately and added to the overall loss map.

We made extensive use of class parameterization in order to easily choose among a set of loss types and combinations thereof for each kind of lossy elements in a unified way. The flexible modular structure of the library allows an easy implementation of new components and loss models. Models of the losses of the basic planetary gearings (planet / planet, planet / ring, planet / sun) as parts of arbitrary planetary gearings have been developed recently and are actually under test.

4 Verification of library models

4.1 Component level verification

For a first validation of the library models, the loss calculations have been compared with the results presented in the literature references which contain loss calculation formulas.

For some components it was also possible to examine special measurements of individual component drag torques. For example, the drag torques of some needle bearings of the gearbox presented later were derived by measurements comparing a complete gearbox and a stripped gearbox where the idler gears of the counter shaft have been removed. Figure 5 shows the calculated drag torques for the regarded idler gears, the sum of these calculated drag torques and the drag torque derived from measurements. The calculation method according to the bearing manufacturer INA [14] was used.

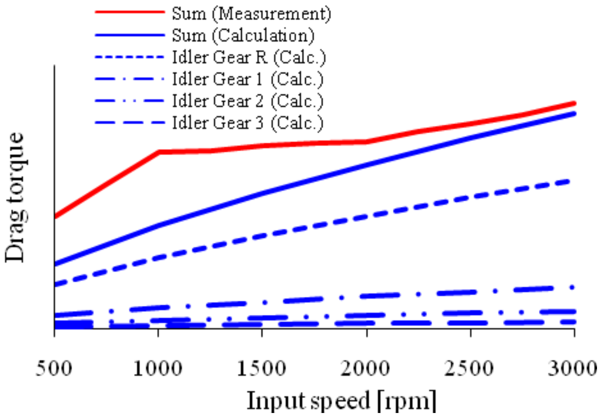


Figure 5: Validation of bearing drag torques of idler gears

The correlation of calculation and measurement is acceptable, especially for speeds above 2000 rpm. The fidelity of the bearing drag torques of the idler gears is mainly affected by the relative speed within the bearing, which is rising from the 3rd to the reverse gear. The automated computation of the reaction forces of the tooth engagements and the bearing forces was checked by analytical calculation of several simple load cases.

4.2 System level verification and sample gearbox loss model

For the system level verification, a complete model of the new BMW 7 speed double clutch transmission (DCT), which was developed in collaboration with GETRAG and introduced in 2008 (BMW M3, 335i Coupe and Convertible), has been set up. The buildup and the functional principle can be seen in fig. 6.

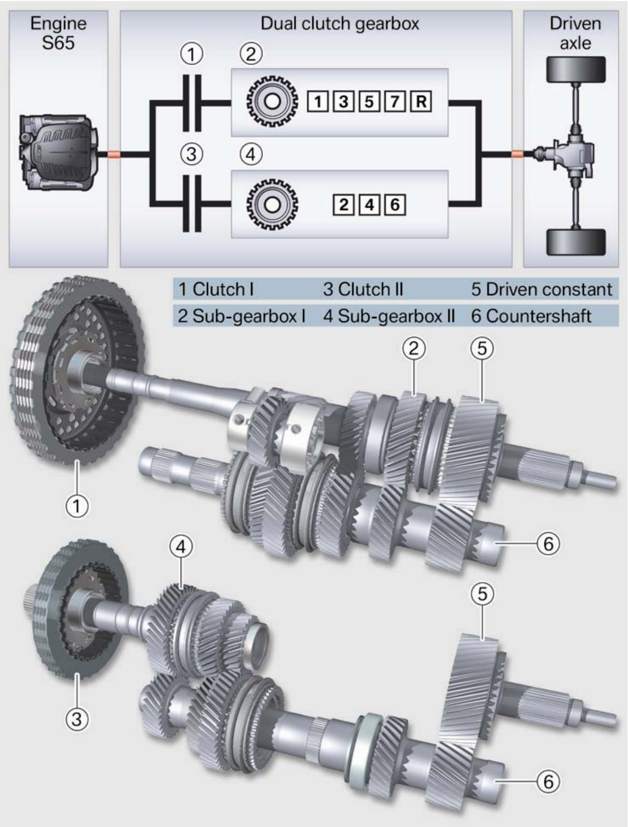


Figure 6: Buildup of the BMW 7 speed double clutch transmission

For the simulation of the drag torques, every lossy component of a gearbox is represented by a corresponding library element. Gears, synchronizers, bearings, seals, rotary unions, clutches and an oil pump have to be considered for loss calculations. The complete Dymola DCT gearbox model is shown in fig. 11.

The gearbox is basically modelled as a rigid multi body system with rotational mechanic flanges at input and output shafts. Inertias of inner shafts and idler gears are reduced to the input and output shafts. The clutches are derived from the original clutch elements of the Modelica standard library. They can be controlled by a torque request signal which is internally interpreted as friction torque (sliding clutch) or maximum transmitted torque (sticking clutch). The gears are shifted by applying forces to the sliding links of the synchronizer units, which are physically modelled including the speed drop caused by friction of the synchronizer cones and the mechanical coupling at the stop position of the sliding links. Since no validated calculation method was available, the losses of the unengaged synchronizers were approximated via drag torque functions depending on the number of cones and the friction radius according to [21], see fig. 7.

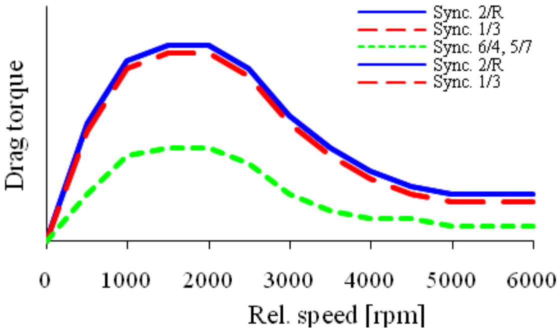


Figure 7: Synchronizer drag torques

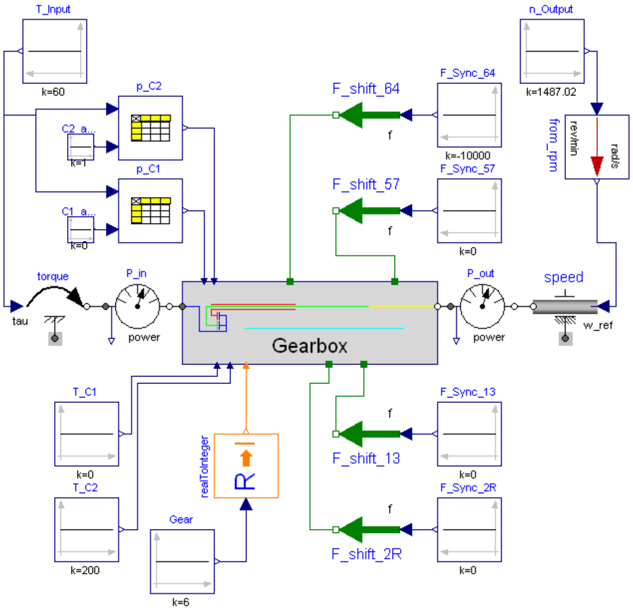


Figure 8: Virtual test bench

The drag torque map of the oil pump was measured separately for several system pressures and could therefore be interpolated from tabulated data depending on input speed and requested system pressure for

closing the clutches. For the gearings, rotary unions, radial shaft seals and bearings, the loss calculation algorithms described in chapter 2 have been used.
In order to calculate the losses of the gearbox for any desired operating point, the gearbox model was integrated in a virtual test bench shown in fig. 8. The actual operating point is adjusted by prescribing the input torque and the output speed. The total losses and the respective overall efficiency can be calculated using the signals of the power sensors at the input and output shaft of the gearbox. Depending on the input torque the pressure level for the clutches, which is mainly influencing the drag torque of the rotary unions, can be derived using simple parameter tables.

4.3 Simulation results

For the validation of the simulation model, two test series were available. A comparison to measurements at 80°C and for input torques between 10 and 100 Nm, varying the input speed from 750 to 2500 rpm, is shown below (figure 9a, 9b). This operating range is very important for the NEDC driving cycle. In the simulation, the loss torques of radial shaft seals were calculated according to Linke [16], the torque independent mesh losses according to Mauz [7], the losses of the needle bearings according to INA/FAG [14] and the losses of the remaining bearings according to the SKF method [15]. In the inactive subgearbox no gear was engaged (no gear prediction). The comparison of measurement and calculation is given in fig. 9a and 9b.

Simulation results show a good correlation with measurements. Except for the 7th gear, the maximum differences of measured and calculated drag torques are below 15%. In the 7th gear, which is engaged by direct coupling of input and output shaft, the calculated discrepancy of the two loads 30 Nm and 100 Nm is significantly less than for the other gears, since no gears are loaded in this case. The remaining torque dependencies are related to the oil pump and the rotary unions. The measurement for the 7th gear (100 Nm) shows an abnormal characteristic, which cannot be explained by the common loss approaches.

Using the validated simulation model it is possible to perform detailed investigations of the partial losses of all gearbox components. Figure 10 shows losses of different components for the 6th gear and 60 Nm input torque. The main amount of losses is caused by the bearings, followed by the oil pump. Only the losses of the synchronizers decrease with rising speed due to the special characteristic shown in fig. 7.

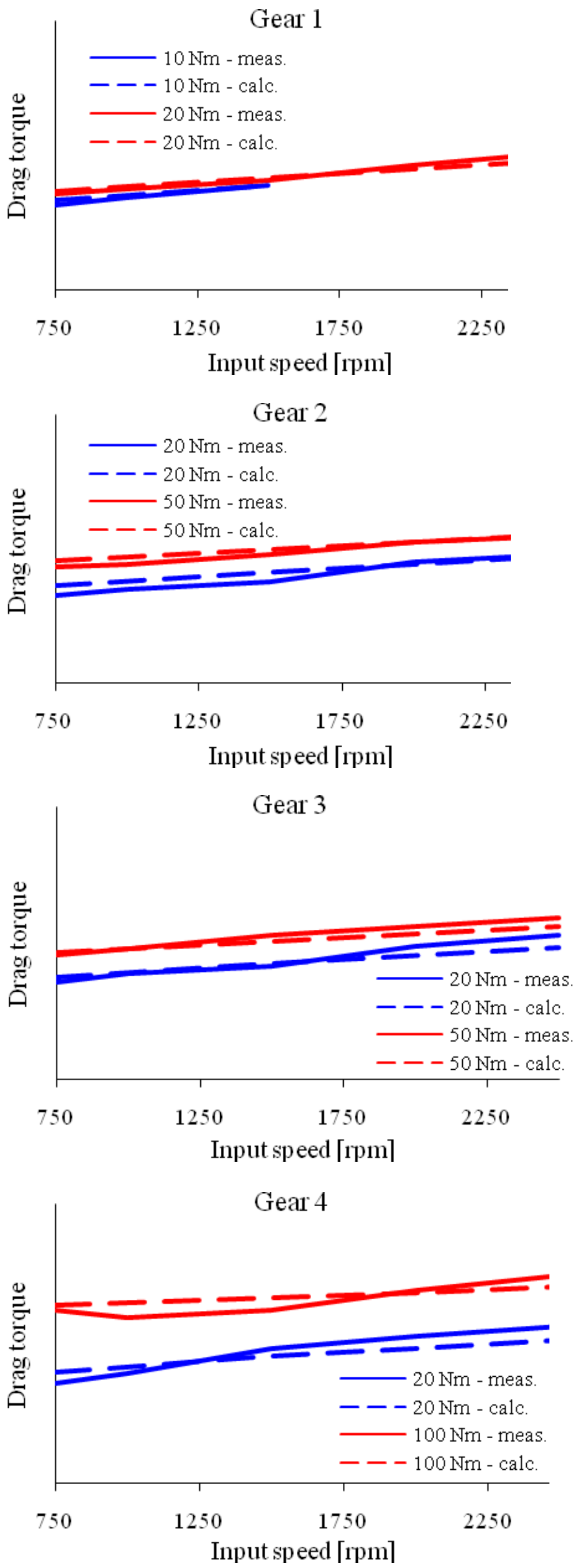


Figure 9a: Comparison of measured and calculated drag torque (80°C), gears 1 to 4

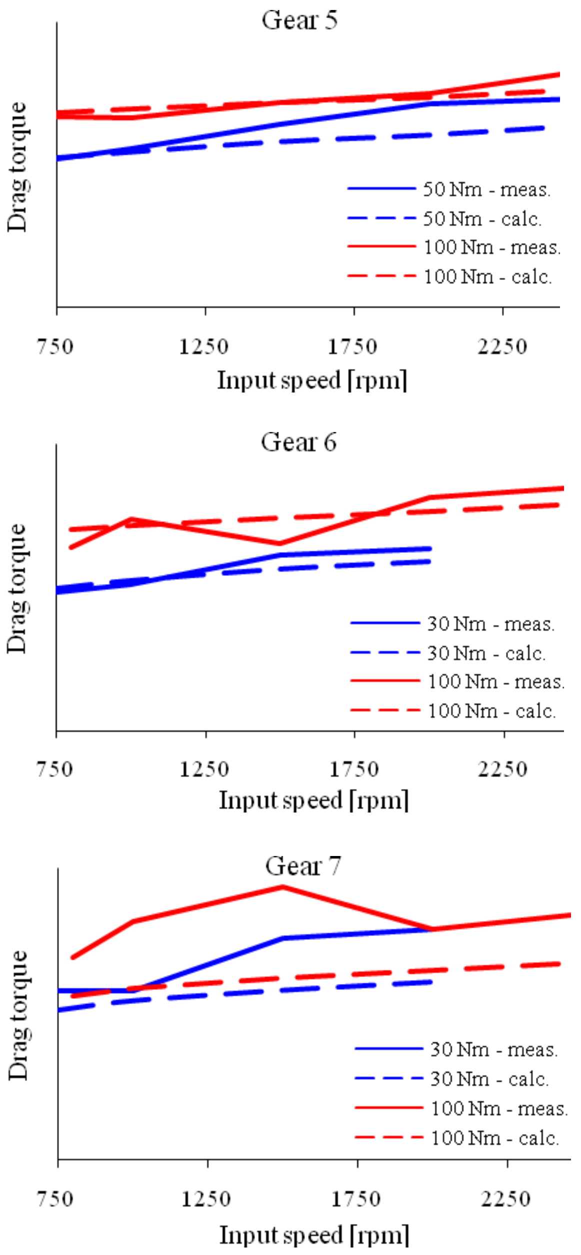


Figure 9b: Comparison of measured and calculated drag torque (80°C), gears 5 to 7

5 Outlook on future work

Our recent investigations focus on 6 and more speed planetary gearboxes. For the calculation of mesh losses and bearing forces, the special relations within the individual planetary gear sets have to be decomposed hierarchically to be able to build models of complex lossy planetary gearboxes out of few base components.

Due to the coupling of many highly nonlinear and in part not continuously differentiable equations describing the drag torques of each single component, numerical problems have to be overcome. They will be investigated in more detail.

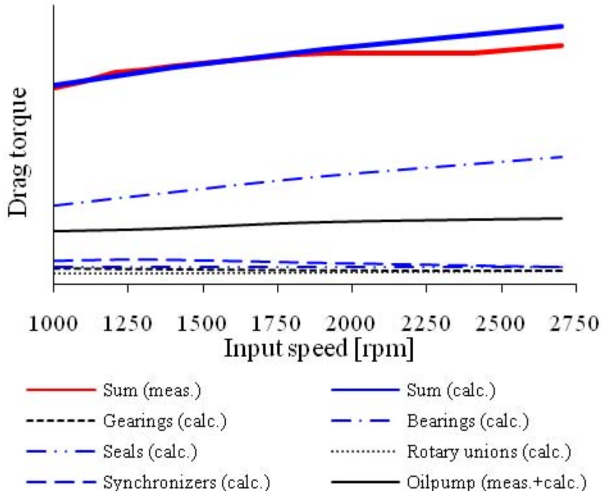


Figure 10: Partial losses (6^{th} gear, 60 Nm, 40°C)

Acknowledgement

The authors would like to thank Mr. Stefan Mayr who reviewed the literature, revised the loss formulas for implementation in Modelica, collected parameter data, tested and verified the models on component and system level during his diploma thesis in mechanical engineering.

References

[1] S. Mayr, Modellierung der Verluste von Getriebekomponenten zur Berechnung von Gesamtgetriebewirkungsgraden. Diplomarbeit TU München und BMW Group, 2008

[2] Forschungsvereinigung Antriebstechnik, Projekt Nr. 69 / I - IV: “Software WTplus”

[3] H. Naunheimer, B. Bertsche, G. Lechner, Fahrzeuggetriebe, Springer, Berlin, 2007

[4] H. Ohlendorf, Verlustleistung und Erwärmung von Stirnrädern. Dissertation TU München, 1958.

[5] L. Schlenk, Untersuchungen zur Fresstragfähigkeit von Großzahnrädern, Dissertation TU München, 1994

[6] A. Wimmer, Verlustoptimierte Verzahnung, FVA Forschungsvorhaben Nr. 372, Heft 731, Abschlussbericht, 2004

[7] W. Mauz, Hydraulische Verluste von Stirnradgetrieben bei Umfangsgeschwindigkeiten bis 60 m/s, Dissertation Uni Stuttgart, 1987

[8] Y. Ariura, T. Ueno, The Lubricant Churning Loss and its Behavior in Gearbox in Cylindrical Gear Systems, Journal of Japan Society of Lubrication Engineers, Vol. 20, No 3, 1975

[9] J. Maurer, Ventilationsverluste, FVA Forschungsvorhaben Nr. 44/VI, Heft 432, Abschlussbericht, 1994

[10] Bearinx ®, Schaeffler KG, Herzogenaurach, Germany

[11] J. Koryciak, Wälzlagerreibmomente, FVA-Forschungsvorhaben Nr. 382, Heft 823, Abschlussbericht, 2007

[12] A. Palmgren, Neue Untersuchungen über Energieverluste in Wälzlagern, VDI-Berichte 20, S. 117-121, 1957

[13] T.A. Harris, M.N. Kolzalas: Roller Bearing Analysis I+II, 5th edition, Taylor & Francis, New York, 2007

[14] INA Wälzlager KG, Basic principles rolling bearings, Friction and increases in temperature, http://medias.schaeffler.de/medias/de!hp.tg.cat/tg_rot*CHEBHCFE;bhHLIZo_3h6b?lang=en

[15] SKF Kugellagerfabriken GmbH, Katalog 2004

[16] H. Linke, Stirnradverzahnung. Hanser Verlag, 1996

[17] J. Kettler, Planetengetriebe-Sumpftemperatur, FVA-Forschungsvorhaben Nr. 313, Heft 639, Forschungsbericht, 2002

[18] M. Gronitzki, Untersuchungen zur Funktion und Auslegung von Rechteckdichtringen für Drehdurchführungen, Dissertation Uni Hannover, 2006

[19] Forschungsvereinigung Antriebstechnik, Projekt Nr. 575 / I: “Synchro Schleppmomente”

[20] www.dynasim.com

[21] T. Skubasz: Untersuchungen von Schleppverlusten an Synchronisierungen, Getriebe in Fahrzeugen 2008, VDI-Berichte 2029, VDI-Verlag, Düsseldorf, 2008

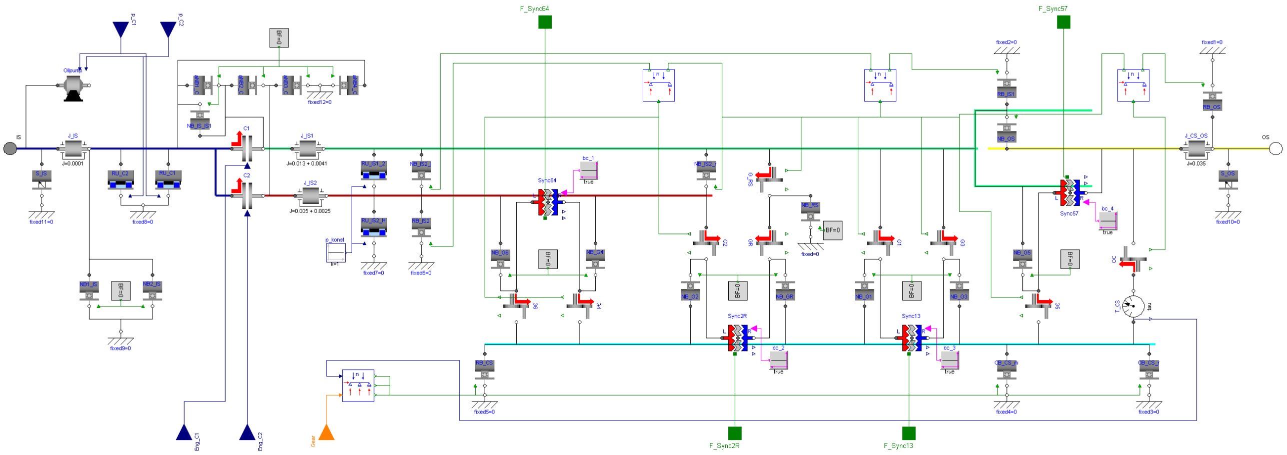


Figure 11: Dymola model of the 7 speed double clutch transmission investigated

Powertrain Torsional Vibration System Model Development in Modelica for NVH Studies

Anand Pitchaikani, Shankar Venkataraman, Kiran Kumar Koppu, John Batteh, Michael Tiller
Emmeskay, Inc
47119 Five Mile Road, Plymouth, MI 48170, USA
anandp@emmeskay.com jbatteh@emmeskay.com mtiller@emmeskay.com

Abstract

For developing high-quality and cost-efficient products, it is important to evaluate and compare system level performance for different configurations early in the development process. This paper will present the development of a vehicle system model in Modelica that is used to study the overall vehicle power-train torsional vibrations that impact Noise, Vibration & Harshness (NVH) characteristics of the vehicle. In this study, a detailed crank-angle resolved, multi-cylinder engine model is constructed, which includes intake/exhaust dynamics, combustion, heat transfer, engine friction and rotational dynamics of piston-crank mechanism. The engine model accurately reproduced real-world engine torque and acceleration fluctuations. The lumped parameter powertrain system model which includes clutch (and associated vibration isolation components), transmission, driveline and chassis is developed and used with the engine model to predict torsional vibrations. This system model is used to understand the powertrain torsional vibration characteristics in different operating regions such as idling, driving and coasting conditions.

To demonstrate the applicability of the developed models, results of unit tests for independent components, especially the engine torque variation and the clutch torsion characteristic, and the system-level quantitative validation with test data are presented. A special model that does fast Fourier transform of the signal on the fly is presented and its role in the analysis discussed. We present a comparison of rattle noise between two compliant clutch (isolator) designs and discuss the rattle metrics used in the analysis. Generic considerations for the deployment of such system level models are also discussed in this paper.

Keywords: gear rattle; engine model; fast Fourier transform

1 Introduction

The noise induced by vibrations of gear-pairs is of great interest to powertrain developers. “Gear rattle”, as this phenomenon is called, is caused by the torsional vibrations of the crank shaft due to engine dynamics (primarily combustion). These cyclic angular accelerations are transmitted from engine to transmission gear pairs and result in undesirable rattle noise. Theoretical and numerical studies are required to improve our understanding of this phenomenon and to enable us to predict such phenomenon so as to improve the underlying design.

Various theoretical and numerical studies on this problem can be found in the literature [1, 2, 3, and 4]. Numerical studies are useful in the automotive industry as they enable a quick analysis of the influence of various parameters and design factors on the dynamic behavior of powertrain. Two numerical methods are usually employed for dynamic analysis of gear rattle. In the “Uncoupled” method, the torsionally loaded path is separated from unloaded gear pairs and is first analyzed. Later, un-loaded gear pairs are modeled as single degree-of-freedom systems where the excitation is that which was obtained in the baseline torsional study. On the other hand, a completely “coupled” model considers contemporary interactions between loaded and unloaded gears. Coupled models are preferred in that they account for all the factors that might have an effect on the rattle noise.

Crowther et al. [5] concluded that metrics measured after impact correlate well, and the relative acceleration between impacting bodies and their relative kinetic energy determine the severity of impact. Several researchers have studied rattle phenomenon in a single maneuver like idling or driving. There are even instances where Modelica® has been used for similar studies [10].

The objective of this modeling effort is to develop an overall generic system model that can be used to study gear rattle phenomenon in various man-

DOI: 10.3384/ecp09430009

euvers. A procedure for development and validation of various powertrain component models is outlined. These validated component models are used to build a complete vehicle model. The models were developed in Modelica using the Dymola® environment. The condition of vehicle coasting with idle speed control is studied in detail using the developed models. For this maneuver, rattle metrics obtained from the simulation model are shown to correlate well with experimental noise measurement. Further based on the understanding provided by the model, an improvement in the isolator design is introduced to reduce the rattle noise during this condition. Simulation and test results that confirm the reduction in rattle noise as a result of this design change are presented.

2 Vehicle Architecture

The vehicle system model is comprised of engine, compliant clutch (isolator), transmission, driveline and chassis components connected in series with a provision to mount the engine and transmission through bearings (Figure 1). The detailed crank-angle resolved engine model produces a fluctuating torque that is required to carry out NVH studies.

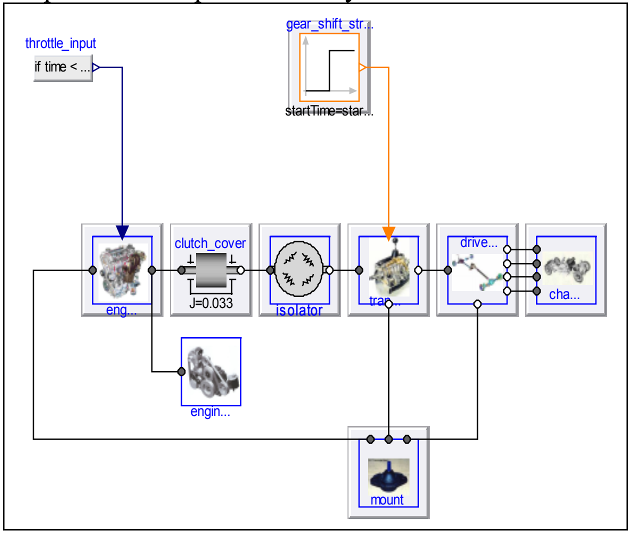


Figure 1 Vehicle Architecture

The non-linear, multi-stage isolator model isolates the engine fluctuations before they get transmitted downstream. The transmission is modeled with certain gears fixed on the input shaft and others on the output shaft. The simple chassis model consists of a lumped vehicle mass and road loads connected by a kinematic tire model. The effects due to tire and suspension compliance are neglected. In the driveline model, a front-wheel-drive system is modeled. The front half shafts are modeled as non-symmetric compliances connecting the transmission to the wheels.

3 Component Models

3.1 Engine Model

A reasonable approach for powertrain torsional vibration analysis is to use experimental engine cylinder pressure data for the various drive conditions as inputs to the vehicle system model. However, especially in the early vehicle development phase, it is difficult to generate pressure traces for engines that are either in development, or for which the available data is limited. Hence it is important to have a detailed physical engine model to predict engine torque fluctuation [8, 9]. This is especially important during transient operation such as cranking, when predicting cylinder pressures would be difficult without a model. Such models can even be used to study the effect of engine manufacturing variation on the engine performance much before the engine goes into production [11]. Another important consideration is that by integrating a detailed transient engine model in the vehicle system model, the drive conditions that involve interactions with the engine control strategy such as idle speed control can be studied. Finally, for high speed operation, pressure data alone is not sufficient since the inertial forces of the piston become significant. The detailed engine model would account for the interaction between the combustion force and the piston inertial force.

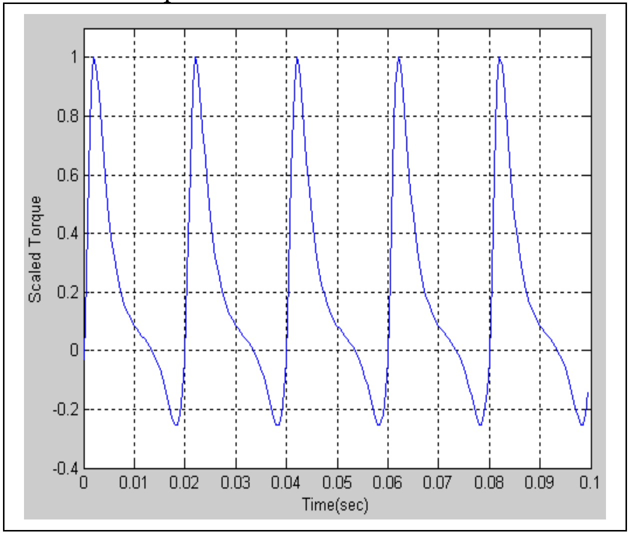


Figure 2 Sample crankshaft torque simulation.

The engine model used in this work was developed in-house to simulate both spark-ignited and diesel engines. The model is parameterized such that relevant engine design parameters can be easily entered into the model from a customized GUI. Figure 2 shows a typical crankshaft torque prediction using the engine simulation model.

This model includes crank angle-resolved, multi-cylinder modeling of the main engine thermodynamics and rotational mechanics including:

- Intake and exhaust breathing
- Combustion
- Heat transfer
- Rotational dynamics of piston, crank-slider mechanism, and crankshaft
- Engine friction (look-up table based)

3.1.1 Model Structure

Figure 3 shows the top-level structure of the detailed engine model. This model contains the hierarchical engine model, the flywheel inertia, a rotational connector for connecting the crankshaft to downstream components, and a signal bus connector used to provide relevant control signals to the engine model.

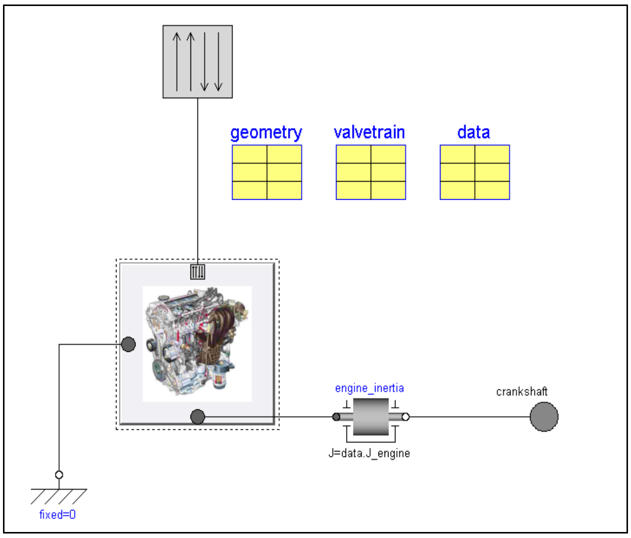


Figure 3 Engine model

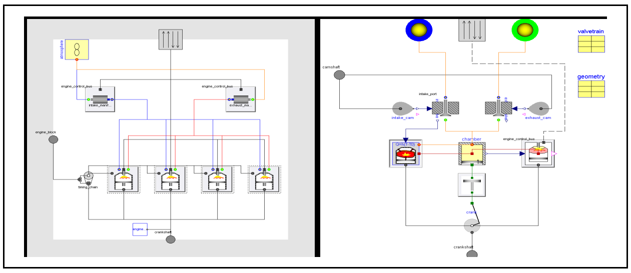


Figure 4 Hierarchical engine model structure.

Figure 4 shows the engine model being hierarchically composed of cylinders, with each cylinder consisting of models for the intake, exhaust and cylinder components. The model has another mechanical connection to the powertrain mounts which accounts for the reaction torque transmitted to the powertrain mount system.

3.1.2 Parameterization

The engine model parameters include engine-specific design data, initial and boundary conditions, and advanced parameters to customize the heat transfer and combustion characteristics of the engine. The parameters are summarized as follows:

- Parameters for initial conditions
- Engine geometry for overall engine specification
- Valve train for specification of the valve geometry and cam timing
- Engine friction look-up table for modeling friction torque (mechanical/rubbing) as a function of engine speed
- Manifold conditions that specify intake manifold temperature and exhaust manifold pressure and temperature
- Heat transfer parameters related to in-cylinder heat transfer
- Combustion parameters for specifying burn rate profile

3.1.3 Engine controller

Dynamic operating conditions are provided to the engine models via an engine controller. The engine controller takes a normalized torque input (0 to 1) and determines the dynamic operating conditions to match the engine torque profile based on a mapping process using the engine model. The engine controller component model takes the normalized torque as input and outputs the dynamic operating conditions to the control bus which will be provided to the engine model. The dynamic operating conditions specified by the controller are as follows:

- Intake manifold pressure (throttling/boost)
- Air-fuel ratio
- Firing flag (true or false)
- Start of combustion (for diesel)
- Spark advance and burn duration (for SI)

3.2 Isolator Model

The isolator model captures the vibration isolation characteristics of the clutch assembly. It is a non-linear spring-damper, modeled in three stages. Each stage is parameterized to incorporate spring stiffness and hysteresis. It is modeled in such a way that the effect of first stage spring is always felt at the output of the isolator, whereas the springs in the second and third stage will be in effect (in parallel to the first stage), based on whether or not the respective backlash is taken up. These stages engage when the relative angle between the input and output shaft of isolator exceeds the specified values of backlash. The model also includes the inertia of various components of clutch like the clutch disc, clutch facing, hub, and flange. This generic model allows a particular stage effect to be present only on one direction (positive/negative torque) by specifying asymmetric values to the specially made backlash element for the positive (clockwise) and negative (anti-clockwise) sides. This helps in parameterizing the model accurately for any isolator characteristic data available. The model for a three-staged isolator is shown in Figure 5.

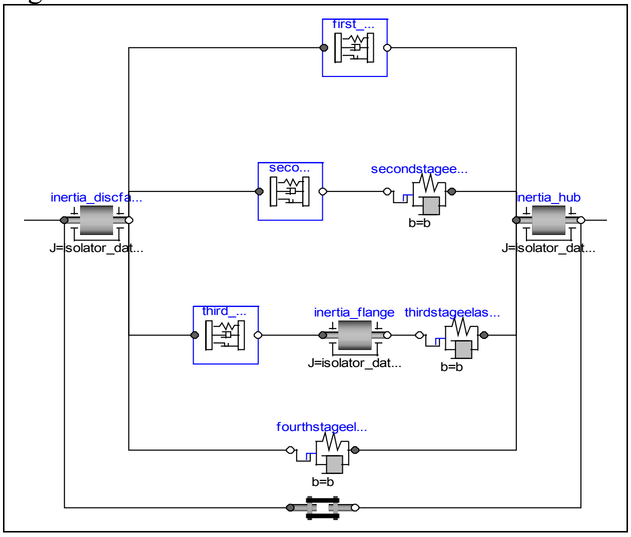


Figure 5 Multi-stage clutch (Isolator) model

3.3 Transmission Model

The transmission subsystem represents the gearing involved in delivering power from the engine to the wheels. One side of the transmission is connected to the engine while the other side is connected to the driveline. Like the engine, the transmission is also connected to the powertrain mounts. The consideration of the mounts is an important aspect that differentiates this architecture from many vehicle level models because it accounts for the influence of reaction torques in the power plant, transmission and driveline on the motion of the powertrain. This part of the physics is particularly important for the transmission because it can be the source of large amplitude, low frequency disturbances not effectively isolated by the mounting system [6, 7].

The transmission model is built from basic components of Modelica Standard Library (MSL) that includes a gear pair with a synchronizer (clutch) in between. Both of the gears in this component are fixed either to the input or output shaft. Any transmission architecture can be modeled by using combinations of this gear pair component.

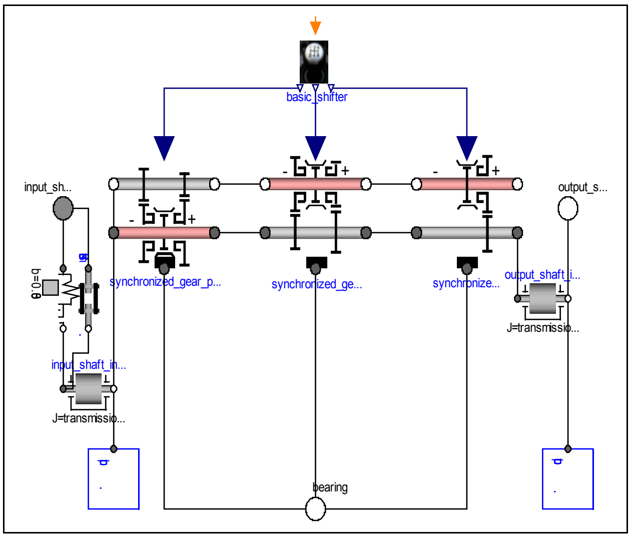


Figure 6 Transmission model

As seen in Figure 6, the transmission model includes the input and output shaft inertias. The shifter component sends the gear selection command to the appropriate gear pair in which the gear needs to be engaged. The commanded gear is then engaged by the synchronizer onto the shaft on which it is floating in neutral condition.

The model includes the coulombic drag torque associated with the input and output shafts. In physical terms, this drag comes from the stir resistance of oil around the transmission input and output shafts. There is a provision in the model to include the backlash between the isolator and the transmission input shaft. If not required, this effect can be eliminated by activating the rigid-bypass model connected in parallel to the backlash model. Rigid-bypass model is capable of locking-up the flanges that are connected to its two sides. This component does not reflect a physical component in the system, it is merely a way to control what details are included in the model.

The gear pair component has the flexibility to include the effect of backlash between the mating gears. The details regarding this backlash model are provided as a code fragment below. The backlash model is instrumented with the following metrics.

- Average_omega: Average relative angular velocity between the mating gears
- Average_alpha: Average relative angular acceleration between the mating gears
- Average_power_contact_damper: Average power associated with contact damper between the mating gears

These metrics are calculated right after each impact between the gear pair. They are averaged over the number of impact events within a chosen time window.

```
model InstrumentedBacklash
  "Backlash w/rattle instrumentation"
  import Modelica.Mechanics.Rotational;
  import Modelica.SIunits.*;
  extends Rotational.ElastoBacklash;
protected
  Real event_rate "Collisions/sec";
  Integer events "Collisions/interval";
  AngularVelocity omega_rel =
                  der(phi_rel);
  AngularAcceleration alpha_rel =
                  der(omega_rel);
  AngularVelocity avg_omega
    "Average rel omega per collision";
  AngularVelocity omega_sum;
  AngularAcceleration avg_alpha
    "Average rel alpha per collision";
  AngularAcceleration alpha_sum;
  Power power_sum;
  Power avg_power
    "Average power per collision";
algorithm
  when sample(0,0.02) then
    event_rate := events/0.02;
    if (events == 0.0) then
      avg_omega := 0.0;
      avg_alpha := 0.0;
      avg_power:= 0.0;
    else
      avg_omega := omega_sum/events;
      avg_alpha := alpha_sum/events;
      avg_power:= power_sum/events;
    end if;
    events := 0;
    omega_sum := 0;
    alpha_sum := 0;
    power_sum:= 0;
  end when;
  when phi_rel>=b2 or phi_rel<=-b2 then
    events := pre(events) + 1;
    omega_sum := pre(omega_sum) +
                 omega_rel;
    alpha_sum := pre(alpha_sum) +
                 alpha_rel;
    power_sum := pre(power_sum) +
                 omega_rel*tau;
 end when;
end InstrumentedBacklash;
```

3.4 Driveline Model

The driveline subsystem models the distribution of transmission output torque to each of the wheels. For many vehicles, this distribution is determined by simple mechanical connections (e.g. differentials in strictly front-wheel or rear-wheel drive vehicles). In other cases, this distribution is actively controlled (e.g. on-demand four wheel drive systems). In this study, a non-active front-wheel drive system is considered. The transmission output shaft connects to a final drive gear followed by a differential that splits the torque to the front two wheels as shown in Figure 7. The final drive gear as well as the differential gear models are taken from the Modelica Standard Library. The driveline subsystem is connected to the mounting system as well. The right and left half shafts are modeled with inertia and compliance.

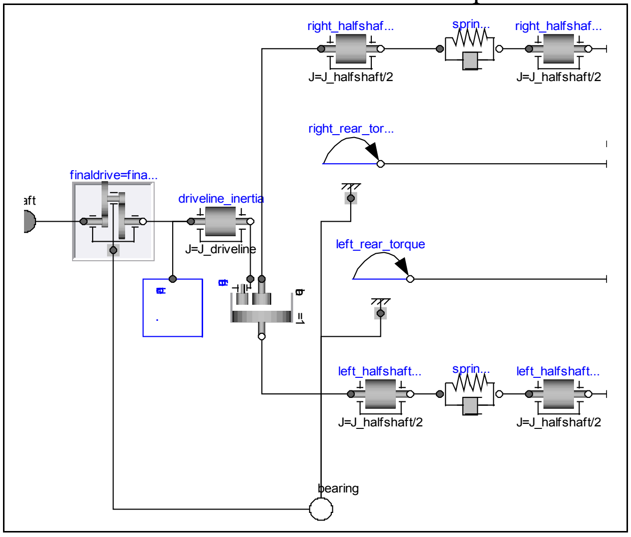


Figure 7 Driveline model

3.5 Chassis Model

The chassis models the tire inertia and neglects the stiffness of the tire and the compliance of the shock absorber. The aerodynamic drag and rolling resistance are modeled using representative equations. The vehicle mass is represented as a translational mass.

4 Component Validation

To study driveline torsional vibration, it is imperative to have an accurate prediction of the engine torque fluctuation, as well as an accurate prediction of the behavior of the transferring system namely the isolator. In this section, validation of the engine and isolator components is presented.

4.1 Engine Model Validation

The vehicle system model has to be employed to study the powertrain torsion characteristics for three different conditions namely, idling, driving and coasting. For this purpose, unit tests were carried out to validate the engine model against engine test data at each of these three conditions.

4.1.1 Engine Vibration at Idling

Here the engine is at steady-state idling condition with engine friction, A/C and alternator loads being applied. Figure 8 shows the acceleration fluctuation levels at the specified idling speed for 2 different engine load levels. The model results show good agreement with the test results.

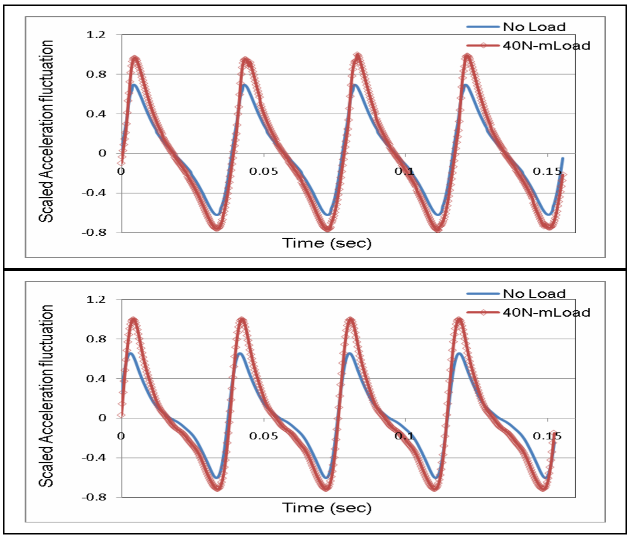


Figure 8 Experimental (Top) vs. Model (Bottom) results for engine vibration at idle condition.

4.1.2 Engine Vibration at Driving

For the engine in driving condition, both acceleration and deceleration validation tests were performed. The acceleration test is done on the vehicle model, with the engine and vehicle accelerating and the transmission in third gear. By sweeping the engine through the operating range (via a tip-in throttle command), the speed fluctuation values at various engine speeds were captured. For the deceleration test, the tip-out maneuver is executed from a higher engine speed.

To facilitate this analysis, a Fourier Transform computation block in Modelica was developed in-house. This model is connected to the engine crankshaft and performs FFT (Fast Fourier Transform) on the engine speed signature on-the-fly. The Fourier transform is computed for a user-specified fundamental frequency and specified number of harmonics of this fundamental frequency. The model computes the continuous Fourier integrals which are used to compute Fourier coefficients, which in turn are used to compute the magnitude and phase for the specified frequencies.

Figure 9 shows the speed fluctuation of the dominant second order harmonic against engine speed. It is noted that the engine speed fluctuation first decreases and then increases as a function of nominal speed. The point at which this trend reversal in speed fluctuation happens is dependent on the relative magnitudes of the inertia forces due to the piston mass and the combustion force.

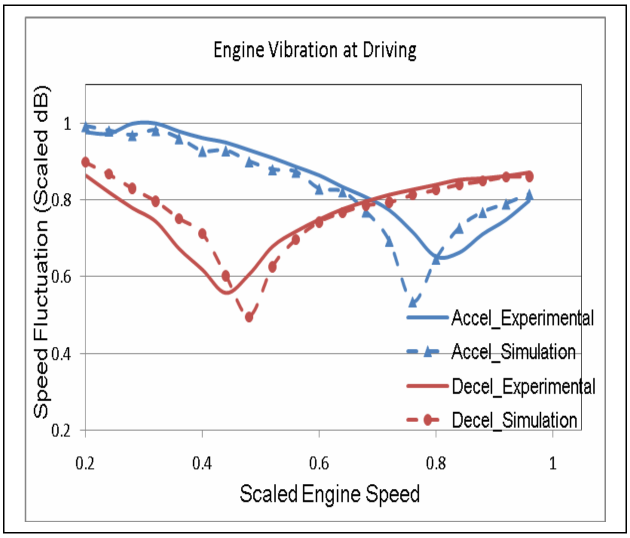


Figure 9 Experimental vs. Model results for engine driving vibration

As seen in Figure 9, a good correlation between the simulation and experimental results for engine driving vibration during both acceleration and deceleration is observed.

4.1.3 Engine Vibration at Coasting

To illustrate the applicability of the vehicle system model to study vibration phenomenon under various conditions, a generic maneuver different from idling or driving, namely coasting was considered. In this maneuver, the transmission is engaged in third gear and the engine is initially at a speed higher than idle speed. The driver lets off the throttle with the transmission still engaged, resulting in the engine speed decreasing due to engine braking.

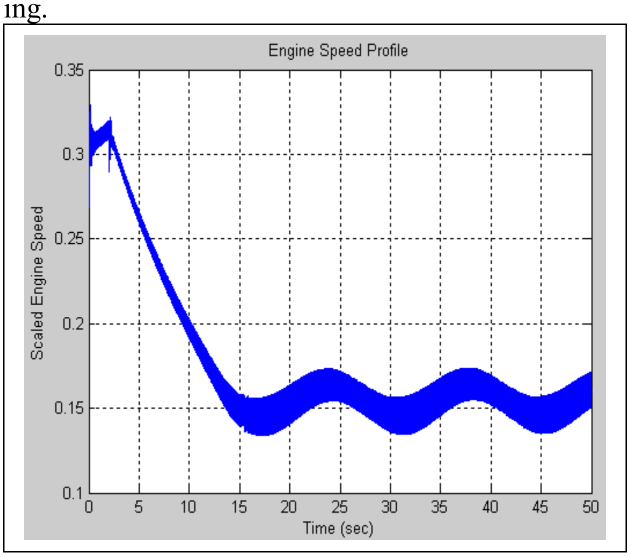


Figure 10 Engine speed prediction during coasting

When the engine speed comes down close to idle, Idle Speed Control (ISC) takes over. To be able to study rattle behavior during such a maneuver, it is first essential to ensure that the engine model is able to simulate the engine speed fluctuation accurately at different stages in the maneuver. For this purpose, an appropriate engine model control to capture the exact behavior during coasting condition is developed. Figure 10 shows the predicted engine speed during coasting condition that includes engine braking and ISC regions. It is noted that the model is able to adequately represent both the mean value as well as the variation in the engine speeds as a qualitative validation of the model result with experimental data was performed.

4.2 Isolator Model Validation

The unit test for the isolator component uses a low frequency sinusoidal torque as input to the isolator model. Figure 11 shows isolator torque plotted against isolator relative angle. The experimental curve was used to determine the parameters for isolator model such as backlash (which governs the onset for each stage) and the stiffness and hysteresis values for each stage.

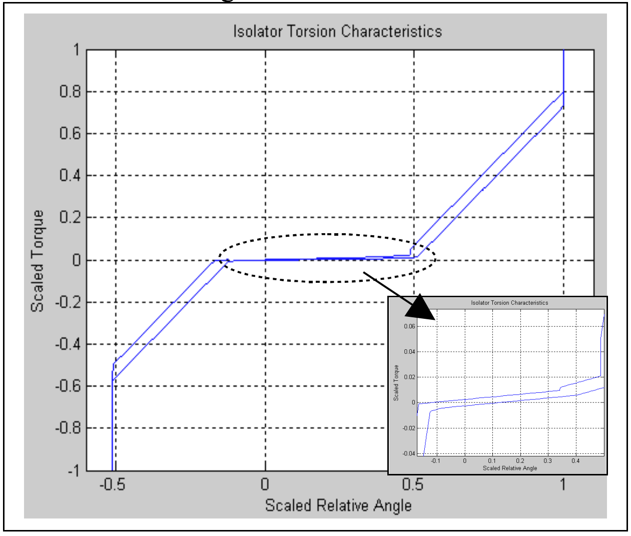


Figure 11 Isolator torsion curve

It is observed that this particular isolator design has 3 stages on the positive side but only the first and third stage on the negative side (see the inset picture of Figure 11).

5 Vehicle System Validation

In the previous sections, component model development and validation for the vibration source component (engine), the vibration transfer component (isolator) and the noise source component (transmission and driveline) was discussed. After the component validation is done for different vehicle conditions, the vehicle system vibration validation and analysis at these conditions needs to be performed. This section presents the validation results for the vehicle system at idle, driving and coating conditions.

5.1 Idle Condition Validation

This test is done on the vehicle model with the engine idling and the transmission in neutral, and hence is intended to show the torsional vibration isolation provided by the isolator first stage.

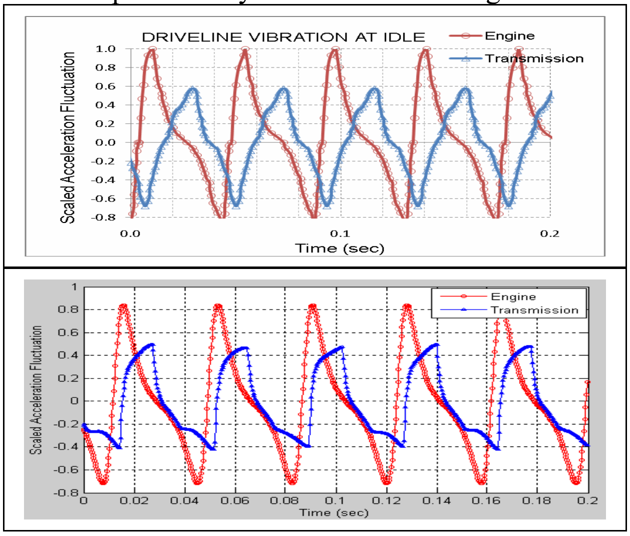


Figure 12 Experimental (top) vs. Model (bottom) results for driveline vibration during idling

Figure 12 shows the torsional vibrations being dampened by the isolator from the engine to the transmission input shaft. The amplitudes at the engine and transmission shaft show good agreement between simulation and experimental data.

5.2 Driving Condition Validation

This test case is very similar to the engine acceleration test, where the engine speed is swept and the corresponding speed fluctuation of engine is captured. Here the speed fluctuation for the transmission is also captured. In this test, the transmission resonance where the transmission speed fluctuation overshoots the engine speed fluctuation near the transmission resonant speed has to be observed. The effect of system parameters such as isolator hysteresis on the system resonance is of great interest.

A frequency domain analysis of the vehicle system model can determine the resonant speed at which the engine speed fluctuation excites the transmission. A frequency analysis of this system will provide us the different resonant speeds of the system.

To perform this analysis, the model linearization functionality of Dymola was used to generate a linear time invariant system from the developed models.

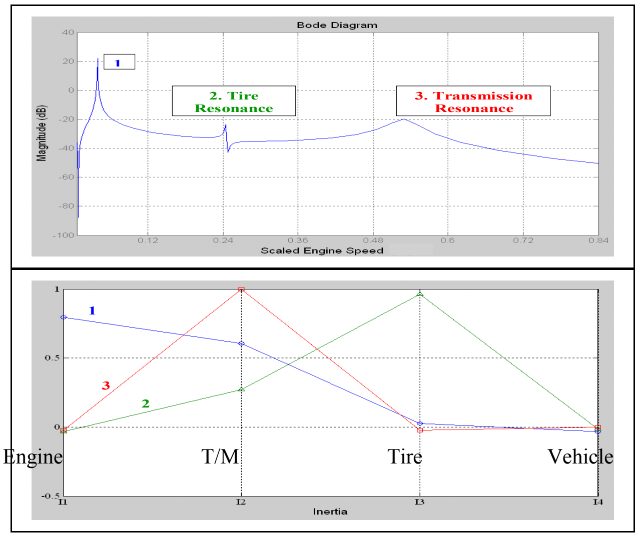


Figure 13 Bode & mode shape plots obtained from linearized model

Figure 13 shows the Bode plot and mode shape plot obtained after vehicle model linearization. The Bode plot represents the transfer function or frequency response of a linear, time invariant system and identifies the system resonant frequencies. The mode shape plot helps identification of the particular system components that are involved in each of the resonant frequencies. For example, the resonance frequencies arising from the transmission and the tire inertias are identified in the Bode plot.

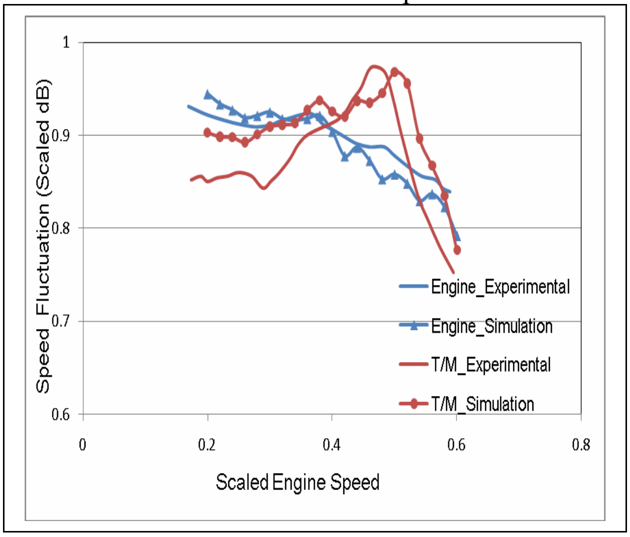


Figure 14 Transmission Frequency Response

After finding the resonant speed, the nonlinear system model is used to find out the magnitude of speed fluctuation for both the engine and transmission. It is observed from Figure 14 that at around the predicted resonant speed, the speed fluctuation of the transmission increases well over that of the engine. The predicted frequency response agrees well with the experimental data as well.

6 Model Based Design

One advantage of a model-based design approach is the ability to analyze the effect of a design change via simulation rather than requiring fabrication and testing of actual hardware. The original isolator design has the characteristics shown in Figure 15. This isolator was observed to have a rattle noise peak during the experiment in test bed. The modified isolator with improved characteristics as in Figure 15 was shown to eliminate the rattle noise peak by the results obtained from the simulation of developed models.

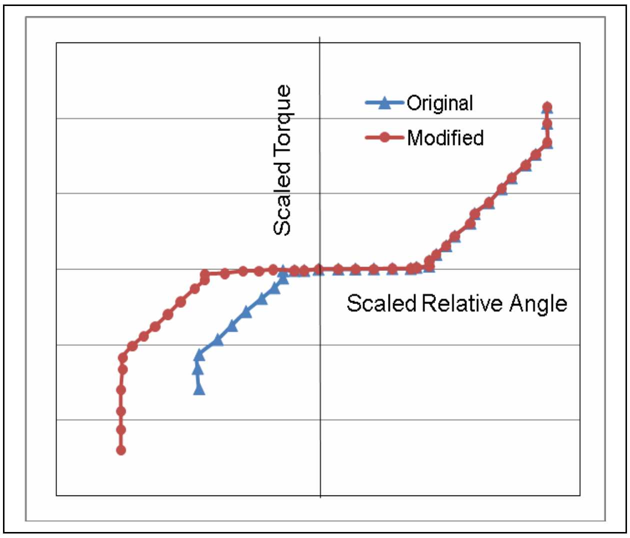


Figure 15 Comparison of Torsion Characteristic of the Original and Modified Isolator

The original isolator design did not include a second-stage spring on the negative side and this resulted in a sudden transition from the negative main-damper (third stage) to negative pre-damper (first stage) which led to rattle. The isolator design can be improved with the addition of a second stage spring in the negative direction as shown in Figure 15 in red color. This design change is implemented in the models and simulated to observe the new rattle metrics.

Figure 16 shows the comparison of the rattle metrics for the original and modified isolators for the driving gear. These rattle metrics were captured in the gear models of the transmission sub-system as explained in section 3.3. Clearly, the magnitudes of the rattle metrics are reduced in the case of the modified isolator, a result which again validates the design proposal for eliminating rattle.

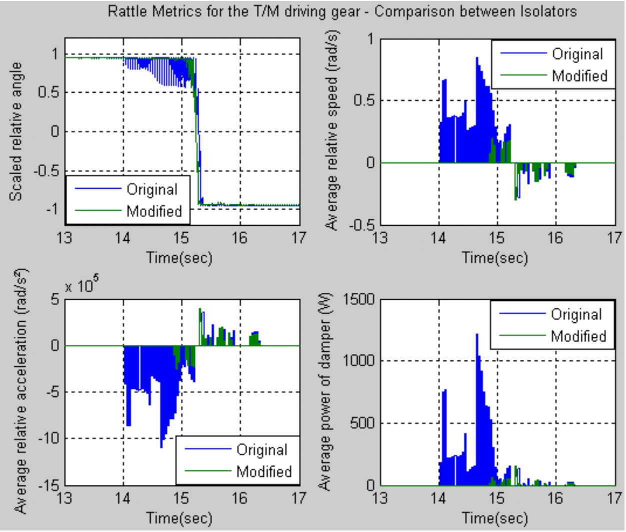


Figure 16 Comparison of Rattle Metrics with the Original and Modified isolator for the T/M driving gear

The conclusions drawn from the model results (Figure 16) were found to agree with the experimental results obtained from the test bed as shown in Figure 17.

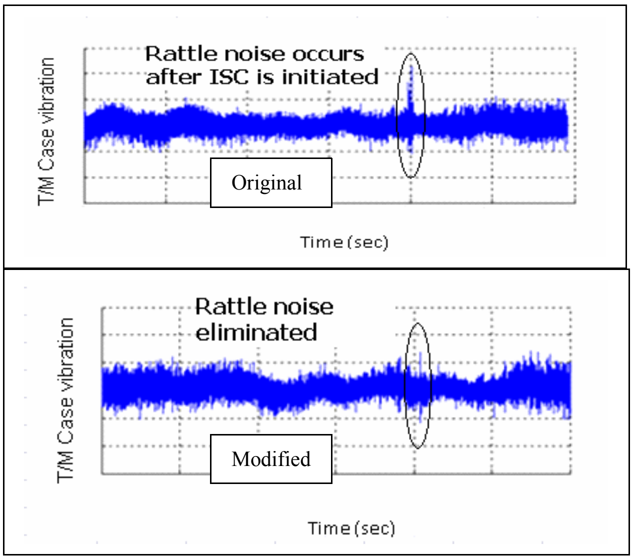


Figure 17 Comparison of vibration between original and modified isolator in test bed

7 Conclusion

In this work, a systematic approach for development and application of an overall vehicle system simulation model to study the powertrain torsional vibration characteristics using Modelica is presented. The resulting comprehensive vehicle system model is capable of analyzing a wide range of vehicle operating conditions such as idling, driving and coasting. Comprehensive tests were done both at the component level and overall system level to demonstrate quantitative model validation.

To illustrate the applicability of the developed system model, a vehicle coasting maneuver was studied in detail and it was found to introduce rattle noise in the system. Having an overall system model was important since it captures the intimate interactions between subsystems. In this particular case, the system model clearly brought out the influence of isolator design on backlash resonance in the transmission. The observation of rattle metrics demonstrated the effectiveness of the model in the identification of a design change in the isolator to eliminate rattle during the maneuver. The effectiveness of the design change was confirmed by vehicle testing. The use of Modelica in modeling a complex system and its use in analyzing complex phenomenon is demonstrated in this work.

In terms of future applications, the simulation model instrumented with the identified rattle metrics could be used to assess likelihood of rattle noise in other vehicle maneuvers without the need to run costly vehicle tests. The system model has been developed in such a way that the component models are readily replaceable. It would be fairly straightforward to integrate alternative components such as a dual mass flywheel (DMF) in the architecture. In this paper, the effectiveness of a design change in the clutch disk isolator towards reducing rattle noise is shown. Similarly design changes in other isolation components such as engine and transmission mounts can be studied in future.

8 Acknowledgements

The authors wish to thank Dr. Swaminathan Gopalswamy and Mr. Sundaresan for their technical inputs throughout the work. The authors also wish to thank Dr. Yasunori Yokojima for his invaluable contribution to this work.

References

[1] R. Brancati, E. Rocca, R. Russo, An analysis of the automotive driveline dynamic behavior focusing on the influence of the oil squeeze effect on the idle rattle phenomenon, Journal of Sound and Vibration 303 (2007) 858-872.

[2] Y. Wang, R. Manoj, W. J. Zhao, Gear rattle modeling and analysis for automotive manual transmissions, Proceedings of Imech, Journal of Automobile Engineering 215 (part D) (2001) 241–258.

[3] R. Singh, H. Xie, R. J. Comparin, Analysis of automotive neutral gear rattle, Journal of Sound and Vibration 131 (2) (1989) 177–196.

[4] R. Brancati, E. Rocca, R. Russo, A gear rattle model accounting for oil squeeze between the meshing gear teeth, Proceedings of Imech 219 (part D) 1075-1083.

[5] A. R. Crowther, C. Janello, R. Singh, Quantification of clearance-induced impulsive sources in a torsional system, Journal of Sound and Vibration 307 (2007) 428-451.

[6] M. Tiller, W.E. Tobler, and M. Kwang, Evaluating Engine Contributions to HEV Driveline Vibrations, 2nd International Modelica Conference (2002) Proceedings, PP 19 – 24.

[7] Michael Tiller, Paul Bowles, Mike Dempsey, “Development of a Vehicle Modeling Architecture in Modelica”, 3rd International Modelica Conference (2003) Proceedings, pp. 75-86.

[8] Newman, C., Batteh, J., and Tiller, M., 2002, "Spark-Ignited-Engine Cycle Simulation in Modelica", 2nd International Modelica Conference Proceedings, pp. 133-142.

[9] John Batteh, Michael Tiller and Charles Newman, “Simulation of Engine Systems in Modelica”, 3rd International Modelica Conference Proceedings, pp. 139-148.

[10] M. Dempsey, S. Biggs and N. Dixon, Simulating driveability using Dymola and Modelica, 4th International Modelica Conference Proceedings, http://www.modelica.org/events/Conference2005/online_proceedings/Session3/Session3a4.pdf

[11] J. Batteh and M. Tiller, "Analytic Evaluation of Engine NVH Robustness Due to Manufacturing Variations", ASME 2005 Internal Combustion Engine Division Spring Technical Conference, pp. 429-437

[12] Modelica Specification, version 3.0, www.-modelica.org/documents/ModelicaSpec30.pdf

[13] M. Tiller, "Introduction to Physical Modeling with Modelica", Kluwer Academic Publishers, ISBN 0-7923-7367-7

[14] Modelica, http://www.Modelica.org.

A Petri Net Library for Modeling Hybrid Systems in OpenModelica

Sabrina Proß Bernhard Bachmann

University of Applied Sciences Bielefeld
Am Stadtholz 24
33609 Bielefeld, Germany

sabrina.pross@fh-bielefeld.de bernhard.bachmann@fh-bielefeld.de

Abstract

For modeling continuous and hybrid Petri Nets with dynamic edge weightings, the already existing Petri Net Libraries were further developed. The new library was implemented in OpenModelica using the SimForge GUI, however it works also with Dymola. With the extensions it is possible to model complex biological as well as production or traffic systems.

1. Introduction

The Petri Nets formalism was first introduced by Carl Adam Petri in 1962 [1]. Today Petri Nets can be found in many different areas. Modeling traffic light crossings, production processes or metabolisms of bacteria are only a few examples. In the last years, they were more and more extended for using them for different kinds of problems.

The first Petri Net Library in Modelica was developed by Mosterman et al. and has been further improved by Otter et al. [2; 3]. Herewith the modeling of “normal” Petri Nets or so-called statecharts is possible. “Normal” Petri Nets are bounded and the Places have the capacity one. No time is associated with their behavior. An external signal can enable or disable the Transitions.

This Petri Net Library was further developed by Fabricius [4]. The extensions are:

- Places can contain an integer number of Tokens.
- The Transitions can be timed. They can have either deterministic or stochastic delays.

The Petri Net Library of the present paper bases on the previous ones. The improvements are:

- Continuous Petri Nets with real numbers of Tokens and continuous firing.
- Continuous and discrete Petri Net elements can be connected to model hybrid Petri Nets.
- The edges can have integer weightings in the discrete case and real ones in the continuous case.
- The edge weightings can be functions.
- The Places can contain a maximum and a minimum amount of Tokens.
- Each edge can have an upper and a lower boundary. The number of Tokens of the respective Place must be between these values so that the connected Transition can fire.
- In the discrete case: If a Place does not contain enough Tokens to fire in all possible Transitions, a random variable decides in which Transitions the Place fires. It is the same if a Place cannot gain Tokens from all possible Transitions because of its maximum value.

Firstly, the new Petri Net Library was developed to model biological systems. Metabolites, enzymes and genes are modeled with Places and Transitions represent the reactions between them [5].

Biochemical reactions, which convert one substance to another, proceed continuously. In order to model these, continuous Petri Net elements were implemented.

Furthermore, the speed of these reactions depends mostly on the current concentration of specific substances which can be now displayed by dynamic edge weightings [6].

Additionally, it should be possible to model gene regulation which contains discrete processes as well as continuous ones. Hybrid Petri Nets, which comprise both discrete and continuous Petri Net elements, are now able to model this [7].

DOI: 10.3384/ecp09430014

The edges can also have upper and lower boundaries. This is necessary for modeling substances which only react when a specific concentration is reached.

In the further development of this library, it became clear that these extensions are not only specific for biological systems. They are also useful in other areas. The present paper illustrates this with an example of the steel production process.

2. Petri Nets

A Petri net is a graphical construction to describe and analyze concurrent processes and non-deterministic procedures. It is a graph with two different kinds of nodes: Places and Transitions, whereas only a Place can be connected with a Transition or a Transition with a Place. A Place is symbolized with a circle and a Transition with a rectangle (see Figure 1).

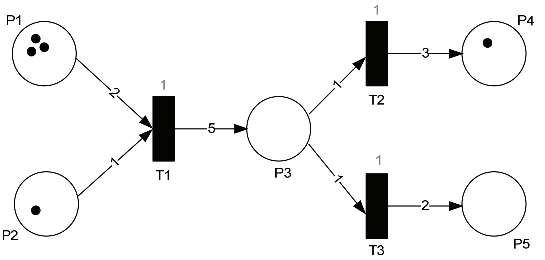


Figure 1: A Petri Net

Every Place contains an integer number of Tokens. In Figure 1, they are shown as black dots. The edges of a Petri Net are provided with weightings and the Transitions with delays. In Figure 1, these are the black numbers at the edges and the grey numbers over the Transitions, respectively. A delay represents the time units that a certain process takes.
A Transition *T* is ready to fire when every Place in its previous area has at least as much Tokens as the edge weighting from the certain Place to *T*. In Figure 1, Transition *T1* is ready to fire because *P1* has three Tokens and must have at least two Tokens, whereas *P2* must contain at least one Token which it actually has. *T2* and *T3* are not ready to fire.
A Transition *T,* that is ready to fire, fires by removing so many Tokens dependent on the respective edge weighting from all of the Places in its previous area. In addition, a specific number of Tokens is laid down in relation to the edge weighting to all of the Places in its past area. After firing *T1* in Figure 1 *P1* has one Token, *P2* zero, *P3* five, *P4* one and *P5* zero (see Figure 2).

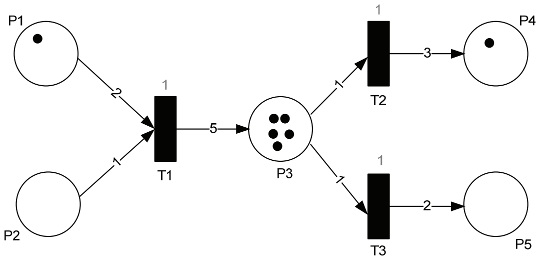


Figure 2: The Petri Net of Figure 1 after firing *T1*

Assumed a Place has only one Token, like in Figure 3, this Token can be either fired in Transition *T1* or in Transition *T2*. Therefore, a decision is necessary which Transition is chosen. One possible solution is that a uniform distributed random variable decides whether *T1* or *T2* gets the Token. It is also thinkable that the edges are weighted. In the example shown in Figure 3, Transition *T1* is chosen with a probability of 80% and *T2* with 20%, respectively.

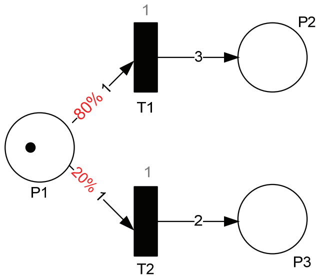


Figure 3: Example for output weightings

One possible extension are Petri Nets with capacities: each Place can only contain a maximum amount of Tokens and must always have a minimum amount of Tokens.
Furthermore, the edges can have threshold and inhibition values. In Figure 4 these are the red numbers in brackets. The first value is the threshold and the second is the inhibition. A Transition is only ready to fire if the connected Places have more or as much Tokens as the threshold value and less or as much as the inhibition value. In Figure 4 is the Transition *T1* ready to fire because this Tansition is only connected with *P1* and *P1* contains three Token. This is more than two and less than five. The Transition *T2* is not ready to fire because the threshold value of the connecting edge between *P1* and *T2* is not achieved.

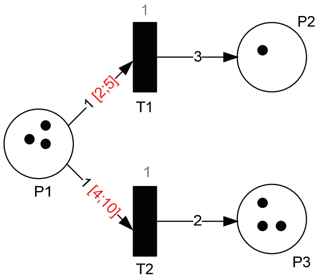


Figure 4: Example for threshold and inhibition values

An additional extension are self modifying Petri Nets which were firstly introduced by Valk [8]. The edge weightings are now functions, which can depend on the current number of Tokens of the respective Places (see Figure 5).

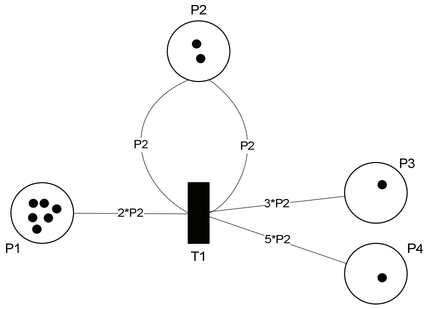


Figure 5: A self modifying Petri Net

2.1. Stochastic Petri Nets

The only difference between Petri Nets described above and stochastic Petri Nets is that the delay is a random variable instead of a fixed value. This random variable can be for example exponential or normal distributed. In the latter case it has to be avoided that the random variable is negative.

2.2. Continuous Petri Nets

A continuous Petri Net is a graphical representation of a differential equation system with the properties of a Petri Net.

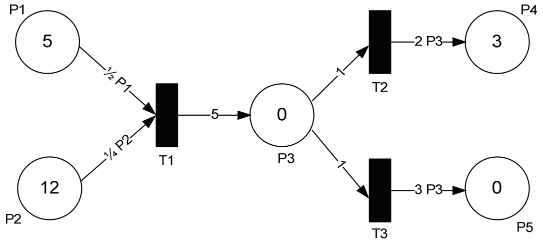


Figure 6: A continuous Petri Net

The number of Tokens in each Place can be real and changes continuously. The edge weightings represent the firing speed for the different branches. The sum of the incoming and outcoming speeds is proportional to the change of Tokens.

The continuous Petri Net in Figure 6 is the graphical representation of the following differential equation system:

$$\frac{\mathrm{dP1}}{\mathrm{dt}} = -\frac{1}{2}\cdot \mathrm{P1}$$

$$\frac{\mathrm{dP2}}{\mathrm{dt}} = -\frac{1}{4}\cdot \mathrm{P2}$$

$$\frac{\mathrm{dP3}}{\mathrm{dt}} = 5-1-1=3$$

$$\frac{\mathrm{dP4}}{\mathrm{dt}} = 2\cdot \mathrm{P3}$$

$$\frac{\mathrm{dP5}}{\mathrm{dt}} = 3\cdot \mathrm{P3}.$$

The difference between the continuous Petri Net and the differential equation system is that if one of the Places in the previous area of the Transition is empty, the Places in the past area will not gain Tokens anymore. If for example *P1* in Figure 6 is empty, then *P3* will not gain any Tokens.

2.3. Hybrid Petri Nets

Hybrid Petri Nets contain discrete and continuous elements. A discrete Transition can be connected with a continuous Place or a continuous Place with a discrete Transition. Connections between discrete Places and continuous Transitions are forbidden.

If a continuous Place is connected with a discrete Transition, the Transition fires by decreasing Tokens continuously in the time of the delay. The slope of the graph is calculated by dividing the edge weighting by the delay. If a discrete Transition is connected with a continuous Place, the Transition fires by adding Tokens continuously in the time of the delay.

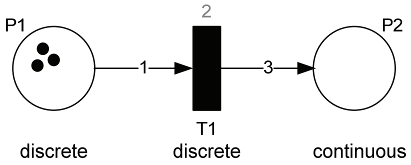


Figure 7: A Hybrid Petri Net

Figure 7 is an example of a hybrid Petri Net. *P1* and *T1* are discrete and *P2* is continuous. After Transition *T1* is ready to fire, *P1* waits two time units before firing one Token. In these two time units, *P2* receives three Tokens continuously. Figure 8 shows these Token progressions.

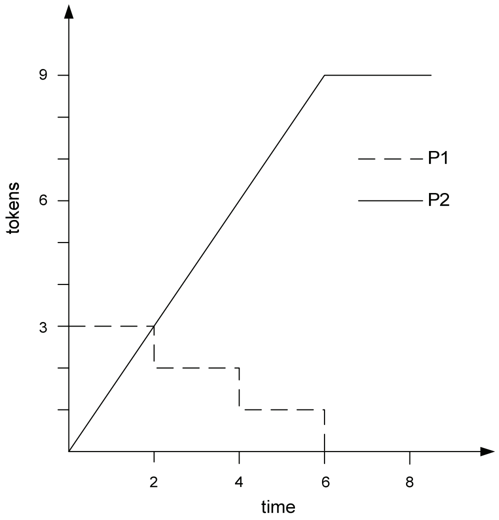
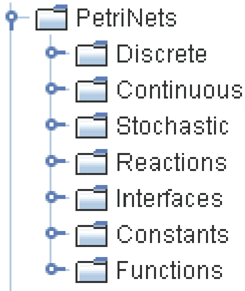


Figure 8: Token progressions of the hybrid Petri Net in Figure 7

3. Petri Net Library

The Petri Net Library is structured in four sub-libraries: Discrete, Continuous, Stochastic and Reactions (see Figure 9). The Reactions Library is discussed in detail in [9].



PetriNets
- Discrete
- Continuous
- Stochastic
- Reactions
- Interfaces
- Constants
- Functions

Figure 9: Structure of the Petri Net Library

Figure 10 shows the icons for the discrete, stochastic and continuous elements of the Petri Net Library. A discrete Place is represented by a turquoise circle and a discrete Transition by a turquoise rectangle. A stochastic Transition is yellow with a turquoise margin and the continuous elements have thick blue margins.

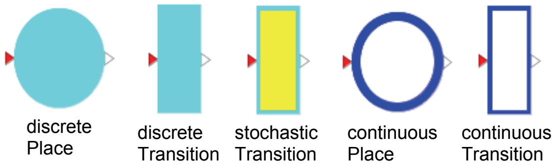
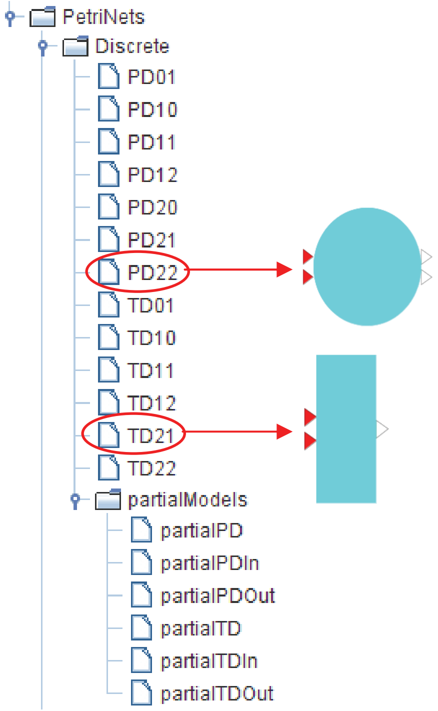


Figure 10: Icons of the Petri Net Library

Every sub-library has general models for Places and Transitions (package partialModels in Figure 11) which are extended to models with fix numbers of input and output connectors. TD21 is for example the denotation for a discrete Transition with two input connectors and one output connector (see Figure 11).

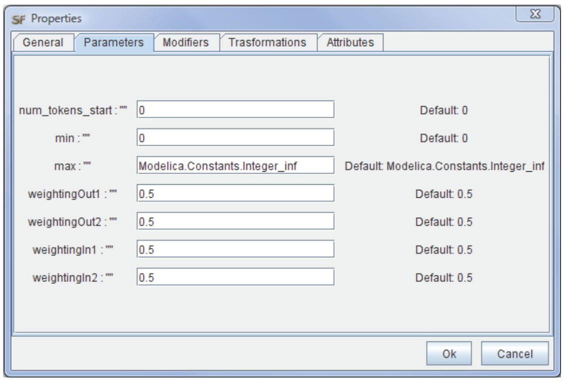


PetriNets
- Discrete
  - PD01
  - PD10
  - PD11
  - PD12
  - PD20
  - PD21
  - PD22
  - TD01
  - TD10
  - TD11
  - TD12
  - TD21
  - TD22
  - partialModels
    - partialPD
    - partialPDIn
    - partialPDOut
    - partialTD
    - partialTDIn
    - partialTDOut

Figure 11: Structure of the discrete Petri Net Library

3.1. Place

In the property-dialog of the discrete and contiuous Place the user can insert the number of Tokens at the beginning of the simulation and the minimum and maximum amount of Tokens that the Place is able to contain (see Figure 12).



Properties

General | Parameters | Modifiers | Trasformations | Attributes

num_tokens_start : ""	0	Default: 0
min : ""	0	Default: 0
max : ""	Modelica.Constants.Integer_inf	Default: Modelica.Constants.Integer_inf
weightingOut1 : ""	0.5	Default: 0.5
weightingOut2 : ""	0.5	Default: 0.5
weightingIn1 : ""	0.5	Default: 0.5
weightingIn2 : ""	0.5	Default: 0.5

Figure 12: Property-dialog of a discrete Place with two inputs and two outputs

In the discrete Place, it is also possible to determine input and output weightings. If a Place is not able to fire in all activated Transitions due to its lack of Tokens, the edges can be weighted. By means of a uniform distributed random variable, it is decided which Transitions receive Tokens. The same principle is applied if a Place is not able to gain Tokens from all ready to fire Transitions due to its maximum value (cf. section 2).

The current number of Tokens is determined in the Place. To do that, two sums are calculated at first. One is the sum of all Tokens that leave the Place and the other one of all Tokens that come inside the Place. In the discrete case, the new number of Tokens is calculated as follows:

```
tokeninout = sumIn > 0 or sumOut > 0;
when tokeninout then
   t = pre(t) + sumIn - sumOut;
end when;
```

In the continuous case, the new number of Tokens is calculated by a differential equation. Of course, additional conditions are considered, i.e. the right hand side of the differential equation may not be negative if the Tokens are equal to the minimum.
After this computation, it is checked whether the Place is empty or full. The current state is reported to the connected input and output Transitions (see Figure 13).
The inState of a continuous Place is only true if the Place is full and the outState is only false if the Place is empty.
The inState of a discrete Place is only true if the Place is full or the Place has just gained Tokens or both. The outState is only false if the Place is empty or the Place has just gained Tokens or both.

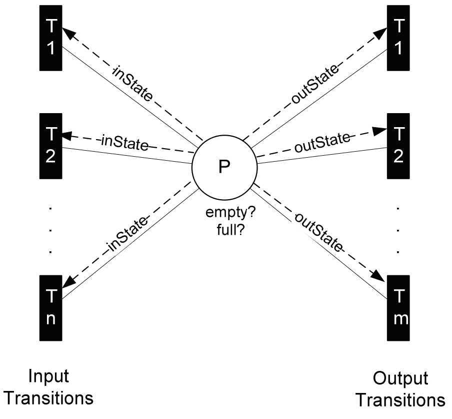
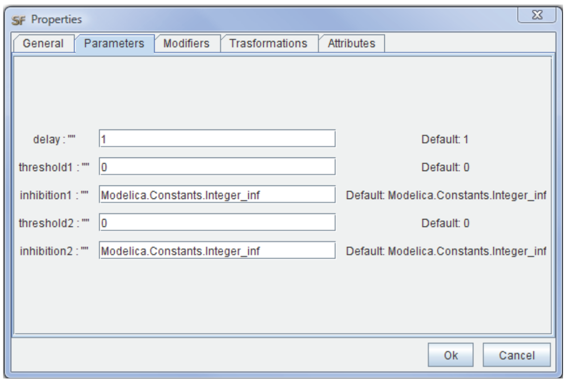


Figure 13: The states of a Place

The inState is the state that is reported to the input Transitions and the outState is the state that is reported to the output Transitions (see Figure 13).

3.2. Transition

In the property-dialog of the discrete Transition, a *delay* can be entered (see Figure 14). If the corresponding Transition is activated, it will take as much time units as keyed in until the Transition fires. In the stochastic case, these are the characteristic values of the corresponding distribution. For example the expectation value *lambda* of an exponential distribution or the expectation value *m* and the standard deviation *s* of a normal distribution. Therefore, the random numbers are calculated by an external C-function.



delay : "" 1 Default: 1

threshold1 : "" 0 Default: 0

inhibition1 : "" Modelica.Constants.Integer_inf Default: Modelica.Constants.Integer_inf

threshold2 : "" 0 Default: 0

inhibition2 : "" Modelica.Constants.Integer_inf Default: Modelica.Constants.Integer_inf

Figure 14: Property-dialog of a discrete Transition with two inputs

There is also the possibility to determine a condition which has to be true so that the Transition is ready to fire. This condition can be entered in 'Modifiers'. For example: con = time>5. In the discrete case a delay can be determine additionally.

The weightings for each edge, which goes in or out of the respective Transition, can be entered in the property-dialog. This has to be done with the aid of 'Modifiers', as functions are also allowed in this case (cf. Section 2). The weightings from the edges that go into the Transition are denoted by *sub1*, *sub2*, … from the top to the bottom. The weightings for the edges, that go out of the Transition, are called *add1*, *add2*, … (see Figure 15).

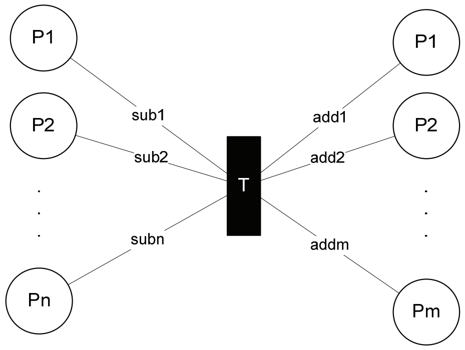
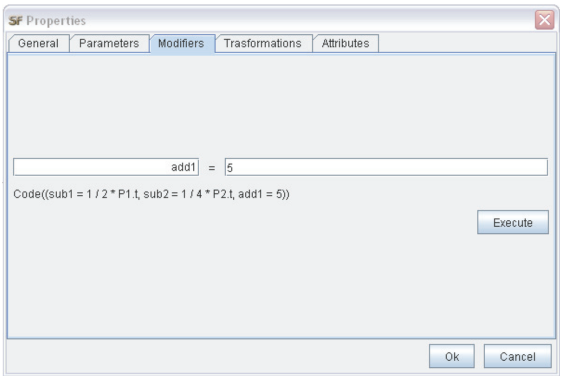


Figure 15: The denotation of the edges weightings

The weightings for Transition *T1* in Figure 6 are keyed in as follows:
sub1 = ½*P1.t
sub2 = ¼*P2.t
add1 = 5.



add1 = 5

Code((sub1 = 1 / 2 * P1.t, sub2 = 1 / 4 * P2.t, add1 = 5))

Figure 16: The entry of *sub* and *add* in 'Modifiers'

If functions are entered as edge weightings, the name of the respective Place has to be typed in with the ending '.t' (stands for Tokens).

In the property-dialog the *threshold* and *inhibition* values for each edge, which goes into the respective Transition, can be entered, too (see Figure 14 and Figure 4). The denotation is like the edge weightings. The boundaries for the first input edge from the top are called *threshold1* and *inhibition1*, for the second these are *threshold2* and *inhibition2* and so on.

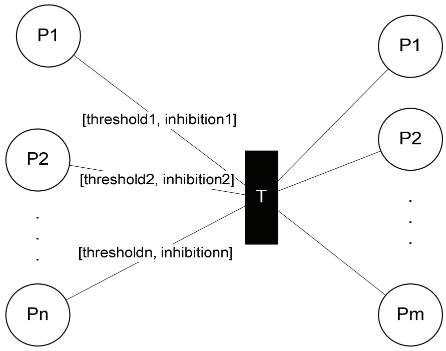

P1
P2
Pn
[threshold1, inhibition1]
[threshold2, inhibition2]
[thresholdn, inhibitionn]
T
P1
P2
Pm

Figure 17: The denotation of the inhibition and threshold values

The Transition decides whether it fires or not. For that, all Transitions check the states of all connected Places. If all states of the input Places are true and none state of the output Places is true and in addition the entered condition is true, the Transition is activated (see Figure 18):

```
activated = if Functions.allTrue(inState)
        and not Functions.anyTrue(outState)
        and con;
```

In the continuous case the activation is equivalent to firing. In the discrete and stochastic case, the activation time is saved and the Transition fires when the corresponding *delay* is passed.

```
when edge(activated) then
     last_activation_time = time;
end when;
delay_passed = activated and
        time - delay >last_activation_time;
```

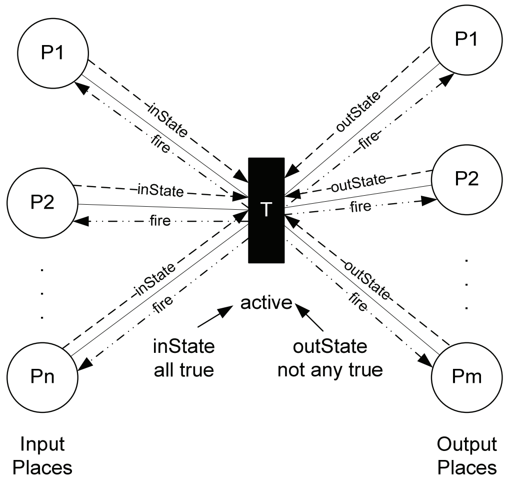


Figure 18: The activation of a Transition

If the Transition fires or not, is reported to the connected input and output Places.

4. Example

Figure 19 shows a simplified example of the production process of crude steel, compare [10].
At first, the iron ore is transported per ship from Brasilia to a stock at the port of Rotterdam. This trip takes generally 14 days. Every 24 days a ship arrives at the port of Rotterdam. But the exact time of arrival is uncertain. The trip can take a little bit longer or shorter because of nature or other conditions. This is modeled with the aid of a stochastic Transition (Transition *ship*). The time of arrival is a normal distributed random variable

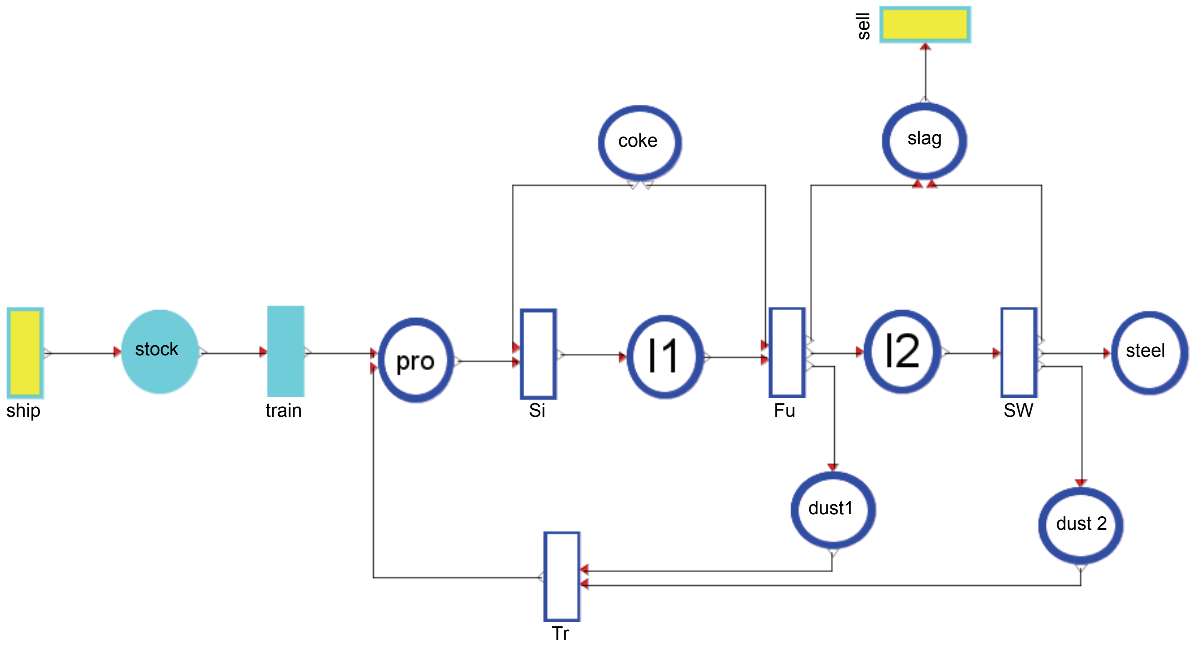


Figure 19: Steel production process

with the expectation value m = 24 and the standard deviation s = 1. A shipload contains 360.000 t iron ore. For this reason *add1* of Transition *ship* is equal to 360.000. The stock at the port can contain at most 720.000 t iron ore. Therefore, the maximal value of the Place *stock* is fixed to 720.000. The start value of this Place is 360.000.

At the next level the iron ore is loaded from the stock to several trains. A train can contain 5000 t iron ore and the drive to the steel production in Duisburg takes 8 hours. The iron ore is delivered "just in time" to the production process. Hence, no other stock is needed. The discrete Transition *train* represents the transport from Rotterdam to Duisburg. The delay is 1/3 day (= 8 hours) and *sub1* = *add1* = 5000. The iron ore (Place *pro*) and the coke (Place *coke*) are mixed in the sintering plant. It accrues the intermediate product sinter (Place *I1*). For one ton employed iron ore 0.2 t coke is needed and 0.73 t sinter is produced. This production step is modeled continuously by means of the Transition *Si*. The edge weightings are the following:

$$sub1 = 0.2 \cdot pro.t$$

$$sub2 = pro.t$$

$$add1 = 0.73 \cdot pro.t.$$

The sinter is further processed in the blast furnace to hot metal (Place *I2*). In addition, the by-products slag (Place *slag*) and blast furnace dust (Place *dust1*) are produced. For one ton employed sinter 0.2 t coke is needed and 0.1 t slag, 0.65 t hot metal and 0.01 t blast furnace dust are produced. The Transition *Fu* displays this. The edges weightings are:

$$sub1 = 0.2 \cdot I1.t$$

$$sub2 = I1.t$$

$$add1 = 0.1 \cdot I1.t$$

$$add2 = 0.65 \cdot I1.t$$

$$add3 = 0.01 \cdot I1.t.$$

The by-product slag is sold to building industry. When 50.000 t slag are produced the company is informed but it is uncertain when the company arrives to pick up the slag and how long this procedure takes. This is modeled with a stochastic Transition with a normal distributed delay (m = ½ and s = 1/8) and *sub1* = 50.000.

In the last production step the hot metal is processed to crude steel (Place *steel*) in the steel works. Slag (Place *slag*) and converter dust (Place *dust2*) are the by-products here.

For one ton employed hot metal 0.13 t slag, 0.8 t crude steel and 0.05 t converter dust are produced. The Transition *SW* represents the steel works. The edge weightings are:

$$sub1 = I2.t$$

$$add1 = 0.13 \cdot I2.t$$

$$add2 = 0.8 \cdot I2.t$$

$$add3 = 0.05 \cdot I2.t.$$

Iron ore can be substituted by blast furnace dust (Place *dust1*) and converter dust (Place *dust2*). This is modeled with the Transition *Tr* and the edges weightings are:

$$sub1 = dust1.t$$

$$sub2 = dust2.t$$

$$add1 = 0.1 \cdot (dust1.t + dust2.t).$$

Following, some simulation results are shown. Figure 20 displays three possible progressions of the stock of iron ore at the port of Rotterdam. Every progression is different because of the stochastic modeling. The stock is limited to 720.000 t iron ore. Hence, this border is not exceed. The iron ore is loaded to trains. Every 8 hours a train drives with 5000 t iron ore to Duisburg. These are the discrete stages in the magnification.

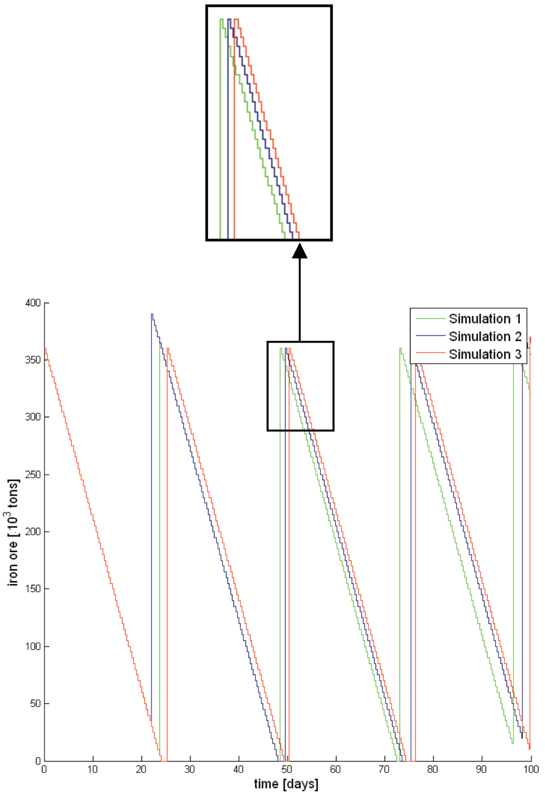


Figure 20: Three simulation results of the iron ore stock at the port of Rotterdam

The iron ore is exhausted in all simulations at specific time points:

Simulation 1 [days]	Simulation 2 [days]	Simulation 3 [days]
48 – 48.5	48 – 49.5	24 – 25.25
72.5 - 73	73.5 – 75.4	49.25 – 50.31
		74.31 – 76.28

This causes bottlenecks in the production process.

The next figure shows the progression of iron ore, sinter and hot metal of simulation 3. The decrease after day 24.3, 49.6 and 74.4 is caused by the exhaused stocks.

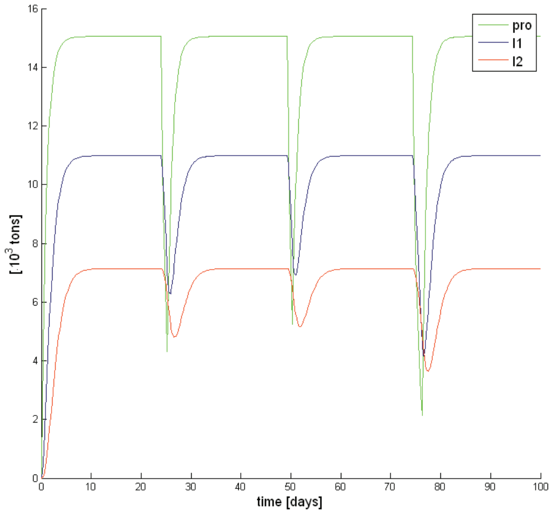


Figure 21: The progressions of iron ore (*pro*), sinter (*I1*) and hot metal (*I2*) (simulation 3)

Figure 22 illustrates the bottleneck in the production process of simulation 3, too. The exhausted stocks are reflected in the amount of crude steel. The production is decreased after every empty stock period.

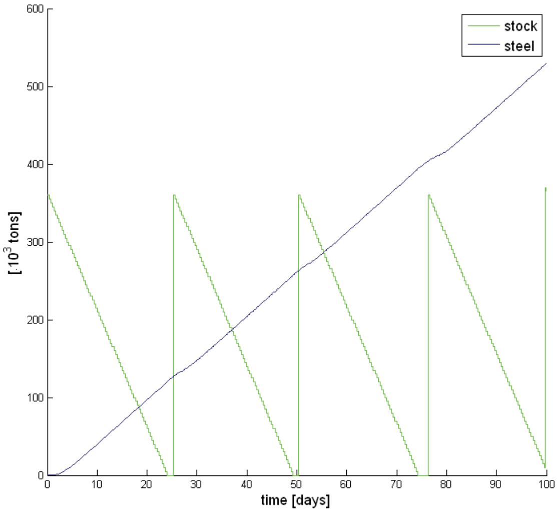


Figure 22: Stock of iron ore (*stock*) and produced crude steel (*steel*) by comparison (simulation 3)

Figure 23 displays the slag progression of simulation 3. When 50.000 t slag are achieved, it is sold to the building industry. The bottlenecks are here visible, too.

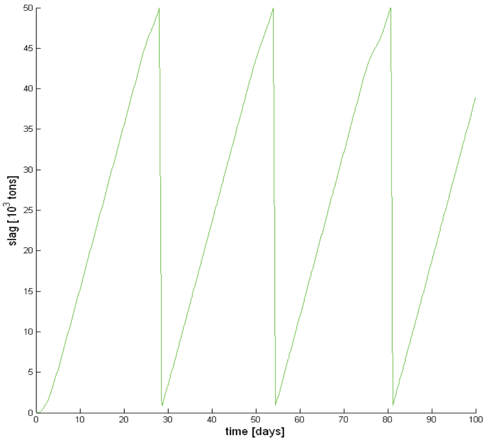


Figure 23: Slag (*slag*) progression of simulation 3

The conclusion of these simulations is that the delivery period of iron ore has to be reduced. The new period has to be big enough that only small bottlenecks appear and small enough that no high stocks accumulate. If for example a period of 22.5 days is chosen, the probability of a bottleneck is 6.7 % and the probability that this bottleneck takes longer than one day is 0.62 %. Now is the task to find the “optimal” solution between bottlenecks and stock costs. Figure 24 shows three simulation results of the progression of the iron ore stock if the delivery period is 22.5 days.

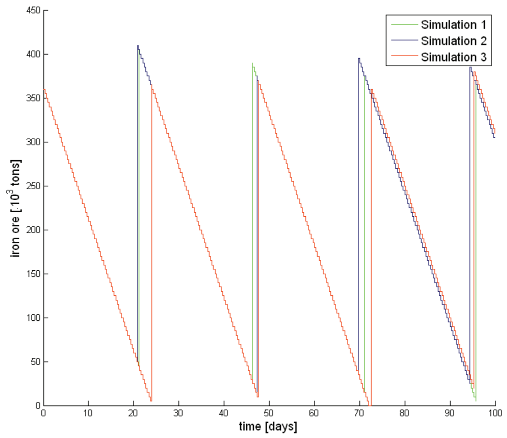


Figure 24: Three simulation results of the iron ore stock with a delivery period of 22.5

5. Conclusions

This paper has shown the new extensions of the Petri Net Library. Now it is possible to model continuous and hybrid Petri Nets with dynamic edge weightings in OpenModelica. These innovations can be applied in different kinds of areas. The paper has demonstrated this with an example of the steel production process. But the Petri Net Library is also useful for the modeling of biological systems [9].

References

[1] **Petri, Carl Adam.** *Kommunikation mit Automaten.* Bonn: Institut für Instrumentelle Mathematik , 1962.

[2] **Mosterman, Pieter J., Otter, Martin and Elmqvist, Hilding.** *Modeling Petri nets as Local Constraint Equations for Hybrid Systems Using Modelica.* Reno, USA , 1998. Summer Computer Simulation Conference .

[3] **Otter, Martin, Arzèn, K.-E. and Dressler, I.** *SateGraph-A Modelica Library for Hierarchical State Machines.* Hamburg , 2005. Modelica Conference. pp. 569-578.

[4] **Fabricius, Stefan M. O.** *Extensions to the Petri Net Library.* 2001.

[5] **Reddy, Venkatramana N., Liebman, Michael N. and Mavrovouniotis, Michael L.** *Qualitative Analysis of Biochemical Reaction Systems*. Compu. Biol. Med. 1996, pp. 9-24.

[6] **Hofestädt, R. and Thelen, S.** *Quantitative Modeling of Biochemical Networks.* In Silico Biology. 1998, 1, pp. 39-53.

[7] **Doi, Atsushi, et al.** *Constructing biological pathway models with hybrid functional Petri nets*. In Silico Biology. 2004.

[8] **Valk, Rüdiger.** *Self-Modifiying Nets: A natural Extension of Petrinets. LNCS.* 1978, 62, pp. 464-476.

[9] **Proß, Sabrina, et al.** *Modeling a Bacterium's Life: A Petri-Net Library in Modelica.* Como, Italy , 2009. Modelica conference.

[10] **Dyckhoff, Harald and Spengler, Thomas.** *Produktionswirtschaft.* Berlin Heidelberg : Springer-Verlag, 2005.

MODELING A BACTERIUM'S LIFE: A PETRI-NET LIBRARY IN MODELICA

Sabrina Proß[1)], Bernhard Bachmann[1)], Ralf Hofestädt[2)], Karsten Niehaus[2)], Rainer Ueckerdt[1)],
Frank-Jörg Vorhölter[2)], Petra Lutter[2)]

[1)] University of Applied Sciences
Am Stadtholz 24
33609 Bielefeld, Germany

sabrina.pross@fh-bielefeld.de

[2)] Bielefeld University
POB 100 131
33501 Bielefeld, Germany

plutter@cebitec.uni-bielefeld.de

Abstract

For modeling biological systems the already existing Petri Net Libraries were further developed with OpenModelica using the SimForge graphical user interface (GUI). The Petri Nets elements were wrapped into models for different reaction types to simplify the modeling process. Additionally, a database connection was implemented for integrating kinetic data. The application of this new Reaction Library is demonstrated by the xanthan production of the bacterium *Xanthomonas campestris* pv. *campestris*. A mathematical model is introduced to predict growth and xanthan production, given an initial glucose concentration. The parameters of this model are estimated with the aid of the Optimization Toolbox in MATLAB.

1. Introduction

1.1. Biological Background

The organism under study, *Xanthomonas*, is a gram negative bacterium of the *Xanthomonadaceae*-family. It is yellow pigmented, rod shaped and has one polar flagellum (see Figure 1).

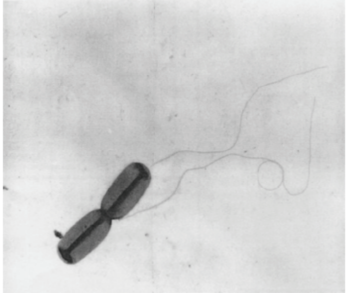


Figure 1: Electron micrograph of *Xanthomonas campestris* [1]

It grows under aerobic conditions and its genome contains about 5 million base pairs [2]. *Xanthomonas* belongs to a group of bacteria that have adopted a plant pathogenic lifestyle. Specifically, *Xanthomonas campestris* pathovar *campestris* harms cruciferous plants like cabbage and cauliflower. Besides its importance as a phytopathogen, *Xanthomonas campestris* pv. *campestris* is known as the producer of the exopolysaccharide xanthan. Xanthan (E-415) is industrially produced and of high commercial significance. e.g. it is employed as a thickening agent and emulsifier. Although substantial efforts have been put into understanding the xanthan synthesis [1], the bacterium's life strategies are far from being well understood.

1.2. Theoretical Background

To describe the influence of external parameters on the production of substances needed for the xanthan synthesis, different kinetic models have been proposed. Unstructured kinetic models consider the bacterium's consumption of food (carbon and nitrogen sources), its growth and its xanthan production through a set of differential equations. As this minimal set of equations treats the bacterium's metabolism as a black box, more complex models had to be established. Garcia-Ochoa et al. introduced a structured kinetic model for *X. campestris* growth that is able to predict different growth rates according to different initial nitrogen concentrations [3]. As we are interested in the question under which circumstances the bacterium's focus is growth on the one hand and which factors lead to xanthan production on the other hand, we still include more details. With a complete genome sequence at hand, the biosynthesis pathways for the production of the exopolysaccharide xanthan could be elucidated

DOI: 10.3384/ecp09430015

[2]. Consequently, we are enabled to set up all the metabolic pathways in question.

2. Methods

2.1. Michaelis-Menten-Kinetics

Once the metabolic pathways are clear, appropriate kinetic equations have to be assigned. As Michaelis-Menten-Kinetics has worked well with similar organisms, we have concentrated on this equation type. The basic reaction describes the transformation of a substrate into a product under enzymatic influence. At this juncture, the following reaction type is the basis:

$$\mathrm{E}+\mathrm{S}\underset{k_{-1}}{\overset{k_1}{\rightleftarrows}}\mathrm{ES}\overset{k_2}{\rightarrow}\mathrm{E}+\mathrm{P}\,.$$

A substrate S links to the active center of the enzyme E and forms an enzyme-substrate-complex ES. ES is converted to the product P, k_i denote velocity constants. The enzyme is set free and can link to another substrate again.

The Michaelis-Menten-Kinetic has the following form:

$$\mathrm{v}=-\frac{\mathrm{d}[\mathrm{S}]}{\mathrm{dt}}=\frac{\mathrm{d}[\mathrm{P}]}{\mathrm{dt}}=\frac{\mathrm{k_{cat}}\cdot[\mathrm{E}]\cdot[\mathrm{S}]}{\mathrm{K_m}+[\mathrm{S}]},$$ where

$[S]$, $[P]$ and $[E]$ are the concentrations of substrate, product and enzyme, respectively. The parameter k_{cat} is denoted as turnover number and specifies the number of substrate molecules that one enzyme molecule converts per second if it is completely saturated with substrate. The Michaelis-Menten constant K_m is the substrate concentration at half-maximum reaction rate (see Figure 2).

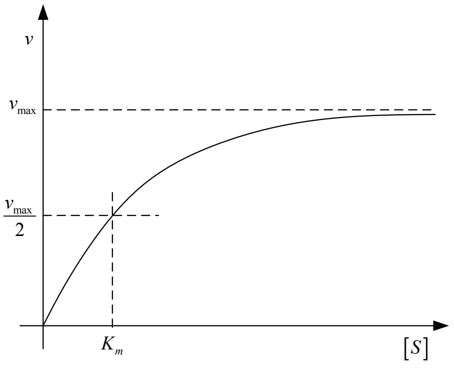


Figure 2: Michaelis-Menten diagram

The negative substrate change is equivalent to the positive product change. It is assumed that the concentrations of free enzymes and enzyme-substrate-complexes do not change during time. They are in a steady state.

2.2. Growth kinetics

For modeling the growth of *Xanthomonas* bacteria it is taken into account that there is a relationship between the exhaustion of glucose and the end of growth. Therefore, two different nutrient limited Growth Kinetics are examined. The first is known as Monod Kinetics [4]:

$$\frac{d[B]}{dt}=[B]\frac{\mu\cdot[N_l]}{K_{N_l}+[N_l]},$$ where $[B]$ is the microbial concentration (biomass), $[N_l]$ is the concentration of the limiting nutrient, μ is the maximum specific growth rate and K_{N_l} is the nutrient concentration at half-maximum specific growth rate.

The second has the following form [5]:

$$\frac{d[B]}{dt}=k\cdot[B]\cdot[N_l],$$ where k is a rate coefficient that depends on physiochemical variables such as temperature, dissolved oxygen and stirrer speed.

2.3. Reaction Kinetics

Reaction Kinetics is used to model the synthesis of xanthan.

The following equation is an example of a stoichiometric equation:

$$r_1R_1+r_2R_2+r_3R_3\rightarrow p_1P_1+p_2P_2,$$ where R_1, R_2 and R_3 are the reactants, P_1 and P_2 are the products and r_1, r_2, r_3, p_1 and p_2 are the stoichiometric coeffients. Reaction Kinetics of this stoichiometric equation is

$$v=-\frac{1}{r_1}\cdot\frac{d[R_1]}{dt}=-\frac{1}{r_2}\cdot\frac{d[R_2]}{dt}=-\frac{1}{r_3}\cdot\frac{d[R_3]}{dt}$$
$$=\frac{1}{p_1}\cdot\frac{d[P_1]}{dt}=\frac{1}{p_2}\cdot\frac{d[P_2]}{dt}$$
$$=k\cdot[R_1]^{r_1}\cdot[R_2]^{r_2}\cdot[R_3]^{r_3},$$

where k is a rate constant.

2.4. Petri Nets

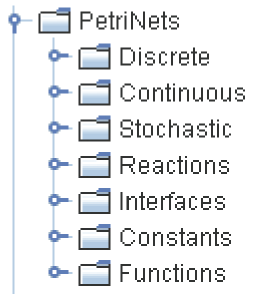
A Petri Net is a graphical construction to describe and analyze concurrent processes and non-deterministic procedures. It is a graph with two different kinds of nodes: Places and Transitions, whereas only a Place can be connected with a Transition or a Transition with a Place. Every Place contains an integer number of Tokens and every edge has a weighting. A Transition is ready to fire when every Place in its previous area has at least as many Tokens as the appropriate edge weighting shows. A Transition that is ready to fire fires by removing as many Tokens as the respective edge weighting indicates - from all of the Places in its previous area. In addition, a specific number of Tokens is stored in all of the Places of its past area - according to the specific edge weighting. To model biological systems, the Petri Net concept has been extended to stochastic Petri Nets, continuous Petri Nets and hybrid Petri Nets, cf [6].

2.5. Petri Net Library

Petri Nets are ideal to model biological processes. Thereby, metabolites, enzymes and genes are modeled with Places, and Transitions represent the reactions between them [7]. In the following it is described how the existing Petri Net Libraries were exetended [6]:

- Continuous Petri Net elements were implemented because biochemical reactions, which convert one substance to another, proceed continuously.
- The speed of these reactions depends mostly on the current concentration of specific substances [8], which can be now displayed by dynamic edge weightings.
- It should be possible to model gene regulation, which contains discrete processes as well as continuous ones. Hybrid Petri Nets, which comprise both discrete and continuous Petri Net elements [9], are now able to fulfil this task.
- The edges can also have upper and lower boundaries. This is necessary for modeling substances which only react when a specific concentration is reached.

The Petri Net Library is structured in seven sub-libraries: Discrete, Continuous, Stochastic, Reactions, Interfaces, Constants, and Functions (see Figure 3).



- PetriNets
  - Discrete
  - Continuous
  - Stochastic
  - Reactions
  - Interfaces
  - Constants
  - Functions

Figure 3: Structure of the Petri Net Library

The next figure shows the icons for discrete, continuous and stochastic Petri Nets elements. For additional information about the functionality, application and implementation of these Petri Net elements see [6]. This paper focuses on the Reaction Library.

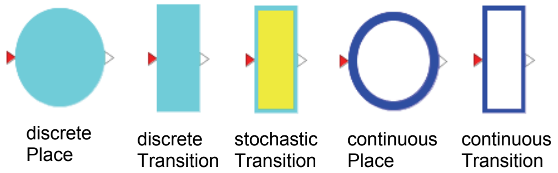
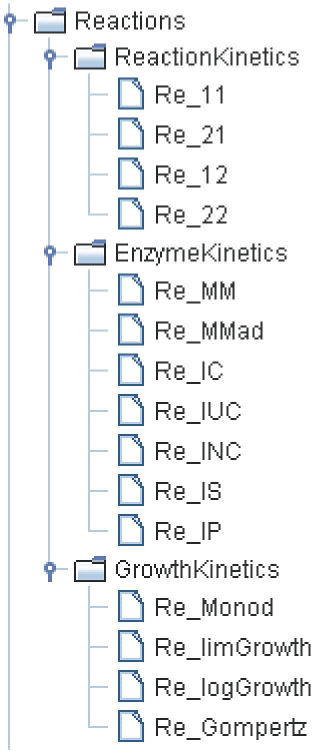


Figure 4: Icons of the Petri Net elements

2.6. Reaction Library

Petri Net elements have been wrapped into appropriate models for different kinds of reactions to simplify the modeling process. These reactions are organized in a sub-library of the Petri Net Library called ‘Reactions’ (see Figure 3). This library is again divided in different sub-libraries to classify the reactions (see Figure 5). These are:

- Reaction Kinetics
- Enzyme Kinetics
- Growth Kinetics.



- Reactions
  - ReactionKinetics
    - Re_11
    - Re_21
    - Re_12
    - Re_22
  - EnzymeKinetics
    - Re_MM
    - Re_MMad
    - Re_IC
    - Re_IUC
    - Re_INC
    - Re_IS
    - Re_IP
  - GrowthKinetics
    - Re_Monod
    - Re_limGrowth
    - Re_logGrowth
    - Re_Gompertz

Figure 5: Structure of the Reactions Library

2.6.1. ReactionKinetics Library

The ReactionKinetics sub-library comprises reactions with the Reaction Kinetics as edge weightings for one or two reactants and one or two products. The reactions for more reactants or products can be extended easily. The reaction $r_1R_1 + r_2R_2 \rightarrow p_1P_1$ for example can be modeled by the reaction Re_21 (see Figure 6).

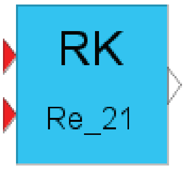
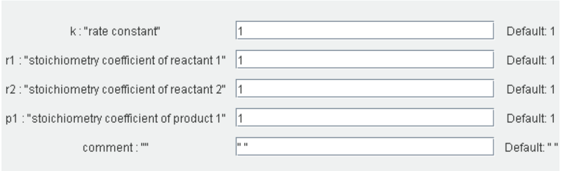


Figure 6: The icon of the reaction Re_21 of the ReactionKinetics Library

The next figure shows the property-dialog of reaction Re_21. The rate constant *k* and the stoichiometric coefficients r_1, r_2 and p_1 can be keyed in.



k : "rate constant"	1	Default: 1
r1 : "stoichiometry coefficient of reactant 1"	1	Default: 1
r2 : "stoichiometry coefficient of reactant 2"	1	Default: 1
p1 : "stoichiometry coefficient of product 1"	1	Default: 1
comment : ""	" "	Default: " "

Figure 7: Property-dialog of the reaction Re_21

2.6.2. EnzymeKinetics Library

The EnzymeKinetics Library consists of reactions that are performed with the aid of enzymes:

- Michaelis-Menten Kinetics (Re_MM)
- Reversible Michaelis-Menten Kinetics (Re_MMad)
- Competitive enzyme inhibition (Re_IC)
- Uncompetitive enzyme inhibition (Re_IUC)
- Non-competitive enzyme inhibition (Re_INC)
- Substrate inhibition (Re_IS)
- Product inhibition (Re_IP)

For more information about the kinetics and enzyme inhibition see [10].

Figure 8 shows a metabolism reaction modeled with the Michaelis-Menten-Kinetics. The icons *sub* and *pro* are Places of the sub-library 'Continuous' of the Petri Net Library, and the Michaelis-Menten reaction between them can be found in the EnzymeKinetics Library.

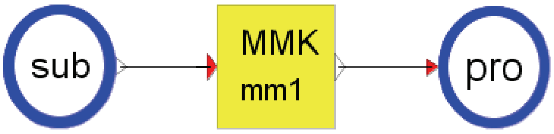
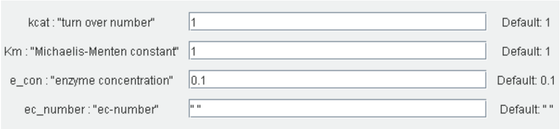


Figure 8: A reaction modeled with the Michaelis-Menten Kinetics

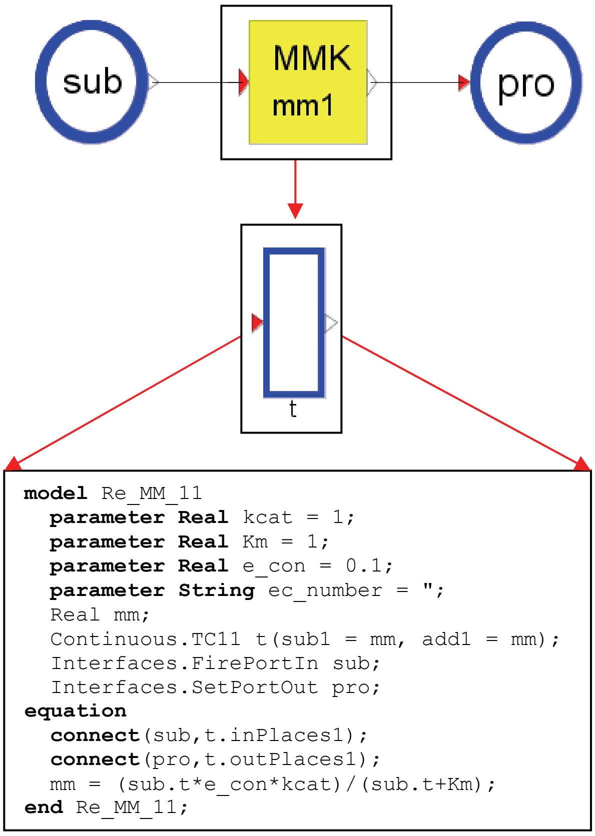
The turnover number k_{cat}, the Michaelis-Menten constant K_m and the enzyme concentration can be entered in the property-dialog (see Figure 9).



kcat : "turn over number"	1	Default: 1
Km : "Michaelis-Menten constant"	1	Default: 1
e_con : "enzyme concentration"	0.1	Default: 0.1
ec_number : "ec-number"	" "	Default: " "

Figure 9: Property-dialog of the Michaelis-Menten Kinetics

The following figure displays the wrapping process of the reaction in Figure 8. The model 'Re_MM' consists of a continuous Transition of the Continuous Library with the Michaelis-Menten Kinetics as edge weightings. The edge weightings in continuous Petri Nets are the right sides of a differential equation.



```
model Re_MM_11
  parameter Real kcat = 1;
  parameter Real Km = 1;
  parameter Real e_con = 0.1;
  parameter String ec_number = ";
  Real mm;
  Continuous.TC11 t(sub1 = mm, add1 = mm);
  Interfaces.FirePortIn sub;
  Interfaces.SetPortOut pro;
equation
  connect(sub,t.inPlaces1);
  connect(pro,t.outPlaces1);
  mm = (sub.t*e_con*kcat)/(sub.t+Km);
end Re_MM_11;
```

Figure 10: Wrapping of the Michaelis-Menten-Kinetics

2.6.3. GrowthKinetics Library

Reactions for modeling growth are organized in the ReactionKinetics Library with two reactions for limited growth:

- Monod Kinetics (Re_Monod)
- Limited Growth Kinetics (Re_limGrowth)

and two for unlimited growth:

- Logistic Growth Kinetics (Re_logGrowth)
- Gompertz Kinetics (Re_Gompertz) [11].

In the case of limited growth the decrease of the limited nutrient is modeled with the differential equation $\frac{d[N_l]}{dt} = -\frac{1}{Y_{BN_l}} \cdot \frac{d[B]}{dt}$, where $[N_l]$ is the concentration of the limited nutrient, $[B]$ is the biomass concentration and Y_{BN_l} is the macroscopic yield of biomass per nutrient unit.

Figure 11 shows an example for modeling with the Monod Kinetics, the nutrient and the biomass are Places of the Continuous Library.

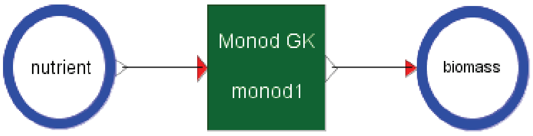
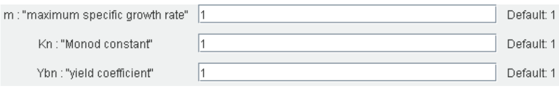


Figure 11: An example for modeling with the Monod Kinetics

The parameters for the Monod Kinetics can be entered in the property-dialog (see Figure 12).

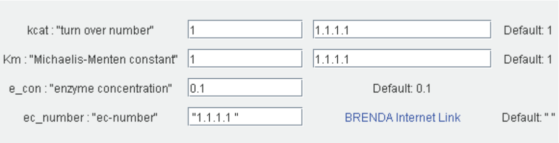


m : "maximum specific growth rate"	1	Default: 1
Kn : "Monod constant"	1	Default: 1
Ybn : "yield coefficient"	1	Default: 1

Figure 12: Property-dialog of the Monod Kinetics (Re_Monod)

2.7. Database Integration

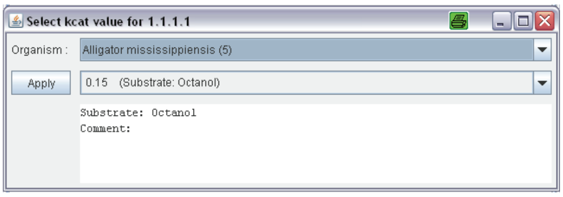
Rafael Friesen has developed a database connection to BRENDA in his master thesis [12], which allows an easy and fast model integration of kinetic data. For using this database connection in combination with the Petri Net Library, a special SimForge version is necessary. Figure 13 shows the new property-dialog of the Michaelis-Menten reaction (Re_MM). Next to the parameters *kcat* and *km* two additional fields show the EC-number filled in at the bottom.



kcat : "turn over number"	1	1.1.1.1	Default: 1
Km : "Michaelis-Menten constant"	1	1.1.1.1	Default: 1
e_con : "enzyme concentration"	0.1	Default: 0.1	
ec_number : "ec-number"	"1.1.1.1 "	BRENDA Internet Link	Default: " "

Figure 13: Property-Dialog of the Michaelis-Menten reaction (Re_MM)

If you press 'enter' in these fields next to the *kcat* or *km* values, the following selection dialog appears.



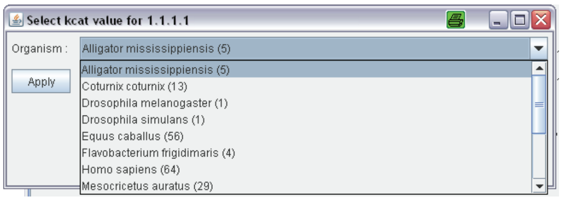
Organism : Alligator mississippiensis (5)

0.15 (Substrate: Octanol)

```
Substrate: Octanol
Comment:
```

Figure 14: Selection dialog

Firstly, the modeled organism can be chosen. In the selection list all organisms with at least one *kcat* or *km* value in the database BRENDA appear.

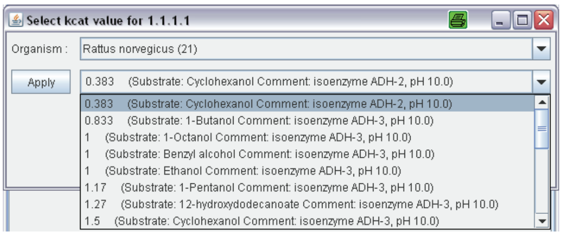


Organism : Alligator mississippiensis (5)

Alligator mississippiensis (5)
Coturnix coturnix (13)
Drosophila melanogaster (1)
Drosophila simulans (1)
Equus caballus (56)
Flavobacterium frigidimaris (4)
Homo sapiens (64)
Mesocricetus auratus (29)

Figure 15: Organism selection

After the organism selection it is possible to choose a proper value. Next to the respective value information about the substrate and experimental conditions is given.



Organism : Rattus norvegicus (21)

0.383 (Substrate: Cyclohexanol Comment: isoenzyme ADH-2, pH 10.0)

0.383 (Substrate: Cyclohexanol Comment: isoenzyme ADH-2, pH 10.0)
0.833 (Substrate: 1-Butanol Comment: isoenzyme ADH-3, pH 10.0)
1 (Substrate: 1-Octanol Comment: isoenzyme ADH-3, pH 10.0)
1 (Substrate: Benzyl alcohol Comment: isoenzyme ADH-3, pH 10.0)
1 (Substrate: Ethanol Comment: isoenzyme ADH-3, pH 10.0)
1.17 (Substrate: 1-Pentanol Comment: isoenzyme ADH-3, pH 10.0)
1.27 (Substrate: 12-hydroxydodecanoate Comment: isoenzyme ADH-3, pH 10.0)
1.5 (Substrate: Cyclohexanol Comment: isoenzyme ADH-3, pH 10.0)

Figure 16: Value selection

If you press 'Apply' the value will appear in the property-dialog of the Michaelis-Menten reaction.

Similarly, you can integrate the dissociation constants of the inhibition reactions.

3. Modeling Xanthan Production

With the aid of the model of the present paper, it is possible to predict biomass and xanthan concentrations of the bacterium *Xanthomonas campestris* pv. *campestris* (Xcc), by a given initial glucose concentration. Both, the growth and the xanthan production, are limited by glucose. Figure 17 shows the model on the top level and Figure 18 shows the Xcc-model, the model behind the grey icon with the denotation 'Xcc'.

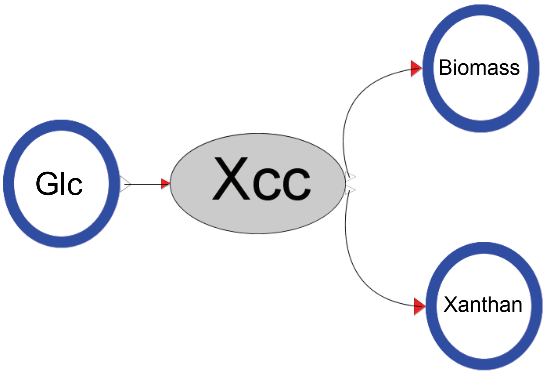


Figure 17: Model on the top level

As can be seen in Figure 18, the bacterium has two choices after glucose intake: It can either opt for growth or for xanthan production.
Xanthan consists of five components: UDP-glucose, UDP-glucuronic acid, GDP-mannose, pyruvate and acetate, the model, however, confines to the three nucleotide sugars (UDP-glucose, UDP-glucuronic acid, GDP-mannose). The synthesis of these three nucleotide sugars to xanthan is modeled with Reaction Kinetics (cf. section 2.3), with one xanthan unit consisting of two units UDP-glucose, one unit UDP-glucuronic acid and two units GDP-mannose.
The growth is modeled with the limited Growth Kinetics and Monod Kinetics for reasons of comparison (cf. section 2.2). The conversion process glucose to biomass is split up into two sub-processes.
The edge weightings of the transition *glcOutIn* are:

```
add1 = sub1 = kg*der(Biomass.t)
```

and of transition growth:

```
add1 = sub1 = kb*glcIn.t
```

(limited Growth Kinetics)

```
add1 = sub1 = mb*glcIn.t/(Kmb+glcIn.t)
```

(Monod Kinetics).

The conversion from one metabolite to another is modeled by the Michaels-Menten Kinetics (cf section 2.1) from the EnzymeKinetics Library and all metabolites are modeled by continuous Places from the Continuous Library.

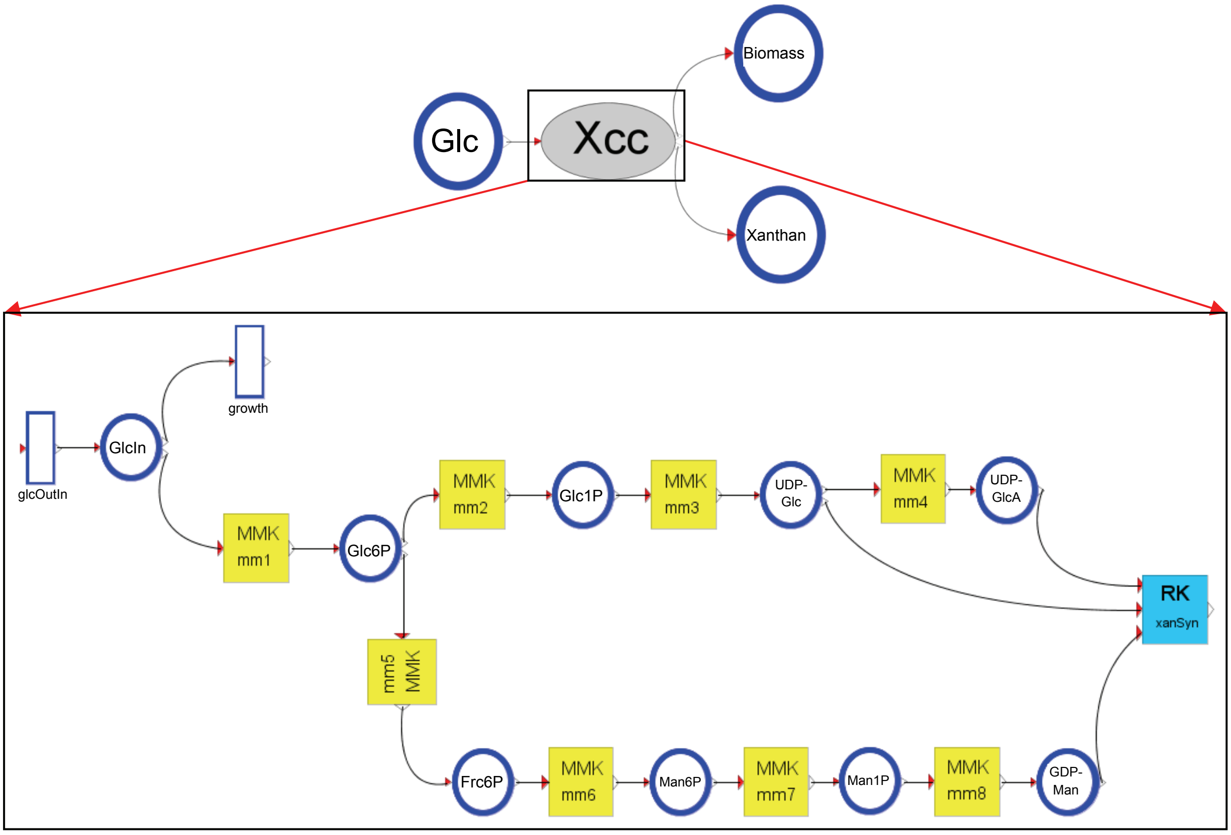


Figure 18: Sub-Model Xcc

4. Experimental Data

To adapt the model parameters to the reality a fermentation was made by Tony Watt [13] under industrial conditions. The bacteria grew in a fermenter with an initial glucose concentration of 12.57 g/l.
For the validation of the model an additional fermentation was made by Tony Watt in a fermenter with an initial glucose concentration of 13.75 g/l.
In both fermentation experiments the xanthan-, biomass-, and glucose-concentrations were measured at several time points.

5. Data-Fitting

The model parameters were fitted to the data of the first fermentation using the MATLAB optimization toolbox. At first, the model of Figure 18 was simplified into a smaller sub-model to estimate the first set of parameters (see Figure 19). These are *kb* for the limited Growth Kinetics, *mb* and *Kmb* for the Monod Kinetics, *kg* for the edge weighting of the transition *glcOutIn,* and the two parameters *kcat1* and *km1* of the Michaelis-Menten reaction *mm1*, respectively. The measured concentrations were glucose and biomass. The concentrations of the glucose inside the bacteria (GlcIn) and of glucose-6-phosphate (Glc6P) were unknown.

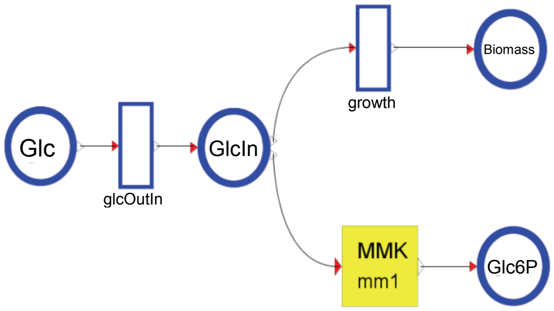


Figure 19: First sub-model for the parameter estimation

The following figures show the results of the adapted biomass concentrations, with Figure 20 displaying the results of the model with the limited Growth Kinetics and Figure 21 representing the model with the Monod Kinetics. Both show a good agreement with the experimental data.

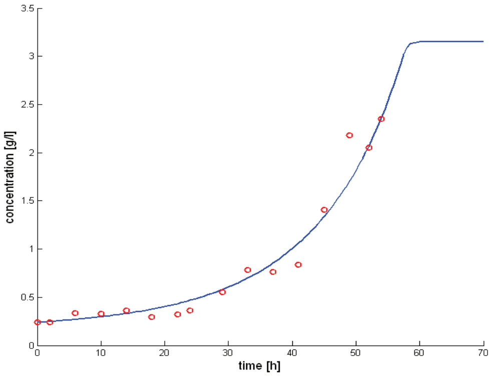


Figure 20: Data-fitting of biomass (limited Growth Kinetics)

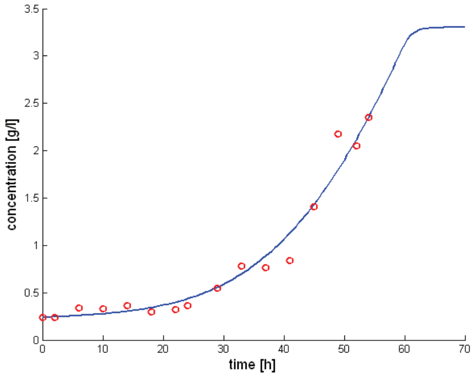


Figure 21: Data-fitting of biomass (Monod Kinetics)

For the next optimization steps the limited Growth kinetics was chosen. Figure 22 shows the data-fitted glucose concentration of this model.

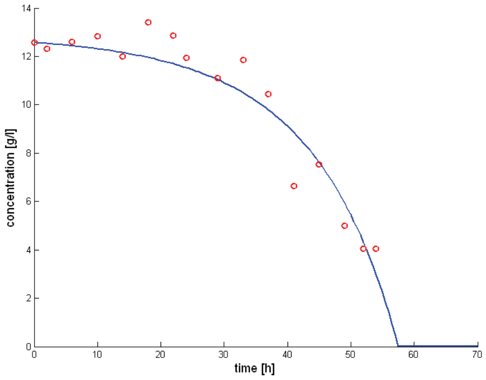


Figure 22: Data-fitting of glucose

The four parameters (*kb*, *kg*, *kcat1* and *km1*) are now fixed and the sub-model of Figure 19 is expanded to estimate the concentrations for the three nucleotide sugars that are necessary to produce the experimental xanthan concentration results (see Figure 23). In this optimization step intermediate metabolites have been left out.

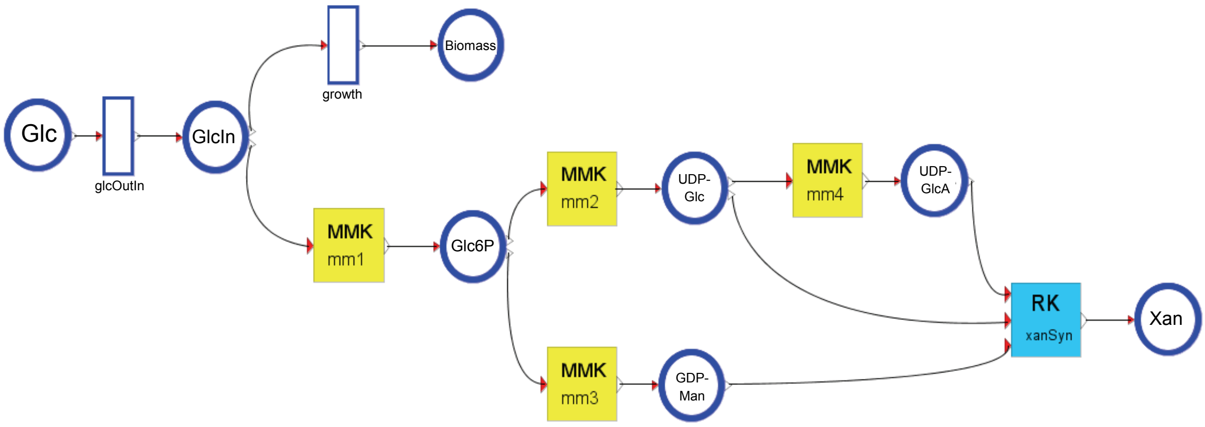


Figure 23: Second sub-model for the parameter estimation

Additionally, the kinetic parameters for the conversion from UDP-glucose (UDP-Glc) to UDP-glucuronic acid (UDP-GlcA) are determined.

Figure 24 displays the results of the data-fitting for the xanthan concentration. Now the *kcat* and *km* values for the conversion from UDP-glucose to UDP-glucuronic acid are fixed and the concentrations of the three nucleotide sugars are saved.

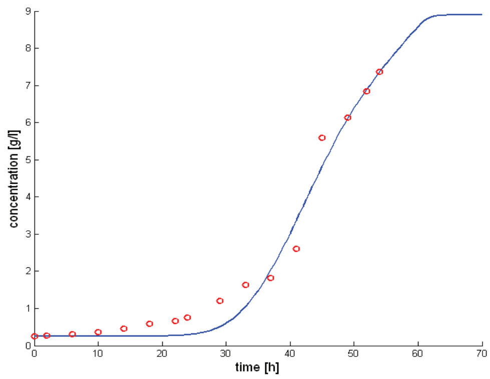


Figure 24: Data-fitting of xanthan

The *kcat* and *km* values for the intermediate metabolites are finally estimated in the last optimization step. These are twelve additional parameters which are adapted to the nucleotide sugar concentrations of the previous optimization step. Figure 25 shows the predicted concentrations of the metabolites for the metabolism in Figure 18.

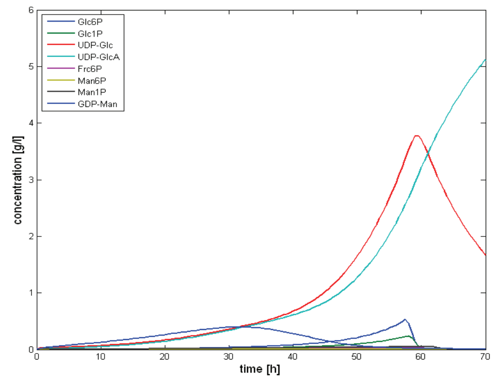


Figure 25: Concentrations of the metabolites

With a data-fitted model at hand, it is now possible to predict biomass growth and xanthan production with different initial glucose concentrations. Figure 26 shows the biomass concentrations for the initial glucose concentrations 3 g/l, 6g/l, 8 g/l and 10 g/l. Figure 27 presents the corresponding xanthan concentrations.

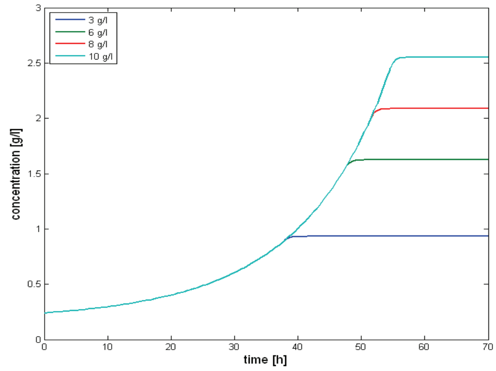


Figure 26: Influence of initial glucose concentration on evolution of **biomass** concentration

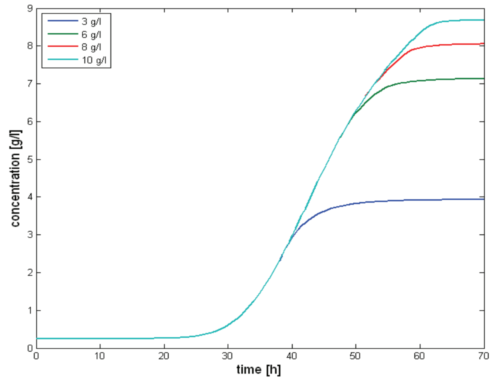


Figure 27: Influence of initial glucose concentration on evolution of **xanthan** concentration

For the validation of the model an additional fermentation was made. The predicted model curves of glucose, biomass and xanthan show a good agreement with the experimental data (see Figure 28, Figure 29, Figure 30).

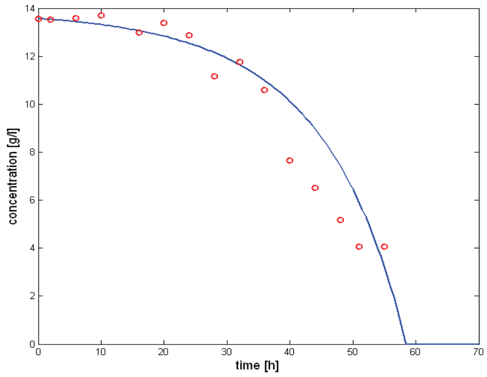


Figure 28: Experimental and model-predicted **glucose** concentration of the second fermentation

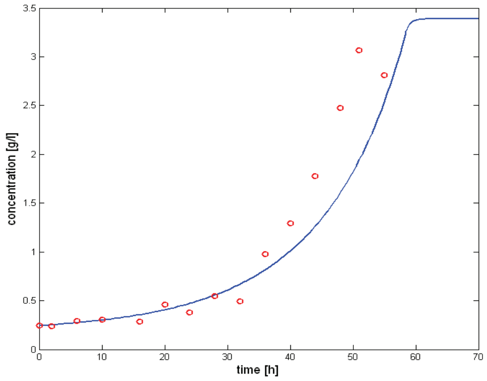


Figure 29: Experimental and model-predicted **biomass** concentration of the second fermentation

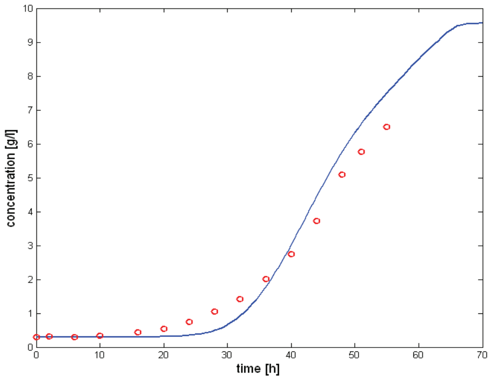


Figure 30: Experimental and model-predicted **xanthan** concentration of the second fermentation

6. Conclusion and Outlook

For modeling biological systems, a Petri Net Library was implemented in OpenModelica with the SimForge GUI. To simplify the modeling process, the Petri Net elements were wrapped into reaction models for different application areas: Reaction Kinetics, Enzyme Kinetics and Growth Kinetics.

With the aid of this library a model was developed for predicting biomass and xanthan concentrations for the bacterium *Xanthomonas campestris* pv. *campestris* given an initial glucose concentration.

The respective model parameters were estimated stepwise by using the MATLAB Optimization toolbox. Validation tests demonstrate a good agreement with measured fermentation data. In the future more experiments are planned to achieve an even better adaptation. Additionally, not only xanthan, biomass and glucose will be measured but also the concentrations of the corresponding metabolites. This measurement is possible with a gas-phase chromatograph that is coupled to a mass spectrometer, the so-called GC/MS analysis. This method allows to determine the concentrations of the metabolites Glucose-6-Phosphate, Glucose-1-Phosphate, Fructose-6-Phosphate, Mannose-6-Phosphate, and Mannose-1-Phosphate .

Furthermore, the quantification of the nucleotide sugars of xanthan can be achieved using high performance liquid chromatography (HPLC). Last but not least further extensions of the model will be considered, i.e. modeling nitrogen as growth limiting factor.

References

[1] **Garcia-Ochoa, F., et al.** Xanthan gum: production, recovery, and properties. *Biotechnology Advances.* 2000, 18, pp. 549-579.

[2] **Vorhölter, Frank-Jörg, et al.** The genome of Xanthomonas campestris pv. campestris B100 and its use for the reconstruction of metabolic pathways involved in xanthan biosynthesis. *Journal of Biotechnology.* 2008, pp. 33-45.

[3] **Garcia-Ochoa, F., Santos, V. E. and Alcon, A.** Structured kinetic model for Xanthomonas campestris growth. *Enzyme and Microbial Technology.* 2004, pp. 583-594.

[4] **Monod, Jacques.** The Growth of Bacterial Cultures. *Annual Review of Microbiology.* 1949, pp. 371-394.

[5] **Quinlan, Alician V.** Kinetics of Secondary Metabolite Synthesis in Batch Cultures When Two Different Substrates Limit Cell Growth and Metabolite Production: Xanthan Synthesis by Xanthomonas campestris. *Biochemical Eng.* 469, 1986, pp. 259-269.

[6] **Proß, Sabrina and Bachmann, Bernhard.** A Petri Net Library for Modeling Hybrid Systems in OpenModelica. *submitted (Modelica Conference 2009).* 2009.

[7] **Reddy, Venkatramana N., Liebman, Michael N. and Mavrovouniotis, Michael L.** Qualitative Analysis of Biochemical Reaction Systems. *Compu. Biol. Med.* 1996, pp. 9-24.

[8] **Hofestädt, R. and Thelen, S.** Quantitative Modeling of Biochemical Networks. *In Silico Biology.* 1998, 1, pp. 39-53.

[9] **Doi, Atsushi, et al.** Constructing biological pathway models with hybrid functional Petri nets. *In Silico Biology.* 2004.

[10] **Berg, Jeremy M., Tymoczko, John L. and Stryer, Lubert.** *Biochemistry.* New York : W. H: Freeman, 2006.

[11] **Gompertz, Benjamin.** On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. *Phil. Trans. R. Soc. Lond.* 115, pp. 513-585.

[12] **Friesen, Rafael.** Petrinets in systems biology: Modelling cell communication with petri nets. Bielefeld, 2009.

[13] **Watt, Tony.** private communication. Bielefeld, 2009.

Creating a Bridge between Modelica and the Systems Biology Community

Jan Brugård[1], Daniel Hedberg[1], Marta Cascante[2], Gunnar Cedersund[3, 4], Àlex Gómez-Garrido[5], Dieter Maier[6], Elin Nyman[3], Vitaly Selivanov[2], Peter Strålfors[3]
[1] MathCore Engineering AB, Teknikringen 1F, 583 30 Linköping, Sweden
{jan.brugard, daniel.hedberg}@mathcore.com
[2] Departamento de Bioquímica y Biología Molecular, Universitat de Barcelona, Spain
[3] Department of Clinical and Experimental Medicine, Linköping University, Linköping, Sweden
[4] Freiburg Institute for Advanced Studies, School of Life Sciences, Germany
[5] Grup de Recerca en Informàtica Biomèdica, Universitat Pompeu Fabra, Barcelona, Spain
[6] Biomax Informatics AG, Martinsried, Germany

Abstract

The Systems Biology Markup Language (SBML) is the leading modelling language within systems biology. It is a computer-readable format for representing models of biochemical reaction networks in software. SBML has been evolving since 2000 thanks to an international community of software developers and users. At the same time the Modelica language has evolved as the leading object-oriented modelling language for convenient, component-oriented modelling of complex physical systems.

As a part of the EC-funded BioBridge project MathCore has developed the Modelica library BioChem and the *MathModelica Systems Biology* toolbox. With the release of version 1.0 of the BioChem library and the *MathModelica Systems Biology* toolbox it will be possible to import and export from Modelica to SBML. The toolbox also allows to publish the Modelica models as interactive HTML pages, with biochemical and experimental enzyme characterizations derived from the BioBridge portal and BioXM knowledge management environment.

Keywords: Modelica tools; SBML; library; biological systems; cellular pathways; translator

1 The BioChem Library

The first beta version of the BioChem library was developed by Larsdotter Nilsson at the Linköping University [1].

The design idea behind the BioChem library is to create a general purpose Modelica library for modelling, simulation and visualization of biological and biochemical systems. The models implemented in the BioChem library describe substances and reactions that can take place in-between these substances in a diverse number of biochemical pathways. An example pathway is shown in Figure 1.

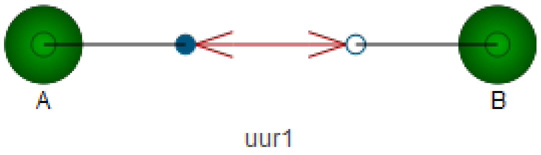


Figure 1 *Simple pathway model using components from the BioChem library.*

Version 1.0 of the library has now been completed and is publicly available [2].

2 Modelica to SBML Translator

A translator that converts Modelica to SBML [3], and vice versa, has been developed. The translator is able to convert SBML models that are compliant with SBML Level 2, version 3. The Modelica models created are Modelica 2.2 and 3 compliant. Modelica models are translated to SBML Level 2, version 1, 2 or 3.

2.1 SBML

SBML is based on XML, making it more suitable to read and write by computers than by humans. SBML is a model representation format for systems biology, created with an objective to become a common intermediate format for software tools. The idea is that a large support for SBML enables the use of a wide range of different tools without having to rewrite models. In principle, a SBML model consists of a number of components, describing biochemical enti-

DOI: 10.3384/ecp09430016

ties (species) and transformations (reactions and rules) forming a biochemical network. In addition it is also possible to describe instantaneous discontinuous state changes using events, and operating assumptions for the model using constraints.

2.2 Design principles

Whereas SBML is a very specific language for describing models within the systems biology field only, Modelica is a much more general and expressive language. One of the challenges in writing a translator lies in finding a way to map different language constructs and features between the two languages. There is no obvious one-to-one mapping as there are usually many different ways of describing things in Modelica. One way to solve this is to try and catch all possible constructs in Modelica and map them to SBML. Another way is to restrict how Modelica models may be constructed in order to reduce the number of possibilities.

In *MathModelica* a combination is used. The BioChem library provides the user with the building blocks that must be used in order for a model to be exportable to SBML. *MathModelica* also provides wizards to help the user build custom components, such as reactions and compartments. These wizards are interactive step-by-step guides, serving two purposes, guiding the user through complex operations and helping the user create models that will be exportable to SBML.

2.3 Verification of the translator

The translator has been tested by importing all curated models from the BioModels database [4] release 14. Table 1 shows the percentage of models that (a) are possible to import to *MathModelica* without failure, (b) give the same simulation result as the reference simulation, and (c) give the same simulation result after exporting to SBML.

Table 1 *Verification of import and export of SBML models. Export has only been tested for models that do not contain the piecewise function.*

	Tested models	Succeeded
(a) Import	216	212 (98 %)
(b) Simulation	212	208 (98 %)
(c) Export	18	18 (100 %)

Three of the models that fail to import because they use comparison between reals, which is not allowed in Modelica, while the fourth model fails because the translator does not yet support delay in SBML events. One of the failing simulations is due to outdated SBML code (to many equations which was allowed previously, but is not allowed in Level 2 version 3), while the three others fail due to numerical reasons.

2.4 Comparison with other tools

The models in the BioModels database have been developed in different SBML tools, such as Systems Biology Workbench (SBW) [5], CellDesigner [6], and Copasi [7].

The libSBML [8] library contains consistency checks to ensure that a model is consistent with the SBML specification. However, this does not ensure that a model works to simulate in a specific software.

Therefore we have performed a comparison between *MathModelica,* Systems Biology Workbench, and CellDesigner to check if the results match the published results from the original tool used for respective model on the BioModels database. Due to time constraints we could not test the full set of available models, instead a subset was chosen, by testing every fifth model (every model with model name ending with either 0 or 5). The result is presented in Table 2.

Table 2 *Comparison between different modelling environments.*

	Successful simulations
MathModelica 2.1 b4	98 % (42 of 43)
SBW 2.7.9	56 % (24 of 43)
CellDesigner 4.01	40 % (17 of 43)

The results indicate that *MathModelica* has a good support for the SBML language, especially in comparison with SBW and CellDesigner, however improvements are still needed to reach a complete support.

2.5 Related work

The BioDyn [8][9][10] software can make external calls to a few other software, including OpenModelica [11][12]. It also includes import functionality from Modelica to SBML. However, SBML models are imported as flat Modelica i.e. the pathway diagram is lost in the translation.

There is also some other related work done, but with limited or no public information available.

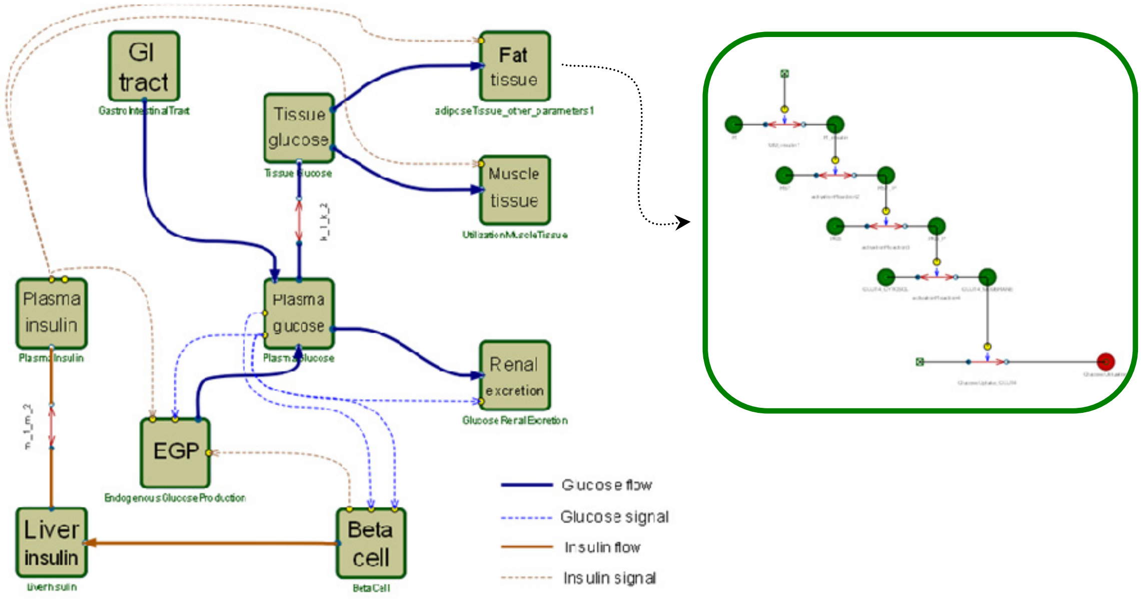


Figure 2 *The top-level of the whole-body model (left) shows the organs, the flows, and the information exchange between the organs, in an underlying level the detailed fat tissue module (right) is found.*

3 Modelling Example

The Dalla Man model [4] is one of the best available models for the whole-body glucose-insulin system, and the model has even been formally accepted as a viable tool when certifying new drugs for diabetes. This model is, however, a non object-oriented model developed in MATLAB.

Based on this model a multi-level object-oriented whole-body model of the glucose-insulin system has been developed using *MathModelica*, MATLAB, and – for the communication between them – SBML.

3.1 Creating an object-oriented model

The original MATLAB model was exported to SBML and inserted in *MathModelica*, and then reformulated into an object-oriented format, in which each object typically correspond to an organ, thus making it possible to successively replace respective organ with more detailed models. The top level of the model can be seen in Figure 2. As long as the input-output profiles of the replaced organ is preserved this can be done with negligible effects on the whole-body level.

3.2 Detailed adipose tissue model

A detailed model for the adipose tissue (fat tissue) was developed (in MATLAB) using mechanistic *a priori* information and existing and novel data from experiments on human fat cells. The parameters of the model were then optimized so that the simulations fitted both the input-output profile of the old model, and the cell-level data.

The model was exported to SBML, and then imported to *MathModelica*, and inserted in the whole-body model. As expected, this new detailed module did not cause any major changes in the whole-body behaviour (Figure 3), even though the intercellular glucose utilization rate (i.e. within the altered adipose tissue) had a somewhat different profile (Figure 4). Hence, the resulting object-oriented model can be said to have a "zoomable" adipose tissue.

To further show the strength of the object-oriented model, a parameter in the insulin signalling cascade was changed, so that the adipose tissue became 5 times less sensitive to insulin. This corresponds to a diabetic fat tissue in an otherwise healthy body. The simulations (Figure 2 and 3) predict that these rather big intracellular changes only leads to slightly higher glucose in the plasma, since a healthy body easily can adapt to the new situation, by slightly increasing the insulin level.

A main strength of the developed model is that it has the ability to translate mechanistically oriented simulations on the biochemical/cellular level, which is the level were drugs act, to the whole-body level, which is the level of clinical interest.

Another important benefit of object-oriented modelling is that the modeller is forced to adopt the cell-oriented experiments to physiologically realistic conditions, compatible with the whole-body dynamics.

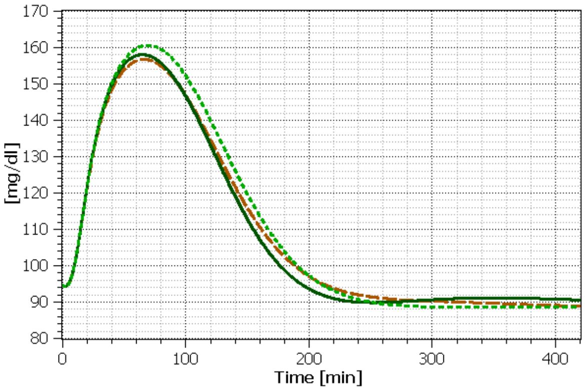


Figure 3 *Simulation results of glucose concentration in plasma of the original model (—), with the mechanistic fat tissue module included (---), and with changed parameters on fat tissue module-level (···).*

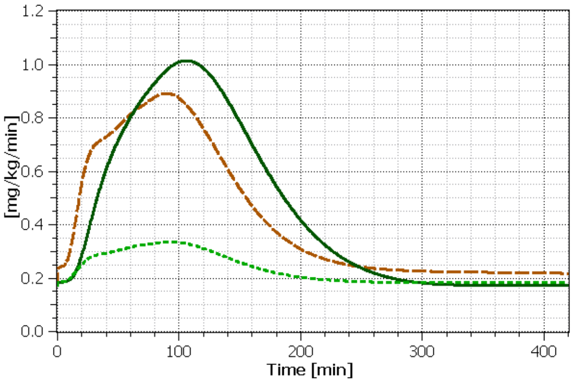


Figure 4 *Simulation results of glucose utilization rate by adipose tissue of the original model (—), with the mechanistic fat tissue module included (---), and with changed parameters on fat tissue module-level (···).*

4 Publishing on the Web

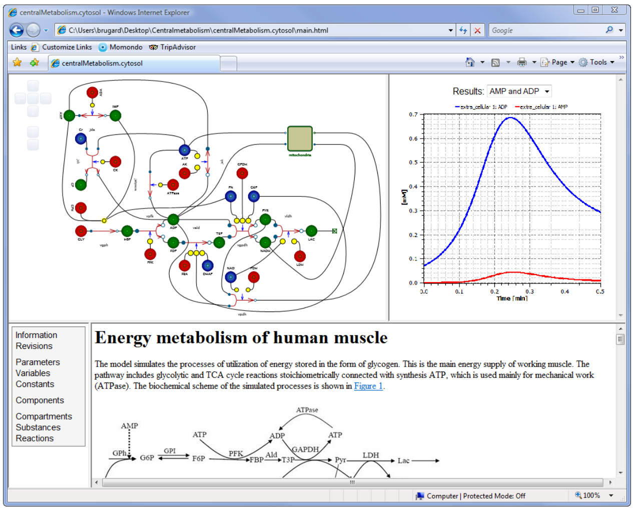
In order to communicate results with others a publish feature has been developed for *MathModelica*. Once a Modelica model has been developed it can be exported to an interactive HTML page.

It is possible to publish not only the selected model, but also its components. The exported model contains interactive model diagram based on the Silverlight technology [14]), model information (documentation), and selected results.

Figure 5 shows an example of a published central metabolism model, originally developed by Selivanov et al [3].

4.1 Publishing on the BioBridge portal

Models can also be published as an integrated part of the BioBridge portal. The BioBridge portal allows modellers (and other users) to access an integrated resource including clinical, genomic, proteomic, and metabolomic information for the metabolic pathways affected by disorders such as chronic obstructive pulmonary disease (COPD), cardiac disease and diabetes. This information is then applied to model the underlying metabolic network. This, in turn, can then be utilized by the modeller for simultaneous analysis of multilevel data in order to improve existing knowledge on complex disorders.



Energy metabolism of human muscle

The model simulates the processes of utilization of energy stored in the form of glycogen. This is the main energy supply of working muscle. The pathway includes glycolytic and TCA cycle reactions stoichiometrically connected with synthesis ATP, which is used mainly for mechanical work (ATPase). The biochemical scheme of the simulated processes is shown in Figure 1.

Figure 5 *Screenshot of the published central metabolism model. Upper left corner shows the (clickable) model diagram of the cytosol pathway, upper right the concentration of AMP and ADP substances, lower left browse menu, and finally the lower right shows the model documentation (at the cytosol level).*

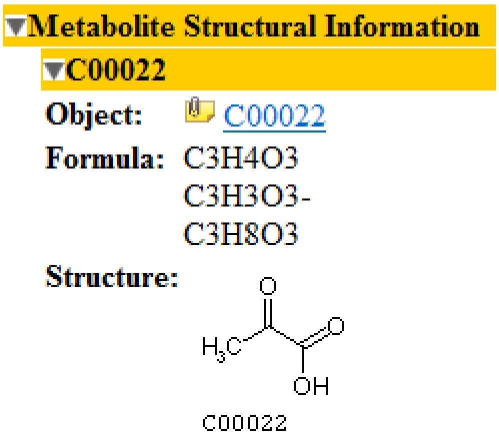
4.2 Connecting to BioXM

The *MathModelica Systems Biology* toolbox also includes the possibility to tag substances in order to associate a specific substance with information available in the knowledge management environment BioXM [17] [18]. When a user selects a substance with a tag, an http request is made to the BioBridge portal and information is retrieved using BioXM, as illustrated in Figure 6.

BioXM directly, or by access to their WebService query interfaces, integrates more than 20 different public databases and ontologies (see

Table 3 for integrated databases) representing a total of 80 793 genes (30 246 human, 27 237 mouse, 23 310 rat), 1 307 pathways, 78 528 compounds, 1 525 474 protein interactions a total of 3 666 313 connections within the knowledge network and the entire Gene Expression Omnibus database. Clinical and experimental data from the two clinical BioBridge studies of chronic obstructive pulmonary disease (COPD) has been integrated with these public resources.

This information is combined with BioBridge literature-mining derived molecular networks for COPD, cardiac disease, chronic systemic inflammation, diabetes and lung and muscle specific signalling sub-networks. Experimental molecular sets encompass gene expression, metabolomics and proteomics data. Exercise and COPD specific kinetic and metabolic data relevant to constrain muscle metabolic models has been extracted from 30 published clinical trials. Finally the mathematical models and probabilistic networks generated within the BioBridge project are fed back into BioXM.



▼Metabolite Structural Information

▼C00022

Object: C00022

Formula: C3H4O3
C3H3O3-
C3H8O3

Structure:

Figure 6 *Metabolite structural information for Pyruvate given by BioXM.*

Table 3: *Public molecular data resources integrated into BioXM*

Source Database	Information Type	Current Statistics	Level of curation	Updates/ Version
BIND	Protein Interaction Molecular complexes Pathways	6256 Interactions	High throughput data submission and hand curated from the literature	last public version 20.3.07
BioGrid	Protein interaction	19 707 interactions	Manually curated from literature Different evidence codes	updated monthly
Biomodels	SBML models	224 SBML models	Partial verification of model simulation	updated monthly
ChEBI	Compound information	15 367	Curated from different data sources	updated weekly
Comparative Toxicogenomics Database (**CTD**)	Compound-gene, Compound-disease and Gene-disease relationships	259 898 relations	Manually curated from the published literature	updated monthly
GEO	Expression data	12 543 studies, 257 312 expression sets	Partial manual curation of metainformation into study sets	updated monthly
EntrezGene	Gene functional information	80 793 human, mouse and rat genes	Curated information integrated from different databases, based on RefSeq genomes	updated weekly
Enzyme	Enzyme related functional information	4 833	Manually curated from the published literature	updated weekly
IntAct	Protein interaction	21 584 binary interactions	Literature curation User submission	updated weekly
KEGG	Pathways	418 pathways	Manually curated from the published literature	updated monthly
LIGAND	Compound information	15 185	Manually curated from the published literature	updated monthly
MIPS Mammalian	Protein Interaction	410 interactions	Manually curated from the published literature	current Release 31.10.07
OMIM	Gene - disease relations	19 769	Curated from the published literature	updated weekly
Pfam	Protein family information	10 340 families	Manually curated from sequence alignments	23.0

5 Limitations and Current Work

The translator is currently in a beta stage and lacks support for unit conversions, and some, as it appears, less commonly used elements of SBML Level 2 version 3, such as initial assignments and constraints. Warnings will be issued if unsupported elements are detected during translation. One element (the piecewise function of SBML) is supported by the SBML to Modelica translator but not yet by the Modelica to SBML translator. Most of these shortcomings will be addressed in the final release of the translator.

The implementation where BioXM is used to access and retrieve information has served as a test implementation and will be extended to make use of the SBML RDF annotations in order to associate entities with biochemical or biological semantics.

During the development of the translator, SBML Level 2 version 4 was released. This version has not yet been looked at and hence is not supported by the translator.

Due to the lack of an accepted standard for storing graphical diagrams of models in SBML, the translator does not support conversion of graphical representations of models. SBML Level 3 is expected to resolve this issue.

While the wizards in *MathModelica* do a good job assisting the user in creating SBML exportable compartments, substances, and reactions in Modelica, they do not let the user edit these components once created. We intend to address this limitation in future releases.

The current version includes Metabolic Control Analysis (MCA) and dynamic sensitivity analysis; however there are various other common tasks like parameter estimation, model reduction, and stochastic simulation that could be added in the future.

6 Conclusions

The ambition is to create a bridge between the systems biology community and the Modelica community. The main benefits of the project is that it makes it possible to 1) convert an existing SBML model and benefit from Modelicas multi domain technology when extending the model, and 2) get direct access to biological databases and tools when developing a biochemical pathway in Modelica.

A public available Modelica library, BioChem, for modelling cellular pathways in Modelica has been developed and a toolbox, *MathModelica Systems Biology*; that allows import/export to SBML, publishing of models and results with connections to biochemical databases has also been developed.

The translation from SBML to Modelica allows people that work with SBML, to import their models to *MathModelica* and use the full strength of the Modelica language to expand their models.

All in all, we believe that hierarchical modelling has a strong future for modelling of complex biological processes, and that *MathModelica*, in communication with SBML-based software and models, has the potential to take a leading role in such efforts.

Acknowledgements

The BioBridge project is supported by the European Commission and is part of the Sixth Framework Programme.

References

[1] Larsdotter Nilsson, E. and Fritzson P., (2003). BioChem - A Biological and Chemical Library for Modelica. Proceedings of the 3rd International Modelica Conference: 215-220.

[2] BioChem library. (Accessed 2009-08-18) http://www.mathcore.com/products/mathmodelica/libraries/biochem.php

[3] Systems Biology Markup Language (Accessed 2009-08-18) http://www.sbml.org

[4] The BioModels Database - A Database of Annotated Published Models. (Accessed 2009-08-18) http://www.ebi.ac.uk/biomodels-main/

[5] Systems Biology Workbench. (Accessed 2009-08-20) http://sbw.sourceforge.net/

[6] CellDesigner. (Accessed 2009-08-20) http://www.celldesigner.org/

[7] COPASI - Complex Pathway Simulator. (Accessed 2009-08-20) http://www.copasi.org

[8] libSBML. (Accessed 2009-08-18) http://sbml.org/Software/libSBML

[9] BioDyn. (Accessed 2009-08-18) http://cbbl.imim.es:8080/ByoDyn

[10] Gómez-Garrido, À. et al. (2008) ByoDyn: integrating computational methods for the analysis of biochemical models. P oster pre-

sented at the 4th Meeting of the Spanish Systems Biology Network (REBS).

[11] OpenModelica. (Accessed 2009-08-18) http://www.ida.liu.se/~pelab/modelica/OpenModelica.html

[12] Fritzson, P. et al. (2006) OpenModelica - A Free Open-Source Environment for System Modeling, Simulation, and Teaching. IEEE International Symposium on Computer-Aided Control Systems Design.

[13] Dalla Man C, Rizza R A and Cobelli C (2007). Meal simulation model of the glucose-insulin system, IEEE transactions on bio-medical engineering, Vol. 54, No. 10, pp. 1740-1749.

[14] Microsoft Silverlight. (Accessed 2009-08-18) http://silverlight.net

[15] Selivanov VA, et al., (2008). The changes in the energy metabolism of human muscle induced by training. Journal of Theoretical Biology 252, 402-410.

[16] The BioBridge portal. (Accessed 2009-08-18) http://www.biobridge.eu

[17] Sameith, K. et al., (2008). Functional Modules integrating essential cellular functions are predictive of the response of leukaemia cells to DNA damage. Bioinformatics 24: 2602-2607.

[18] Losko, S. et al. (2006) Knowledge Networks of Biological and Medical Data: An Exhaustive and Flexible Solution to Model Life Science Domains. Lecture Notes in Computer Science 4075: 232-239.

Optimization of a Pendulum System using Optimica and Modelica

Pontus Giselsson[a] Johan Åkesson[a,b] Anders Robertsson[a]
[a)]Dept. of Automatic Control, Lund University, Sweden
[b)]Modelon AB, Sweden

Abstract

In this paper Modelica and Optimica are used to solve two different optimal control problems for a system consisting of a pendulum and a cart. These optimizations demonstrates that Optimica is easy to use and powerful when optimizing systems with highly non-linear dynamics. The optimal control trajectories are applied to a real pendulum and cart system, in open loop as well as in closed loop with an MPC-controller. The experiments show that optimal trajectories from Optimica together with MPC feedback is a suitable control structure when optimal transitions through non-linear dynamics are desired.

Keywords: Optimal control, Optimica, Modelica

1 Introduction

Optimal control problems for dynamical systems with non-linear dynamics often lead to non-convex optimization problems. These problems are usually difficult to solve and lots of time and effort is usually spent on transforming the optimal control problem into a numerical optimization problem. In this paper we use the high-level languages Modelica together with Optimica to solve two different optimal control problems for a pendulum and cart system. The Modelica and Optimica combination allows the user to concentrate on how to formulate the optimal control problem, rather than on how to transform it into a numerical optimization problem. The pendulum dynamics are highly non-linear which makes this an appropriate application to show the efficiency of the Optimica and Modelica combination. The first optimization problem considered in the paper is to swing up the pendulum from its downward position to the inverted position in as short time as possible. The second problem is to move the cart from one position of the track to another in as short time as possible, while the pendulum end point, i.e., the end of the pendulum that is not attached to the cart, must avoiding an elliptical obstacle.

We also present experimental results where the optimal control trajectories are applied to a real pendulum on a cart system. There is a close match between the optimal trajectories and the real system trajectories when no or small disturbances are present. This demonstrates that optimal control is applicable to the real process. Of course, we get good experimental results when the process is accurately described by the model. When larger disturbances are present, e.g., in the initial conditions, the optimal control trajectories applied to the real process result, as expected, in state trajectories that are far from the optimal ones. A Model Predictive Controller (MPC) is introduced to minimize the influence of these disturbances. Experimental results show that the combination of optimal control feed-forward and MPC-feedback is a suitable control structure for these problems where optimal transitions through non-linear dynamics are desired.

The remainder of the paper is organized as follows. In Section 2 an introduction to the Modelica extension Optimica is given. Section 3 describes the cart and pendulum process used in the paper. In Section 4 we state and solve two optimization problems using Optimica and Modelica. Results from the optimizations are applied to the real pendulum, in open loop as well as in closed loop with an MPC-controller, in Section 5. Section 6 describes how Optimica and this particular application is used in the education at the Dept. of Automatic Control, Lund University. Finally in Section 7 we give some conclusions.

2 Optimica and JModelica.org

Modelica does not offer explicit support for formulation of dynamic optimization problems. In particular, means to express quantities such as cost function, con-

DOI: 10.3384/ecp09430094

straints, optimization interval, and optimization parameters are lacking. In an effort to extend Modelica to also include high-level formulation of dynamic optimization problems, the Optimica extension was proposed [1]. The Optimica extension is supported by the novel Modelica-based open source platform JModelica.org [8].

2.1 JModelica.org

JModelica.org is a novel Modelica-based open source project targeted at dynamic optimization [2], [3]. JModelica.org features compilers supporting code generation of Modelica models to C, a C API for evaluating model equations and their derivatives and optimization algorithms. The compilers and the model C API has also been interfaced with Python [6] in order to enable scripting and custom application development. In order to support formulation of dynamic optimization of Modelica models, JModelica.org supports the Optimica extension [1]. Optimica offers constructs for encoding of cost functions, constraints, the optimization interval with fixed or free end points as well as specification of transcription scheme.

The JModelica.org platform contains an implementation of a simultaneous optimization method based on orthogonal collocation on finite elements [5]. Using this method, state and input profiles are parametrized by Lagrange polynomials, of order three and four respectively, based on Radau points. This method corresponds to a fully implicit Runge-Kutta method, and accordingly it possesses well known and strong stability properties. By parameterizing the variable profiles by polynomials, the dynamic optimization problem is translated into a non-linear programming (NLP) problem which may be solved by a numerical NLP solver. This NLP is, however, very large. In order to efficiently find a solution to the NLP, derivative information as well as the sparsity patterns of the constraint Jacobians need to be provided to the solver. The simultaneous optimization algorithm has been interfaced with the large-scale NLP solver IPOPT [10], which has been developed particularly to solve NLP problems arising in simultaneous dynamic optimization methods.

The choice of a simultaneous optimization algorithm fits well with the properties of the dynamic optimization problems treated in this paper. In particular, simultaneous methods handle unstable systems well, and also, state and input inequality constraints are easily incorporated.

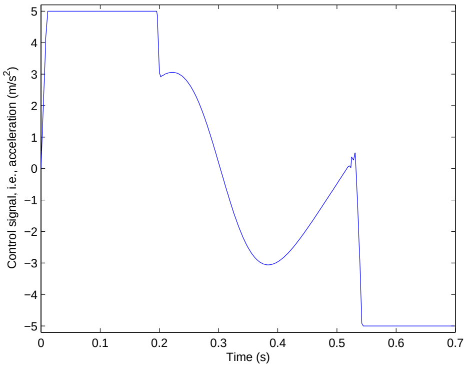


Figure 1: Control signal in the constrained double integrator example in Section 2.2.

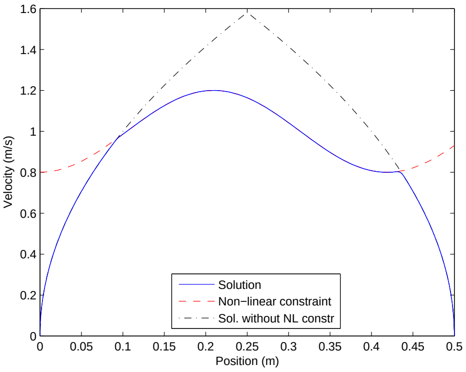


Figure 2: Phase plot of x and $\dot{x}$ in the constrained double integrator example in Section 2.2. Also the non-linear constraint and the solution to the unconstrained problem are plotted.

2.2 Optimica example

In this section, the Optimica syntax is explained by stating and solving a double integrator optimization problem. The example will also serve as an evaluation of the accuracy of the Optimica solution compared to the optimal solution. The following optimization problem is solved:

$$\begin{array}{ll} \min\limits_{u} & t_f \\ \text{subject to} & \ddot{x} = u \\ & 0.2\cos 15x + \dot{x} \leq 1 \\ & |u| \leq 5 \\ & x(0) = 0 \qquad \dot{x}(0) = 0 \\ & x(t_f) = 0.5 \quad \dot{x}(t_f) = 0 \end{array} \tag{1}$$

where t_f is the final time, u is the control signal, x is the position and $\dot{x}$ is the velocity. The non-linear constraint is added to make the problem a bit more complex. A Modelica model for a double integrator is:

```
model DoubleIntegrator
  package SI = Modelica.SIunits;
  SI.Position x(start=0);
  SI.Velocity x_dot(start=0);
  input SI.Acceleration u;
equation
  der(x) = x_dot;
  der(x_dot) = u;
end DoubleIntegrator;
```

An Optimica specification of the problem is:

```
optimization DIopt (objective=finalTime,
                    startTime=0,
                    finalTime=(free=true,
                               initialGuess=1))
  DoubleIntegrator DI(u(free=true,
                        initialGuess=0.0));
constraint
  DI.x(finalTime)=0.5;
  DI.x_dot(finalTime)=0;
  0.2*cos(15*DI.x)+DI.x_dot <= 1;
  DI.u <= 5;
  DI.u >= -5;
end DIopt;
```

In the first line of the Optimica specification the optimization objective is specified. In this case the objective to be minimized is the final time. Then the Modelica model of the dynamical system that is used in the optimization is specified and u is chosen to be the decision variable. Then all constraints, inequality as well as equality constraints, are listed.

The solution to (1) is obviously to accelerate with maximum positive acceleration until, or if, the constraint is reached. Then continue with maximum allowed velocity until deceleration is needed to reach $x = 0.5$ and $\dot{x} = 0$. This behaviour is clearly seen in the Optimica solution of the problem, Figures 1 and 2.

3 The Process

The Department of Automatic Control in Lund has a history of designing and building laboratory processes. One of the latest processes that are built in-house is the pendulum and cart process depicted in Figure 3. This process is used in this paper to demonstrate the applicability of Optimica and optimal control.

3.1 Cart control

The cart is driven by a DC-motor which is controlled in a cascaded structure. See Figure 4 for a schematic view of the cascaded control structure. There is an inner loop that controls the current through the DC-motor. P_1 represents the current dynamics which behaves like a first order system with a time-constant of 0.17 ms. C_1 represents the PI-controller in the current loop that controls the current, i, to its reference, i_r. The current reference, i_r, is set by the outer loop that controls the cart velocity. The current dynamics are fast in comparison to the velocity dynamics, which makes $i_r \approx i$ a good approximation. The transfer function from i to v, i.e. P_2, is ideally an integrator with a gain. The velocity dynamics are controlled with another PI-controller, C_2. The reference to the velocity control loop, v_r, is integrated from an acceleration reference, u, since acceleration is our desired control signal. This cascaded control structure is suitable when fast closed loop dynamics from v_r to v is desired. Since $v_r \approx v$ is a good approximation, we have double integrator dynamics from control signal, u, to cart position, x.

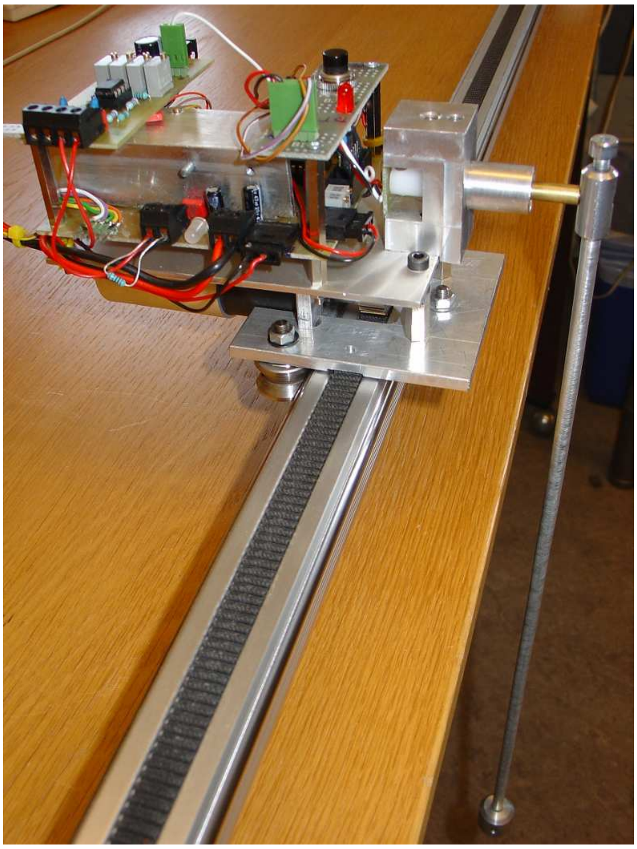


Figure 3: Photo of the cart and pendulum system described in Section 3.

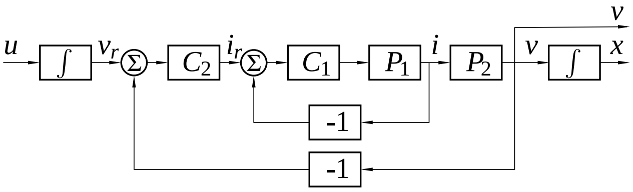


Figure 4: Cascaded control structure for the cart control.

3.2 Hardware setup

On the cart there are two Atmel ATmega16 micro processors. The current controller, C_1 in Figure 4, discussed in Section 3.1, is running on one of them at a sampling rate of 28.8 kHz. This micro processor gets the current reference, i_r, from the other micro processor, where the velocity controller, C_2, is running at 1 kHz. This second micro processor also communicates with a PC via the serial interface. This communication is performed at frequencies around 50 Hz. From Matlab/Simulink on the PC, the velocity reference, v_r, is sent to the velocity controller on the micro processor. The velocity reference is obtained by integrating the acceleration reference, u, on the PC-side. Since a smooth acceleration profile of the cart is desired, the velocity reference needs to be updated more frequently than at 50 Hz. Therefore the acceleration reference, u, is also sent to the velocity controller from the PC. The velocity reference is updated in the micro processor at a frequency of 1 kHz according to $v_r(t) = v_r(t_0) + u(t_0)(t - t_0)$, where t_0 is the time when the last references was received from the PC, $t \in [t_0, t_0 + h]$ and h is the PC communication sampling time. These updates are consistent with the velocity reference in the next sample from the PC which is $v_r(t_0 + h) = v_r(t_0) + u(t_0)h$.

One alternative would be to send only the acceleration reference to the micro processor and to calculate the velocity reference there. This would imply that in order to stop the cart, it must be controlled with a feedback loop on the PC. With our implementation structure the cart can easily be stopped by setting the velocity reference to zero.

The PC also receives cart position and pendulum angle measurements as well as velocity estimates from the micro processor. This enables for us to, on the PC, create another level of feedback loops in the cascaded control structure.

3.3 System modeling

Due to the low level control of the cart, described in Section 3.1, the cart behaves as a double integrator. When x is the cart position and u the control signal, we have the following cart dynamics

$$\ddot{x} = u$$

The pendulum dynamics are well known; let θ be the pendulum angle and we get

$$\ddot{\theta} = -\frac{g}{l}\sin\theta + \frac{a}{l}\cos\theta$$

where $\theta = 0$ is defined to be the pendulum downward position, g is the gravitational acceleration, l is the pendulum length and a is the horizontal acceleration of the pendulum pivot point. This horizontal acceleration, a, is equal to the cart acceleration, $\ddot{x}$, and thus equal to the control signal, u. This gives us the following model for the complete system

$$\ddot{\theta} = -\frac{g}{l}\sin\theta + \frac{u}{l}\cos\theta \tag{2}$$

$$\ddot{x} = u \tag{3}$$

```
model Pendulum
  package SI = Modelica.SIunits;
  parameter SI.Length l = 0.4;
  constant SI.Acceleration g = 9.81;
  SI.Position x(start=0);
  SI.Velocity x_dot(start=0);
  SI.Angle theta(start=0);
  SI.AngularVelocity theta_dot(start=0);
  SI.Position x_p;
  SI.Position y_p;
  Real u_dot(unit="m/s3");
  input SI.Acceleration u;
equation
  der(x) = x_dot;
  der(x_dot) = u;
  der(theta) = theta_dot;
  der(theta_dot)=-g/l*sin(theta)+1/l*cos(theta)*u;
  der(u) = u_dot;

  x_p = x-l*sin(theta);
  y_p = -l*cos(theta);
end Pendulum;
```

Listing 1: A Modelica model for the pendulum and cart system.

A schematic view of the full system is found in Figure 5, where S_{pend} represents the non-linear pendulum dynamics (2) and G_{cart} represents the double integrator dynamics (3), z is a vector containing the states, $z = (x\,\dot{x}\,\theta\,\dot{\theta})^T$ and S represents the full system. The position of the cart and the pendulum angle are defined such that the pendulum end point in the horizontal direction, x_p, and in the vertical direction, y_p, are given

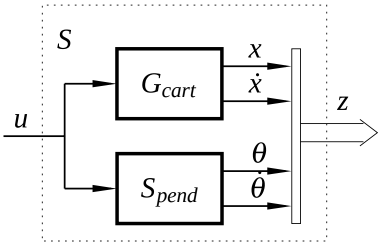


Figure 5: Schematic view of the system.

by

$$x_p = x - l\sin\theta$$
$$y_p = -l\cos\theta$$

The Modelica model that describes this pendulum and cart system is found in Listing 1.

4 Optimization

In this section we use Modelica and Optimica to solve two different optimization problems based on the pendulum and cart model. The first problem is to swing up the pendulum in as short time as possible. The second problem is to move the pendulum and cart from rest at one cart position on the track to another, while the end point of the pendulum must avoid an elliptical obstacle. Also in this second problem the objective to be minimized is the final time.

4.1 Time-optimal Swing-up

The optimization objective is to swing up the pendulum from the downward pendulum position to the inverted pendulum position in as short time as possible. The cart should stop at the same position as it started and the cart and angular velocities should be zero at the final time. The control signal, i.e., the cart acceleration, u, is limited to the interval ± 5 m/s^2. Its derivative, $\dot{u}$, is limited to the interval ± 100 m/s^3. The cart track is limited, which lead to constraints in the cart position. The cart position must satisfy -0.5 m $\leq x \leq 0.5$ m. The optimization problem is stated mathematically in (4)

$$\begin{array}{lll} \min\limits_{u} & t_f & \\ \text{subject to} & \ddot{\theta} = -\frac{g}{l}\sin\theta + \frac{u}{l}\cos\theta & \\ & \ddot{x} = u & \\ & -0.5 \leq x \leq 0.5 & \\ & |u| \leq 5 \quad\quad |\dot{u}| \leq 100 & \\ & \theta(0) = 0 \quad\quad \dot{\theta}(0) = 0 & \\ & x(0) = 0 \quad\quad \dot{x}(0) = 0 & \\ & \theta(t_f) = \pi \quad\quad \dot{\theta}(t_f) = 0 & \\ & x(t_f) = 0 \quad\quad \dot{x}(t_f) = 0 & \end{array} \qquad (4)$$

where t_f is the final time. The Modelica and Optimica codes that describe the optimization problem are found in Listings 1 and 2 respectively. The resulting time optimal state and control trajectories are found in Figures 6 and 7. The pendulum angle changes sign two times during the swing-up. It starts with a positive angle, switches to negative and finally it reaches its inverted position with a positive angle. This means that the optimal swing-up is performed with three swings before the inverted position is reached. In [4] the minimum number of pendulum swings needed for swing-up, given a maximum acceleration of the pendulum pivot point, a_{max}, is analyzed. Three swings are needed if $0.388g \leq a_{max} \leq 0.577g$ which is equivalent to 3.81 m/s^2 $\leq a_{max} \leq$ 5.66 m/s^2. This analysis is not directly applicable to our setup since cart constraints and acceleration rate limitations are not considered in the analysis in [4]. When cart terminal position and acceleration rate constraints are chosen as in our setup, three swings are needed for swing-up if 4.45 m/s^2 $\leq a_{max} \leq$ 7.70 m/s^2. This interval is obtained simply by solving the swing-up problem with different acceleration constraints in Optimica. The fact that our problem with additional constraints requires more acceleration to swing-up the pendulum with a fixed number of swings, is not surprising. The max-

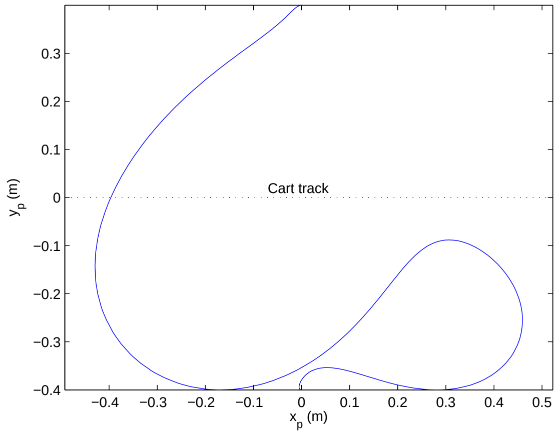


Figure 6: Optimal trajectory of the pendulum end point for swing-up problem in Section 4.1.

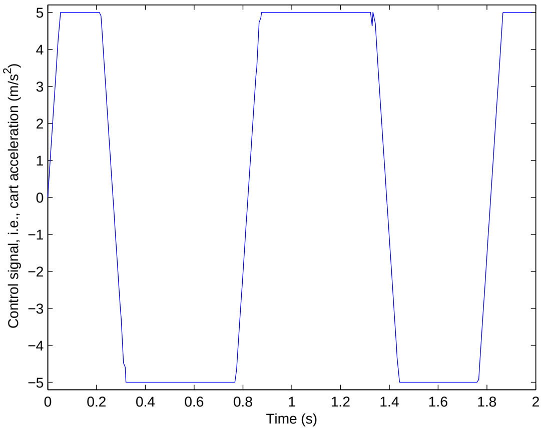


Figure 7: Optimal control signal, i.e., cart acceleration, for swing-up problem in Section 4.1.

```
optimization Swingup (objective=finalTime,
                      startTime=0,
                      finalTime=(free=true,
                                 initialGuess=1))
  Pendulum pend(u(free=true,initialGuess=0.0));
constraint
  pend.x(finalTime)=0;
  pend.x_dot(finalTime)=0;
  pend.theta(finalTime)=3.1415;
  pend.theta_dot(finalTime)=0;

  pend.x <= 0.5;
  pend.x >= -0.5;
  pend.u <= 5;
  pend.u >= -5;
  pend.u_dot <= 100;
  pend.u_dot >= -100;
end Swingup;
```

Listing 2: An Optimica model for time optimal swing-up of the pendulum.

imum acceleration in our example is 5 $\mathrm{m/s}^2$ which is within the interval where a minimum of three swings are needed.

In [4] they also discuss an energy based swing-up strategy that was originally proposed in [11]. The idea of the method is to control the system to the energy-level that corresponds to the inverted pendulum position using maximum acceleration in either way. When this energy based approach is applied to this system, with $a_{max} = 5\ \mathrm{m/s}^2$, the pendulum reaches its inverted position when the cart position is approximately 3m from its starting point. This position is far outside the track, which is why this energy based method is not directly applicable when track limitations are present.

4.2 Optimization with path-constraints

In this optimization problem we want the cart to start at rest at position $x = 0$ with the pendulum in the downward pendulum position, $\theta = 0$. At the final time, the cart and pendulum should be at rest at position $x = 0.8$ m and pendulum angle $\theta = 0$. We also introduce an additional constraint stating that the end point of the pendulum must never enter an elliptical area described by

$$\left(\frac{x_p - 0.5}{0.05}\right)^2 + \left(\frac{y_p + 0.4}{0.3}\right)^2 = 1$$

Due to the track limitations we need the cart position to satisfy $-0.1\ \mathrm{m} \leq x \leq 0.9$ m. The control signal limitations are the same as in the previous optimization problem, i.e. $-5\ \mathrm{m/s}^2 \leq u \leq 5\ \mathrm{m/s}^2$ and $-100\ \mathrm{m/s}^3 \leq \dot{u} \leq 100\ \mathrm{m/s}^3$. The objective of the optimization is to reach the final states as fast as possible. The optimization problem is described mathematically in (5)

$$\begin{array}{lll}
\min\limits_{u} & t_f & \\
\text{subject to} & \ddot{\theta} = -\frac{g}{l}\sin\theta + \frac{u}{l}\cos\theta & \\
& \ddot{x} = u & \\
& x_p = x - l\sin\theta & \\
& y_p = -l\cos\theta & \\
& \left(\frac{x_p-0.5}{0.05}\right)^2 + \left(\frac{y_p+0.4}{0.3}\right)^2 \geq 1 & \\
& -0.1 \leq x \leq 0.9 & \\
& |u| \leq 5 \qquad |\dot{u}| \leq 100 & \\
& \theta(0) = 0 \qquad \dot{\theta}(0) = 0 & \\
& x(0) = 0 \qquad \dot{x}(0) = 0 & \\
& \theta(t_f) = 0 \qquad \dot{\theta}(t_f) = 0 & \\
& x(t_f) = 0.8 \quad \dot{x}(t_f) = 0 &
\end{array} \tag{5}$$

where t_f again is the final time. The codes in the corresponding Modelica and Optimica files are found in Listings 1 and 3 respectively. This problem turns out to be more difficult to solve than the swing-up problem. Actually it is not easy to find a solution that is feasible, i.e., that satisfies all constraints. In order to solve this problem we need to give the solver an initial guess that is feasible and not too far away from the optimum. One crucial decision to make is if the pendulum should follow behind the cart over the obstacle, or if it should go in front of the cart. It turns out that if the pendulum follows behind the cart we get

```
optimization Path (objective=finalTime,
                   startTime=0,
                   finalTime=(free=true,
                              initialGuess=1))
  Pendulum pend(u(free=true,initialGuess=0.0));
constraint
  pend.x(finalTime)=0.8;
  pend.x_dot(finalTime)=0;
  pend.theta(finalTime)=0;
  pend.theta_dot(finalTime)=0;

  pend.x <= 0.9;
  pend.x >= -0.1;
  pend.u <= 5;
  pend.u >= -5;
  pend.u_dot <= 100;
  pend.u_dot >= -100;

  ((pend.x_p-0.5)/0.05)^2+((pend.y_p+0.4)/0.3)^2>=1;
end Path;
```

Listing 3: An Optimica model for the path following problem.

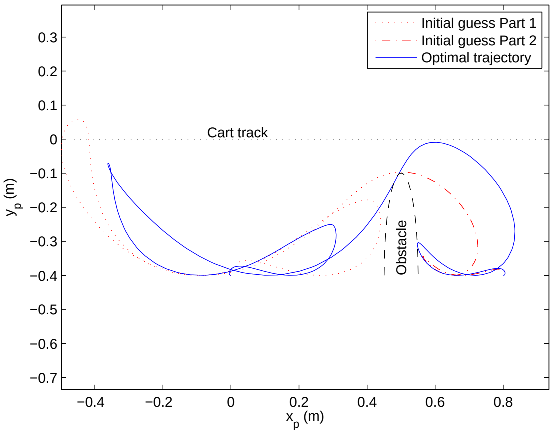


Figure 8: Optimal trajectory of the pendulum end point for path constrained problem in Section 4.2. Also the pendulum end point in the two parts of the initial guess is plotted.

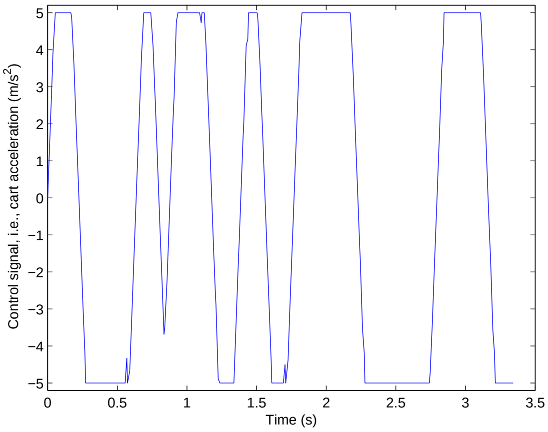


Figure 9: Optimal control signal, i.e., cart acceleration, for path constrained problem in Section 4.2.

very large oscillations after passing the obstacle. It is time-inefficient to damp these resulting pendulum oscillations because of the track and control limitations. Thus the time-optimal solution must have the pendulum in front of the cart when passing the obstacle. To help the optimizer finding this solution the problem is divided into two smaller and easier subproblems.

The first subproblem is an altered version of the original problem (5). The elliptical constraint is removed and the final constraints are set to

$$\begin{array}{ll} \theta(T) = -\frac{75.52\pi}{180} & \dot{\theta}(T) = 0 \\ x(T) = 0.1127 & \dot{x}(T) = 1.4 \end{array} \qquad (6)$$

This terminal point of the optimization corresponds to when the pendulum is precisely above the obstacle with the pendulum leaning in the forward direction.

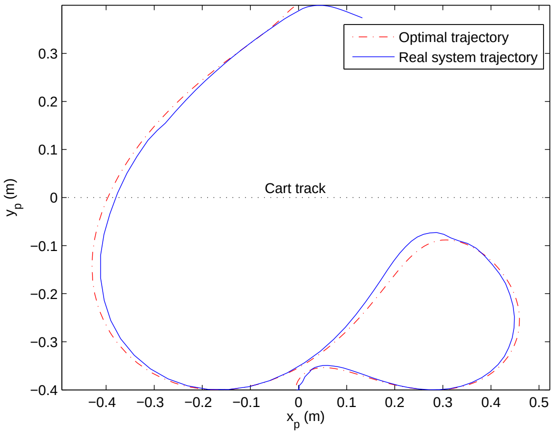


Figure 10: Experimental results for swing-up problem when control trajectory applied in open loop as described in Section 5.1. The optimal pendulum end point trajectory is also plotted for comparison reasons.

The terminal cart velocity, $\dot{x}$, is set to a positive value since we want the cart to have a forward motion over the obstacle. The angular velocity of the pendulum, $\dot{\theta}$, is set to zero which makes it possible for the pendulum angle to decrease directly after passing the obstacle. The terminal cart and pendulum angular velocities are chosen intuitively to enable a fast transition from above the obstacle to the terminal point of the original problem (5).

The second subproblem continues from where the first subproblem terminated. The initial conditions in the second subproblem are the same as the terminal constraints of the first subproblem, (6). The terminal constraints of this second subproblem are the same as in the original problem, (5). This means that the pendulum continues on the other side of the obstacle until it reaches the terminal point.

The resulting optimal trajectories of the two subproblems are then merged and given as an initial guess when solving the original problem. Given this initial guess, the solver converges to the optimal solution. The resulting pendulum end point movements for the two parts of the initial guess and for the optimal solution are found in Figure 8. The control signal for the optimal solution is found in Figure 9. The final time for the merged initial guess is 3.39 s while the optimal solution has a final time of 3.34 s. The first part of the initial guess takes 2.18 s while the second part is performed in 1.21 s. The corresponding first and second parts of the optimal solution take 1.94 s and 1.40 s respectively. This means that the intuition behind the choice of terminal constraints for the first subproblem

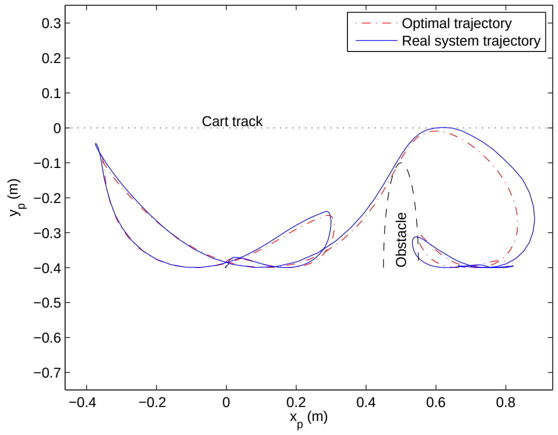


Figure 11: Experimental results for path constrained problem when control trajectory applied in open loop as described in Section 5.1. The optimal pendulum end point trajectory is also plotted for comparison reasons.

(6), i.e., to enable for a fast second part, is good. The second part of the initial guess is fast and the merged initial guess is not very far from the optimal one in terms of the optimization objective, namely the final time, t_f.

5 Experiments on the real Pendulum

In this section the optimal control trajectories obtained in the previous section are applied to the real system. These experiments will serve as an evaluation of how well the model describes the actual system and it will show the practical applicability of optimal control trajectories in a real system.

5.1 Open loop results

Figures 10 and 11 show how the real system responds to the optimal control trajectories. The figures also show the optimal trajectories from the previous section for comparison reasons. The trajectories are very similar, which means that the model of the system is accurate.

In the optimizations it is assumed that the initial conditions of the pendulum and cart are such that the cart is at rest at position, $x = 0$, and the pendulum is at rest at angle $\theta = 0$. If the experiments are performed with initial conditions of the pendulum that do not satisfy the assumed ones, i.e., if the pendulum is swinging when the experiment is started, we get results as shown in Figures 12 and 13. The magnitude of the initial swings are approximately 45° in these experiments. The figures show that we are far from reaching our objectives when this kind of disturbances are present. To make the optimization results usable in reality, we need feedback to take care of deviations from the optimal trajectories.

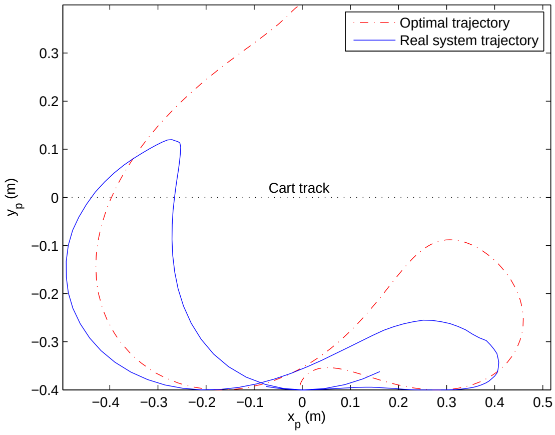


Figure 12: Pendulum end point trajectory for the real system when pendulum is swinging initially and no feedback is used as described in 5.1.

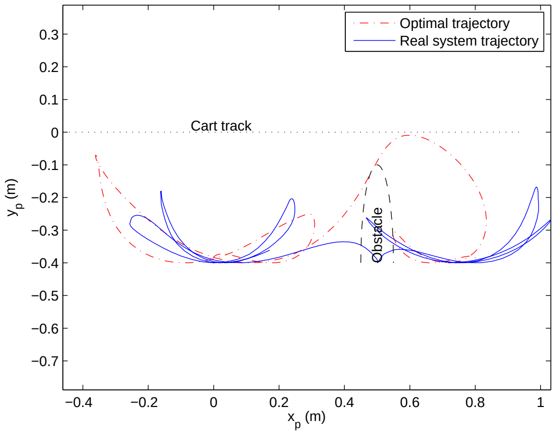


Figure 13: Pendulum end point trajectory for the real system when pendulum is swinging initially and no feedback is used as described in 5.1.

5.2 MPC-Feedback

Model Predictive Control feedback (MPC) is introduced to take care of disturbances to the system. A schematic view of how the feedback is introduced is found in Figure 14 where z and S are defined as in Figure 5. In MPC, a finite time-horizon optimization problem with state and control constraints is solved in every sample. In our setup, deviations from the opti-

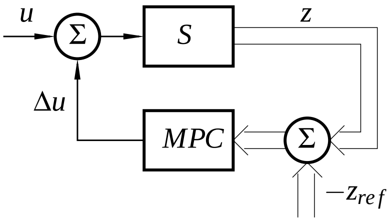


Figure 14: A schematic view of how the MPC-feedback is introduced.

mal state and control trajectories are minimized, such that control magnitude and cart position constraints are not violated.

The mathematical and implementational aspects of the MPC-feedback is beyond the scope of this paper and will be described in a future paper. The results when applying the MPC-feedback to the real system are, however, relevant to show that the optimal feed forward trajectories must be accompanied with feedback to be useful in reality. Experimental results of optimal trajectory feed-forward in combination with MPC-feedback are visualized in Figures 15 and 16. The experiments are performed with initial pendulum swings. Also here the initial swings have a magnitude of around 45° to be comparable to the results in the previous section. Due to the initial swinging, the trajectories are far from the optimal ones in the beginning but the feedback brings the system closer with time. If the feedback control authority is large enough, the original objectives of the optimizations can be achieved despite errors in the initial conditions. The figures show that we have enough control authority in these experiments since we manage to swing-up the pendulum in the first experiment and avoid the obstacle in the second experiment, as desired.

6 Teaching

Optimal control of the cart-pendulum system was introduced as a new laboratory exercise in the course on Nonlinear Control and Servo systems (FRTN05) at the Dept. of Automatic Control, Lund, in 2009, see [9].

The cart system has previously been developed as a general module for different control experiments and has been used as a test bed in both student and research projects as well as in other courses [7].

The preliminary evaluation of the new computer and laboratory exercises has been very positive from both the students as well as from the lecturer and the teaching assistants. Optimal control has already before played an important role in the course curriculum, but was mainly focused on the theoretical aspects and lacked from the gap between constrained-low-order-pen-and-paper-problems and more realistic examples and applications. Here Optimica has played an important role to bridge that gap and to complement the previous course contents.

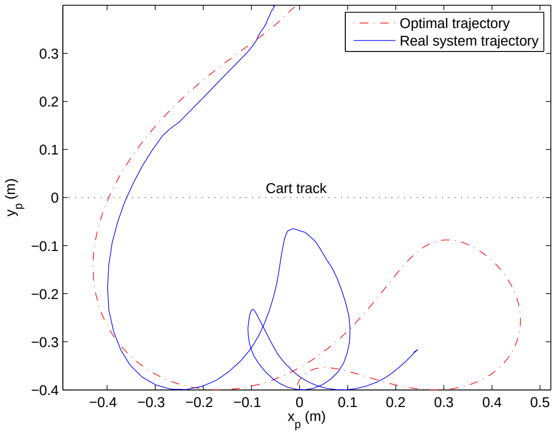


Figure 15: Pendulum end point trajectory for the real system when pendulum is swinging initially and feedback is used as described in 5.2.

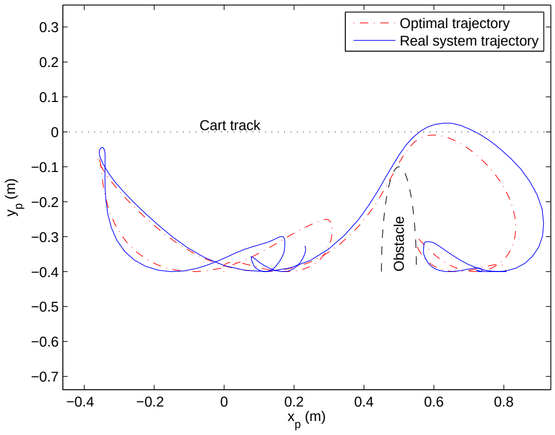


Figure 16: Pendulum end point trajectory for the real system when pendulum is swinging initially and feedback is used as described in 5.2.

The new software gives the the students the possibility to concentrate on the formulation of the optimal control problem separately from the system modeling and to experimentally evaluate how solutions change with respect to the cost function and to the constraints.

Obtaining a numerical solution naturally raises the question of accuracy, but also to related questions on sensitivity to initial conditions and to discrepancies of the model and the real plant. In the lab exercises this is evaluated where pure feedforward solutions are compared to the combination of a feedforward reference from the optimal control problem together with feedback around this trajectory, similar to what has been outlined in Sections 4 and 5.

7 Conclusions/Future Work

The second optimization problem with path constrained pendulum end point movements shows that optimal control problems can be difficult to solve. Although the Optimica tool is very powerful, one needs to understand the problem and sometimes supply an initial guess to help the solver converging to the correct solution.

The results of the Optimica optimizations are open loop control trajectories. When applying these to a real system, everything must be accurately modeled and only very small disturbances may be present to get good results. This is however rarely the case, which is why we need feedback that controls the actual state trajectories towards the optimal ones. This combination of optimal feedforward and feedback has shown to be very efficient when optimal transitions through nonlinear dynamics are desired.

In this paper, optimal trajectories are pre-calculated using Optimica and MPC-feedback is used to stay close to the optimal trajectories. An extension to this work would be to instead of pre-calculating the optimal trajectories, rather let an MPC-controller run with Optimica in real time. Then the optimization problems stated in (4) and (5) would be solved in each sample with different initial conditions. The initial conditions would be the measured state variables at the current sample. The main difficulty would be to ensure fast enough computations for this to be implementable in a real time application.

References

[1] Johan Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *In 6th International Modelica Conference 2008*. Modelica Association, March 2008.

[2] Johan Åkesson, Tove Bergdahl, Magnus Gäfvert, and Hubertus Tummescheit. Modeling and optimization with modelica and optimica using the jmodelica.org open source platform. In *Proceedings of the 7th International Modelica Conference 2009*. Modelica Association, September 2009.

[3] Johan Åkesson, Magnus Gäfvert, and Hubertus Tummescheit. Jmodelica—an open source platform for optimization of modelica models. In *Proceedings of MATHMOD 2009 - 6th Vienna International Conference on Mathematical Modelling*, Vienna, Austria, February 2009. TU Wien.

[4] Karl Johan Åström and Katsuhisa Furuta. Swinging up a pendulum by energy control. *Automatica*, 36:278–285, February 2000.

[5] L.T. Biegler, A.M. Cervantes, and A Wächter. Advances in simultaneous strategies for dynamic optimization. *Chemical Engineering Science*, 57:575–593, 2002.

[6] Python Software Foundation. Python Programming Language – Official Website, 2009. http://www.python.org/.

[7] Per-Ola Larsson and Rolf Braun. Construction and control of an educational lab process - the gantry crane. In *Reglermöte 2008, Luleå*, June 2008.

[8] Modelon AB. JModelica Home Page, 2009. http://www.jmodelica.org.

[9] Pontus Giselsson. Laboratory Exercise in Nonlinear Control and Servo Systems, 2009. http://www.control.lth.se/course/FRTN05/labs/lab3/lab3.html.

[10] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–58, 2006.

[11] Magnus Wiklund, Anders Kristenson, and Karl Johan Åström. A new strategy for swinging up an inverted pendulum. In *Preprints IFAC 12th World Congress*, volume 9, pages 151–154, Sydney, Australia, July 1993.

Optimized Control of Hot-Gas Cycle for Solar Thermal Power Plants

Jan Gall Dirk Abel
IRT – Institut für Regelungstechnik, RWTH Aachen, Steinbachstr. 54, D-52074 Aachen, Germany
j.gall@irt.rwth-aachen.de
Nils Ahlbrink Robert Pitz-Paal
DLR – Deutsches Zentrum für Luft- und Raumfahrt, Linder Höhe, D-51147 Köln, Germany
Joel Andersson Moritz Diehl
Electrical Engineering Department (ESAT-SCD), KU Leuven, B-3000 Leuven, Belgium
Cristiano Teixeira Boura Mark Schmitz Bernhard Hoffschmidt
SIJ – Solar-Institut Jülich, FH Aachen, Heinrich-Mussmann-Str.5, 52428 Jülich, Germany

Abstract

In this paper, the overall modeling approach for an optimized control of a hot-gas cycle with its different components for solar thermal power plants is pointed out.
For control purposes a linear model-based controller (MPC) was implemented in Modelica based on an external state-of-the-art QP solver linked to the Modelica model.

Keywords: solar energy, control, optimization

1 Introduction

1.1 Background

One possible answer to address climate change is using solar instead of fossil energy. Among other technologies central receiver systems (CRS) using air as heat transfer medium are being investigated. A demonstration plant (STJ) has just been completed.
The STJ uses 18000 m² of sun-tracking mirrors (heliostats) to heat up air to 700 °C which in turn generates superheated steam, driving turbine and generator. A storage system can take up the thermal energy for one full-load hour. By adjusting the rate of the volume flow of two blowers, it is possible to charge or discharge the storage during operation. The Virtual Institute of Central Receiver Power Plants (vICERP) has been founded to solve the demanding requirements for the optimal plant control under the strongly fluctuating energy input.

1.2 Scope of Paper

In this paper, the overall modeling approach for an optimized control of a hot-gas cycle for solar thermal power plants is pointed out. A detailed description of the modeling of the receiver and the heliostat field can be found in an affiliated conference paper by Ahlbrink et al. [1]. The emphasis of the modeling work lays on the development of dynamic component models to be used in control systems. Depending on the control task, the discretization has to be adapted. Main components of the hot-gas cycle are the solar thermal receiver and the storage system. The steam cycle is preliminarily only included as heat sink.

2 Modeling

The modeling efforts are shared among the vICERP partner institutions. Therefore, it is crucial to use a common model setup to ensure a proper use of the models. A common test platform provides the necessary interfaces, so that new, improved modules can easily be integrated and tested. The models are based on the open source library Modelica_Fluid [2]. The vICERP library uses a finite volume approach with staggered grid method implemented with flow and volumes elements [3]. The mass and energy balances are considered in the volume element. A formulation of the balance equations from Hirsch [4] is implemented using pressure and specific enthalpy as state variables. The momentum equation is reduced to a pressure drop equation and formulated in a flow element. Models like the receiver, storage system, steam generator are setup in a way that the models end with a flow element. Thus, to the outside they

DOI: 10.3384/ecp09430053

behave like a pressure drop element. Volume models are needed to interconnect the components.

Figure 1 shows the top-level of the model developed in Dymola/Modelica. Several different components can be identified in the figure: the heliostat field and receiver on the top left, the storage in the middle, a simple model of the water steam cycle on the right and the two blowers on the bottom right. The following sections give a brief introduction to the models.

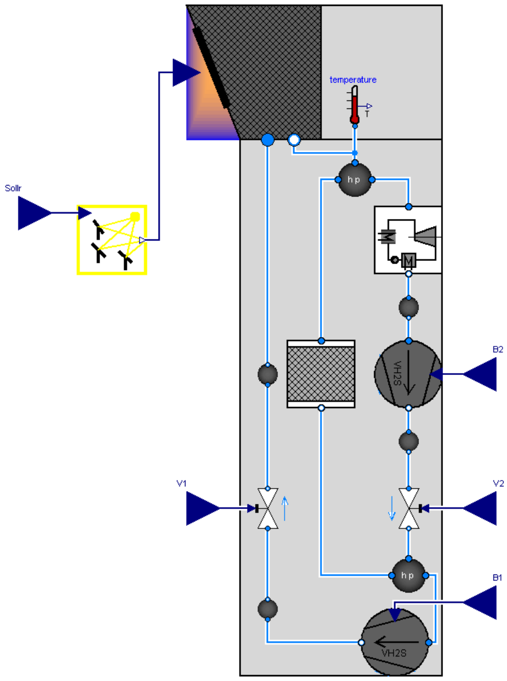


Figure 1. Screenshot of the model in Dymola

2.1 Heliostat field and Receiver

The 2200 heliostats that focus the sunlight onto the receiver are calculated by a special Monte-Carlo raytracing code, called STRAL [1], which generates a flux map on a surface which coincides with the receiver. The receiver is modeled in Modelica. The output is an averaged temperature for the air mass flow entering the hot-gas system.

2.2 Hot-Air Pipes and Blowers

The models for hot-air pipes are simplified using one volume and one flow element for each pipe. The blower models include the characteristic curve of the blower provided by the manufacturer. Implemented in a lookup table, this map allows the calculation of a resulting mass flow given the power input and pressure difference between inlet and outlet port of the blower.

2.3 Storage

A thermal storage system is used as a buffer that stores energy at times of high irradiances and enables operation of the plant after sunset or during periods of reduced solar input. The developed storage model enables the analysis of different operation conditions of the power plant. The storage behavior is similar to that of a regenerator. The hot air flows through the storage material and heats it up. During discharge, the air flows in reverse direction and cools down the storage material, while being heated up.

The storage model is divided into storage cells. Each cell element describes the characteristic material and flow phenomena, which are included in differential equations. Thus, each cell element computes two temperatures which represent the local temperature of the storage material and the local temperature of the fluid.

The model enables the description of charging, discharging and stand-by operation. Additionally heat losses during stand-by periods are calculated. Thus, temperature profiles inside the storage can be computed for any time in the simulation process.

Figure 2 shows the temperature profile for the 100%- and 0%-storage capacity load situation.

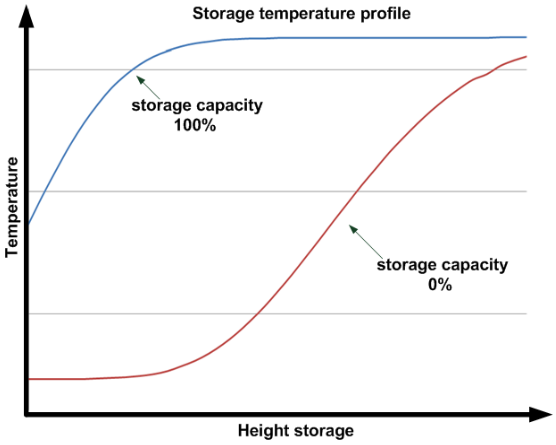


Figure 2. Temperature profile of the storage system for 100%- and 0%- storage capacity

2.4 Heat Sink

Whereas in the final system the steam cycle will be modeled in detail, it is – at this stage – merely integrated as a heat sink, featuring qualitatively the steam cycle's anticipated behavior.

3 Control

3.1 Basic automation scheme

The simulation of the operational behavior of the complete plant requires an integrated control scheme within the model to ensure compliance with given limits of absolute and gradient values. As a first concept, a basic automation scheme has been developed based on a wiring of SISO control loops with PID controllers. This scheme should on one hand assure a safe operation of the plant under normal operation conditions and on the other hand be a measure of performance for more sophisticated control schemes. The tuning of the different controllers has been done in MATLAB using a response optimization technique. An extract of the scheme is depicted in Figure 3.

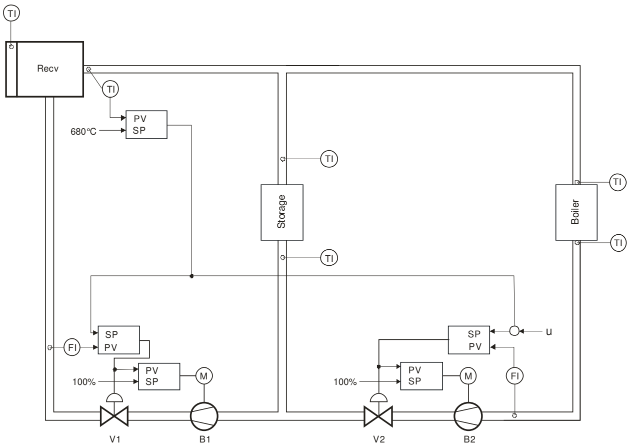


Figure 3. Basic automation scheme

The measurement signals for the control scheme are different volume flows and temperature information. Actuating variables are the speed of the two blowers and different valves located in the air cycle. The main goal of the control scheme is to maintain the outlet air temperature of the receiver constant at 680 °C. This is achieved by controlling the air volume flow through the receiver. As a consequence of an increasing volume flow through the receiver, the temperature of the outgoing air decreases. The temperature difference from the design point is used in an outer control loop of a cascaded structure, which feeds the required volume flow as setpoint to the inner control loop. The inner loop accesses two actuators for adjusting the volume flow, a blower and a valve mounted directly after the blower. The blower is obviously necessary to generate the air flow. The use of the valve is justified for two reasons. First, the blower itself has a low pressure drop during standstill periods such that an airstream just flows through it if the stream is generated by the other blower installed in series. Second, the blower is limited to a minimal rotational speed. Therefore, the valve is closed appropriately to set volume flows below the threshold given by the blower itself.

3.2 Model predictive control

The vICERP project includes the application of a *model predictive controller (MPC).* This makes use of the dynamic model of the plant that has been developed for the simulation to predict future behavior of the plant with regard to changes in actuating variables.

With an MPC approach, it is also possible to include a natural objective function (maximize produced energy, minimize risk of boiler shutdown during transients, minimize time to start-up etc.) as well as imposing first-principle constraints such as bounds on variables or periodicity constraints.

The scope of this paper is limited to a *linear* model predictive controller, which aims at demonstrating the concept as well as the basic software coupling.

It is the goal of the project to make extensive use of the full *nonlinear* model of the plant to find an optimal controller that works in nominal operation as well as during start-up, shut-down and during sudden changes in the weather conditions (if these changes can be predicted, this prediction should be taken into account). This will be described in more detail in the outlook of this paper.

3.3 Linearized model

The MPC controller is based on a linear state space model of the form:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned} \tag{1}$$

This model was obtained from the non-linear Modelica model by using the *linearizeModel* command.

3.4 MPC implementation

Based on the above representation the controller is able to predict the future behavior of the plant regarding to a future trajectory for the input (and possible disturbances). This can be expressed in an equation of the form

$$\begin{aligned} &Y(k) = \\ &\Psi \cdot x(k) + \Upsilon \cdot u(k-1) + \Theta \cdot \Delta U(k) + \Xi \cdot D_m(k) \end{aligned} \tag{2}$$

for suitable matrices Ψ, Υ, Θ and Ξ [9]. The different terms represent the free and forced response of the plant, together with the response to future trajectories of the inputs and disturbances. Combined with a given reference trajectory for the outputs and additional linear constraints on the states and inputs, this can be reformulated as an optimization problem of the form

$$\min_{\Delta U(k)} \Delta U(k)^T \cdot H \cdot \Delta U(k) - G^T \cdot \Delta U(k) \qquad (3)$$

with Hessian matrix H and gradient vector G. This is a standard optimization problem known as the *Quadratic Programming (QP)* problem.

3.5 QP Solver

The MPC controller requires the above quadratic program to be solved at each sampling time. This is carried out with the QP solver qpOASES [5], which uses an online active-set method particularly suited for MPC problems [6].

To make qpOASES, which is written in C++, fully compatible with Modelica, a C interface has been written. By using Modelica's *external objects*, the QP solver is able to retain memory between calls. This fact can be expected to grow importance once the MPC controller is extended to nonlinear models.

The Modelica interface to qpOASES is available upon request from the authors of the paper (LGPL license).

4 Simulation Results

For evaluation of the model and different control schemes the simulation results according to the following scenario are presented in Figure 4.

The plant is operating in its stationary design point (i. e. outlet temperature at the receiver is at 680 °C) with a constant solar irradiation. At time *t = 100 s*, a sinusoidal disturbance with a period of 600 s and an amplitude of 50% of the previous irradiation acts on the input. After 1200 seconds the input remains constant again.

The upper part of the figure shows this disturbance of the solar irradiation. In the lower part the responses of the air outlet temperature at the receiver with different controllers are depicted. The main goal of this feedback control is disturbance rejection, i. e. it should assure a constant outlet temperature by adjusting the air volume flow through the receiver appropriately.

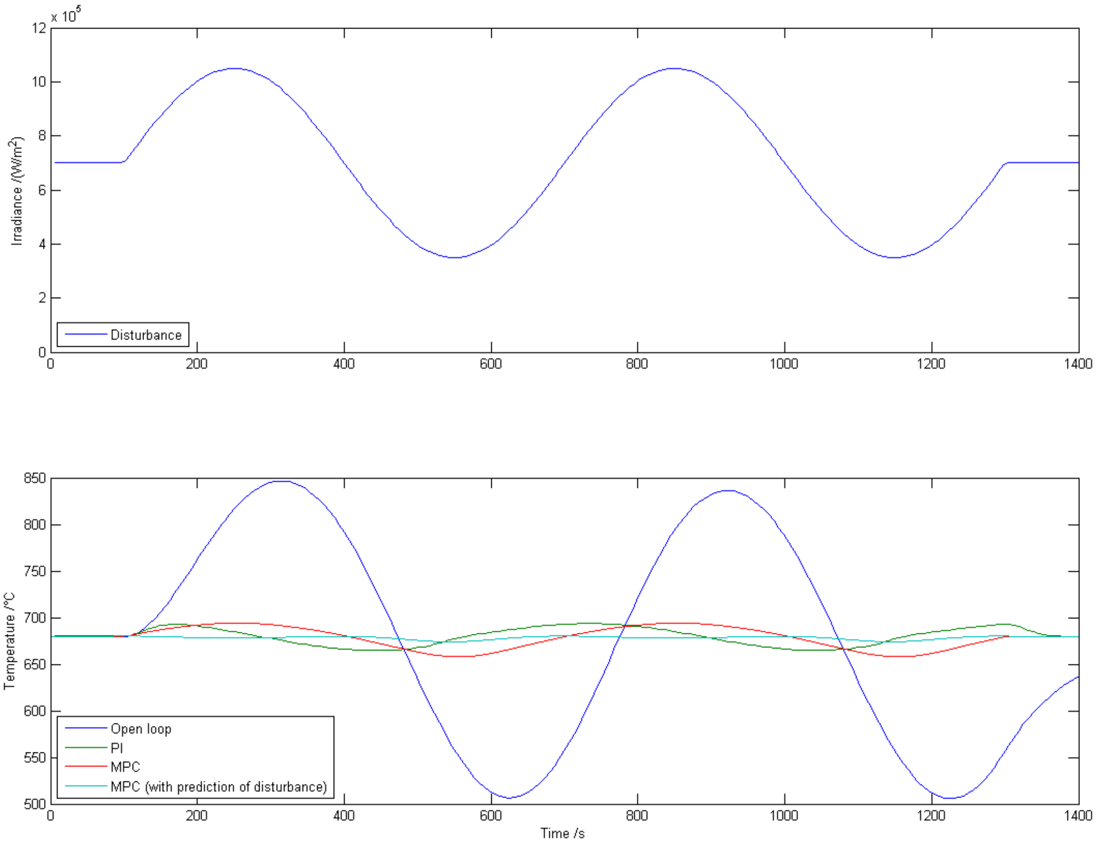


Figure 4. Simulation results

The first case is just an open loop simulation of the system, i. e. no control actions are performed at all and all manipulated variables, especially the setpoints of the blowers, remain constant.

The second curve shows the resulting characteristics if the control scheme as depicted in Figure 3 based on PI controllers is used. In this case the maximal deviation of the outlet temperature is about 15 °C.

In the following two cases the implemented MPC-block was used to control the air temperature. Therefore the PI controller in Figure 3 which uses the air temperature as measurement variable in the outer control loop was simply replaced by the MPC-block. The inner controllers directly manipulating the actors remain the same. Although the model-based controller has inherently the ability to cope with systems with multiple in- and outputs, it this case it is just used with a single in- and output. The controller has a sampling interval of 0.5 s and uses an internal model with 25 states at a discretization interval of 2 seconds. It has a prediction horizon of $n_p = 150$ and a control horizon of $n_u = 30$ (i. e. it predicts the response of the plant 300 seconds in the future). As one can see in Figure 4, the controller shows comparable results to the PI controller.

In the fourth case, the MPC-block was extended to also incorporate the influence of the disturbance on the system by feedforward control. If the supplied solar energy can not only be measured, but also predicted (e. g. by weather forecast or vision-based [8]), it is possible that the MPC also uses this information for prediction. In this case the controller achieves the best performance with only minimal deviation from the setpoint at 680 °C.

5 Conclusion

In this paper we have presented a first-principle model for a central receiver solar power plant with open volumetric receiver. The model includes the different components of the plant, e. g. receiver, storage, and is used for simulation and optimization purposes of both the separate components and also the plant behavior as a whole.

For control purposes a generic linear model-based controller (MPC) was implemented and achieves reasonable results. The implementation is based on an external state-of-the-art QP solver linked to the Modelica model for the calculation of optimal control actions.

Future work aims at not only using optimal control for the air cycle as presented in this paper, but also to extend this approach to other areas of the plant, e. g. storage regulation.

6 Outlook

6.1 Non-linear MPC

A non-linear MPC controller is obtained if the linear state space model (1) is a replaced with a continuous state space model of the form:

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t), p, t) \\ y(t) &= g(x(t), u(t), p, t) \end{aligned} \qquad (4)$$

The dynamics are now described by a non-linear ordinary differential equation (ODE) and the discrete time *k* has been replaced by the continuous time *t*. Included is also the dependence of a set of parameters *p*.

A model of the form (2) is always available since simulating a translated Modelica model is always equivalent to integrating a (possibly hybrid) *differential-algebraic equation*, due to construction of the Modelica language [10].

By adding a quadratic objective function and introducing the prediction horizon T_p and control horizon T_c, we obtain an *optimal control problem in differential-algebraic-equations* of the form:

$$\begin{aligned} \underset{x(\cdot),u(\cdot),y(\cdot),p}{\text{minimize}} \quad & \int_0^{T_p} \left\| y - y_\text{ref} \right\|_Q^2 dt + \int_0^{T_c} \left\| u - u_\text{ref} \right\|_R^2 dt \\ \text{subject to} \quad & \dot{x}(t) = f(x(t), u(t), p, t) \\ & y(t) = g(x(t), u(t), p, t) \\ & y_\text{lb} \le y \le y_\text{ub} \quad \text{(output bounds)} \\ & u_\text{lb} \le u \le u_\text{ub} \quad \text{(control bounds)} \\ & p_\text{lb} \le p \le p_\text{ub} \quad \text{(parameter bounds)} \\ & x(0) = x_0 \quad \text{(initial value)} \end{aligned} \qquad (5)$$

where P is a positive definite and Q is a positive semi-definite matrix. Bounds are denoted by the subscript “ub” and “lb” for upper and lower bounds respectively.

Problem (5) is an infinite dimensional optimization problem that can be efficiently solved by parameterization of both the control *u* and the state *x* using a *simultaneous* method such as *direct multiple shooting* and *collocation*.

State-of-the-art numerical methods for solving such and similar dynamic optimization problem have been implemented in the software package ACADOtoolkit [12], developed by OPTEC. The software is available open-source under the LGPL license, allowing it to be linked also with commercial code, and work is underway to fully integrate it with Modelica.

The integration consists of two parts. Firstly, it should be possible to call ACADOtoolkit from Modelica. This is made possible by implementing a plain C interface which can be called from Modelica code using external objects just like the qpOASES interface. ACADOtoolkit is written exclusively in C/C++ and avoids linking with external software, so it is very suitable to use together with Modelica tools as well as on embedded systems. Real-time optimal control is one of the aims of the project.

The second, much larger part of the integration consists of extending the software so that it can use models formulated in Modelica. Small dynamical models can easily be coded directly in C++, but for complex models, a better solution is to import the model equations into ACADOtoolkit. This will undoubtedly require the extension of the software to deal with e.g. *hybrid systems* and *integer valued controls*.

There are efforts to standardize the interaction between equation-based, object-oriented modeling languages such as gProms and Modelica on one hand, and computer algebra tools and other mathematical software on the other [7]. The two open-source Modelica projects OpenModelica and JModelica [11] both offer the possibility to export the flattened simulation problem (i. e. variables, initial values, model equations, etc.) in the ModelicaXML format which in turn uses MathML to describe the model equations.

To make also ACADOtoolkit conformant with this standard, so that models defined in ModelicaXML code can used by the software, an XML module is being developed. This module parses the ModelicaXML code and translates the model equations into ACADOtoolkit's internal (symbolic) representation.

A further standardization is to use the Optimica, a language extension for the formulation of optimal control problems, to formulate the optimal control problems. This provides an abstraction which is useful to help engineers and scientists formulate optimal control problems in a structured way [11].

Acknowledgements

The authors would like to thank for financial support granted by the Initiative and Networking Fund of the Helmholtz Association, the state of North Rhine-Westfalia, and the European Union/European regional development fund.

References

[1] Ahlbrink N., Belhomme B., Pitz-Paal R.: *Modeling and Simulation of a Solar Thermal Power Plant with Open Volumetric Air Receiver.* Conference Proceedings, Modelica Conference 2009, Como

[2] Casella F., Otter M., Proelss K., Richter C., Tummescheid H.: *The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks.* Conference Proceedings, Modelica Conference 2006, Vienna, September 4-5 2006

[3] Tummescheid H.: *Design and Implementation of Object-Oriented Model Libraries using Modelica.* Thesis, Department of Automatic Control, Lund Institute of Technology, Lund, August 2002

[4] Hirsch T.: *Dynamische Systemsimulation und Auslegung des Abscheidesystems für die solare Direktverdampfung in Parabolrinnenkollektoren.* Fortschrittsberichte VDI, Reihe 6, Nr. 535, VDI Verlag, Düsseldorf, 2005

[5] www.qpoases.org

[6] Ferreau H. J., Bock H. G., Diehl M.: *An online active set strategy to overcome the limitations of explicit MPC.* International Journal of Robust and Nonlinear Control 18:816-830 (2008)

[7] Casella F., Donida F., Lovera M.: *Beyond Simulation: Computer Aided Control System Design Using Equation-Based Object Oriented Modelling for the Next Decade.*
EOOLT 2008, Paphos

[8] López-Martínez M., Vargas M., Rubio F. R.: *Vision-Based System for the Safe Operation of a Solar Power Tower Plant.* Proceedings of the 8th Ibero-American Conference on AI: Advances in Artificial Intelligence, Vol. 2527, pp. 943 – 952, 2002

[9] Maciejowski J. M.: *Predictive Control with Constraints.* Prentice Hall, 2002

[10] Fritzson P.: *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley-IEEE Press, 2003

[11] Åkesson, J.: *Optimica – An Extension of Modelica Supporting Dynamic Optimization*, Conference Proceedings, Modelica 2008

[12] www.acadotoolkit.org

Feedback Loop Optimization for a Distillation System by applying C-Code Controllers with Dymola

Hansjörg Kapeller   Dragan Simic
AIT Austrian Institute of Technology, Mobility Department, Electric Drive Technologies
Giefinggasse 2, 1210 Vienna, Austria
hansjoerg.kapeller@ait.ac.at   dragan.simic@ait.ac.at

Abstract

The process engineering domain covers a large field of different disciplines such as thermodynamics, electrical engineering or chemistry. On one hand just the knowledge about these different disciplines to approach and develop standalone-solutions is a big challenge where on the other hand the configuration and the control of the system routines are crucial steps. Nowadays most applications in process engineering are automated and digital signal processors (DSP) are widely used to implement control modules for different automatic control systems in an efficient and flexible way. Nevertheless, without dynamically applicable and real-time capable models it seems nearly impossible to estimate real operating conditions (e.g. controller parameter settings) in process engineering according to real requirements.

This paper presents the simulation model of a thermal control circuit for determining the distillation properties of petrochemical end products modeled in *Modelica* and performed by using the simulation tool *Dymola*. This simulation environment improves the design of interdisciplinary models and allows the optimization of the entire feedback loop.

Keywords: process engineering, automatic control systems, simulation, interdisciplinary models, optimization

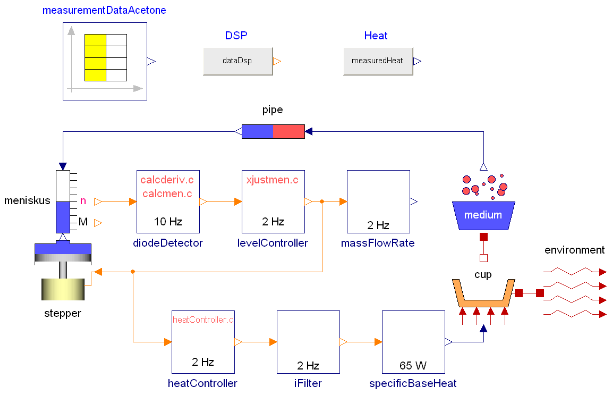


Figure 1: Scheme of the investigated measurement device for determining the distillation properties of petrochemical end products

1 Introduction

The measurement device for determining the distillation properties of petrochemical end products is depicted schematically in Figure 1. It is a device in which different processes of thermodynamics, chemistry, mechanics, electrical measuring and control technology must be controlled. The mode of operation is based on the vaporization of the investigated medium, e.g. acetone, which condenses in a collecting vessel again. The heating energy is controlled by the heat controller and the condense level of the vessel is controlled by a stepper motor, respectively. In case of equilibrium and on condition that all controllers are working in a stable operating point the medium vaporizes and condensates in the vessel by keeping a constant level until all - in case of a pure substance - is exhausted or - in case of a mixture - the next component reaches the inherent evaporating temperature.

The challenge in this process engineering application is to parametrize the level control and the heating control [1]. Both controllers are not independent: if the stepper motor controller does not work adequatly, e.g. the motor rotates too fast, the level in the vessel sinks too quickly and the operating point becomes unstable. The incapacity resp. inertia of the heat controller

DOI: 10.3384/ecp09430054

leads to an insufficient vaporization of the medium and therefore to an insufficient condensation rate with the consequence, that the level decrease can not be compensated. A flexible, timely and systematic way is to model the entire measurement device using the object-oriented modeling language *Modelica* [2].

2 Modeling and Simulation in Dymola

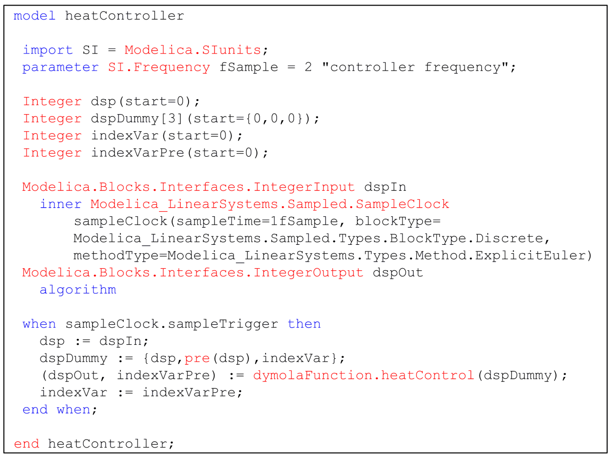
For modeling and simulating integrated and complex systems the software platform *Dymola* is used. *Dymola* is an environment using the objects described in *Modelica* syntax, which allows the software designer to create models of any kind of objects that can be described by algebraic and ordinary differential equations. With *Dymola* simulations of the behavior and interaction betweens systems of different engineering fields, such as mechanical, electric, thermodynamic, hydraulic, pneumatic, thermal and control systems are possible. The modeling language itself is open which means that users are free to create their own model libraries or modify standard libraries to better match the users individual modeling and simulation needs.

2.1 Simulation Models

To ensure, that the simulation reflects the measurement device as best as possible, all components - except the controllers - are modeled in *Modelica* code, whereas the C-code algorithms of the DSP controllers and all variable names have been retained unchanged and are coupled directly as external C-functions to the rest of the closed loop control system. Following this strategy, the model allows to tune the controllers in the same way as it occurs in the real measurement device by adapting the C-routines of the DSP controllers and the obtained simulation results based on the same C-routines.

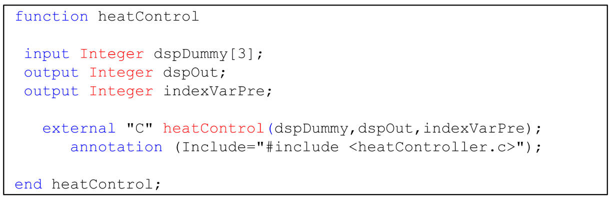
To emulate the DSP properties in *Dymola*, the external C-routines (e.g. the heat controller) are invoked using a sample clock. The clock frequency is given by $T_{clock} = \frac{1}{f_{sample}}$ and corresponds to the processing time when the heat controller routine will be executed on the DSP. Thus every clock-event the *Dymola*-function `heatControl.mo` will be executed (cp. Figure 2).

In Figure 3 the source code of this *Dymola*-function in which the C-routine itself will be invoked and processed is presented (cp. the sequence beginning with `external "C"`).



```
model heatController

 import SI = Modelica.SIunits;
 parameter SI.Frequency fSample = 2 "controller frequency";

 Integer dsp(start=0);
 Integer dspDummy[3](start={0,0,0});
 Integer indexVar(start=0);
 Integer indexVarPre(start=0);

 Modelica.Blocks.Interfaces.IntegerInput dspIn
   inner Modelica_LinearSystems.Sampled.SampleClock
       sampleClock(sampleTime=1fSample, blockType=
       Modelica_LinearSystems.Sampled.Types.BlockType.Discrete,
       methodType=Modelica_LinearSystems.Types.Method.ExplicitEuler)
 Modelica.Blocks.Interfaces.IntegerOutput dspOut
   algorithm

 when sampleClock.sampleTrigger then
   dsp := dspIn;
   dspDummy := {dsp,pre(dsp),indexVar};
   (dspOut, indexVarPre) := dymolaFunction.heatControl(dspDummy);
   indexVar := indexVarPre;
 end when;

end heatController;
```

Figure 2: *Dymola* model of the heat controller. When the clock-event occurs, the *Dymola* function `heatControl.mo` will be executed



```
function heatControl

 input Integer dspDummy[3];
 output Integer dspOut;
 output Integer indexVarPre;

   external "C" heatControl(dspDummy,dspOut,indexVarPre);
      annotation (Include="#include <heatController.c>");

end heatControl;
```

Figure 3: *Modelica* code of the *Dymola* function `heatControl.mo`. Represents the interface to the external C-routine `heatController.c`

2.1.1 Level Meter – Diode Array

Figure 4 depicts the model of the vessel equipped with two input connectors. One connection provides the input for the vaporized distillate (cp. *Distillate_in*), the second one represents the control the input-variable for the level control (cp. *Stepper_connect*). In real life the level meter is realized by a diode array consisting of 16 diodes, where effectively 10 diodes are evaluated. The remaining 6 diodes are used, to delimit the upper and lower recoverable level positions. In the simulation environment a corresponding discrete signal is generated, which feeds the *diodeDetector*-model. The *diodeDetector*-model invokes the C-function routines `calcderiv.c` and `calcmen.c` and maps the discrete level signals into a proper range of values (nominal value amounts 5) which are available after the inherent DSP process time of 0.1 s as output, again.

Figure 5 presents two generated outputs taken from a simulation run during evaluation procedures. The red curve shape (10 diodes evaluated) can be compared with a gray curve, which represents the obtained output of discrete position values, when all 16 diodes

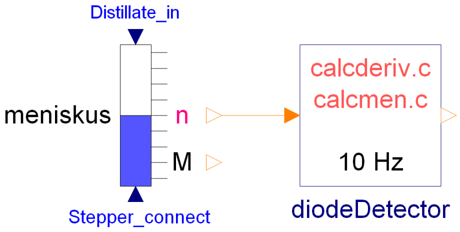


Figure 4: *Dymola* model of the vessel including the C-function routines for the level output generation

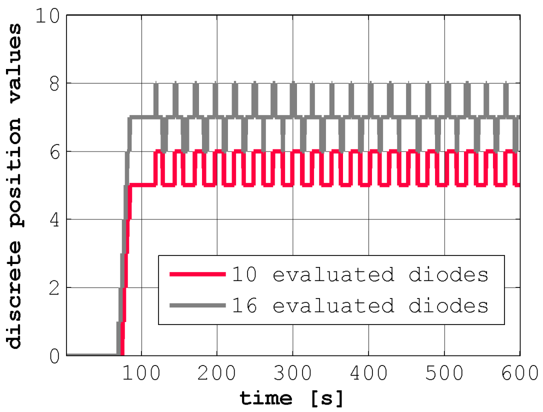


Figure 5: Obtained output of the level meter in *Dymola*. Comparison of two curve shapes: red curve 10 evaluated diodes, gray curve 16 evaluated diodes

were evaluated. A diode array with higher resolution gives rise to a higher frequent, albeit smaller ripple, which leads to positive effects on one hand for the stepper motor - less influence due to the mass inertia - on the other hand for the heat controller, too.

2.1.2 Level Control

Figure 6 depicts the previous model extended with a PI-controller which provides the control variable for the level control thus the control loop for adjusting the frequency of stepper motor can be closed. The stepper motor controller is implemented as external C-routine and corresponds identically to the implemented PI controller on the digital signal processor. The DSP process time for this C-code amounts $0.5\,s$, the process time for *diodeDetector*-model amounts $0.1\,s$ and is implemented corresponding to the real measurement device. With other words: the provided input sample rate for the level controller is higher $(10\,Hz)$ than the generated output of the level controller $(2\,Hz)$ which acts as input for the stepper motor, again.

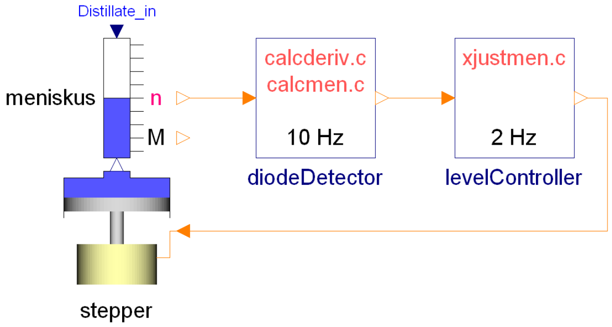


Figure 6: Extended vessel model with additional PI-controller which provides the control variable for the level control

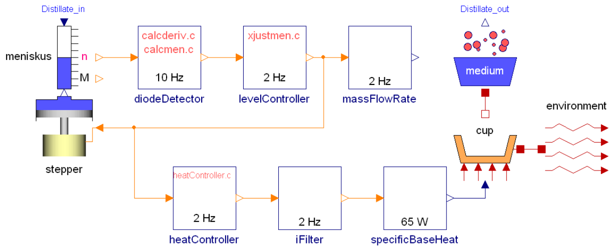


Figure 7: Implementation of the heat controller closed loop in *Dymola* which corresponds to the real measurement device

2.1.3 Heat Controller

The control signal for the stepper motor is also used as input for the heat controller (`heatController.c`) which is implemented as PID-controller. The output of the heat controller is filtered (*iFilter*) in order to smooth the control variable if unexpected discontinuities occurs. This signal will be charged with an acetone-specific base heat and is connected to the cup-heater. Thus the controlled heating energy causes the evaporation of the medium (acetone) and a desired flow rate will be achieved (*Distillate_out*). All components regarding the heat controller are depicted schematically in Figure 7.

2.2 Simulation of the Entire Measurement Device Model

If the two interfaces *Distillate_out* and *Distillate_in* are connected using a pipe model (*pipe*), then the feedback loop for the entire model can be closed (anticipated in Figure 1 already) and simulated in order to ensure the validity comparing measurement data with simulation data. Only after a successful validation, modifications in parameterization or other appropriate measures can be taken into account. The measurement data are imported in *Dymola* using the block

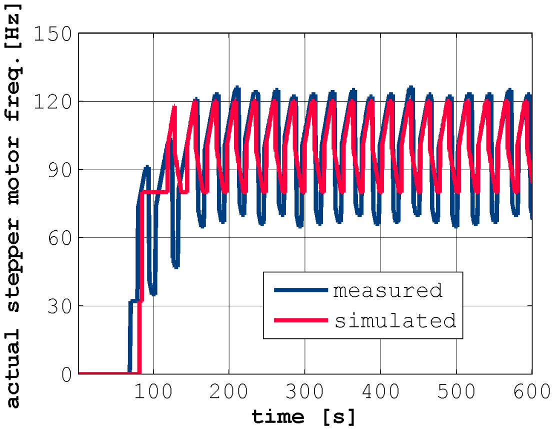


Figure 8: Comparison of measured and simulated actual stepper motor frequency

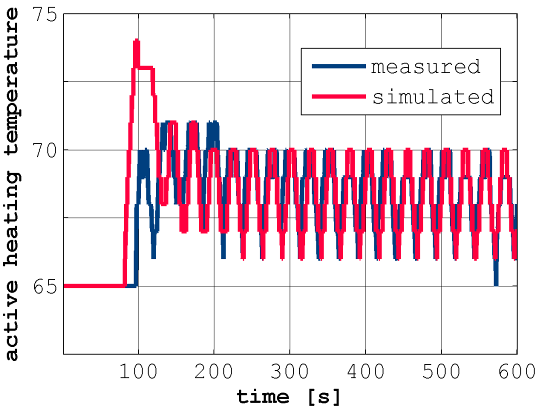


Figure 9: Comparison of measured and simulated active heat temperature

measurementDataAcetone, the recorded DSP output is also available. The comparison between measurement and simulation is the subject matter of chapter 3.

2.2.1 Validation of the Stepper Motor Control

In Figure 8 the actual stepper motor frequency (blue curve) - plotted from the real measurement device when determining the distillation properties of acetone - and the corresponding simulation curve shape (red curve) are presented. The comparison of both, the measured curve shape and the simulated curve shape shows a good approximation of the developed measurement device-model in the simulation environment *Dymola*.

2.2.2 Validation of the Heat Controller

In Figure 9 the active heating temperature regulated by the heat controller (blue curve from measurement data) - when determining the distillation properties of acetone - and the corresponding simulation curve shape (red curve) are presented. The comparison of both, the measured curve shape and the simulated curve shape shows a good approximation of the developed measurement device-model in the simulation environment, again.

3 Discussion of the Simulation Results

As in Figure 8 and Figure 9 already illustrated, the validation of the simulated measurement device is ensured adequately. The further investigation leads to a first conclusion, that the significant oscillating heating tempertuare destabilizes the entire measurement device. A suitable way to smooth this signal is not to smooth the heat controller output as anticipated in chapter 2.1.3 already and realized here, but to smooth the heat controller input, corresponding to the common approach recommended by several control engineering theories [3]. As mentioned above, the heat controller input signal (implemented as PID-controller) is derived from the control signal for the stepper motor and is a strong oscillating signal (cp. Figure 8, nominal value: $120Hz$) and from point of reasonableness it makes sense to smooth this signal.
A different approach, but with similar effect, can be achieved, when a diode array with higher resolution is used (e.g. 16 evaluated diodes), which leads to less ripple in the provided heat controller input signal. These modification and further effects of controller and filter parameter changes will be investigated in the next chapter.

3.1 Parameter Tuning - Improvements

The measurement device for determining the distillation properties of petrochemical end products is depicted in Figure 10 again, though with the already mentioned modification to smooth the *heatController* input. To enhance the filtering effect, the number of averaged signals in the filter block is increased from 32 values up to 64. Thus the smoothed filter output is generated by averaging the last 64 values and implies an additional improvement for the entire system performance. To reduce the signal-ripple distinctly (allows more effective signal-smoothing), a diode array with higher resolution (e.g. 16 evaluated diodes) could

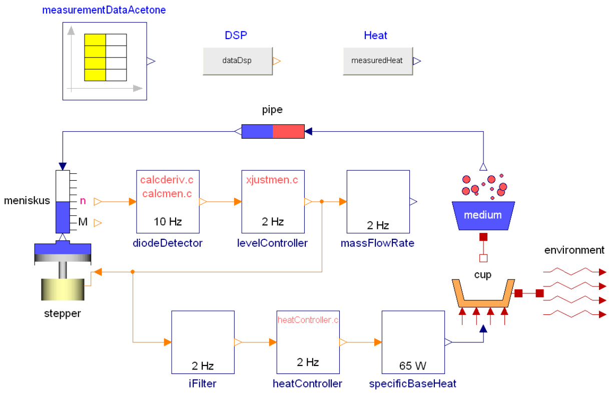


Figure 10: Scheme of the supposed measurement device for determining the distillation properties of petrochemical end products

be used. Without improvements, the achieved active heating temperature from the heat controller indicates curve shapes, as shown in Figure 9 already. With the modifications and when the heat controller parameters are tuned in a proper way, too, the controlled heating energy leads to an optimized evaporation of the medium (acetone) and therefore a more constant flow rate can be achieved, however.

3.1.1 Consequences

Figure 11 depicts the achieved improvement when the *heatController* input is filtered by using an enhanced filter, where the number of averaged signals in the filter block is increased from 32 values up to 64 and when the heat controller parameters are tuned in experimental way: the derivative control component is deactivated and the proportional gain is increased double the amount. When applying these modifications (note that the heat controller is a PI-controller and not a PID-controller anymore) the obtained active heating temperature curve shape shows a significant less oscillating curve shape.

Finally the investigation of the resulting flow rate in the system is of interest and the comparison of the suggested improvements are presented in Figure 12. The red curve shape shows the flow rate of acetone (desired flow rate $10\mu l/s$) simulating the entire model without improvements and where all parameters remain unchanged.

The green curve represents the flow rate of acetone when the *heatController* input is filtered by using the enhanced filter and when the heat controller parameters are tuned. This graph shows a signficant improvement in the flow rate regarding minor oscillating and closer approximating to the desired flow rate value.

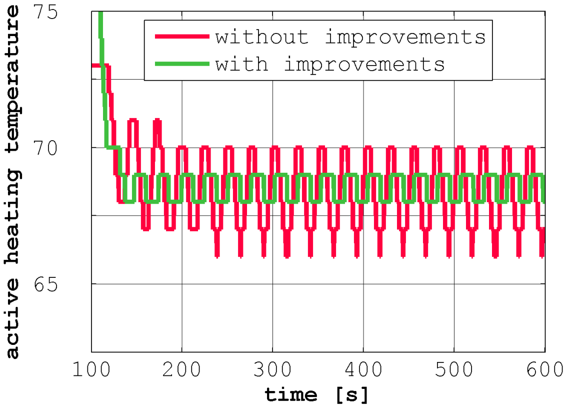


Figure 11: Comparison of the simulated active heating temperature curves - without improvements and with improvements

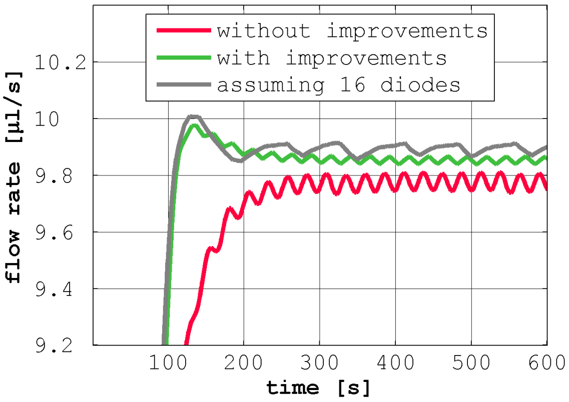


Figure 12: Comparison of the of the simulated curves for the acetone flow rate - without improvements and with improvements

The gray curve shape shows a further enhanced time behavior and results if additionally to the proposed improvements a diode array with higher resolution (e.g. 16 diodes) is used.

4 Conclusions

The purpose of this contribution was the implementation of the entire distillation model in such way, that the simulation reflects the measurement device as best as possible. Following this strategy, the model allows to improve the measurement device and to tune the controllers in the same way as it occurs in the real measurement device by adapting the C-routines of the DSP controllers, whereas time can be saved and a benefit in flexibility can be obtained. The entire *Modelica* model of the measurement device for determining the distillation properties of petrochemical end products was presented and through the use of the simulation environment *Dymola* the design of interdisciplinary models and the design of the entire feedback loop can be revised. The recommended parameter modification especially for the heating controller and the changes regarding the control structure (i.e. the right selection which signals should be filtered) show a significant improvement in the total system behavior.

References

[1] O. Föllinger, *Regelungstechnik*, Hüthig Verlag, Heidelberg, 8 edition, 1994.

[2] Peter Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, IEEE Press, Piscataway, NJ, 2004.

[3] O. Föllinger, *Optimierung dynamischer Systeme*, R. Oldenbourg Verlag, München, 1985.

Modelling of high temperature storage systems for latent heat

Andreas Stückle
Institute of Technical Thermodynamics, German Aerospace Center (DLR)
Pfaffenwaldring 38-40, 70569 Stuttgart, Germany
andreas.stueckle@dlr.de

Abstract

There is a huge demand for heat storages for evaporation applications. Thermal storage systems are used to increase the efficiency of thermal systems by an improved adaption of energy availability and energy demand.

In this paper a possible solution for modular storage systems from 200-600 °C and pressures up to 100 bar is presented. The application of steam as a working medium requires the availability of isothermal storage if charging/discharging should take place at almost constant pressure. The saturation temperature range is between 200°C and 320°C. Therefore nitrate salts are used as phase change material (PCM). The solution developed at DLR is characterized by a modular concept of tube register storages surrounded by both sensible and latent heat storage material.

The focus in this paper is on modelling of the PCM storage. A model is introduced for melting and freezing of the PCM. To perform with an acceptable heat transfer rate inside the PCM, fins are used to increase the overall thermal conductivity. Instead introducing mean storage material parameters, like thermal conductivity or specific heat capacity, the geometry of the finned tube is modelled by using discrete elements. Therefore the model allows detailed studies on heat transfer during space and time. The fin design can be varied in order to find an optimal configuration. A set of partial differential equations is created and solved. When considering a stand alone system, that means tube, fin and PCM, without a connection to other components, investigation is quite effective. In case of the PCM storage there is the big advantage, compared with a sensible regenerator, that the almost constant fluid temperature, when evaporating or condensing, leads to a uniform temperature distribution in fluid flow direction. Therefore only a very rough discretisation in axial direction is needed, what even allows bonding with other components e.g. from the Modelica Fluid Library.

Sensible storages as they are used for preheating and superheating have a characteristic temperature gradient in axial direction. To describe their thermal behaviour concentrated models, using dimensionless numbers, are used.

Keywords: latent heat; storage system; high temperature

1 Introduction

The range for heat storage for evaporation applications is wide. One of the big topics today is the equipment of thermal solar power plants for direct steam generation [1] with thermal storage to increase their total electricity output. Direct steam generation means that the heat transfer fluid is water and steam. Heat accumulators are a key for electricity suppliers to guarantee safety of supply and to achieve good results.

The DLR is working on solutions for storage systems from 200-600°C and pressures up to 100 bar. The saturation temperature range of the saturated steam is between 200°C and 320°C. Due to the saturation temperature, the melting range of the PCM is between 200°C and 320°C. Therefore, nitrate salts are used.

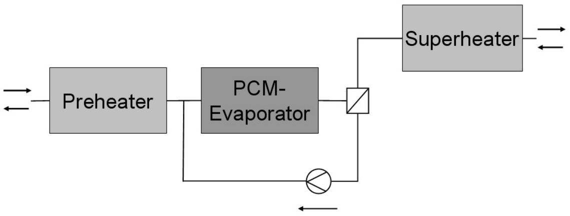


Figure 1: Modular Storage System

The storage systems developed at DLR are characterized by a modular configuration of sensible and latent heat accumulators (see Figure 1). The sensible accumulators are made of a special high temperature concrete. Both species are based on a tube register surrounded by the storage material. The tube register separates the heat transfer fluid from the storage material. According to the Rankine process, a storage system has to deliver superheated steam when discharging. Therefore, the storage system for direct steam generation is divided into modules. Feed water is heated up close to the evaporation point in a preheater storage. Evaporation takes place in the evaporator system consisting of the evaporator PCM-storage, a steam separator and a recirculation pump [2]. The last part of the process is the superheating of the saturated steam. Preheater and superheater storage are both sensible accumulators.

When charging the system, the fluid flows in the opposite direction. The superheated steam is cooled down in the superheater storage; thereafter condensed in the evaporator storage; finally the liquid water is cooled down in the preheater.

Heat transfer in solids is characterized by partial differential equations. Additionally, a model to describe the melting and solidifying of the PCM was developed. Convectional heat transfer as well as condensation and evaporation processes affect the heat transfer at the tube's inner surfaces and are described by a Nusselt correlation for evaporation or condensation [3]. Also, pressure losses are calculated by correlations. A steam- and water model is used to compute state variables. Conservation laws for heat and mass flow have to be solved.

A storage system for direct steam generation is characterized by a modular design. That means that there will not be a monolithic storage block but several units. Therefore, an object oriented approach is convenient, particularly if the storage model will be connected in the future with a power block or a collector field model. Furthermore, peripheral equipment such as pumps and valves have to be modelled and controlled. Because of its interdisciplinary features and modular character, Modelica is used.

The aim of this work is to provide a tool to design and optimize storage systems and analyze especially transient effects.

2 PCM Storage Model

2.1 Discrete Storage Models

In the first package of this work, discrete models of "Storage-Tubes" were developed. Connecting these elements in series and in parallel forms a whole storage module [4]. The storage models are two-dimensional formations fragmented into differential elements. That means that the cylindrical storage element is differentiated in axial and radial directions. The third dimension, which is the angle dependence around the tube axis, is neglected because it is assumed that the heat transfer is axially symmetric. This assumption is particularly proper for vertical tube registers. Vertical design is applied for high temperature PCM storages because of various constructive reasons.

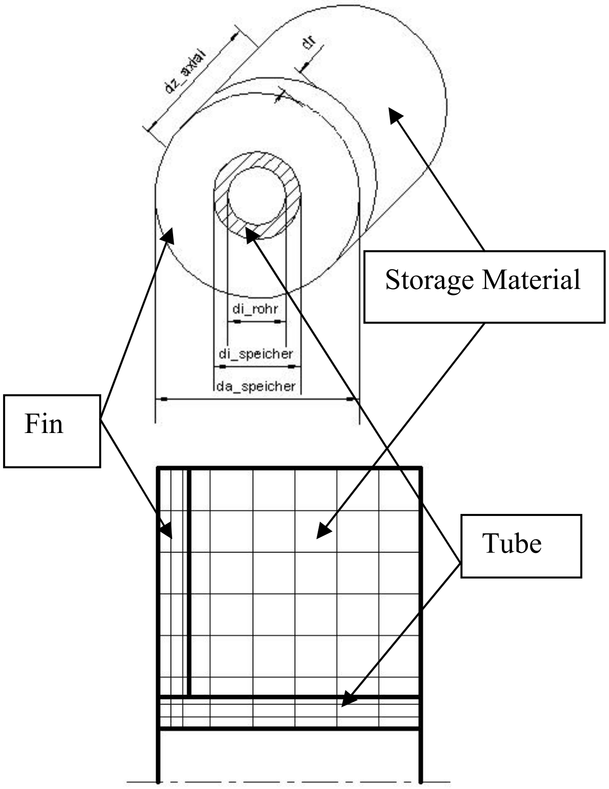


Figure 2: Model and two-dimensional grid of a storage tube

The discrete elements as depicted in Figure 2 are all composed in the same structure. Each element is defined by coordinates x1 and x2. In the Modelica language, indices are used to call the element.

Mass, heat capacity, substance values and state variables are concentrated in a centre point of the finite volume element, which is from here on called cell.

This concentrated mass is surrounded by thermal resistivities. They depend on the heat conductance value of the cell, the distance between the centre point and the heat transfer across the cross-sectional area between two cells (see Figure 3).

The cells are coupled by “connectors” to transfer the flow variable “heat flow rate” and the state variable “temperature”.

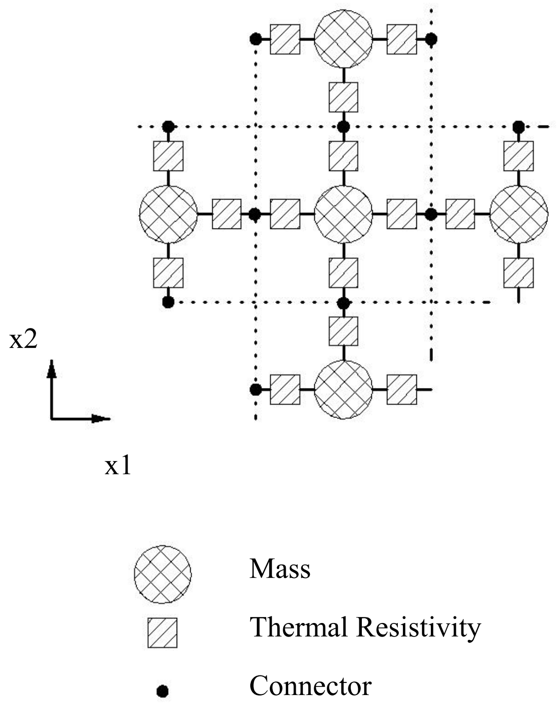


Figure 3: Structure of Discrete Elements

2.2 Energy Balance

The following differential equation describes the enthalpy change of a cell (left side) by the heat flow over the cell’s borders (right side).

$$\frac{d}{dt}\int_V \rho \cdot c_p \cdot T \cdot dV = -\int_S \vec{J} \cdot \hat{n} \cdot dS$$

The heat transfer mechanism on the right side can be convection, conduction or even radiation.

In this paper, only convection on the inside of the storage tubes and conduction inside the material is considered. This leads to the following form whereas for the conduction the heat flow is the product of the cross section, the heat transfer coefficient and the temperature gradient perpendicular to the cross section. For the cylindrical shape, a form factor is used.

At the inner tube side the convective heat coefficient is derived from a Nusselt-correlation.

$$-\int_S \vec{J} \cdot \hat{n} \cdot dS \Rightarrow A \cdot k \left.\frac{\partial T}{\partial x}\right|_S$$

2.3 PCM Model

To model melting and freezing of cells, a model with a specific heat capacity for the solid phase and liquid phase is used. In theory, the enthalpy of a pure medium rises in a step change when melting or freezing. From experiments, we know that in engineering applications with nitrate salts there will not be a phase change at a certain temperature but over a small temperature interval of ca. 1K.

This effect is also useful for numerical calculations where step changes should be avoided to guarantee convergence and to increase calculation performance. Good performance is even reached with a range of 0.1K. As there is no sense in engineering applications to declare such a small range, the melting range is assumed to be 1K.

An effective specific heat capacity for the melting range is defined by the specific enthalpy of fusion divided by the temperature melting range.

$$c_{p,melt} = \frac{h_{melt}}{\Delta T_{melt}}$$

The following figure shows how the enthalpy of a PCM increases with the rise of temperature.

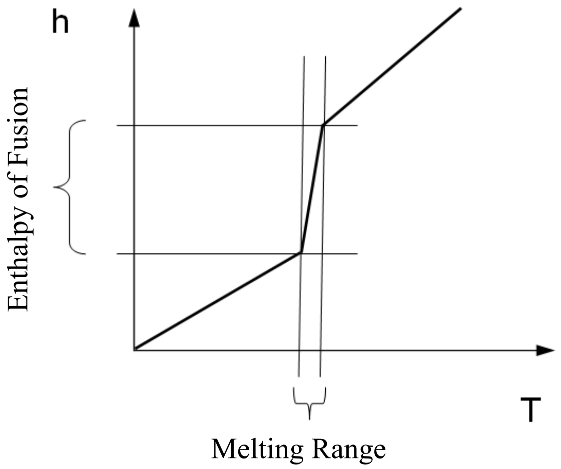


Figure 4: Enthalpy of the PCM

The PCM is treated as a solid even for the liquid phase. That means that there are no convective heat transfer mechanisms if the PCM is liquid. They would improve the performance when charging by mass transfer inside the liquid phase.

The output performance of a thermal storage is critical because the discharge rating is usually higher or equal to the charging rating. When discharging the PCM storage, the PCM first solidifies at the surface of the tube and the fins. Therefore, the heat transfer mechanism is conduction.

2.4 Comparison with a Fluent Model

Plausibility of the model was checked by a 3-D Fluent calculation (see Figure 5, the broken line is the Fluent result, the continuous line from Modelica). The Fluent standard PCM Model is used. As in Modelica convectional heat transfer in the liquid phase is neglected.

The temperatures of the cells lying on the line from the chamfer between tube and fin to the cell with the longest distance from tube and fin were compared (see Figure 2). The rugged temperatures from the Modelica model result from the coarse discretisation, the Modelica grid consists of only 90 knots, but the total time to melt the PCM completely is almost the same. For this case, the PCM was molten in Fluent within 8514s compared to 8726s in Fluent, which is less than 2.5 % difference.

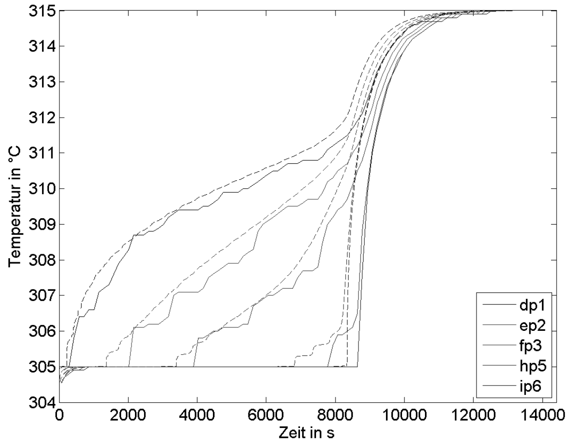


Figure 5: Comparison of the Modelica Model with Fluent

The Fluent calculation took several hours whereas the Modelica model only takes a few seconds. The advantage of the Modelica model is the much higher performance, which even allows coupling of the model into a system simulation.

3 Application

3.1 Dimensioning of Fins and Tubes

These models are used to configure fins for different applications, especially finned tubes and fins for a claimed power in-/output by best utilization.
For evaporating water, high heat flux rates are needed. Therefore, different kinds of fins can be used to conduct the heat flow from the solidifying PCM into the storage material or backwards.

Performance of fins of different materials and geometries is shown. The melting and freezing of the PCM is examined, especially the melting front, which controls power in- and output. The melting front can be visualized as a two dimensional plot using Matlab as depicted in Figure 6.

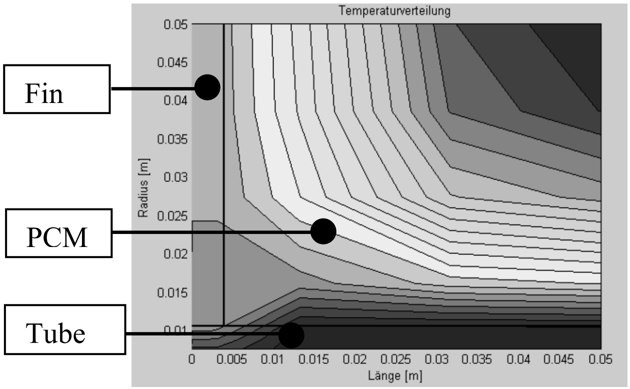


Figure 6: Temperature Field displayed in Matlab

In the following figure, two fins are compared. Integral heat flow rates are depicted. A cycle of complete melting and solidification is shown. The storage was initialized with a homogenous temperature 1 K below the melting temperature. Therefore, before melting 0.5 K of sensible heat is needed. Heating was stopped until the temperature of all cells was at least 0.5 K above the melting range. That means all cells are liquid and slightly superheated. Cooling is done with the same gradient and until the PCM is subcooled 0.5 K. As expected, the model shows a symmetric behaviour.

The peak in heat flow rates at the beginning of charging and discharging are due to the highest temperature gradients at these moments.

$\dot{Q}_{total}$, the total transferred heat flow rate, is the heat flow rate transferred between the heat transfer fluid and the inner surface of the tube. $\dot{Q}_{Fin}$ is the flow rate between the fin and PCM and analogously $\dot{Q}_{Tube}$ between the tube and PCM.

The importance of the heat conducting role of the fins in PCM evaporator applications is underlined when considering that about 85% of the transferred heat flows through the fin.

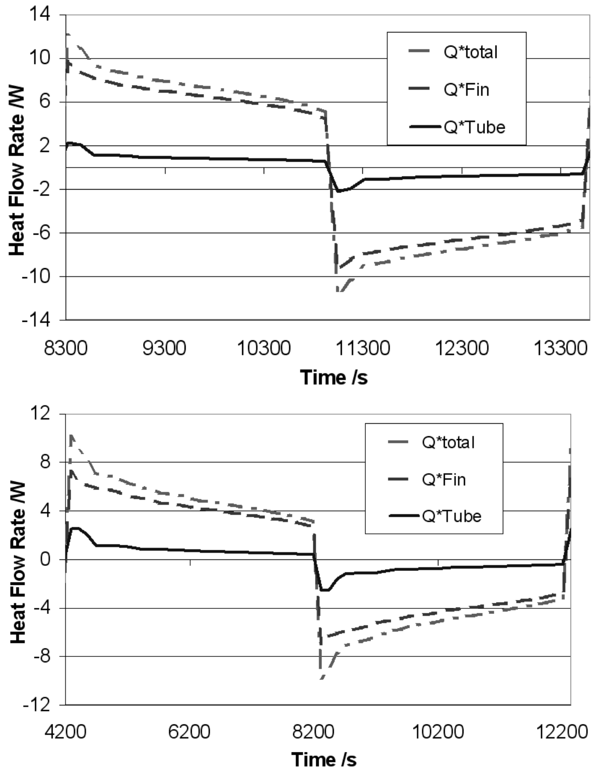


Figure 7: Heat Flow Rates in different Fin Configurations

The tube, fin and PCM mass is the same in both cases. Therefore, the amount of total enthalpy is the same. In the first case, which allows the higher transfer rate, the fins have half the thickness compared to the second case.

With the same amount of material, about 50 % higher heat flow rate is achieved.

3.2 Modeling of a whole Evaporator Storage

In the next step, the tube-fin-PCM-model is multiplied along the tube within a discrete element of the fluid flow. Because of the almost constant temperature of a fluid when condensing or evaporating, the axial discretisation can be relatively rough. This is done in the following example of a PCM-storage evaporator unit (see Figure 8).

While discharging, the source supplies the storage module with almost saturated water. The fluid exits the storage with a mass fraction of saturated steam at the outlet. This mass fraction is separated in a steam drum. The saturated steam flows into the sink. The remaining water is recirculated through the storage by a pump.

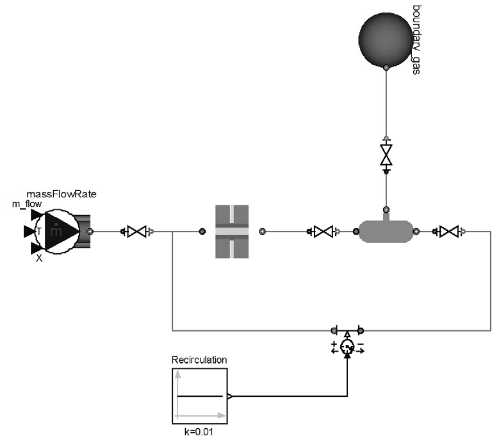


Figure 8: Example of a PCM-Evaporator Model

4 Conclusion and Outlook

A model to design a PCM storage equipped with finned tubes was developed. Rating and characteristic can be calculated. Further, this model is adequate to be coupled with other components, e.g. from the Modelica Fluid Library, to model whole systems.

As depicted in Figure 8, the source and sink used to simulate the sensible storages so far as shown in Figure 1 will be replaced by sensible storage models using coupled differential equations. They can be solved easily in Modelica after transformation into a system of differential algebraic equations (DAEs).

References

[1] Steinmann, W.D., Eck, M. Buffer storage for direct steam generation. Solar Energy, 2006.

[2] Buschle, J., Steinmann, W.D., Tamme, R. Analysis of steam storage systems using Modelica. 5th International Modelica Conference, 2006.

[3] Baehr, H.D., Stephan, K. Wärme- und Stoffübertragung. Springer-Verlag Berlin Heidelberg, 2006

[4] Steinmann, W.D., Buschle, J. Analysis of thermal storage systems using Modelica, 4th International Modelica Conference, 2005.

Modelling of Residential Heating Systems using a Phase Change Material Storage System

Corinna Leonhardt Dirk Müller
Institute for Energy Efficient Buildings and Indoor Climate,
E.ON Energy Research Center, RWTH Aachen University
Jaegerstr. 17/19, 52066 Aachen, Germany
cleonhardt@eonerc.rwth-aachen.de

Abstract

Modern heating systems for buildings need a supply temperature of approximately 35 °C. Standard heat storage systems do not work very efficiently with small supply temperature differences, because of the low sensible heat storage capacity. In contrast to the sensible heat storage a phase change material (PCM) storage system uses the phase change process to store energy at small temperature differences [3], [4], [5]. In this paper a thermo hydraulic model of a PCM storage is developed and implemented by using Modelica, so that dynamic modelling is possible. To show the advantages of latent heat storage (LHS) the PCM storage model has been combined to build a standard heat pump system model with a PCM storage instead of a sensible water storage and the overall system is analysed.

Keywords: PCM; latent heat storage; heat pump; thermo hydraulic modelling

1 Introduction

The topic of this research project is to study a latent heat storage device for modern heating systems with a supply temperature of approximately 35 °C. The integration of the storage system into a heat pump system should equalize the work load profile leading to a higher yearly averaged coefficient of performance. Additionally, the necessary heat pump peak power will be decreased because the heat pump is able to load the storage system over night time. The center of research is the development of computer simulations to examine the feasibility and advantages of a latent heat storage system and of course to find an efficient way of using the power of the PCM.

The institute for Energy Efficient Buildings and Indoor Climate focuses on the research of the reduction of energy consumption of buildings and indoor climate. There are two main research groups in the institute. The first one works on energy systems and the second one focuses on room airflows and indoor comfort. According to these topics, one research approach for example is the thermo hydraulic modelling and simulation of energy systems and single components with Modelica. The institute uses and develops its own libraries [1]. Up to now one library for the thermal building behaviour and a second one for HVAC systems have been created. With the help of these libraries and the Modelica Standard Library a model of a phase change material storage system is created and analysed.

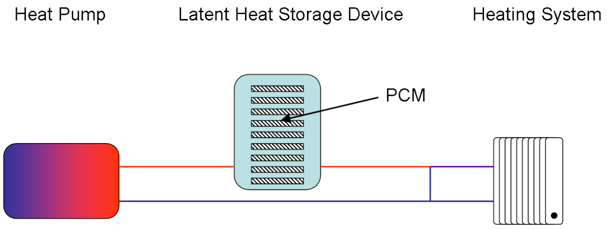


Figure 1: Heat Pump System in Combination with Latent Heat Storage

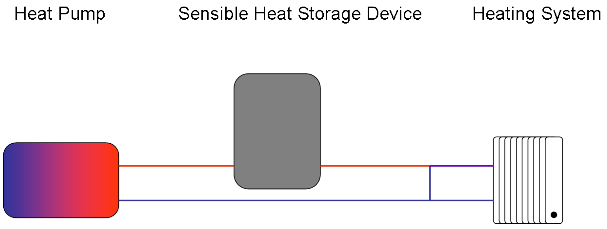


Figure 2: Heat Pump System in Combination with Sensible Heat Storage

DOI: 10.3384/ecp09430025

This paper describes the modelling and simulation of a latent heat storage device and its integration in a heat pump system. First the development of a PCM storage device is shown and then the single components of the heat pump systems are mentioned. Finally a comparison between a heat pump system with a sensible water storage to a heat pump system with a latent heat storage is made and the results are shown in this paper.

The pictures (fig. 1 and fig. 2) show a schematic view of the overall systems. The first system describes a simple residential heating system in combination with a latent heat storage (fig. 1). The second one is the reference case. Therefore, the whole set-up is equal to the first one, but this time the latent heat storage device is replaced by a simple buffer storage with thermal layering.

2 Latent Heat Storage Device

2.1 Heat Transfer Equations

A thermo hydraulic model of a latent heat storage device is developed. The model is based on energy balance, which is given by:

$$\sum \dot{Q}_i = m_{cell} \cdot c_{cell} \cdot \frac{dT_{cell}}{dt} \tag{1}$$

The sum of all heat fluxes $\dot{Q}_i$ is equal to the product of mass m_{cell}, heat capacity c_{cell} and the derivative with respect to time of the temperature $\frac{dT_{cell}}{dt}$. In this paper a modified heat capacity is used and described by an *arc tangent* function [2] of the specific enthalpy:

$$h = h_t \cdot \left[\frac{\arctan((T - T_t) \cdot r_t)}{\pi} + 0.5 \right] + c \cdot (T - T_0) \tag{2}$$

first term (latent heat part)

h_t specific enthalpy of transition
T_t temperature of transition
r_t width of transition

second term (sensible heat part)

c specific heat capacity
T_0 reference temperature of the system

So that the heat capacity is given by:

$$c_p = \frac{h_{trans}}{\pi \cdot [((T - T_{trans}) \cdot r_{trans})^2 + 1]} + \widehat{c_p} \tag{3}$$

Fig. 3 shows an example of the heat capacity and enthalpy function of an invented PCM, which is based on a typical paraffin with a phase change by 320 K. This latent heat storage material has a characteristic phase change temperature of 320 K, a phase change enthalpy of 180 kJ/kg and a heat capacity of 2.4 kJ/(kgK). For the width of the peak in the function of the specific heat capacity curve the factor r_t is set to the value two.

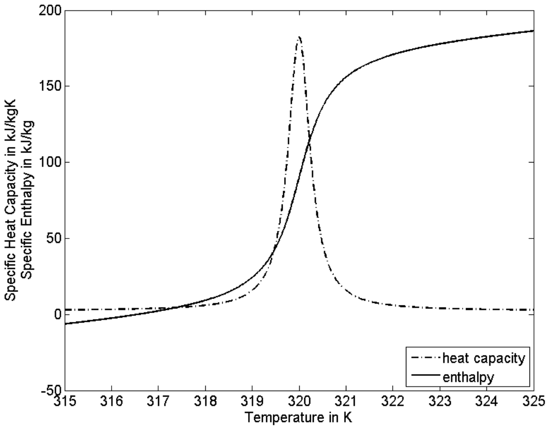


Figure 3: Specific heat capacity and enthalpy

2.2 Model of Latent Heat Storage

With the help of these functions a new latent heat storage model has been created. The new latent heat storage device consists of a box with several PCM plates, which are flowed by water. The new library consists of several sub packages, for example: components, database and systems.

The components package includes all parts of a PCM storage. For instance it includes the plates of different phase change materials and several latent heat storage devices, which differ in the set-up and in the number of PCM plates. Generally a PCM storage device is made up of several single plates in combination with standard pipes of the Modelica Fluid Library (fig. 4). The figure shows one example of latent heat storage device, which exists of 12 plates of PCM and a water flux, which flows in a meandering course across the plates.

The plates are filled with the phase change material.

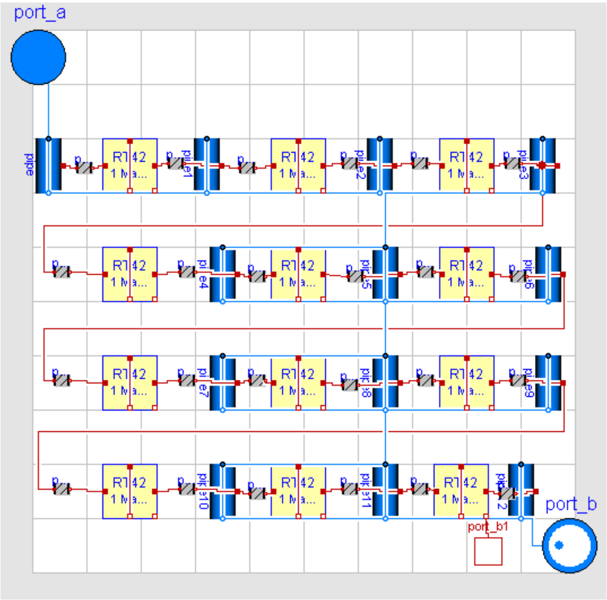


Figure 4: Structure of a PCM Storage Device

Up to now there is no conduction and no mass of the plate itself implemented, so that heat transfer from the latent heat storage material to the pipe wall is ideal. In order to get the best layout of the PCM storage plate the design of an existing storage plate for cooling systems is adapted to the heating system requirements.

One latent heat storage plate can be divided in two directions (see fig. 5).

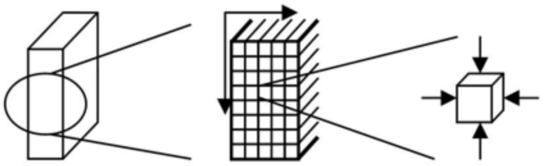


Figure 5: Structure of the Plates

In order to get the opportunity to describe the behaviour of one latent heat storage plate in two directions, it consists of discrete elements (finite volumes). Every single element is described by one energy balance (see fig. 6), which is realized by one capacity block and four conductivity blocks, so that the temperature distribution in the latent heat storage device can be examined.

Fig. 6 shows the four heat conduction elements and one capacity block, which are used. The single

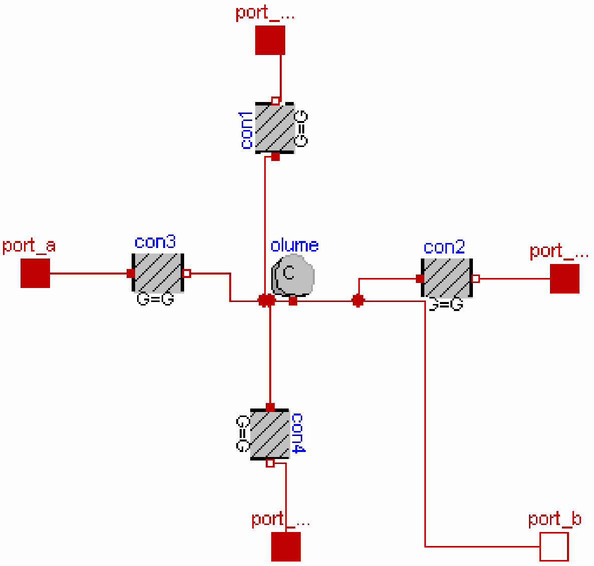


Figure 6: Structure of the PCM-Volume

volumes are connected with each other, so that the whole plate is created.

Due to the fact that thermodynamic properties change during the phase change process the capacity block and the conduction block of the Modelica Standard Library are adapted (see Model PCM _ capacity and Model PCM _ conductivity).

```
model PCM_capacity
Real c_p;
Real T( start=T_start);
  Modelica.Thermal.HeatTransfer...
  Interfaces.HeatPort_a port;

equation
T = port.T;
m_cell*c_p*der(T) = port.Q_flow;
c_p=((h_trans/(pi*((T-T_trans)*...
    r_trans)^2+1))+c);

initial equation
  if steadyStateStart then
    der(T) = 0;
  end if;
end PCM_capacity;
```

The capacity block is duplicated and modified by using the equation (3) mentioned before (see Model PCM _ capacity, equation section).

Another point is that the conductivity of the latent heat storage material is a function of the actual state of aggregation, so that the heat conduction, lambda, is a function of the amount of the liquid and solid mass fraction as shown in the model PCM _ conductivity.

```
model PCM_conductivity
...
if frac_liquid < 0 then
lambda=lambda_solid;
elseif frac_liquid < 1 then
    lambda=frac_liquid*lambda_liquid+...
    (1-frac_liquid)*lambda_solid;
else
lambda=lambda_liquid;
end if;

end PCM_conductivity;
```

Fig. 7 describes another aspect, which has to be kept in mind by using a latent heat storage device, the question of the amount of PCM, which has to be used to keep the building warm.

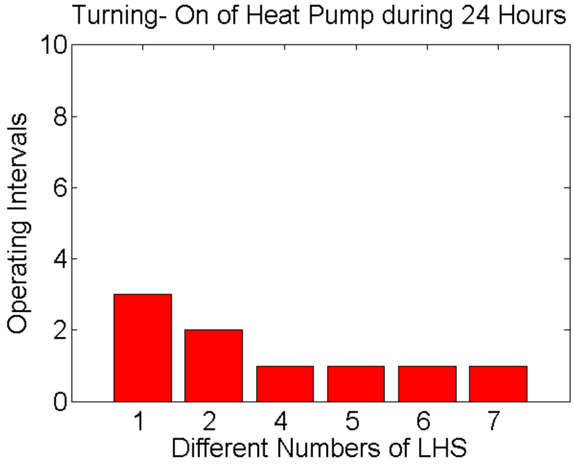


Figure 7: Comparison of different numbers of LHS

The simulation set-up is based on a cool day and a building, which is built up of five identical rooms. As the name already suggests, the simple house model describes the behaviour of a residential building. It consists of five rooms with the same geometrical and thermo hydraulic qualities.

As fig. 7 shows, the heat pump has to turn on three times to keep the rooms warm, if there is only one latent heat storage device. According to this, the heat pump has to turn on once, if there are 7 PCM storage devices integrated.

For the actual study six latent heat storage devices have been taken, which means a total amount of mass of PCM of about 170 kg.

3 Heat Pump System

At the Institute for Energy Efficient Buildings and Indoor Climate an independent library for the whole heat pump system has been developed [1]. This enables us to take off all components for the heat pump system, there. Therefore all components can be directly put together to an overall system (see fig. 8 and fig. 9).

The heat pump model is described in detail in paper [1]. This model consists of two heat exchangers (condenser and evaporator). Both heat exchangers allow changing energy between the heat pump cycle and the medium of the heating cycle.

The standard buffer storage is also a component, which can be taken off the heat pump library. It consists of several layers. Within the model any number of them can be chosen. In this study the buffer storage consists of five layers, so that thermal layering is also taken into account.

The heating cycle is controlled by a heating curve, which is calculated in respect to the ambient temperature.

4 Combined Simulation

A comparison of two heat pump systems has been made. The first system consists of simple buffer storage with thermal layering and in contrast to the first system, in the second one the buffer storage is replaced by a latent heat storage device.

In fig. 8 the buffer storage is presented in the middle. On the left the heat pump cycle is presented and on the right the heating cycle can be seen.

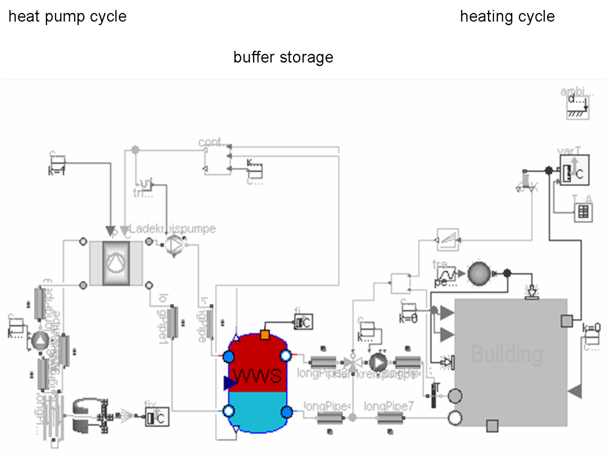


Figure 8: Heat pump system with buffer storage

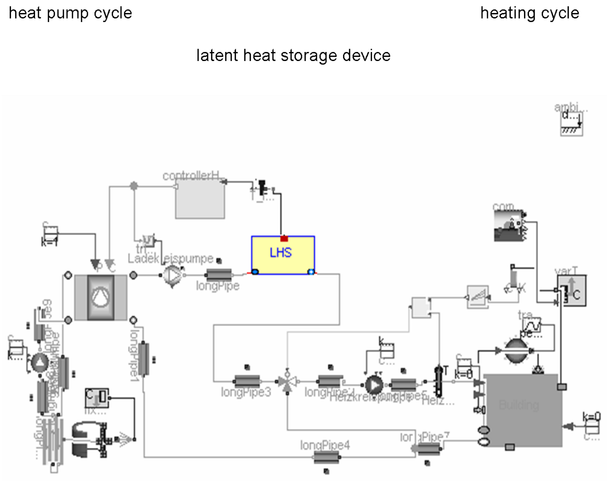


Figure 9: Heat pump system with latent heat storage

The set-up of fig. 9 is built up analogously. Only the storage system, the buffer storage, is replaced by a latent heat storage device.

Both systems are simulated for a cool day in May and the results are shown in fig. 10 and fig. 11.

4.1 Results

Fig. 10 shows the results of the first simulation run of the two systems (fig. 8 (warm water storage) (WWS)) and fig. 9 (latent heat storage) (LHS)) and in fig. 11 the results after an optimization of the overall system are described. In the first diagram of fig. 10 and fig. 11 the results of the heat pump power are described. The second diagram shows the heat pump condenser temperature and the last diagram shows the ambient temperature and the room temperature.

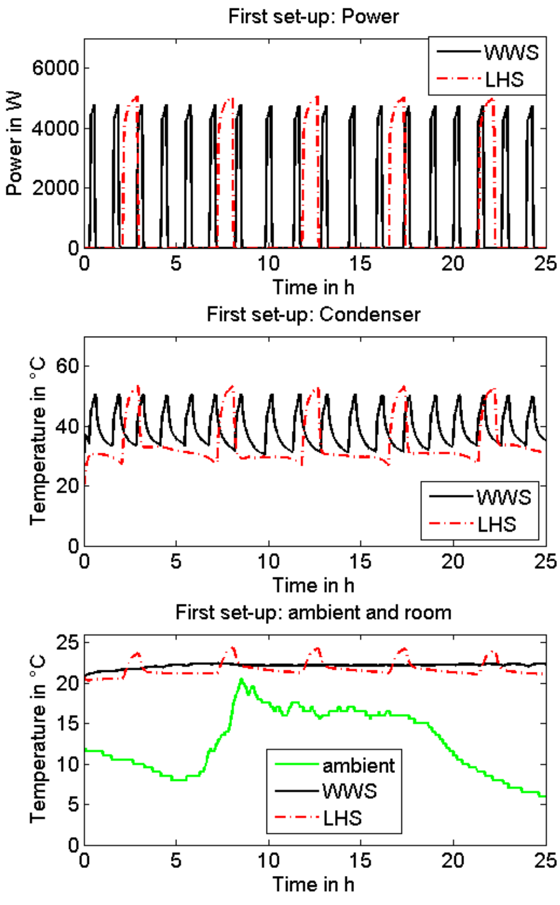


Figure 10: Results of first set-up

The results of fig. 10 and fig. 11 show that the latent heat storage device improves the heat pump behaviour. On the one hand the switching between on and off of the heat pump can be reduced by integrating a PCM storage instead of the buffer storage, so that heating costs can be spared. On the other hand it is even possible to reduce the amount of energy, because of the low supply temperature, which reduces the energy losses. So another effect should be lower heat pump temperatures, but up to now they are only a little bit lower after the optimization, but this still has to be analysed.

4.2 Conclusions

The first simulations of the latent heat storage device show the potential of such a storage system and that its integration into heat pump systems can decrease the heat pump power.

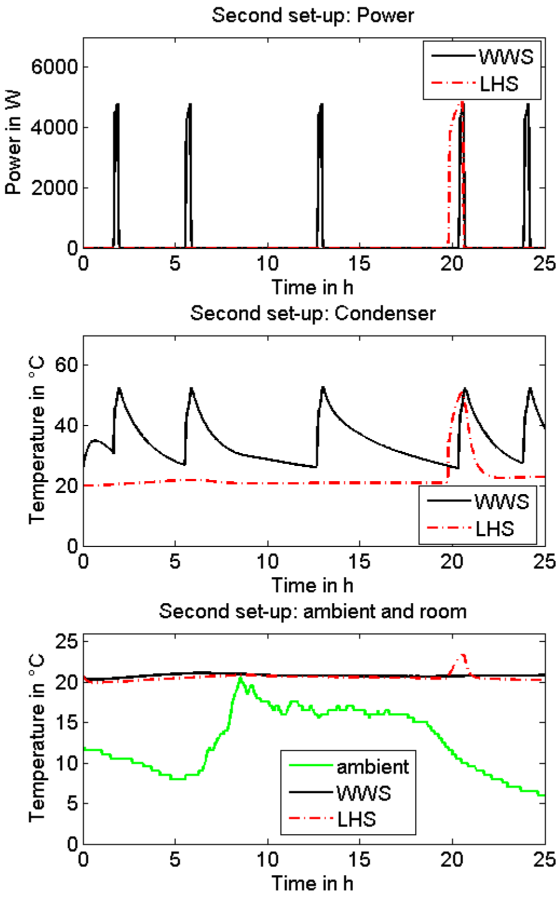


Figure 11: Results of the second set-up

The existing latent heat storage component itself will be improved, for example by adding the conduction and mass of the storage plate. Another point is that the heat transfer to the water in respect to the specific surface area of the plates will be integrated.

One further application, which will be simulated, is the combination with a solar thermal collector. And another interesting addition of the existing simulation model would be the integration of a domestic hot water tank to the overall system.

5 Acknowledgement

The authors would like to thank the E.ON gGmbH for the financial support of the project.

References

[1] Huchtemann, K. Advanced simulation methods for heat pump systems: Modelica 2009, September 20-22.

[2] Buschle J. Analysis of steam storage systems using Modelica: Modelica 2006, September 4-5.

[3] Fachinformationszentrum Karlsruhe (Hg.) Wärmespeicher: BINE- Informationsdienst, Karlsruhe 2005

[4] Mehling, Cabeza Heat and Cold Storage with PCM: Heat and Mass Transfer, Springer, Berlin 2008

[5] htttp://www.zae-bayern.de / deutsch / abteilung-1 / arbeitsgebiete / latentwaermespeicher / einführungg.html (Februar 2009)

Rapid Thermal Analysis of Rigid Three-Dimensional Bodies with the Use of Modelica Physical Modelling Language

Corey Bolduc   Chahé Adourian
Canadian Space Agency, Department of Space Technologies
6767 Route de l'Aéroport, St-Hubert, Quebec, Canada
coreybolduc@yahoo.com Chahe.Adourian@asc-csa.gc.ca

Abstract

Quick analysis of thermal systems without the use of CAD models may be very valuable for rapid-prototyping. Such a need led to the heavy modification of the Heat Transfer package found in the Modelica standard library. The library in question revolves around a three-dimensional generic block composed of a variable number of interconnected elements with individually assignable thermal and physical properties. When applying the accompanying library functions, one may create moderately complex thermal structures with non-uniform thermal properties. Furthermore, other tools available also allow the user to attach chains of blocks of different resolutions, as well as insert one block into another to create composite models with the possibility of internal heat generation.

Keywords: Thermal Analysis, Modelica, Finite Element Analysis, Matlab

1 Background

Thermal analysis software may be generally categorized into one of two groups, each with their own respective strengths and weaknesses. The first group represents packages such as the 1-D HeatFlow library in Modelica which relies on primitive nodal techniques to produce relatively crude analyses of thermal models, but in short order and with little effort. On the opposite side of the spectrum lie the other group consisting of highly developed Finite-Element Analysis software such as NASTRAN, ANSYS, and CATIA. Although the latter group offers superior fidelity, CAD models are required for analysis, which is often time consuming in itself to produce. Somewhere in the middle lie a group of hybrid tools which may be considered as rapid thermal analysis tools, and whose importance is only beginning to be recognized. They are characterized by being capable of modeling thermal models that go beyond 1-D but at the same time don't require a CAD model in order to create them. We describe in this paper the implementation of such a thermal modeling library with the use of the Modelica language.

2 Introduction

The need for a rapid thermal analysis tool arose naturally with the development of a multidisciplinary satellite simulator. Among other criteria, a means of predicting the temperature for specific sections of the spacecraft at any point in orbit became necessary. By combining multiple internal and external heat sources as those produced by solar radiation and various onboard power systems, a complete and reasonably accurate analysis may be executed for both transient and steady-state scenarios. Utilizing generic thermal models, one may quickly create specialized components which may be "snapped" together, thereby creating complex combinations quickly.

The overall objective is visualized by the components seen in figure (1). The top part of the figure shows the satellite being modeled with its two solar panels on the left and the right and the main body in the middle. These different components are connected mechanically, electrically and thermally. Below the satellite are three Modelica device blocks each containing behavioral models from each of the physical disciplines mentioned previously (and more). Their structure reflects that of the spacecraft. On the left and right are the Modelica models of the solar panels and in the middle the main body. Each device model possesses proper interfaces in each discipline to connect to its neighbors and more importantly for the purposes of this paper they each contain a *Thermal Block* model shown below the device. The *Thermal Blocks* in turns

DOI: 10.3384/ecp09430036

is composed of *Thermal Elements* which are shown as nodes on a grid at the bottom of the figure.

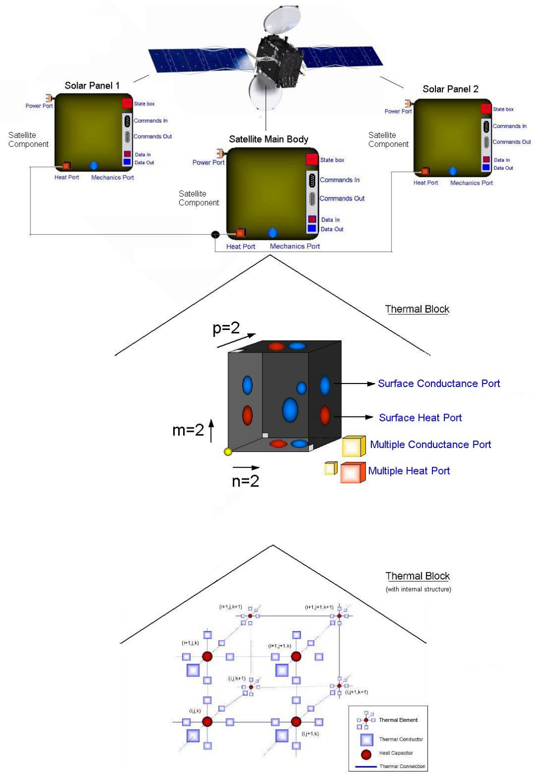


Figure 1: Example

The *Thermal Element* was designed such that it allows and simplifies the creation of hierarchically structured thermal components that fits nicely within the device models and can be representative of the hierarchical structure of real systems. This is accomplished by providing various types of ports permitting direct access to heat flow and temperature variables within *Thermal Elements* and *Thermal Blocks*. Designers have then the freedom to apply different boundary conditions (e.g. temperature, or heat flow) to individual elements and define thermally dynamic components/devices such as a battery or a heater.

Each *Thermal Element* in *Thermal Block* may be assigned thermal properties different from those of its neighbors' such as specific heat (heat capacitance), thermal conductance and initial temperature. Thermal conductance is of particular interest since the ability to assign it a value of zero effectively prevents any heat flow through an element, resulting in what accurately represent a cavity. The assignment of the aforementioned properties is carried out by use of base functions (three shapes for each individual thermal property and density). For each property, these functions are available for spherical, rectangular prismatic, cylindrical distributions, and are used to assign a particular property value within the shape's boundary and another on its outside.

Other tools developed also give the user the option to interface blocks of dissimilar resolution, or to insert multiple blocks into a single block in order to represent a satellite containing interacting components of varying fidelity, and heat sources through conduction and radiation (future development).

Despite its powerful simulation capabilities, Modelica (and Dymola) leave much to be desired in terms of user interfacing and three-dimensional graphic visualization of simulation results. In order to expedite results analysis, a program was created in Matlab to display an animation of the temperature and heat flow distribution within a body (or bodies) as a function of time. It is the hope that in the future further advances of the Matlab code will lead to an independent executable which will be able to act as the interface between the user and Dymola, thereby facilitating the initial configuration and debugging of models.

3 Thermal Element

The basic building block of the thermal model is the thermal element representing the thermal behavior of a cubic shape with isotropic thermal properties. The mathematical modeling follows from finite volume methods as explained in [1]. In the following sections, we provide the details of this model.

3.1 Physics

The *Thermal Element* is comprised of six thermal conductors to represent thermal connections from any of the six faces of the cube and connected to a central heat capacitor modeling the cube capacitance via a heat port (Figure 2). Two thermal conductors are placed on each side of the heat capacitor in each direction. The thermal conductors transport heat and are normally assigned a constant thermal conductance value given by equation (3.2) where G_i is the heat conductance, k is the thermal conductance, A_i is the cross-sectional area and L_i is the length of the cube in the direction

$$i \in \{+x, -x, +y, -y, +z, -z\} \tag{3.1}$$

of heat flow.

$$G_i = k\,\frac{A_i}{L_i/2} \tag{3.2}$$

To calculate the heat flow through each thermal conductor equation (3.3) is used where T_i and T_c are respectively the temperatures of surface A_i and the center of the cube. Heat flow into the cube has positive sign.

$$Q_{ThermalConductor_i} = G \, \frac{T_i - T_c}{L_i/2} \tag{3.3}$$

The heat capacitor modeling the heat storage capacity of the cube completes the thermal model of the single cube. Given the specific heat capacity c and the mass of the cube its thermal capacitance C is calculated using equation (3.4).

$$C = c\, m \tag{3.4}$$

Heat flow into the capacitor is given by equation (3.5) where $\frac{dT_c}{dt}$ is the time rate of change of the capacitor's (or cube's) temperature.

$$Q_{HeatCapacitor} = C \, \frac{dT_c}{dt} \tag{3.5}$$

Because a heat flow is associated with each thermal conductor and the conductors are thermally coupled to the central heat capacitor, the total heat flow into the capacitor (or equivalently the cube) is the sum of heat flows through the conductors (or cube surfaces) and is given by equation (3.6).

$$Q_{HeatCapacitor} = \Sigma Q_{ThermalConductor_i} \tag{3.6}$$

The *Thermal Element* is a single node in a rectangular grid discretisation of a physical object given by the *Thermal Block* described in section (4).

3.2 Coding

The thermal element is assembled using standard components from the Modelica library. Relevant components include thermal conductors and a heat capacitor from *Modelica.Thermal.HeatTransfer* library and constant sources from the *Blocks.Sources* library. These are arranged as shown in figure 2.

3.3 Interfaces

A *Thermal Element* has many different interfaces useful under different circumstances.

Default Heat Ports In the basic use case of a *Thermal Element*, the elements are packed side by side and their heat ports connected individually top-to-bottom, left-to-right and front-to-back in order to form the grid nodes of a single *Thermal Block*.

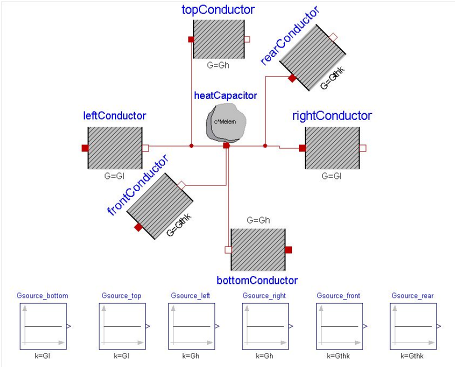


Figure 2: Thermal Element model

Multiple Heat Port In some situations that we will encounter later, it is convenient to have a single connector which combines all six heat-ports of a *Thermal Element*. A custom connector called *HeatPortMulti* was created for this purpose and is part the *Thermal Element*'s interfaces.

Conductance Ports In addition to the heat ports, there are six signal output ports that hold the conductance values G_i and are used when connecting to a resolution adapter (i.e. Mapper or Surface Interface). Under some circumstances, it is necessary for a *Thermal Element* to share with connected thermal components its conductance value for correct physical modeling.

4 Thermal Block

The *Thermal Block* pictured as a grid in figure (3) is the main physical representation which the user initializes and analyzes after simulation. It represents a block of material in the shape of a rectangular prism and is composed of *Thermal Elements* set on a cartesian grid. Setting the parameters of the individual *Thermal Elements* is the method by which the user defines the actual shape of the prism. This is equivalent to a Finite Element Model with the parameters at each grid point determining both the type or even presence of material together with its physical properties. For example, an empty cavity inside the prism can be modeled by setting the conductance to zero at grid points in the cavity. Different materials may be specified at different grid points again by varying the parameter values at these points.

4.1 Mathematical Model

The *Thermal Block* is assumed to be a rectangular prism with height H (x-direction), length L (y-

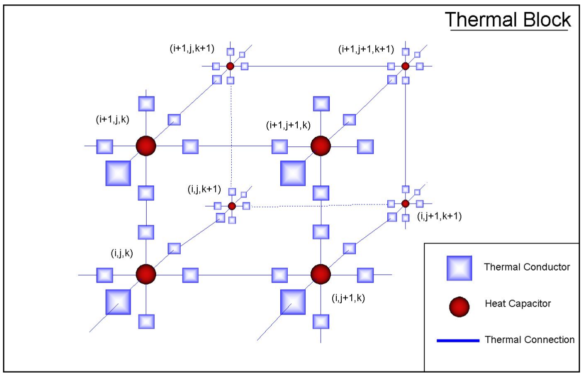


Figure 3: Thermal Block

direction) and thickness Thk (z-direction). The grid resolutions or the number of *Thermal Element* nodes in each direction are specified by integers N_x, N_y and N_z which the *Thermal Block* used to create a lattice structure of *Thermal Elements* connected by their heat ports.

Thermal conductivity, specific heat capacity, initial temperatures and mass-density distributions must also be provided and are used to initialize the *Thermal Elements*. The physical dimensions and grid resolutions are used to calculate *Thermal Element* lengths L_i and cross-sectional areas A_i in each direction i which in turn are used to calculate the individual thermal parameters C and G_i.

4.2 Physical Property Distributions

By default, the aforementioned properties are set to be filled uniformly with a default value (arbitrarily chosen to be that of aluminum) and can be changed by the user. However, uniform distributions are not very interesting and another option is required.

Components with complex geometries and non-uniform thermal properties are made possible with the aid of pre-defined functions. We defined example functions that provide the parameter distributions necessary to model cylinders, spheres and rectangular prisms of constant k, c, T_0 and ρ values. Thermal property values for both inside and outside the shape in question are specified. Moreover, the center of each shape function may be positioned anywhere within the *Thermal Block*, and superimposed on one another. For example, two create a thermal model representing two parallel cylinders side by side we take a single *Thermal Block* and apply a superposition of two Cylindrical distributions with proper offsets.

By extending the use of distribution functions, one may create cavities in the material by setting the conductance of certain thermal elements to zero, thereby effectively preventing heat flow through them as well as reducing the number of equations to solve. The hollow function originally prevented the connection of heat ports between neighboring *Thermal Elements*. This was counterproductive as these connections were needed when inserting a *Thermal Block* into another. Therefore, the current cavity function works by assigning a thermal conductance value of zero to the thermal element to be isolated from its neighbors. Given the resulting “empty space” within the material, this does not forbid the possibility to connect another block to the cavity walls.

4.3 Interfaces

The *Thermal Block* has various interfaces for connecting to other thermal components. There are six interfaces to connect to the surfaces of the block. The internal grid elements of the block are also visible allowing for example to connect an external heat source to the center of a *Thermal Element* or to insert a block into another.

In order to allow for these complex interconnections, access to heat and conductance value information were devised through the creation of specialized ports. The first, referred to as a HeatPortMulti port, includes six standard heat ports (one per *Thermal Element* surface) each of which connects automatically when HeatPortMulti’s are connected to one another. Such a port was implemented in order to facilitate the insertion of one block into another by the Mapper which is covered later. The second type of port, the ConductanceMultiPort port, serves the same function as the former, but for conductance.

Both types of ports were implemented as arrays of size and dimensions equal to those of the *Thermal Elements* within the *Thermal Block*, and were connected to the corresponding sides of the appropriate element before connecting to one another in sequence.

With an infrastructure established to access the internal elements of a *Thermal Block*, six standard heat port arrays and six conductance port arrays, each of dimension two, were introduced and connected to each side of the *Thermal Block* in order to facilitate surface-to-surface connections. These side ports are created conditionally only when connecting a particular side, so as to avoid the creation of mathematically problematic extra equations.

4.4 Block Insertion

Given the grid structure of the *Thermal Blocks*, it is natural to consider overlaying two such grids. Such a scenario arises when we want to model components contained within components. For example the satellite's external metallic frame contains within it a multitude of components including the computer, sensors, actuators, batteries, etc. We would like one *Thermal Block* to represent the satellite's structure with a cavity within. Another block could represent the battery which is smaller than the first block and is contained within it. Inserting one block into another can be seen as an overlaying of the two thermal grids at the intersection of the two physical objects. Thermal connections must be established at the interfacing surfaces. The Mapper component enables this by accessing the internal elements of each block and properly mapping them to each other.

5 Surface Interface

The *Surface Interface* model serves to thermally couple the external surfaces of two thermal blocks with different grid resolutions. When attempting to thermally connect two such bodies, one finds that their respective array indices cannot be correlated to each other one-to-one. Instead a more complicated linking mechanism must be implemented which requires not only connections to the surface heat ports but also knowledge of the conductance of each of the surface elements. Therefore the surface interfaces of *Thermal Blocks* contains both heat port and conductance port arrays of dimension two on each end. The array connectors of the *Surface Interface* are initialized manually by the user and must match the array sizes of the surface arrays on the adjacent bodies . Information on heat and conductance from each body is passed through both ends to a custom thermal conductor array which receives its conductance value externally through a conductance connection. In order to implement physically correct heat distribution, algorithms were implemented to distribute the heat flow from one element to the appropriate number of elements on the opposite end.
Below is a sample of the distribution algorithm for the connection between the input heat port array and a thermal conductor array:

```
for i in 1:s loop
  for j in 1:t loop
    for Mi in (1:sp) loop
      for Mj in (1:tp) loop
        M1[Mi + (i - 1)*sp, Mj + (j - 1)*tp].G
          = CPL[i, j] / (areaRatio1-1);
        connect(HPL[i, j],
          M1[Mi + (i - 1)*sp,
          Mj + (j - 1)*tp].port˙a);
       end for;
    end for;
  end for;
end for;
```

M1 is one of the external thermal conductor arrays, sp and tp are the corresponding dimension sizes to s and t and (either u or v depending on surface rotation), $areaRatio1 = u \times v$, and HPL is the heat port array on the left side.

Interface parameters The parameters s and t represent the number of elements in each planar direction of the interacting surface of the left body (u and v for the right body). The algorithm essentially functions by finding the product of the surface dimensions and their counterpart dimensions on the opposing side (e.g. $areaRatio1 = u \times v$). The heat flow and thermal conductance values for each body surface element are individually supplied to a number of thermal conductors equal to that of the number of elements on the surface of the other body. The thermal conductor itself does not in fact utilize the received thermal conductance value immediately, but instead increases or decreases its value in order to avoid the intermediate thermal conductor arrays interfering with heat flow, which would ultimately skew the results. Included in the surface interface model is the option to rotate the right surface with respect to that of the left surface (input). Such a feature becomes necessary when constructing objects such as a cube out of four separate blocks.

Contact Resistance In the physical world, when two objects come into contact with one another, there exists a thermal resistance known as thermal contact resistance. Thermal contact resistance represents the thermal resistance created by the two materials not coming into complete contact with one another due to microscopic roughness and resulting spaces. These spaces may either contain a fluid or vacuum, each of which will inhibit the heat flow between the materials. In order to simulate the aforementioned phenomenon, a standard thermal conductor array is implemented in the interface whose thermal conductance ($G = A/Ri$) is based on the contact area and the interface resistance. Contact resistance tables exits for a variety of materials, finishes, roughness, temperature and other conditions.

Surface Mapping In order to distribute the heat flow accurately through the surface interface from one block to another, two separate thermal conductor arrays are employed. This is shown in figure (4). The two resolution values (e.g. m and p) for each surface are multiplied by their directional counterpart to find a product ($m \times p$). The thermal conductor arrays are then established as being size $(mSide_1 \times mSide_2) \times (pSide_1 \times pSide_2)$. The intermediate thermal conductor array is based on a standard (library) thermal conductor, but fitted with a heat and conductance port, thereby allowing it to have its G inputted by the G of the connecting element in the adjacent block. With current value of G known for each element in a block and the desired resolution change, the equivalent G may be computed. Therefore, the computed G serves to increase, or decrease resistance as required across the surface interface, in order to obey heat conservation.

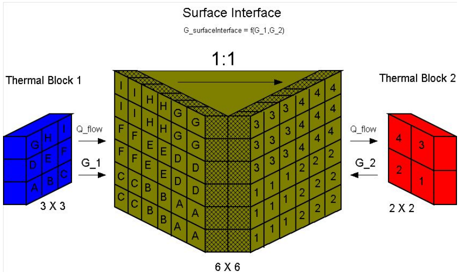


Figure 4: Surface Interface

6 Mapper

A three-dimensional extension to the surface interface is the Mapper. The Mapper allows for one body of a particular resolution, to be thermally coupled within another block of a different resolution. Such an option would allow one to be able to insert a power supply into a certain section of the satellite's main body.

7 Animation

Because of the somewhat limited data visualization capabilities in Dymola when working with three-dimensional data, a visualization tool was developed using Matlab. At the present moment the visualization tool accepts Dymola analysis data from the thermal model run and renders a three-dimensional animation. Visualization allow color-coded animation of the temperature scalar data as well as heat flow vector information. Figure (5) shows an example heat distribution within a *Thermal Block* at a specific point in time. The visualization capability is of great help in determining whether our model is behaving properly and for debugging purposes.

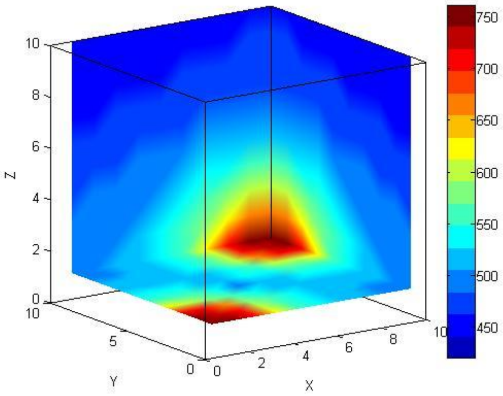


Figure 5: Temperature Distribution Snapshot

8 Conclusion

In conclusion, we have developed an experimental thermal modeling library using Modelica that lies halfway between simple 1-D modeling tools and advanced CAD-based ones. The block diagram modeling capabilities of Modelica were used to provide high-level snap-on thermal models that could represent 2-D and 3-D models with non-uniform thermal distributions. Further, mechanisms to connect these 3-D models in complex ways were developed including the capacity to insert blocks one into the other following the idea of hierarchical composition.

References

[1] Earl A. Thornton. Thermal Structures for Aerospace Applications. AIAA, 1996: 98-99.

SoundDuctFlow: A Modelica Library for Modeling Acoustics and Flow in Duct Networks

Helmut Kühnelt Thomas Bäuml Anton Haumer
Austrian Institute of Technology, Mobility Department
Giefinggasse 2, A-1210 Vienna, Austria
{helmut.kuehnelt,thomas.baeuml,anton.haumer}@ait.ac.at

Abstract

SoundDuctFlow, a Modelica library for the joint calculation of acoustic and flow quantities in HVAC (Heating, Ventilation and Air Conditioning) ducts is presented. Modeling the sound propagation in ducts by one dimensional acoustic two-port methods is a well-established technique for the acoustic characterization of large HVAC duct networks. Two different approaches of acoustic modeling will be considered in the framework: When resonant phenomena are insignificant, it is often sufficient to apply band-averaged models to predict the sound power level and the transmission loss within the individual components of the duct network. For the low frequency range where linear plane wave propagation is valid, acoustic two and multi-port models based on a transmission matrix formulation of the sound pressure can be applied for high frequency resolution and phase accurate calculations. For the prediction of the mean air flow in the duct network pressure loss models are applied. The coupling of acoustic and flow elements permits the simulation of flow acoustic phenomena.
For setting up large networks a smooth work flow is vital for the user: The simulation is easily set up using the GUI provided by Dymola. External parameterisation ensures persistent data management. The resulting system of equations is automatically pre-processed and solved by Dymola.
In this paper an overview of this new library is given together with exemplary applications.

Keywords: acoustics; flow; ducts; flow noise; HVAC; modeling; simulation; Modelica library

1 Introduction

The joint calculation of sound level, air flow rate and pressure loss in large HVAC duct networks as well as the prediction of the noise level in the passenger compartment gains more and more in importance in the early phase of design. The unified Modelica framework for one-dimensional acoustic and flow simulation introduced here serves as a tool for concept modeling. Used by the design engineer in an early design phase, it has to meet several requirements: The prediction method has to be fast and computational efficient. All kinds of available data should be used, from analytic and (semi)empiric models to data obtained from measurements and three-dimensional acoustic and fluid dynamics computer simulations to characteristic diagrams and sparse point data provided by component manufacturers. It also should be extendable to add physical phenomena like heat transfer or transport of humid air and complex flow-acoustic interactions. Finally the framework should be open for integration into a complete system simulation.

2 Modeling acoustics in duct networks

In modeling acoustics of duct networks usually following assumptions are made: Only the steady state of the system has to be regarded. The dimensionality of the systems often can be reduced from 3-D to 1-D. The acoustic properties of each component can be characterized by an acoustical two-port (or a multi-port in the case of a branching), a grey box whose transfer behavior can be calculated by means of 1-D to 3-D methods, like finite (FEM) or boundary element methods (BEM),

DOI: 10.3384/ecp09430115

independently from each other. Two standard one-dimensional approaches are available:

First, the sound power-based description has been widely used and forms the basis of the most standards and guidelines for the analysis of sound in ducts, e.g. see ASHRAE [1, 2] or VDI [3]. It can be applied for frequencies well above the plane wave cut-off. The sound power level within a duct network can be determined by summation of the losses and gains in sound power level at the connecting interfaces of the components from the fan towards the terminal sections of the network. All contributions resulting from wave reflections are neglected in this approach. This makes the prediction procedure very simple, but reduces the reliability considerably.

Second, the plane wave based description of ducts, mufflers and networks [4, 5, 6] covers the low frequency range. A variety of analytical plane wave two-port models is available in the literature. The acoustical transmission matrix of non-standard components can be determined by 3-D FEM/BEM or unsteady CFD simulations in case of non-zero mean flow. In HVAC duct networks with their large lateral dimensions, however, the frequency range of plane wave propagation is rather limited. Nevertheless, accurate sound prediction at lower frequencies below a few hundred Hz is quite important, since absorbing liners are ineffective at those frequencies. In Table 1 the plane wave cutoff frequency, f_c, is given for some exemplary circular ducts.

Table 1: Plane wave cutoff frequencies for circular ducts.

diameter	f_c	typical usage
10 mm	20 kHz	brass wind instrument
50 mm	4 kHz	car exhaust pipe
300 mm	670 Hz	HVAC ducts

2.1 Sound power-based description

The sound power-based description of sound transmission in ducts is valid for frequencies well above the plane wave cut-off. Wave reflections are neglected. The transmission loss of a duct element is considered. The variable of state then is the sound power level L_W:

$$L_W = 10 \log_{10} \left(\frac{P_{ac}}{P_0} \right) \mathrm{dB} \tag{1}$$

with a reference sound power level $P_0 = 10^{-12}$ Watt. Typically L_W is specified for octave or 1/3-octave frequency bands. The summation of levels L_i obeys the following rule:

$$L_{sum} = 10 \log_{10} \sum_{i=1}^{n} 10^{L_i/10} \tag{2}$$

The acoustic elements based on the sound power description are extended from one, two or multi-ports. Each port is represented by a signal-based connector holding two arrays of incoming and outgoing L_W in N frequency bands. Since there are no reflections considered, in passive elements like silencers, each incoming L_W signal is mapped onto the outgoing L_W independently of its counterpart:

$$\begin{aligned} L_{WiA}^{out} &= L_{WiB}^{in} - \Delta L_{WiB \to A} \\ L_{WiB}^{out} &= L_{WiA}^{in} - \Delta L_{WiA \to B} \end{aligned} \tag{3}$$

with ΔL_{Wi} the reduction of the sound power level at the ith band. This allows bidirectional usage of the components.

In-duct sources, like fans or bends contributing regenerated flow-noise, add their forward and backward contribution to the total sound pressure level according to eq. (2)

Figure 1 gives some examples of acoustic components already included in *SoundDuctFlow*, like different types of pipes, bends, silencers, fans.

2.2 Plane wave description

In the wave based description the steady state, frequency dependent plane wave solution of the wave equation is regarded. This approach is valid only for the low frequency region below the plane wave cut off. Although this region is rather limited because of the large lateral dimensions usually found in HVAC duct networks, an accurate calculation is nevertheless important, since absorbing materials are rather ineffective at those frequencies. Two physically equivalent methods can be applied to characterize the individual duct element.

In the *scattering matrix method* the sound pressure p is decomposed into forwards and backwards traveling plane waves:

$$p(x,t) = \hat{p}(x) \exp^{\mathrm{i}\omega t} = p^{+} \exp^{\mathrm{i}(\omega t - kx)} + p^{-} \exp^{\mathrm{i}(\omega t + kx)} \tag{4}$$

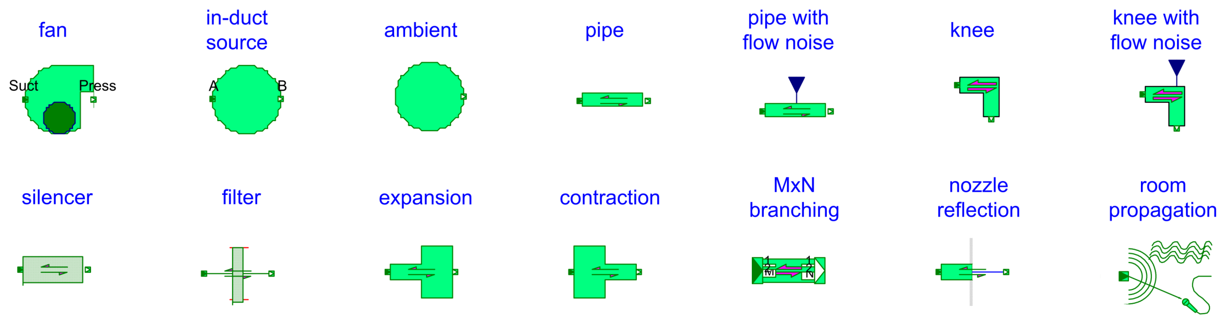


Figure 1: Examples of acoustic elements.

where $k = \omega/c_0$, p^+ and p^- are the amplitudes of forwards and backwards traveling waves. The acoustic particle velocity then is:

$$\hat{v} = \frac{1}{\rho_0 c_0}\left(p^+ \exp^{-ikx} - p^- \exp^{ikx}\right). \tag{5}$$

A four pole scattering matrix S maps the forward and backward propagating sound pressures of one port onto the second port:

$$\begin{pmatrix} p_1^+ \\ p_1^- \end{pmatrix} = \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix} \begin{pmatrix} p_2^+ \\ p_2^- \end{pmatrix} \tag{6}$$

This formulation is preferred in connection with the experimental determination of the acoustic reflection coefficient $R = p^-/p^+$ at the interfaces of duct elements.

The *transmission matrix method* describes the relation between acoustic pressure, p, and acoustic volume flow, q, in forward and backward sections of the single element by the matrix T:

$$\begin{pmatrix} p_1 \\ q_1 \end{pmatrix} = \begin{pmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{pmatrix} \begin{pmatrix} p_2 \\ q_2 \end{pmatrix} \tag{7}$$

This formulation is preferred for modeling in Modelica since the potential variable p and the flow variable q directly can be used to constitute the physical connector.

A collection of plane wave models will be included into the *SoundDuctFlow* library.

2.3 Modeling of junctions

In the *sound power-based* description, at junctions the incident sound power is distributed over the total of all outgoing duct sections. By definition there are no reflections. The proportion of the sound power transmitted from section i to section j, $W_{i \to j}$, then is

$$W_{i \to j} = W_i \frac{S_j}{S_{tot} - S_i} \tag{8}$$

where W_i is the sound power incident at the duct section i, S_i, S_j and S_{tot} are the incident, outgoing and total cross-sectional areas.

In the *plane wave* model, the reflection of waves at a multiple junction is considered. As an example a T-shaped junction between three pipes of cross-sectional surfaces A_1, A_2 and A_3 is examined here. From the equation of Bernoulli we find that the acoustic pressures p_1 and p_2 in the duct just before and after the side-branch have to be the same as the pressure p_3 at the mouth of the side-branch:

$$p_1 = p_2 = p_3 \tag{9}$$

The conservation of mass yields that the sum of the acoustic volume flows at the junction is zero:

$$q_1 + q_2 + q_3 = 0 \tag{10}$$

3 Modeling flow and pressure loss in duct networks

The 1-d modeling of air flow in HVAC ducts and other components is based on several approximations: The air flow in HVAC ducts with its low Mach number is regarded as incompressible, steady state and fully stabilized. In most cases it is also turbulent. The state variables for fluid flow are the static pressure as the potential variable and the mass flow rate as the flow variable. The pressure loss coefficient $K = \frac{\Delta P}{\rho U^2/2}$ relates the pressure difference ΔP between the two ports of an flow duct element to the kinetic energy density of the flow. It depends on the Reynolds

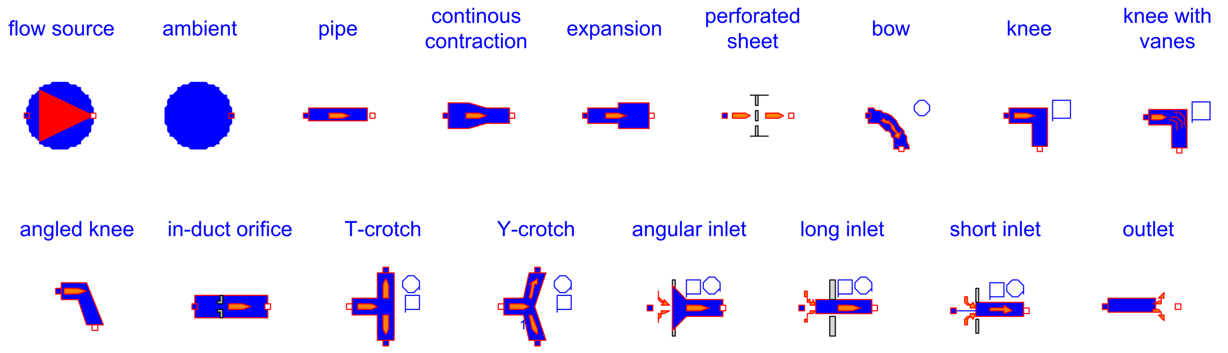


Figure 2: Examples of flow elements.

number, on the relative roughness and on the cross-sectional shape of the duct.

Analytic models, like the Colebrook-White equation, as well as measurement data, given as characteristic diagram or interpolated formula [7, 8, 9] or tabulated manufacturer data serves as input for the various flow models. Effects by turbulence, adverse velocity gradients, cross-sectional shape, surface roughness, curvature, interaction between elements in not fully developed flow regions result in corrective terms. Within the Modelica library flow elements for standard duct components are available (Fig. 2).

4 Complex duct components

Complex compound flow-acoustic duct components (see Fig. 3) can be assembled from simple elements representing either the acoustic or the flow part of a duct element using a special connector that combines the acoustic and the flow connector. This allows the modeling of non-standard components.

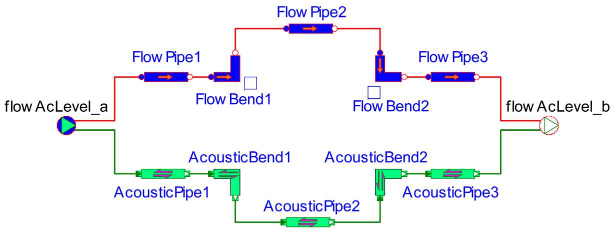


Figure 3: Complex compound flow-acoustic duct component utilizing a combined flow-acoustic connector.

5 Flow-acoustic interaction

Flow-acoustic coupling can also be modeled on this basis. Inside duct elements, like bends or orifices, the high Reynolds number flow triggers flow noise. In most cases the back-reaction of the sound field on the flow is negligibly weak. This effect can be modeled by extracting the state of the fluid flow and feeding it into the acoustic element. There the additional flow noise is calculated by (semi)empirical models, as given in [3, 10] for instance. An example of a complex compound pipe component with flow noise is given in Fig. 4.

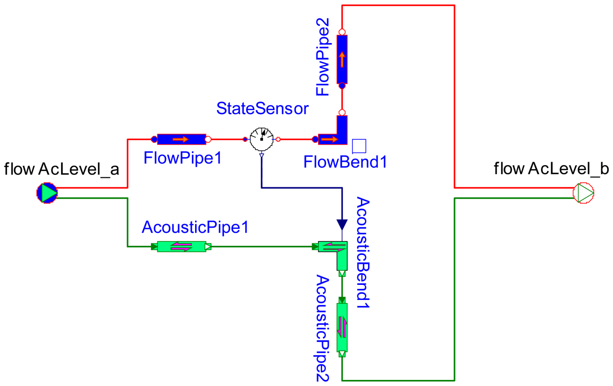


Figure 4: Compound flow-acoustic duct component in which the mean flow velocity generates flow noise at the bend.

6 Work flow and practical issues

When dealing with large models, the parameterisation of each component in the Dymola GUI is error-prone and not very convenient for the user,

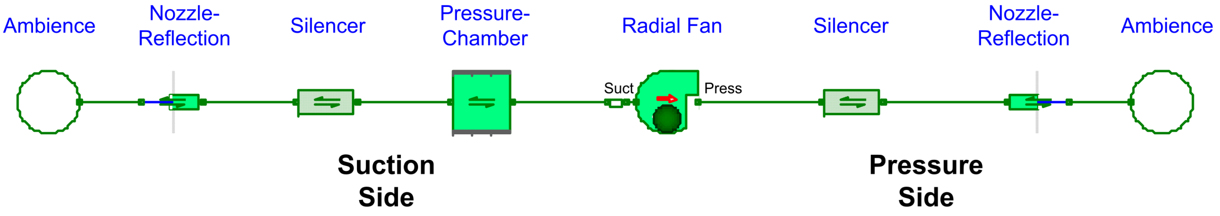
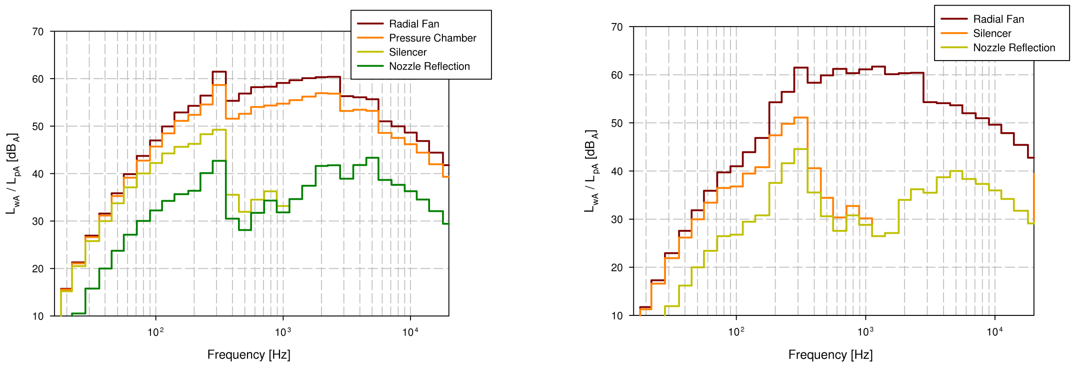


Figure 5: Acoustic representation of of a ventilation unit.



(a)

(b)

Figure 6: A-weighted L_W in third octave bands at the suction side (a) and pressure side (b) components of a ventilation unit.

especially when the parameters are varied repeatedly within the design process. Therefore we use an external parameterisation procedure to ensure consistent data handling and a smooth work flow: After setting up the model in the Dymola GUI, all model parameters and connections are extracted into an XML structure. This XML structure then is filled in with the parameters' numerical values and read in at the run time of the simulation. All results are also written to an XML file for external postprocessing procedures like report generation.

7 Examples

As a first example, the acoustic representation of a ventilation unit is presented in Fig. 5. Here it it is not connected to the duct network but in a configuration for measuring the radiated sound power from its exits. Therefore the nozzle reflections at the intake and exhaust pipe terminations have to be included. The predicted A-weighted L_W at each element is shown in Fig. 6. Prediction and measurement of the A-weighted L_W at the exits (Table 2) agree quite well.

Table 2: Measured and predicted A-weighted sound power level of a ventilation unit.

Position	Sound power level L_{wA}	
	Measurement	Prediction
Pressure Side	52 dB(A)	50.4 dB(A)
Suction Side	54 dB(A)	51.9 dB(A)

The ventilation unit is then installed in a small duct network (Fig. 7). The predicted third band SPL at each element is shown in Fig. 8.

8 Conclusions and perspectives

The concepts of sound propagation in HVAC systems together with modeling of the air flow are realized in the new Modelica library *SoundDuctFlow*, providing a joint prediction of the noise level and the pressure loss in the duct network. The concept of acoustical connectors based either

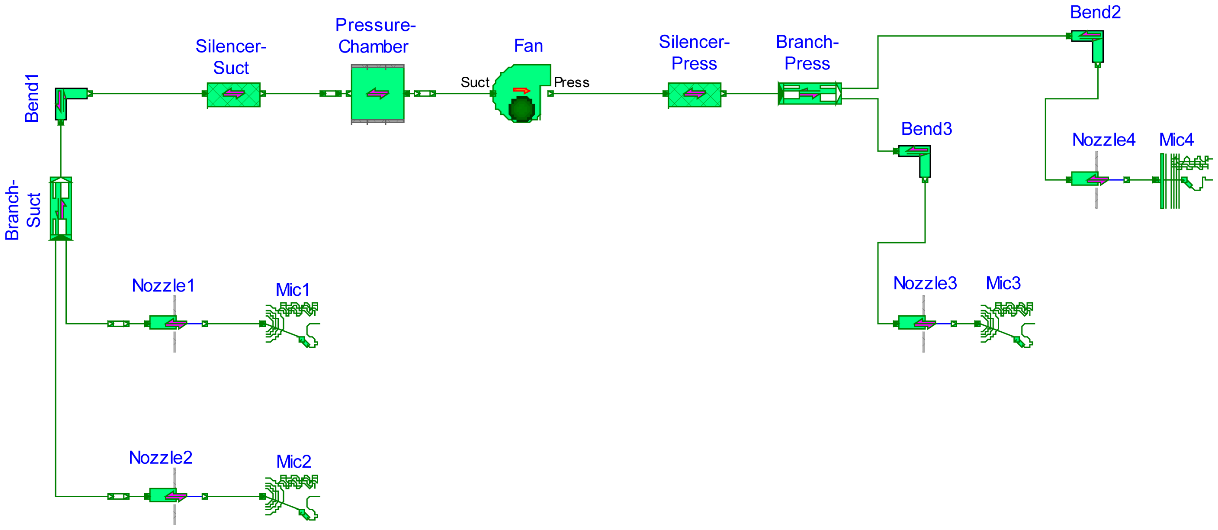
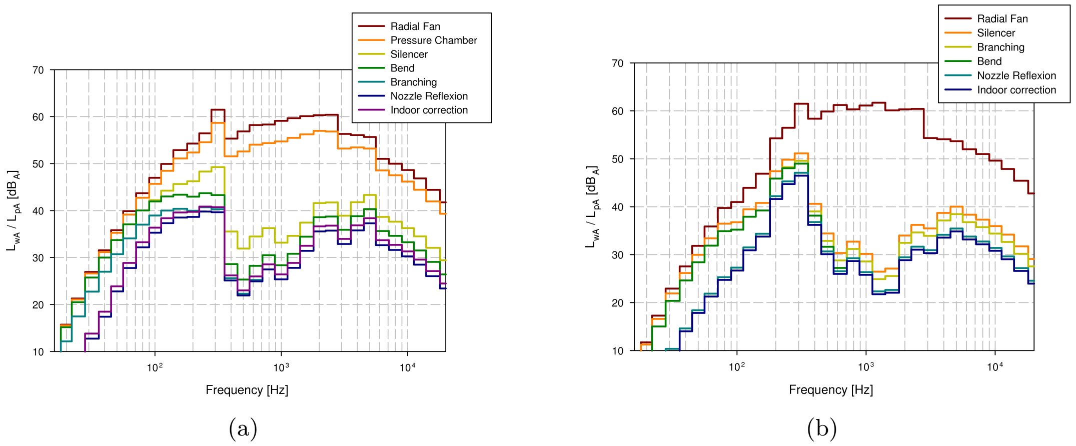


Figure 7: Acoustic representation of a ventilation duct network.



(a)

(b)

Figure 8: A-weighted L_W at each element of a ventilation duct network along two paths from the fan to a listener, (a) at the suction side and (b) the pressure side.

on a sound power level description or a plane wave description was discussed. The effects of the geometric conditions on the mean air flow is modeled by 1D pressure loss models. The triggering of noise by flow effects is also modeled. The modeling of the emitted noise level of a ventilation unit agrees quite well with acoustical measurements.

SoundDuctFlow will be extended to incorporate also heat transfer, transport of humid air and complex flow-acoustic interactions. For the last point, however, there will be the need for new models deduced from complex 3D computational aeroacoustic simulations.

9 Acknowledgments

We are very grateful to Gerhard Karlowatz, Liebherr-Transportation Systems, Austria, for providing real life network examples and measurement results.

References

[1] Reynolds, D. D. and Bledsoe, J. M., *Algorithms for HVAC Acoustics*, ASHRAE Inc., Atlanta, 1991.

[2] ASHRAE, Sound and Vibration Control, *2007 ASHRAE Handbook – HVAC Applications*, ASHRAE Inc., Atlanta, 2007 (Chapter 47).

[3] VDI, Association of German Engineers, *VDI Guideline 2081*, VDI, Düsseldorf, 2001.

[4] Munjal, M. L., *Acoustics of Ducts and Mufflers*, Wiley, New York, 1987.

[5] Boden, H. and Abom, M., Modelling of Fluid Machines as Sources of Sound in Duct and Pipe Systems, *Acta Acustica*, 3 (1995), 545-560.

[6] Boden, H. and Glav, R., Exhaust and Intake Noise and Acoustical Design of Mufflers and Silenecers, in *Handbook of Noise and Vibration Control*, ed. by Crocker, M. J., John Wiley & Sons, 2007.

[7] Miller, D. S., *Internal Flow Systems*, BHR Group Limited, Cranfield, UK, 1996 (2nd ed).

[8] Idelchik, I.E., *Handbook of Hydraulic Resistance*, Begell House, 1996 (3rd ed).

[9] Recknagel, Sprenger, Schramek, *Taschenbuch für Heizung und Klimatechnik. R. Oldenbourg Verlag*, München Wien, 1999 (69th ed).

[10] Nelson, P. A. and Morfey, C. L., Aerodynamic sound production in low speed flow ducts, *Journal of Sound and Vibration*, 79(1981), 263–289.

Introduction of the 3D Geometrical Constraints in Modelica

Régis PLATEAUX Olivia PENAS Faïda MHENNI
Jean-Yves CHOLEY Alain RIVIERE
LISMMA – EA2336 (SUPMECA)
3, rue Fernand Hainaut - 93407 Saint-Ouen Cedex France

{firstname.lastname}@supmeca.fr

Abstract

Mechanical Modelica Libraries enable one to model instances of geometrical objects and their absolute positioning. But in order to facilitate the design by taking into account the size and the position of objects, we need relative positioning variables and parameters. In this paper, we propose to meet this challenge by adding in Modelica a set of 13 geometrical constraints and by implementing them.

Keywords: Modelica 3D, Geometrical Constraints, TTRS, Relative positioning

1 Introduction

In previous studies [1], to facilitate the design of a mechatronic product [2], we proposed to integrate the entire downward design cycle in order to achieve a modelling continuity. To do this, we proposed a hybrid design methodology based on several tools, languages and methodologies including Modelica [3], in Dymola [4] environment.

Modelica models integrate some geometrical elements such as position of gravity point, mass, volumes and inertia matrix elements. However, the current environments don't allow until now to improve entirely our method. Figure 1 presents the desired 3D Modelica framework enabling a simultaneous representation in the "modelling" design window of the 2D logical diagram and of the 3D geometric class and keeping the 3D view for the simulation

Actually they usually propose:

- 2D icons containing only geometrical parameters (absolute positions and dimensions) in the modelling interface,
- realistic 3D representations of Modelica objects in the simulation interface with the same geometrical parameters,

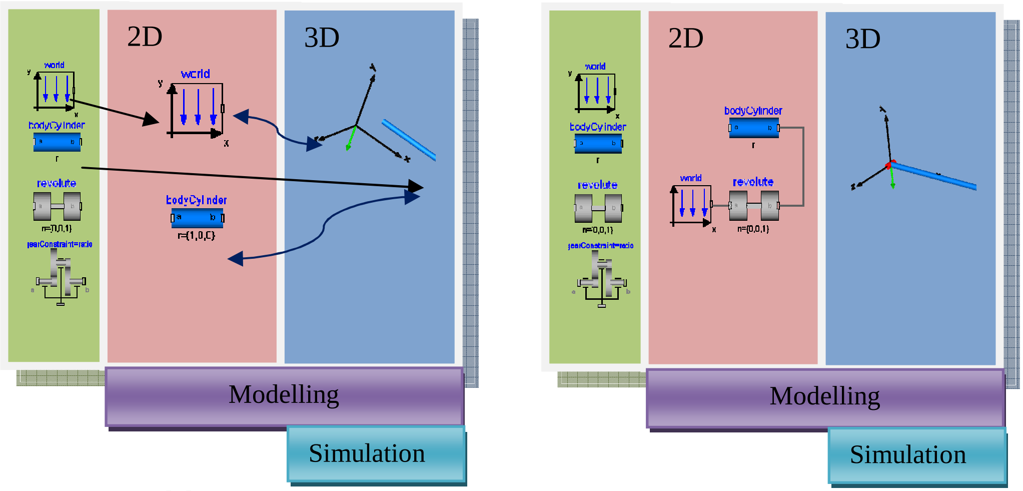


Figure 1. 3D Modelica Environment

DOI: 10.3384/ecp09430038

But:

- no geometrical variables, and
- a framework without any dynamic link between both of these interfaces.

For example, the geometrical structure of the model in Figure 2 cannot easily be anticipated in 3D. We must wait for the simulation to know which of structures has been defined.

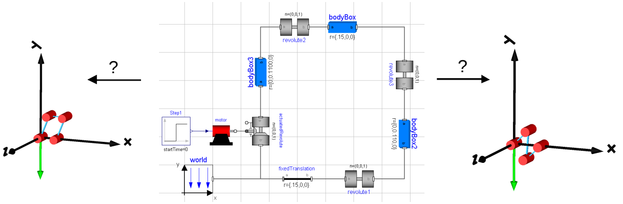


Figure 2. Two possible representations of the geometry of a model: parallelogram or twisted parallelogram.

That is why our aim is to transform geometrical parameters into geometrical variables in both interfaces and to propose for the design one new modelling interface with both views (2D and 3D). Thus today we tackle the first problem by modifying libraries of Modelica objects in order to switch geometrical instances in objects modelled by constraints. It will later help the transition from 2D to 3D modelling for the second step [5].

To do this, the first point is to introduce explicitly geometrical constraints as Modelica objects.

2 Towards an explicit relative positioning in Modelica

Considering the model already developed in our laboratory for the geometrical tolerancing [6], we implement geometric constraints with the « Topologically and Technologically Related Surfaces » (TTRS) theory [7].

2.1 TTRS Objects in Modelica

Any surface or association of real surfaces of an object is related to a kinematic invariance class named TTRS. There are 7 classes of TTRS classified according to increasing degrees of freedom (DOF) in Table 1.

Table 1 7 Classes of TTRS

Classes	Symbol	DOF
Identity	$\{E\}$	0
Revolute	$\{R_{D,P}\}$	1
Prismatic	$\{T_D\}$	1
Helical	$\{H_{D,P}\}$	1
Cylindrical	$\{C_D\}$	2
Spherical	$\{G_P\}$	3
Planar	$\{S_O\}$	3

Kinematic joints can be expressed by TTRS.

Each TTRS is characterised by a MRGE (Minimal Reference Geometric Element). Each MRGE is made up of a combination of one point, one line and/or one plan, but does not take into account the intrinsic dimensional aspect of the object.

Figure 3 shows the example of a cone, which is represented by the TTRS “Revolute Surface” whose MRGE representation is one point and one line.

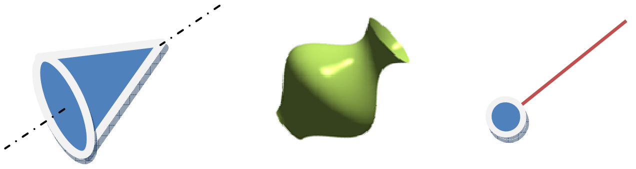


Figure 3. Cone representation: TTRS Revolute & MGRE point/line.

In order to assemble two geometrical objects, i.e. to define geometrical constraints between two TTRS, 44 associations are identified depending on the relative orientations and positions with regards to the other. In turn each association forms TTRS. They correspond to the most elementary formulation of a kinematic connection between objects.

Example: association of two Revolute TTRS

Given $\{R_{D1,P1}\}$ and $\{R_{D2,P2}\}$:

If $D1 \neq D2, \{R_{D1,P1}\} \cup \{R_{D2,P2}\} = \{E\}$

If $D1 = D2, \{R_{D1,P1}\} \cup \{R_{D2,P2}\} = \{R_{D1,P1}\}$

Finally the passage to its MRGE enables us to have only 13 possible cases of constraints (Table 3 p.5).

These constraints between MRGE numbered from C1 to C13 are expressed by means of algebraic expressions and parameters.

For example: the association of 2 beams by application of the C12 constraint (Figure 4) corresponds to:

Given (M_1, $\mathbf{u_1}$) and (M_2, $\mathbf{u_2}$) two sliding points and vectors which belong respectively to D1 and D2 lines, we have:

$$C12 = \left\{ \begin{array}{c} \mathbf{u_1} = \mathbf{u_2} \\ \mathbf{u_1} . \overrightarrow{\mathbf{M_1 M_2}} = 0 \\ \mathrm{d} = \left\| \overrightarrow{\mathbf{M_1 M_2}} \right\| \end{array} \right\}$$

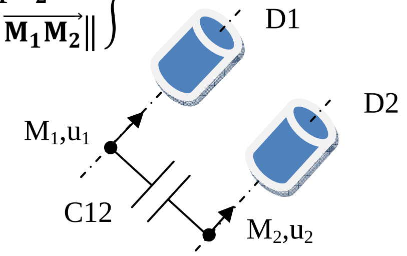


Figure 4. C12: line-line, parallel, distance.

2.2 MRGE Expression in Modelica/Dymola

The set of related MRGE may be implemented with Modelica: each MRGE is an object (point, line and plan).

In Modelica language, each object is associated to another by means of its topological connexion performed through its connectors. The elementary connector for the MRGE is the point. The implementation choice was to represent the MRGE line by its affine view using the previous elementary MRGE point.

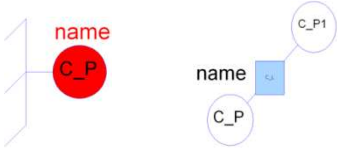


Figure 5. Modelica implementation of the MRGE Point connector and of the MRGE Line with 2 point connectors and 1 line connector.

2.3 The 13 Constraints

With MRGE, the 13 constraints are generated thanks to their algebraic expressions. The model is now designed by constraints: they are no longer instances of geometrical objects but their equivalent constraints (e.g. Figure 5) with real geometrical variables and parameters.

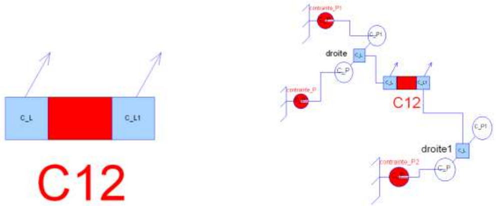


Figure 6. Example of the C12 Constraint in Modelica: two lines constrained parallel.

3 Academic Application

Models currently developed take into account only geometric variables. Further improvements will integrate mechanical ones.

One goal is to come closer to the design habit. *What you see is what you mean* is another one. So when we express perpendicularity, parallelism or any geometric constraint, our point of view is to declare it.

After a short description of the treated example, we will study the method used with current models. Then we will build it step-by-step with the new constraint objects.

3.1 Design Point of View

The studied mechanical system is an automatic rising barrier called “Sinusmatic”. The SINUSMATIC barrier is adaptable to the dimensions and speed to most applications. Its particularity results from patented kinematics for his bellcrank that transforms the continuous circular movement into an approximately sinusoidal 1/4 round one.

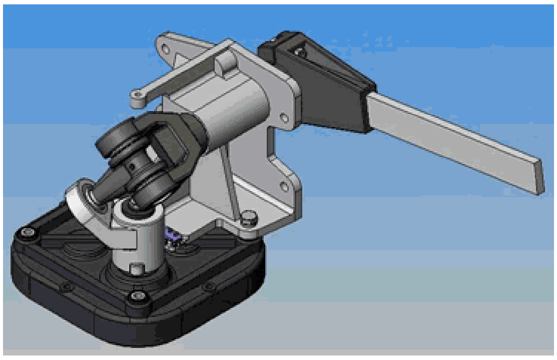


Figure 7 Currently Sinusmatic

Its mechanical structure is schematized in Figure 8.

It is composed of S0: frame, S1: plateau, S2: socket, S3: ball, S4: crosspiece and S5: fork axis.

Joints are expressed in Table 2.

Table 2 Sinusmatic Joints

	S1	S2	S3	S4	S5
S0	$\{R_{D,P}\}$				$\{R_{D,P}\}$
S1		$\{E\}$			
S2			$\{G_P\}$		
S3				$\{C_D\}$	
S4					$\{R_{D,P}\}$

Some design requirements are specified:

- Axis of the cylindrical joint go through the centre of the ball-and-socket (S3/S4),

- Centre D of S4/S5 joint is the intersection of S1/S0 et S5/S0 axes,
- Axes of (S3/S4) and (S4/S5) joints are converging.

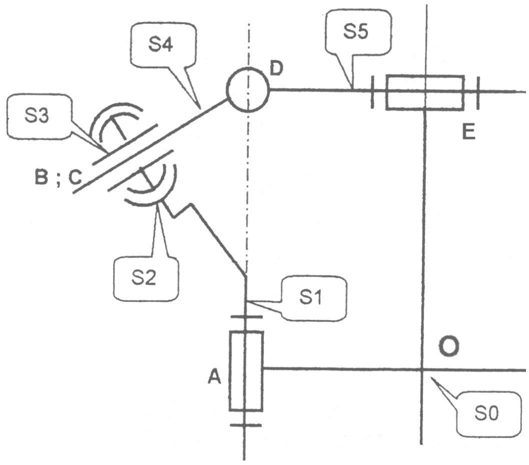


Figure 8 Kinematic Diagram

3.2 Current approach

We need to express the structure by means of vectors as Figure 9.

The direct impact is that we need all values or a set of consistent values.

Moreover the sense depends on the choice of the connected initial point.

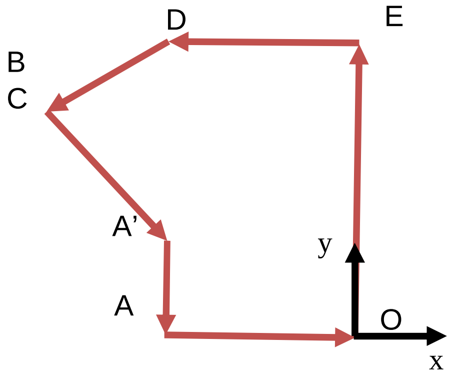


Figure 9 Vectorial Model

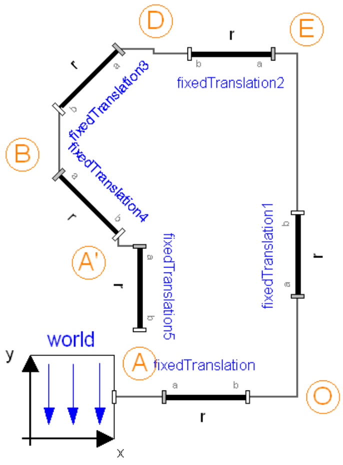


Figure 10 Modelling with Vectorial Approach

3.3 Method with constraint objects

With this approach we may draw (Figure 11 in Dymola framework) all characteristic nodes (green circles) with lines (blue rectangles).

The entire model is non-simulable. Thus we add constraints that express the design needs. Whatever expressed now is definite.

The geometric loop may be simulated to obtain the result in Figure 12.

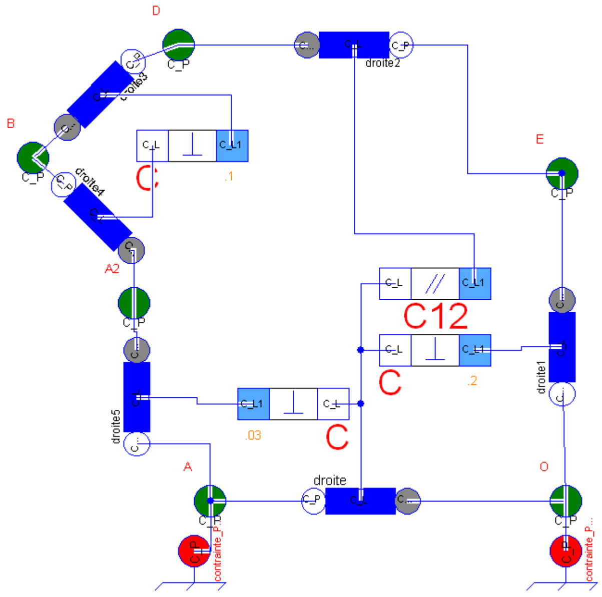


Figure 11 Geometric Constraint Modelling

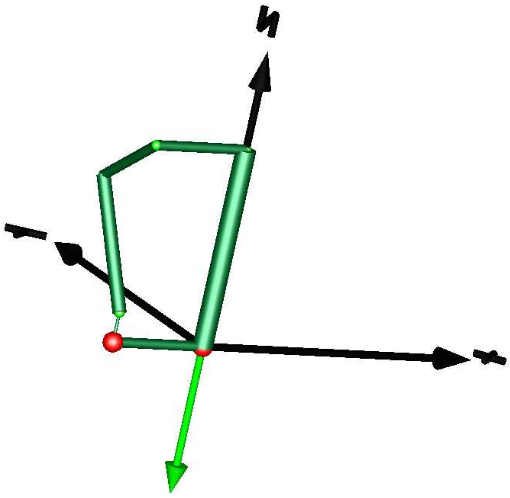


Figure 12 Result of Constraint Modelling

4 Conclusions

We use the interface properties with 3D environment of Modelica/Dymola framework to add design functionalities for geometrical integration.

This modelling by constraints in Modelica is not only used to transform geometrical instances to geometrical variables but also to integrate multi-physics and geometry variables, relying on the requirements data.

It will be developed more deeply in further papers.

References

[1]. **Plateaux, Régis, et al.** *Towards an Integrated Mechatronic Design Process.* Málaga : IEEE, 2009. ISBN 978-1-4244-4195-2.

[2]. **Ferretti G., Magnani G., Rocco P.** Virtual prototyping of mechatronic systems. 2004, Vol. 28, (2), pp. 193–206. doi:10.1016/j.arcontrol.2004.02.002.

[3]. **Modelica Association.** Modeling of Complex Physical Systems. *Modelica.* [Online] http://www.modelica.org/.

[4]. Dymola – DYnamic MOdeling LAboratory with Modelica (Dynasim AB). [Online] http://www.dynasim.com.

[5]. **Hadj-Amor, H.J.** *Contribution au prototypage virtuel de systèmes mécatroniques basé sur une architecture distribuée HLA - Expérimentation sous les environnements OpenModelica-OpenMASK.* Toulon : Supmeca, 2008. LISMMA (EA 2336) thesis.

[6]. **Clément, André, Rivière, Alain and Temmerman, Michel.** *Cotation tridimensionnelle des systèmes mécaniques.* Paris : PYC Edition, 1994.

[7]. **Clément A., Rivière A., Serré P., Valade C.** *The TTRS: 13 Constraint for Dimensionning and Tolerancing ", 5th CIRP Seminar on Computer Aided Tolerancing,.* The University of Toronto, Canada : s.n., April 27-29, 1997.

[8]. **Fritzson, P.** Introduction to Object-Oriented Modeling and Simulation with OpenModelica. *IDA - The Department of Computer and Information Science.* [Online] [Cited: 03 01, 2009.] http://www.ida.liu.se/~pelab/modelica/OpenModelica/Documents/ModelicaTutorialFritzson.pdf.

[9]. **Plateaux, R., et al.** *Méthodologie intégrée de conception d'un produit mécatronique.* Marseille : CFM, 2009. (in reviewing).

Table 3 The 13 constraints

	●	╱	▱
●	$C1 : O_1 = O_2 \rightarrow \{S_{O_1}\}$	$C4 : O_1 \in D_2 \rightarrow \{R_{D_2}\}$	$C3 : \{R_D\}$
	$C2 : O_1 \neq O_2 \rightarrow \{R_{O_1O_2}\}$	$C5 : O_1 \notin D_2 \rightarrow \{E\}$	
╱		$C11: D_1 = D_2 \rightarrow \{C_{D_1}\}$	$C8: D_1 \perp P_2 \rightarrow \{R_{D_1}\}$
		$C12: \left\{\begin{matrix} D_1 \parallel D_2 \\ D_1 \neq D_2 \end{matrix}\right\} \rightarrow \{T_{D_1}\}$	$C9: D_1 \parallel P_2 \rightarrow \{T_{D_1}\}$
		$C13: \left\{\begin{matrix} D_1 \nparallel D_2 \\ D_1 \neq D_2 \end{matrix}\right\} \rightarrow \{E\}$	$C10: D_1 \angle P_2 \rightarrow \{E\}$
▱			$C6: P_1 \parallel P_2 \rightarrow \{G_{P_1}\}$
			$C7: P_1 \nparallel P_2 \rightarrow \{T_{P_1 \cap P_2}\}$

Using Modelica for Interactive Simulations of Technical Systems in a Virtual Reality Environment

Jens Frenkel[1] Christian Schubert[1] Guenter Kunze[1] Kristian Jankov[2]
Dresden University of Technology, Institute of Mobile Machinery and Processing Machines
Muenchner Platz 3, D-01066 Dresden, Germany
CNH Baumaschinen GmbH
Staakener Strasse 53-63, D-13581 Berlin (Spandau), Germany

Abstract

Simulation has become an essential tool in the development of construction machinery. In addition to the validation of technical features, the assessment of man-machine interaction has become more important within complex working environments. In cases where most attention is paid to the human as the operator, simulations have to fulfil special requirements. Allowing the user to interact with the system implies the need for real time simulation as well as flexible hardware integration and a powerful visualisation. Therefore a modular software framework called SARTURIS[3] has been developed meeting all these requirements. In order to support flexible multidomain modelling the Modelica language is being used. This paper presents SARTURIS and its applications, focusing on the integration of Modelica based on OpenModelica using the example of a wheel loader. Since OpenModelica is not yet able to deal with the Modelica Multibody library, a Python-based tool called PyMbs has been developed. It allows comfortable description of multibody systems and export to Modelica code as well as other formats.

Keywords: real time simulation; construction machinery; virtual reality; OpenModelica;

1 Introduction

A main focus of research is to study the impact of the operator on mobile machinery. Due to the low level of automation in such machines, the stress on components during usage heavily depends on the way a machine is operated. Studying this influence provides essential information needed during the design process of such a machine. Obtaining these information from experimental data requires a real prototype and increases demands on cost and time. Furthermore, it is extremely difficult to provide equal conditions for each experiment which makes results hardly comparable. Using simulation instead is a more efficient way of achieving those results without facing the aforementioned problems. It also can be used for studying dangerous manoeuvres without endangering man and machine. Thus, simulation proves to be a valuable tool (see Figure 1).

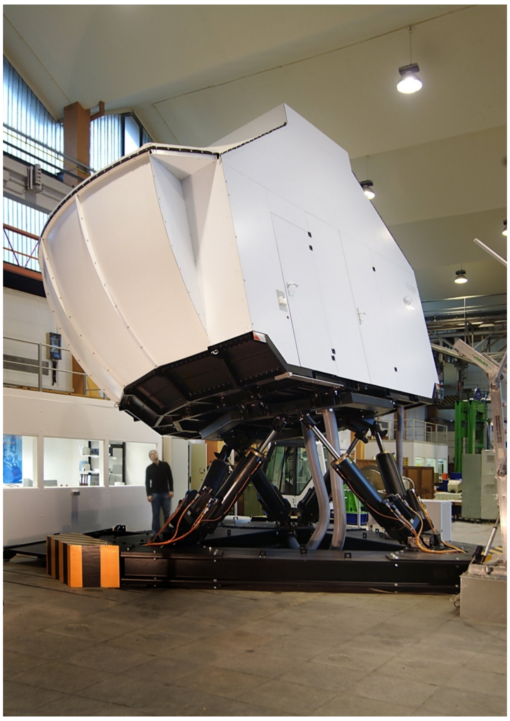


Figure 1: Motion platform

[1] TU Dresden http://tu-dresden.de/bft
[2] CNH Baumaschinen GmbH http://www.cnh.com
[3] SARTURIS BMBF support code: 01ISC24A
. INPROVY BMBF support code: 02PC1110

DOI: 10.3384/ecp09430080

Only if a sophisticated model of the whole machine is implemented, significant results can be obtained. Such models always involve different domains like mechanics, hydraulics and control. The Modelica language is ideally suited for describing these models, since it has been designed to support multi-domain modelling [6]. Moreover, its object-oriented approach allows reuse and substitution of submodels, simplifying the creation of a model [8].

The inclusion of an operator, however, is very challenging. Obtaining a mathematical operator model, which has to be able to react and decide, is virtually impossible. Therefore Virtual Reality (VR) is the only viable option leading to a simulation with a "human in the loop". There are a lot of publications describing the use of VR for analysing the influence of drivers' behaviour, e.g. [1][2][3][4]. Furthermore, the development of the human-machine interface is supported by new methods of VR technologies. The articles point out that the current adoption of these technologies in the industrial sector is rather low [5]. Current simulation systems support the modelling and use of VR technologies only to a minor degree. There is no Modelica tool known to the authors that is specialised in interactive VR simulations.

A tool is needed that not only carries out calculations in real time, but also offers realistic graphics as well as a support for a large variety of input and output devices. The simulation framework SARTURIS, specifically developed at our institute towards interactive VR simulation, meets all the aforementioned requirements including support for Modelica as primary modelling language.

2 SARTURIS

The simulation framework SARTURIS[1] has been developed at Dresden University of Technology in cooperation with industrial partners within a publicly funded (BMBF) research project [10] [11] [12]. SARTURIS allows interactive simulation of technical systems in a virtual reality environment. In order to achieve the best compromise between performance, portability, and development methodology, SARTURIS is based on C++ and uses freely available libraries.

SARTURIS itself is merely a slim application featuring a module loader establishing a framework for individual software components and thereby enabling reusability (Figure 2). Efficient creation of new software components is guaranteed through the use of Model Driven Architecture (MDA). The interaction between these modules along with their parameterisation is specified in XML-files. Each software component has its own XML type definition describing its usage and configuration as well as the interaction with other components. Thus automated syntax checking or even code completion is available. XML files are either written as plain text or assembled using a graphical user interface (GUI).

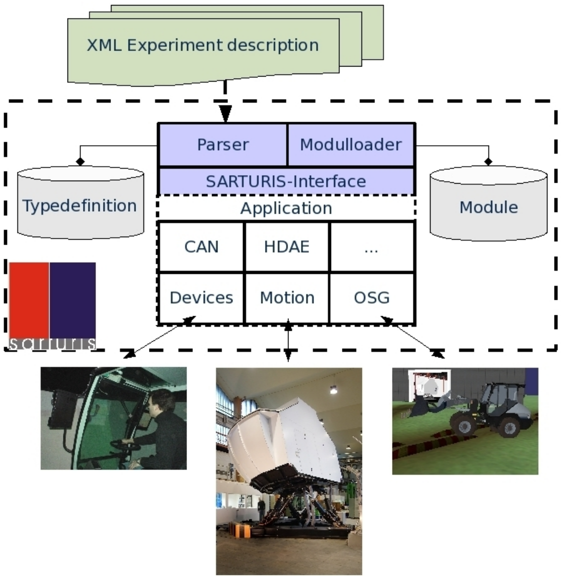


Figure 2: Sarturis Framework

Software components belonging to the same field of functionality are encapsulated within the same module. For instance the module OpenSceneGraph (OSG) [13], see Figure 2, contains all necessary functions to achieve a realistic 3D Visualisation. A comprehensive set of modules has already been created. Graphical User Interfaces can be defined within the XML files by using a module referencing the GIMP-Toolkit (GTK) [14]. In order to integrate miscellaneous input and output devices a module featuring Controller Area Network (CAN) communication has been implemented [15]. Even our motion platform (Figure 1) can be operated via SARTURIS by means of a corresponding module.

To support the development of additional modules several different interfaces have been designed in C++. Every implementation of a technical model inherits from an according interface like *HDAESystem*.

[1] BMBF support code: 01ISC24A

3 Integrating Modelica Models into SARTURIS

3.1 Initial Situation

Every technical system which is to be simulated within SARTURIS forms a module on its own, that contains a class which inherits from the interface *HDAESystem*. Therefore it was necessary to translate the system equations directly into C++ code. Although C++ is extremely powerful as a programming language it is ill-suited for modelling purposes. Hence, the integration of a new model was very tedious and error-prone. Moreover, the resulting code was neither reusable nor maintainable.

The Modelica language on the other hand facilitates convenient modelling of technical systems. The model description is reusable and easily maintainable through its equation-based and object-oriented approach. Furthermore it is very flexible due to the acausal description.

In order to combine the strengths of Modelica and the capabilities of SARTURIS, a transformation of Modelica code into C++ code was needed. OpenModelica [22] proves to be the best solution, since it is able to translate Modelica models into C code. Beyond that, the usage of open-source software is very beneficial to universities since it offers great flexibility and can be used for teaching. Furthermore, every user has the opportunity to get involved in the development of the simulation software through the OpenModelica Consortium.

3.2 OpenModelica Code Export

Before discussing the integration of the OpenModelica Code Export into SARTURIS, a brief introduction on how the OpenModelica Compiler (OMC) translates Modelica code into a simulation shall be given. This process is divided into different stages which are shown in Figure 3. First, the given coherent Modelica model is translated into a flat model where all of the object-oriented structures are removed yielding a system of differential and algebraic equations. Next, this system of equations is analysed and optimised with regards to numerical integration. Consequently the resulting system of equations is passed to a code generator which converts the optimised system of equations into C Code.

The so-called C Code Export yields the following files:

- Text file (*init file*) containing all initial values and information about the system and the solver.
- C source code file (*c model*) containing all equations arranged so that they can be readily used with the supplied solver.
- C source code file (*c model functions*) containing all functions both external and internal which are used in the model.
- precompiled libraries (*sim libs*) containing all model independent functions needed for linking

The *c model* and the *c model functions* are compiled and linked against the *sim libs* using an appropriate C/C++ Compiler resulting in a stand alone programme which runs the simulation and stores the result in a text file. This file contains all the values of the states, their derivations and the algebraic variables that occured during the simulation.

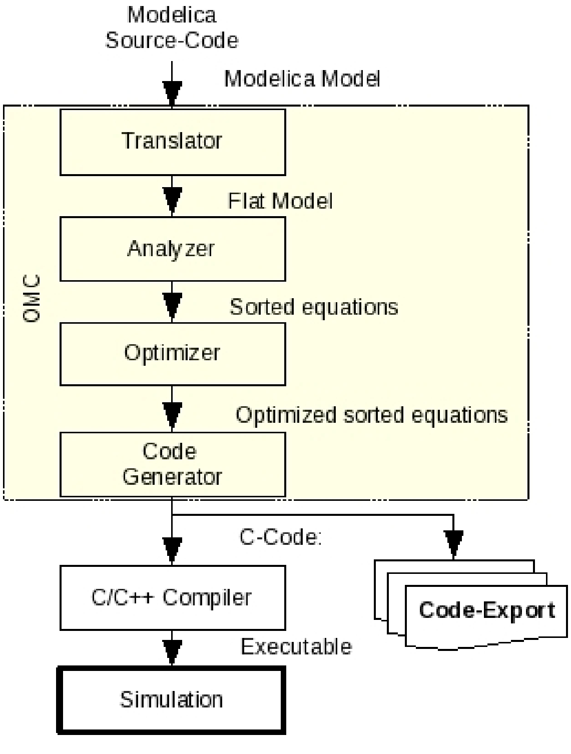


Figure 3: Translation Stages from Modelica Code to a Simulation. According to [9], p. 10

3.3 Integrating the OMC Code Export into SARTURIS

To transfer C code generated by the OpenModelica Compiler automatically into C++ code which can readily be used within SARTURIS as a module, the following two steps have to be carried out.

In the first step information about the model has to be gathered in order to generate the C++ class interface along with its XML type definition as described in section 2. Following information is needed.

1. names of all
   (a) inputs
   (b) outputs
   (c) parameters
   (d) states
   (e) algebraic variables
2. default values of all parameters
3. initial values of all states

The *init file*, as part of the OMC Code Export, provides most of the information except 1.a and 1.b. All information but 2. and 3. is stored within the *c model*. Since there is no single file containing all the information needed, both, the *init file* and the *c model* have to be evaluated.

The second step is to convert the C code generated by the OMC into C++ code implementing a SARTURIS module. A major requirement is to minimise changes within the C code. Ideally, it should be possible to use it without any modifications at all by encapsulating it into a wrapper class. Unfortunately this does not seem to be possible as discussed in the following sections. Furthermore, the system of differential equations and the solver should be separated. Thus results of different solvers can be compared without having to recompile the model. So far the DASSL-solver which comes with OpenModelica and some standard solvers like an Explicit Euler and a fourth order Runge Kutta solver have been implemented. A current student project is dealing with the integration of the SUNDIALS package [16].

3.4 Automation

A tool called OpenModelicaToSarturis (OM2S), see (Figure 4), has been developed which automates the procedure outlined in the previous subsection. It enables the user to generate a SARTURIS module without writing a single line of C++ code. Thus, models developed in Modelica can be easily used within SARTURIS for interactive VR simulations.

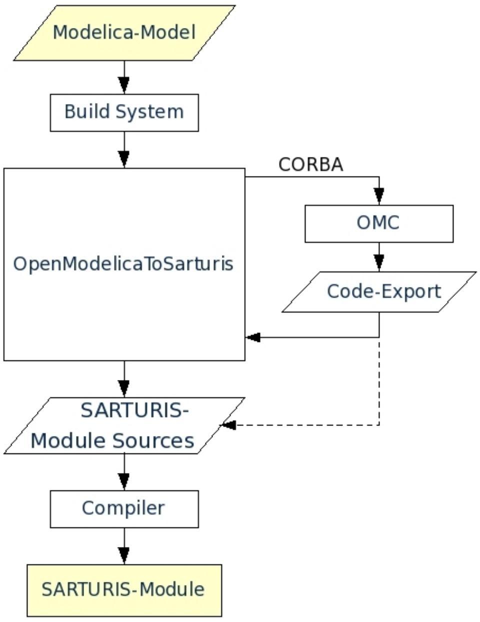


Figure 4: OpenModelicaToSarturis: Automated Translation of Modelica Models into a SARTURIS Module

The whole compilation process from a Modelica model to a SARTURIS module is coordinated by CMake [17]. Thereby, custom build rules can be defined easily and it is possible to detect utility programmes, libraries and include directories in a platform neutral manner. It generates makefiles and workspaces which can be used with any supported compiler. At the beginning of the compilation process OM2S is started which then launches the OMC. Communication between OM2S and OMC is achieved via CORBA, offering a convenient interface to trigger the translation of a Modelica model. Subsequently, OM2S turns the C code into C++ code implementing a SARTURIS module. Simultaneously, a sample SARTURIS configuration is generated featuring a diagram for each state as well as sliders for each input value. After the compilation process is finished, SARTURIS can be launched with the sample configuration [18]. It allows validating the results immediately and it may also be used as a template for more complex settings.

3.5 Difficulties And Suggested Solutions

This section describes the difficulties encountered during the integration of the OMC Code Export into SARTURIS and the measures taken to overcome them. In addition, suggestions to improve the OMC Code Export regarding usability are given.

3.5.1 Gathering System Information

Gathering system information needed for the class interface as well as the XML type definitions, requires a parsing of the *init file* and the *c model*. Parsing the *init file* can be achieved using internal functions of the *sim libs* which are called at every start of a simulation run. Analysing the C code however proves to be much more challenging. The current implementation reads the C code line by line looking for unique keywords. All names of the state variables, for instance, are stored within the static array *state_names* which always has the form of *char* state_names[2]={"h", "v"};*. Once such a line is identified, all relevant information is extracted.

Clearly, a change in the formatting of the source code will inevitably leads to an abstraction of wrong information or none at all. Thus, parsing generated C code in order to gather information about the system should generally be avoided. One feasible solution is to extend the *init file* by the names of the inputs and outputs. Thus all information could be extracted from a single text document. Furthermore it allows changing the values of inputs which are assumed to be zero otherwise. One might also consider to change the formatting of the *init file* into a standardised format like XML. This would enable checking the syntax against a language definition and use readily available parsers to extract all the information.

Another possible solution is to extend the CORBA interface by single or multiple commands that return all system information. Although this is very elegant from a programmer's point of view, it limits the possibilities of usage. It would not be possible to gather system information, if only the code export but no OMC was available.

3.5.2 Using the C Code

Encapsulating the system of differential equations into its own class poses a problem since the C code generated by the OMC makes use of global variables. Namely the structure *DATA*, which contains the values of all the states, algebraic variables and parameters is always referenced via a global variable called *localData*. The usage of global variables, however, has to be avoided when dealing with classes since it causes unwanted interference between different instances of the same class. Consequently, every function has to be converted into a function of the class. It can be achieved by adding a prefix, consisting of the name of the class, to every function definition.

Again, parsing and changing the provided C code is not an elegant solution since changes to the C code might cause this method to fail. In order to avoid altering the C code and to allow the use of a wrapper class, all global variables should be eliminated. If a subroutine needs access to *localData* it should provide a pointer to this structure in its function definition such that the caller is able to pass the structure. The interface of the DASSRT solver for example also features two pointers, namely *rpar* und *ipar*, see Figure 5. They can be used to pass lists of real and integer parameters to DASSRT. Beside parameters like start time and stop time, the DASSRT interface expects multiple pointers to user functions. These user functions perform the calculation of the residuals, Jacobian matrix or constraints, respectively. All these user functions are called by DASSRT and need access to the information stored in *localData*. The parameter *ipar* could be exploited passing the address to *localData* via a static typecast from *long** to *DATA** and back, see Figure 6. Since both types are pointers no conflicts regarding the size of the variable have to be expected.

```
void  DDASRT(
 int (*res) (..., double *rpar, long* ipar),
 ...
 double *rpar,
 long *ipar,
 int (*jac) (..., double *rpar, long* ipar),
 int (*g) (..., double *rpar, long* ipar),
  ...);
```

Figure 5: Interface Solver DDASRT

Thus it is possible to change all global variables into local ones and thereby to increase the usability of the C code.

```
...
static DATA* localData;
...
int functionDAE_res(..., long int* ipar)
{
  ...

  return 0;
}
```

```
...
int functionDAE_res(..., long int* ipar)
{
  ...
  DATA* localData;
  localData=(DATA*)ipar;
  ...

  return 0;
}
```

Figure 6: local DATA* vs. global DATA*

3.5.3 Implementation

In order to prove feasibility of the suggested solutions, the following changes have been implemented into a local copy of the OMC source code:

1. Extending the *init file* by the names of inputs and outputs

2. Turning localData into a local variable

It has been proved that these changes allow a much more convenient subsequent use of the source code generated by the OMC.

4 PyMbs

A major drawback connected with the usage of OpenModelica is the missing support for the Modelica.Mechanics.Multibody library. Since multibody systems form an essential part in the study of mobile machinery, a tool called PyMbs written in Python has been created at Dresden University of Technology. Using sympy [21], a library for symbolic mathematics, PyMbs generates the equations of motion of arbitrary holonomic multibody systems having the standard form

$$\dot{p} = v$$

$$M\dot{v} + h = f + \left(\frac{d\Phi}{dp}\right)^T \lambda$$

$$\Phi(p) = 0$$

where p is the vector of generalised positions, v the vector of generalised velocities, λ the vector of constraint forces or Lagrange multipliers respectively, M represents the system mass matrix, h is the vector of the gyroscopic and centrifugal forces, f is the vector of all external and elastic forces and $\mathbf{\Phi}$ contains all holonomic constraints. PyMbs is able to export this system of equations as Modelica code which can then be used within a Modelica model and simulated using OpenModelica. It can also be exported as a MATLAB or a Python file for the use with standard solvers. In order to avoid extremely long equations when calculating the mass matrix M or the vector h explicitly, a recursive formulation [7] exploiting the structure of the multibody system has been implemented. Arising kinematic loops may either be closed by introducing kinematic constraints or using predefined and precalculated kinematic loop objects describing the relation between dependent and independent coordinates. This choice either leads to a DAE or ODE formulation, respectively.

Figure 8 shows an exemplary implementation of a model of a crane crab and a load (see Figure 7). The crab may move horizontally in one axis and the load may rotate around the crab. A force is applied to the crab modelling the effect of a drive. Figure 9 shows the Modelica code, automatically generated by PyMbs. Note, that the model is defined as *partial* since there is no equation defining the magnitude of the driving force. In order to equip PyMbs models with connectors from the Modelica Standard library a new model inheriting all equations from the PyMbs model should be created. Inside the new model mechanical connectors can be instantiated and associated with the corresponding variables as shown in Figure 10.

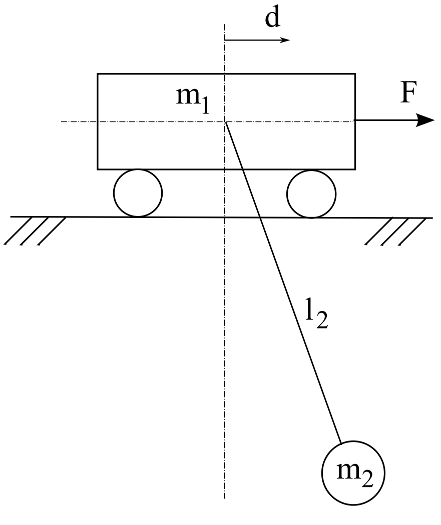


Figure 7: Crane Crab

```
from PyMbs.Input import *

# Set up a new MbsSystem
world=MbsSystem([0,0,-1])

# Define Input and Parameters
F = world.addInput('Force', 'F')
m1 = world.addParam('mass␣1', 'm1', 10)
m2 = world.addParam('mass␣2', 'm2', 1)
l2 = world.addParam('length', 'l2', 1)
I2 = world.addParam('inertia␣2', 'I2', 1/12)

# Define Bodies and Coordinate Systems
crab = world.addBody(name='Crab', mass=m1)
load = world.addBody(name='Load', mass=m2,
                     inertia=diag([0,I2,0]))
load.addCoordSys('joint', p=[l2,0,0])

# Connect Bodies Through Joints
world.addJoint('TransCrab', world, crab, 'Tx',
               startVals=1)
world.addJoint('RotLoad', crab, load.joint, 'Ry')

# Add Sensors and Force Elements
world.addLoad('DrivingForce', 'PtPForce',
              crab, world, F)
world.addSensor('Position', 'Distance',
                crab, world, 'd')

# Calculate Equations of Motion and Generate Code
genEquations(world, explicit=True)
genCode('mo', 'CraneCrab_PyMbs')
```

Figure 8: PyMbs Source Code of a Crane Crab

```
// This file was generated by PyMbs
partial model CraneCrab_PyMbs
        // Positions
        Real[2] q (start={1,0})
               "q_TransCrab,q_RotLoad";
        // Velocities
        Real[2] qd (start={0,0})
               "qd_TransCrab,qd_RotLoad";
// Inputs
        Real F;
// Parameters
        parameter Real I2 = 0.083 "inertia_2";
        parameter Real g  = 9.81  "gravity";
        parameter Real l2 = 1     "length";
        parameter Real m2 = 1     "mass_2";
        parameter Real m1 = 10    "mass_1";
// Sensors
        Real[2] d;
// Variables
protected
        Real[2]   WF_DrivingForce;
        Real[2,2] M;
        Real[2]   h;
        Real[2]   f_gravity;
        Real[2]   f_ext;
        Real[2]   f;
equation
        der(q) = qd;

        d = {abs(q[1]),q[1]*qd[1]/abs(q[1])};

        WF_DrivingForce = {q[1]/abs(q[1]),0};

        M = {{m1+m2,l2*m2*sin(q[2])},
             {l2*m2*sin(q[2]),I2+m2*l2^2}};

        h = {l2*m2*qd[2]^2*cos(q[2]),0};

        f_gravity = {0,-g*l2*m2*cos(q[2])};

        f_ext = F*WF_DrivingForce;

        f = f_ext+f_gravity;

        M*der(qd) = f - h;

end CraneCrab_PyMbs;
```

Figure 9: PyMbs Modelica Output

With only few enhancements to the model of the crane crab, PyMbs is able to generate an interactive graphical output(see Figure 11). It enables the user to check the consistency of the model by manipulating the generalised coordinates via sliders. The effect on the multibody system can be evaluated ad hoc.

In case the available collection of joints, force elements and sensors do not suffice, PyMbs can be extended very easily due to its object oriented structure. Moreover, it takes only very little effort to implement further output formats.

PyMbs is freely available. For further information please contact one of the authors.

```
model CraneCrab
  extends CraneCrab_PyMbs;
  import Modelica.Mechanics.Translational.*;
  // Mechanical Connector
  Interfaces.Flange_b flange;
equation
  flange.s = d[1];
  flange.f = F;
end CraneCrab;
```

Figure 10: Usage of PyMbs Output in Modelica

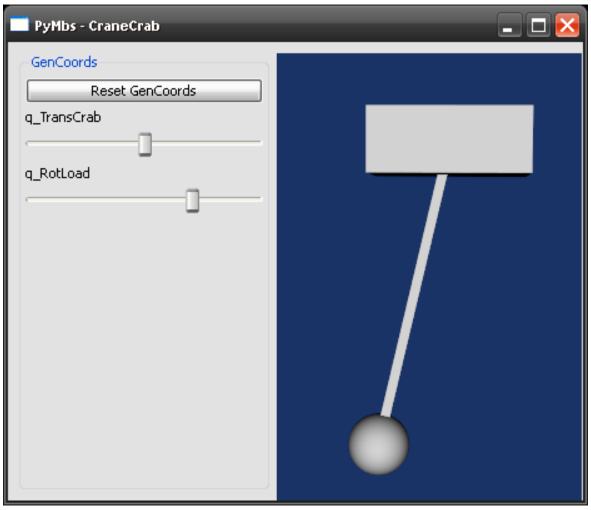


Figure 11: Graphical PyMbs Output of the Crane Crab

5 Example Models

Based on the described tool chain, several models have already been implemented. In this paper, a wheel loader shall be presented (Figure 12 and 13). The purpose of the wheel loader model is the assessment of innovative operational controls and novel assistance systems. A realistic driving experience is achieved by connecting the model via SARTURIS to a motion platform (Figure 1). In order to provide the user with his familiar operating environment, the manufacturer provided a real driving cab, which has been installed onto the motion platform and is integrated via CAN Bus (Figure 14).

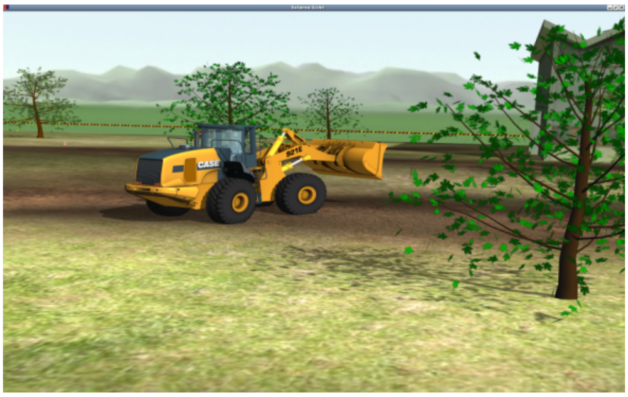


Figure 12: Virtual Reality Simulation of a Wheel Loader (Exterior View)

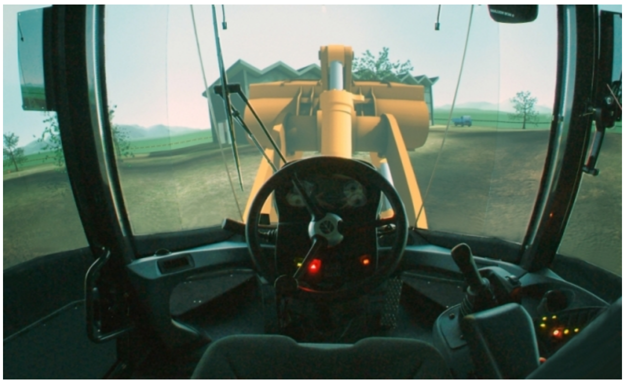


Figure 13: Virtual Reality Simulation of a Wheel Loader (Operator View)

The mechanics of this model have been described using PyMbs, exported to Modelica and equipped with mechanical connectors. Hydraulics, drivetrain as well as control systems and a tire model [23] have been modelled directly within Modelica. This model was then translated into a SARTURIS module and integrated into the simulation environment. It is now possible to run a model of the wheel loader on the motion platform which is equipped with the control units (pedals, joystick, steering wheel...) from the real machine. They allow interaction with the simulation through an operator in real time.

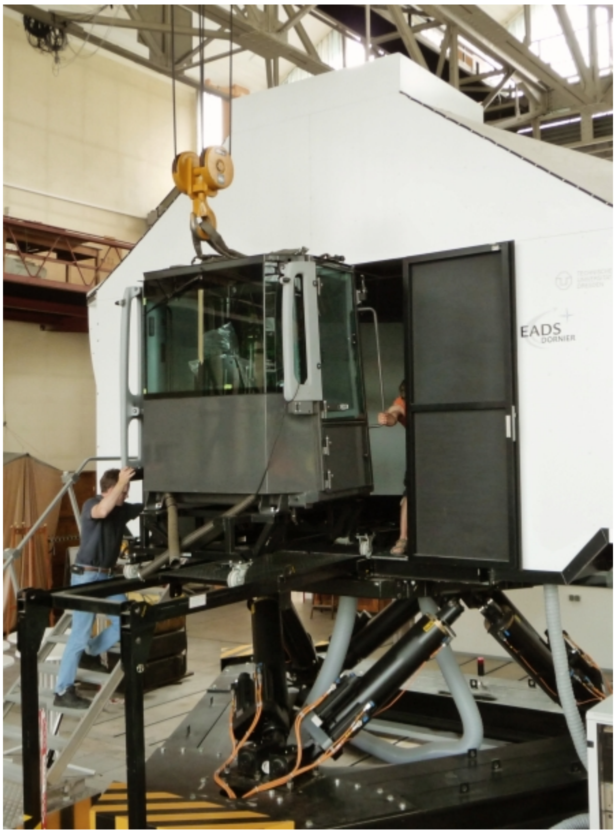


Figure 14: Changing the Cabin of the Motion Platform

6 Future Work

Increasing pressure on costs and time foster a need for a more efficient design process. Prior research has facilitated simulation processes that enable companies to shorten the design process by using virtual prototypes. However, these methods are hardly used in the branch of mobile machinery. Machines comprise numerous components manufactured by different companies. Thus, successful simulation requires a cooperation of the manufacturer and his suppliers. Due to the risks connected with transferring crucial information a cooperative simulation process has not yet been established in industry. INPROVY[2] [20], a research project coordinated by the Dresden University of Technology, aims at overcoming those obstacles by providing methods that allow simulations across company borders. Furthermore, our research focuses on the reuse of available information and its administration as well as its protection.

[2] BMBF support code: 02PC1110

7 Conclusion

It was shown that it is possible to conduct real time simulations in a virtual reality environment with Modelica by using our simulation framework SARTURIS. A tool called OpenModelicaToSarturis has been developed which automatically converts Modelica models into SARTURIS modules using OpenModelica. The lacking support of the Modelica.Mechanics.Multibody library by OpenModelica has been overcome by using PyMbs. PyMbs is a tool, developed in Python, which allows modelling of holonomic multibody systems. It offers different output formats like Modelica, MATLAB and Python code. The presented methods and tools are very beneficial to support the modelling and use of interactive VR technologies.

Dresden University is very interested in cooperation with other universities which might want to use SARTURIS for their research.

References

[1] Koo, T. Y.; Bae, C. H.; Kim, B. Y.; Rowland, Z.; Suh, M. W.: Development of a driving simulator for telematics human-machine interface studies.-Proceedings of the Institution of Mechanical Engineers, Part D (Journal of Automobile Engineering) * Band 222 (2008) Heft 11

[2] Zschocke, A. K.; Albers, A.: A method to examine links between subjective and objective evaluations of steering torque utilising a model-based approach.-FISITA, World Automotive Congress, 32 * (2008)

[3] Pasetto, M.; Gamberini, L.; Manganaro, A.: Potential of immersive virtual reality models in studies of drivers' behaviour and interventions to improve road safety.-PRESENCE, Annual International Workshop on Presence, 11 * (2008)

[4] VTT Technical Research Centre of Finland: HumanICT - New Human-Centred Design Method and Virtual Environments in the Design of Vehicular Working Machine Interfaces VTT Working Papers.-ISBN-Nr.: 978-951-38-6625-9

[5] Strassburger, S; Schulze, T.; Fujimoto, R.: Future trends in distributed simulation and distributed virtual environments.-WSC, Winter Simulation Conference, 40 * (2008)

[6] Beater, P.; Otter, M.: Multi-Domain Simulation: Mechanics and Hydraulics of an Excavator. In: Proceedings of Modelica 2003 conference, 2003

[7] Fisette, P.; Samin, J. C.: Symbolic generation of large multibody system dynamic equations using a new semi-explicit Newton/Euler recursive scheme. Archive of Applied Mechanics, Vol. 66, Issue 3, pp. 187-199 (1996)

[8] Fritzson, P.: Principles of Object-Oriented Modeling and Simulation with Modelica2.1.Wiley-IEEE Press, 1 2004.

[9] Fritzson, P.; et al.: OpenModelica System Documentation.www.openmodelica.org, 1 2008.

[10] Penndorf, T.; Kunze, G.: Codegenerator fuer die Echtzeitsimulation von Mehrkoerpersystemen.-ASIM 2006 19. Symposium Simulationstechnik, Universitaet Hannover, September 2006

[11] Penndorf, T.: Universelles Framework zur Abbildung von Maschinenmodellen in virtuellen Umgebungen. In: Schriftenreihe der Forschungsvereinigung Bau- und Baustoffmaschinen (2006) 34

[12] Penndorf, T.; Kunze, G.: "Durchgespielt"-Interaktive Simulation von Baumaschinen. IX-MAGAZIN FUER PROFESSIONELLE INFORMATIONSTECHNIK Heft 08/2007.-Heise Zeitschriften Verlag, Hannover

[13] OpenSceneGraph, http://www.openscenegraph.org/

[14] GTK, http://www.gtk.org/

[15] CAN in Automation e. V.. http://www.can-cia.org

[16] SUNDIALS (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) http://www.llnl.gov/CASC/sundials/

[17] CMake, http://www.cmake.org/

[18] Frenkel, Jens: Integration von OpenModelica in das Programmsystem SARTURIS.-TU

Dresden, Professur fuer Baumaschinen- und Foerdertechnik, Diplomarbeit,Februar 2009

[19] Schubert, C.; Frenkel, J.: PyMbs Userguide.- TU Dresden, Professur fuer Baumaschinen- und Foerdertechnik, Forschungsbericht, September 2009

[20] www.inprovy.de

[21] sympy, http://code.google.com/p/sympy/

[22] OpenModelica, http://www.openmodelica.org/

[23] Zimmer, Dirk and Otter, Martin(2009)'Real-time models for wheels and tyres in an object-oriented modelling framework',Vehicle System Dynamics,99999:1,

Interactive Simulations and advanced Visualization with Modelica

Tobias Bellmann
Institute of Robotics and Mechatronics, German Aerospace Center (DLR)
Münchner Straße 20, 82234 Weßling, Telefon: +49 8153 28-0, Fax: +49 8153 28-2243

Abstract

In this paper a Modelica library for interactive simulation and advanced visualization called ExternalDevices is introduced and presented. Providing support for standard input devices like keyboard and joystick as well as for communication via UDP and shared memory, this library allows the user to interact with a running simulation and process the output data of the simulation in other processes capable of UDP connections. An advanced visualization system replaces the standard Dymola visualization and offers additional features like full-screen viewing, transparency and support for flexible bodies.

Keywords: interactive simulation; visualization; simulation; network; flexible bodies

Introduction

Simulations with Modelica normally are not designed for interactive control. In the standard Modelica 3.1 library, no blocks for input devices or other control possibilities are existent. Nevertheless it can be help- and useful to interact with a running multi-physics simulation, either to reduce the effort needed for the generation of input data for the simulation, or to react directly to the results of a running simulation.

The integrated visualization of the Modelica MultiBody Library is vendor-specific. It is therefore limited to the specified visualization methods provided by the simulation tool. The visualization definitions of Modelica 3.1 are limited to some basic features like some elementary shapes and untextured .dxf CAD files.

To overcome the missing interaction and visualization possibilities, the *ExternalDevices* Library provides a set of blocks and techniques to allow interactive simulations, as well as an advanced real-time visualization of the running simulation, considerably extending the scope of operation especially for multi-body simulations. The *ExternalDevices* library is structured in the following functional packages:

- Input devices: Blocks for the direct control of simulation states by the user
- Communication devices: Blocks allowing the simulation to communicate with other processes via network or shared memory
- External visualization: Blocks and models replacing vendor-specific visualization systems and adding additional visualization possibilities.

The *ExternalDevices* library links either to static or dynamic C++ libraries to provide this additional functionalities. It is available for Dymola 7.x, in a version for Windows, a version for Unix/Linux is planned. Every Modelica implementation able to link external C libraries can use the *ExternalDevices* library and the visualization system.

1 Input devices

Input devices are needed for interactive control of the simulation states, for example to trigger events or to control actors of a multi-body simulation. For this purpose the ExternalDevices library provides blocks for three common PC input devices:

Keyboard: This block allows the monitoring of single keyboard keys, and has a boolean output for the chosen key state. Several blocks can be used parallel, each with a selectable key ID like VK_Return for the return key.

DOI: 10.3384/ecp09430056

Figure 1: Input devices blocks

Joystick: This block includes support for three axis, eight buttons joysticks for Modelica. The joysticks must be configured and calibrated correctly in the Windows system control panel. Several blocks at the same time are usable with separate joystick IDs, allowing the use of more than one joystick attached to the PC. The three outputs are from the type Real and are normed from -1 to 1 for each joystick axis.

SpaceMouse: The SpaceMouse block includes support for the 3dConnexion Spacemouse, a six DOF input device originally developed at the Institute of Robotics and Mechatronics of the german aerospace center (DLR). This devices consists of a pressure-sensible handle, which can be pushed and rotated to manipulate objects in three-dimensional space. The output connectors of this block also are normed from -1 to 1 and all buttons of the Spacemouse are retrievable via a boolean output vector.

2 Communication devices

It is often useful to control a simulation via network or to process the simulation output data in another program or simulation. The *ExternalDevices* library supplies the user with blocks to communicate with external processes via UDP or Shared Memory. As an example, one simulation can provide input data for another simulation via UDP.

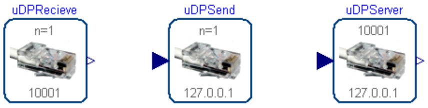


Figure 2: Network devices blocks

UDPRecieve block: This block introduces an UDP client communication input for Modelica. The incoming data must be binary coded double vectors, in the form [double 1,.....,double n] with the selectable length n. The listening port can be selected and must be unique on the system. More than one block in a model may be used under this premise.

UDPSend block: This block allows the sending of Real vectors with selectable length n. Parameters are the target port and IP address. More than one block may be used in a model.

UDPServer block: Combination of UDPSend and UDPRecieve functionality.

Shared memory Block: A shared memory block, using the QT framework from QT Software [1] allows the communication between two processes on the same computer system. Supporting Real, Integer and Boolean vectors it can be used as an interface to other parallel running simulations or processes with the same QT shared memory interface. Several blocks can be used within the same model by defining different memory storage IDs.

3 A model-based approach for visualization

Including the complete Modelica 3.0 standard for visualization of multi-body models, this library furthermore allows the user to build more complex visual environments within the Modelica model, to be simulated on an external visualization tool, DLR SimVis. The visualization package uses network communication to transmit the visualization data from the simulation to the external visualization tool. This tool, DLR SimVis, provided together with this library is based on OpenGL [4] and the OpenSceneGraph [2] 3D scenegraph.
The main purpose is the visualization of multi-body simulations, replacing vendor-specifig graphic engines with a presentable graphic engine, supporting full-screen viewing and a large variety of textured 3D CAD file formats.
Every object visualized in the external viewer is provided by a visualization component in Modelica, containing all necessary data for correct visual representation.

3.1 Object-oriented approach for visualization components

The object-oriented approach of the *ExternalDevices* visualization system easily allows to integrate the visualization components in to physical components. The complete informations necessary for the visualization are already existent in the physical component and can be used in the visualization block. Figure 3 shows the integration of a visualization block in the model of a spring. The information about the spring's position, orientation and length is available via the frame connectors, other parameters like winding number, wire diameter e.g. can be chosen via parameter dialog.

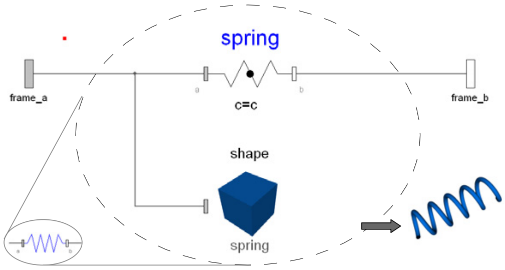


Figure 3: Integration of the visualization component in a physical model

3.2 Comparison with existing physical visualization systems

Most existing visualization systems rely on a central configuration file defining the scene and the input channels for a specific visualization task. The MathWorks VR Toolbox [5] for example uses VRML Files, containing the complete kinematic and degrees of freedom for scene manipulations as well as the CAD Data. In the corresponding Simulink model, the VR Block provides the inputs for the desired degrees of freedom in the scene. These inputs are now connected with the according signals of the Simulink physics model. This leads to increased configuration efforts if the model has to be changed, because both the VRML Configuration and the signals have to be adapted. Another example for the separation of model and visualization system is the VisEngine of Aerolabs GmbH [6]. The VisEngine uses a configuration file defining the used CAD data for the scene and the input channels (e.g. UDP, tables from the file system, etc.) for moving the CAD objects.

In both cases, a change in the model requires a modification in the visualization. With the object-oriented approach of *ExternalDevices* combined with Modelica, this is not necessary, because of the complete integration of the visualization into the model components.
Because of an additional software layer, hidden from the user, there is no need for complicated additional signal connections. The visualization data is collected in a data core controlled by an *ExternalObject* construction and transmitted to the external visualization viewer (see Figure 4).

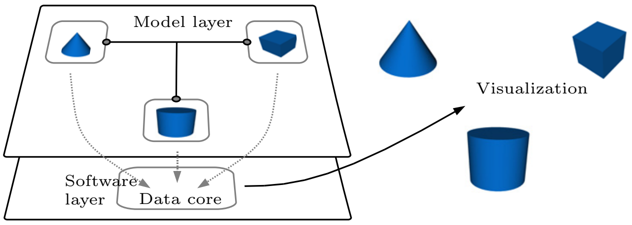


Figure 4: Model layer and software layer with data administration (hidden)

3.3 Modified Modelica 3.0 standard library

One strength of Dymola's visualization system is the automatic generation of the scene via the visualization properties of every multibody system. This is done via implementing a visualization definition in every part of the Multi-body library. These definitions all inherit the *Modelica.Mechanics.MultiBody.Visualizers-.Advanced.Shape* block, which is the connection to the Dymola visualization. By replacing this block with a modified variant, the complete model visualization is redirected to the external visualization. This modification allows to use every existing model with the external Visualization.
Since Modelica 3.1 the vendor-specific library elements like the Shape block are grouped in an additional service library *ModelicaServices*. Analogue to the modified Modelica library, it is possible by providing a customized *ModelicaServices* implementation to redirect the visualization data to the external viewer DLR SimVis.

3.4 External viewer software DLR SimVis

The visualization data is sent from the simulation process via network to the external viewer software SimVis. This software is responsible for the interpretation of the visualization data and for the rendering of the scene. Because of the use of a network communication, the Viewer software does not have to run on the same system as the simulation, and therefore more computing power for the simulation can be provided. Based on the open source scene graph OpenSceneGraph [2] a wide range of formats for CAD data is supported. The continuing development of OpenSceneGraph provides the base for highly detailed visualizations. Utilizing the ffmpeg video en/decoding library [3], several video codecs for video grabbing are supported. The following features already are implemented:

- Fullscreen mode
- Support for multiple cameras
- Multi-monitor support
- Textured CAD files (.dxf, .stl, .3ds, .obj, ...)
- Video grabbing, formats: (MPEG4, MS MPEG4 2/3 (.avi), Flash video (.flv), Huffman Encoding (lossless), Windows Media Video (.wmv))
- Video grabbing with free configurable bit rate and replay speed
- Wireframe mode, Stereo mode (anaglyph and indirectly by OpenGL graphics card drivers)
- Precise replay controls including a jog-dial

3.5 Object-oriented network protocol

In order to reduce the amount of visualization data to be transported over the network connection, an optimized protocol is necessary. In the implementation of the *ExternalDevices* library, an object-oriented approach with data and packet objects has been chosen. While the data objects handle the data storage and the tasks of serialization / deserialization, the packet objects are responsible for data object administration. This includes an incremental packaging of data, where only data objects, which changed in the last time step are included in the transmitted data. A lossless transport protocol provided, like TCP/IP, this method reduces the needed communication bandwidth significantly, because static properties of visualization elements must not be communicated every time step.
Every packet is identified and assigned to a visualization element by an unique, automatically generated ID. Figure 5 provides an overview of the network communication architecture.

4 Visualization package content

The visualization package is structured in subpackages for Shape blocks, Camera blocks, Light blocks, Energy Flow blocks and Effect blocks.

4.1 The *UpdateVisualization* block

Vis
animate=true

The *UpdateVisualization* block is responsible for the integration of the visualization blocks in the simulation process. Similar to the *MultiBody.World* block, it will be automatically inserted as an *inner* component if a visualization block is used in the model. The *UpdateVisualization* block controls parameters like the IP address and port for the communication with the visualization software SimVis and can be used to disable the complete visualization.

During a simulation run, the block triggers a time event every time visualization data shall be sent to SimVis. This update interval time can be varied via a parameter and should be < 0.04 s for 25 frames per second during real-time simulations. If the update time event is triggered, every block sends its data to the data core triggered by the Boolean variable *UpdateVisualization.send.*

4.2 Shape blocks

Every Shape block has a frame connector as input. The resulting forces and torques of such a block is zero, so the block only has visual and no dynamic or kinematic effects. This blocks can be used as

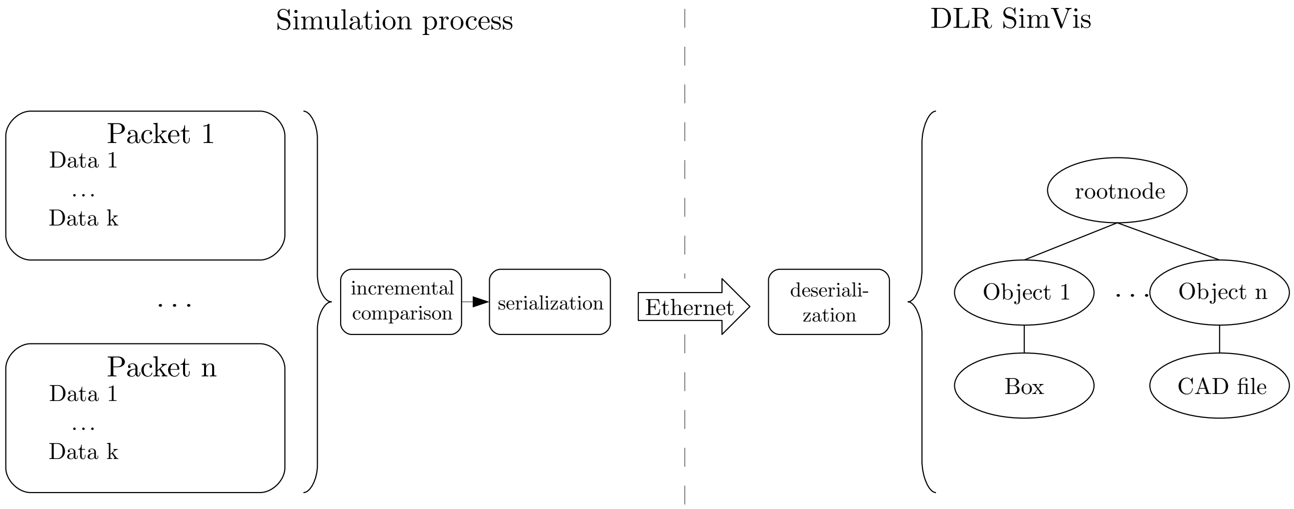


Figure 5: Basic principle of visualization network architecture

integrated components in a Modelica multi-body model:

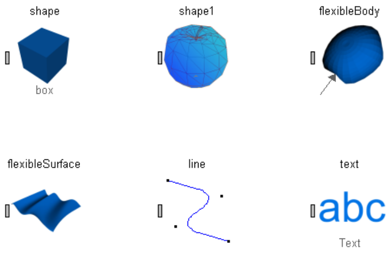


Figure 6: Shape blocks

Elementary Shape: This block represents the supported basic shapes, such as boxes, spheres etc. The available primitives are:

Basic shape type	Existent in Modelica 3.0	Existent in library
Box	Yes	Yes
Sphere	Yes	Yes
Cone	Yes	Yes + Features
Spring	Yes	Yes
Cylinder	Yes	Yes
Pipe	Yes	Yes
Beam	Yes	Yes
Gearwheel	Yes	Yes + Features
Coordinate System	Yes	Yes
Grid	No	Yes

Every *ElementaryShape* can be parametrized in size, color, transparency and reflection behavior (for specular highlights). For the more complex shapes like spring, gearwheel and cone, additional parameters can be set. Beyond the standard parameters, required by the Modelica 3.1 standard, some additional parametrization is possible, for example the operating angle of gearwheels allows the construction of bevel gears.

FileShape: The *FileShape* block allows to use 3D CAD models, supporting numerous file formats like .obj, .dxf, .3ds, .stl, and every other file formats supported by the OpenSceneGraph plug-in system. The key features of this block are support for textured 3D models and additional parameters like transparency and a wireframe modus. The loaded 3D model can be scaled in x,y,z directions.

Line The *Line* block implements linear or Bezier interpolated lines, with n control points relative to the frame connector of the block.

Text Shape: This block allows to place 2D texts in the scene, aligned to a specified direction or to the screen. Font, character size and color are parameterizable.

Text Shape with Value: In addition to the *TextShape*, this block has a Real input connector. The text in the visualization is followed by the input value of this connector, allowing the display of simulation data in the visualization.

4.3 Visualization of flexible bodies

Especially for the use in the DLR *FlexibleBodies* library, a visualization module for flexible bodies is available in the library. For topologically simple objects (e.g. beams or tori) a parameterizable surface can be used, visualizing an array of points (see Figure 7)

For more complex models, an spatial interpolating algorithm is implemented. This algorithm allows to deform CAD models visually according to a displacement set of n control points ($n << n_{CAD}$) and interpolates the CAD data points spatially between the control points. With this algorithm the number of points to be communicated to the visualization system can be reduced drastically, as only the displacement set has to be calculated and submitted to the visualization (see Figure 8).

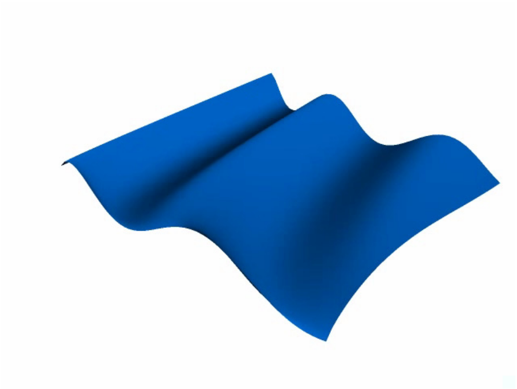


Figure 7: quadratic, parameterizable surface

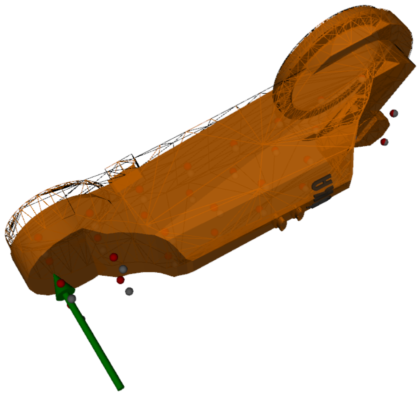


Figure 8: Deformed(wireframe) component of an industrial robot

4.4 Energy flow visualization

The visualization package supports the visualization of energy flows with several blocks (see Figure 9). The energy flow is represented visually by a transparent pipe (can be deactivated) with moving arrows inside. The spatial configuration of the pipe is specified within the model by parameterizing the single components of the pipe (MultiBody Library compatible) with informations like length, diameter, radius of curved segments etc. The basic flow elements available are *StraightPipe*, *CurvedPipe* and *FlexiblePipe* (flexible interpolated). For visualization purposes, the color, size and speed of the arrows can be dynamically changed during the simulation.

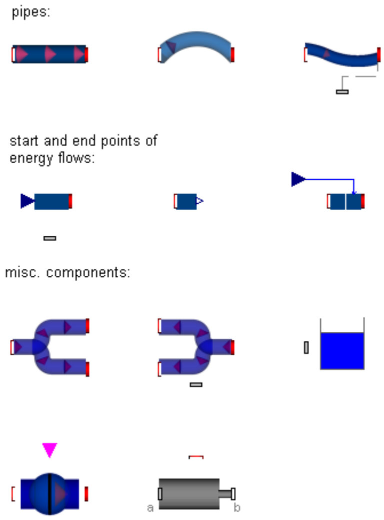


Figure 9: Energy flow blocks

Every energy flow pipe has to begin with a *FlowBegin block* and ends with an *FlowEnd* block. The start position of a pipe is defined by a MultiBody frame connector, whereas the flow speed of the pipe indicators can be set by an Real input of the FlowBegin block. The interconnection between the pipe segments is handled by a special connector containing the connecting frame of the segment, the flow through the pipe and an ID of both connected pipe segments. The IDs are necessary to provide information about the assembly of the energy flow system for the visualization software as a double-linked list.

In order to avoid kinematic loops, the frame information of the pipe segments is encapsulated

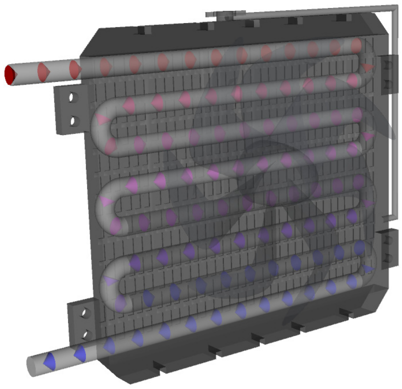


Figure 10: Car Radiator with coolant flow

in the connector and should not be accessed directly. If a connection between two defined points is desired, the *FlexiblePipe* block can be used to exactly construct a pipe connecting these two points. This is done by a Bezier interpolation algorithm and can be updated during the simulation to visualize a flexible connection between two moving points.

4.5 HUD Elements:

This objects support the generation of simple head-up-displays, allowing the placement of text, bar graphs and bitmaps. This elements can be combined to form e.g. analogue instruments (see Figure 11) or digital gauges. The displayed values are updated during the simulation and allow a direct insight into the connected states of the simulation. The head-up-display is placed as a 2D overlay over the 3D scene.

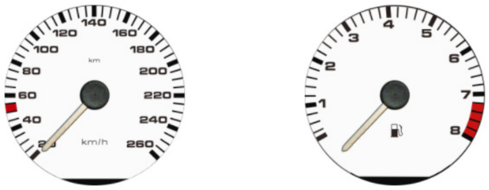


Figure 11: Several bitmaps combined to an analogue tachometer and rpm gauge

4.6 Cameras

The visualization system supports multiple camera views. If there is no dedicated camera in the model, a standard view will be used. For every new camera block in the model, an additional subwindow is available in the visualization viewer, showing the scene from the cameras perspective. Every camera can have its own background color, viewing distance and field of view. A full screen mode allows to display the camera perspective on the complete display, multiple displays are supported. The following camera blocks are available:

FreeCamera: A free movable camera, initialized at the camera-frame connectors start position. The cameras position and perspective can be adjusted in the viewer with the computer mouse.

FixedCamera: The perspective and position of this camera is fixed. The position of the camera only can be changed by the simulation itself, and no user interaction is possible. The direction of the camera view can be parametrized.

FollowCamera: This camera is centered on the position of the camera's reference-frame connector. As the reference connector moves, the camera keeps focused on this position. The position of the camera is defined via the camera frame connector.

DynamicFollowCamera: The position of this camera is defined, like the *FollowCamera* via the camera frame connector position. But unlike the Follow Camera it is no direct coupling but a delayed following behavior characterized by a PT 1 system. The time constant of this following behavior can be parametrized.

AttachedCamera: A camera with free adjustable perspective, but with a position defined by the camera's reference-frame connector. This camera is useful for the observation of a dedicated object from different perspectives.

4.7 Lights

To create a well lightened scene, this sub-package provides Light blocks. Without a dedicated lighting block in the model, a standard light will be created and placed at the position of the camera.

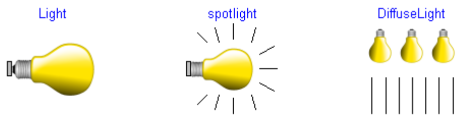


Figure 12: Lighting blocks

Light block: This is the most flexible lighting block, including the complete OpenGL definition for lights. Ambient, specular and diffuse light colors as well as directional lighting and attenuation of light are parameterizable.

Spotlight block: To reduce the configuration effort, this block provides a preconfigured spotlight with a selectable color and a spot angle and direction. The specular and diffuse color are set to the same value as the light color.

Diffuse Light block: This block provides a preconfigured directional light, with selectable color and light direction. This light can be used for environment lighting, as it produces parallel light rays like sunlight.

4.8 Effects

This sub-package contains blocks for additional visual effects. In the current release version of the *ExternalDevices* library, the available effects are weather and particle effects.

Weather effect block: Especially for environment visualization, this block provides the three effects fog, rain and snow for more realism. The fog effect, combined with a reduced viewing distance of the camera can be used to reduce the viewing distance in the scene, in order to increase the frame rate of the visualization.
The Rain and Snow effect are done with a particle system. A wind strength can be parametrized. All weather effects can be triggered with a boolean input by the simulation. Weather effects have no distinct position but are located around every camera.

Particle effect block: With this block simulation of smoke or fire with variable intensity is possible. A wind strength can be parametrized as well as particle size and -lifetime. The ParticleEffect block can be connected to MultiBody systems with a frame connector defining the position of the particle origin.

5 Application examples

The following samples are generated with the ExternalDevices library and are demonstrating selected models with external visualization.

Example 1: Electric motor Figure 13 shows a electrical engine propelling a rotor. In Figure 14 the associated Modelica model is shown, with a magnification of the motor. File Shape blocks are used to represent the CAD Data of the motor and the rotor, a Flow Shape block visualizes the energy flow between motor and rotor.

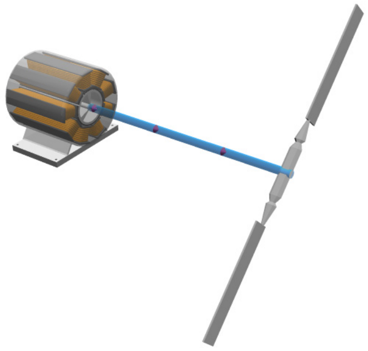


Figure 13: Electrical motor example

Example 2: Hybrid vehicle Figure 15 shows a screen shot of a visualization of a hybrid vehicle, rendered with SimVis. Advanced rendering effects like the transparency of the chassis allow an insight into the car's components. The simulation is controlled with a SensoDrive steering wheel via CAN bus, and a Logitech pedal system via USB joystick input. In Figure 16 the complete driving simulation, as shown on the FISITA world automotive conference 2008 in Munich can be seen. There are actually two simulations running, a driving simulation and a simulation of a robot based motion simulation (see also Example 3). The driving simulation renders two camera perspectives on the left and upper display, while the motion simulator is shown on the right display. The motion simulator receives acceleration

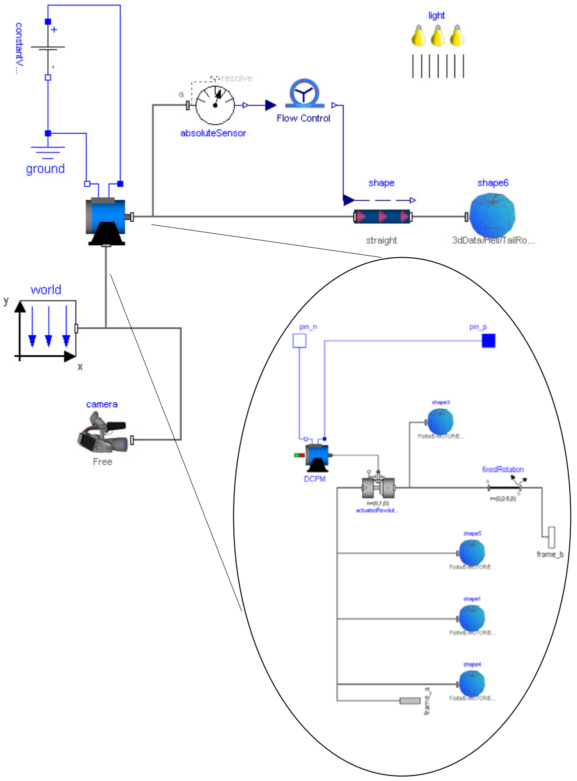


Figure 14: Model of the example

and angular velocity data from the driving simulation and creates a trajectory simulating these motions.

Example 3: Robot visualization Figure 17 shows a visualization of a KUKA KR500/1 robot (Source CAD Data: kuka.com) used as motion simulator. The effects of specular highlights are visible, generating a more plastic look of the robot. In this simulation, the robot joint angles are received via UDP from the real KUKA KR500/1's control computer, the path-planning of the motion simulator is done in a Modelica model and transmitted back to the robot control.

5.1 General performance

During internal tests and projects, scenes with 400 dynamically moved objects have been created, reaching frame rates >25 fps. Models with a memory sizes up to 600 MB (uncompressed) have been loaded and used as scenery. With the increasing rendering power of actual graphic processing units, CAD models with numbers of vertices $> 10^6$ can be displayed with acceptable frame rates.

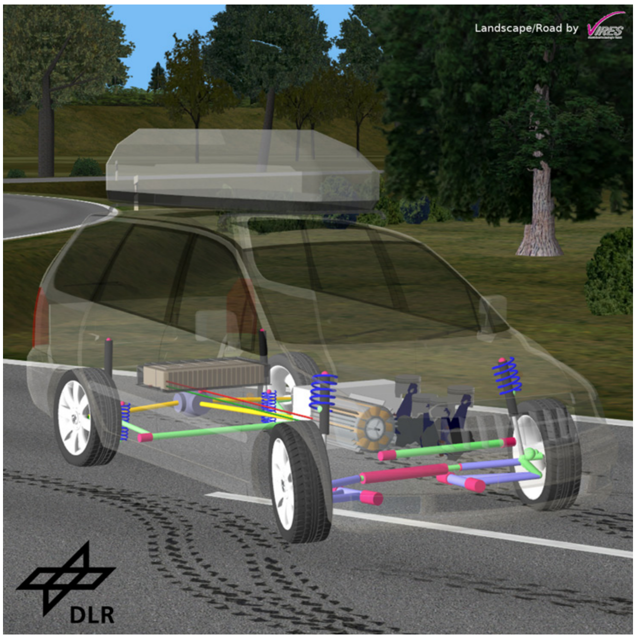


Figure 15: Visualisation of a hybrid vehicle

Figure 16: Driving simulator shown at FISITA 2008

6 Conclusion and Outlook

The *ExternalDevices* library has shown its usefulness during DLR internal tests, especially for creating dynamic, interactive simulations. The possibility to interact with the simulation reduces the effort to generate input trajectories and allows the user to determine the progress of the simulation.

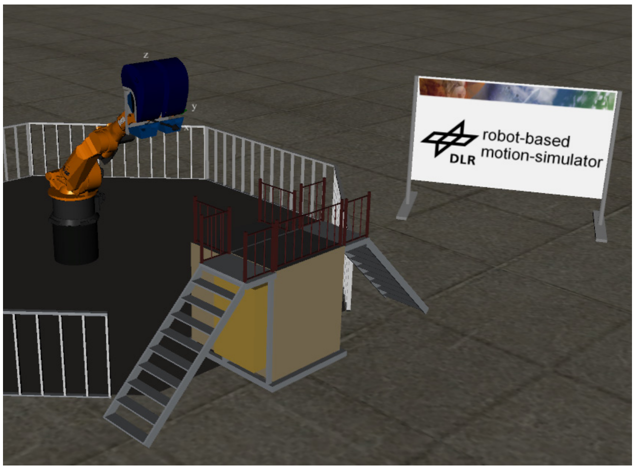


Figure 17: Visualisation of KUKA KR500/1 industrial robot

With UDP and shared memory interfaces, the cooperation of several simulations or reading and writing to other external sources is feasible.
The complete replacement of vendor-specific visualizations and the shift to a platform independent set of Modelica blocks allows much more flexibility in the development of visualization solutions.
The further development of the library will now focus to extend the support of input instruments (e.g. six-axis, force-feedback Joysticks) and the improvement of the visualization, with a new, modular HUD system and full integration of particle systems. Another very focused project aims to integrate plug-ins for loading large terrain databases into the SimVis framework.

References

[1] QT Software: http://www.qtsoftware.com/products/

[2] OpenSceneGraph: http://www.openscenegraph.org/

[3] FFMEPG: http://ffmpeg.org/

[4] OpenGL - The Industry Standard for High Performance Graphics: http://www.opengl.org/

[5] Simulink 3D Animation 5 - User's Guide, MathWorks Inc, http://www.mathworks.de/access/helpdesk/help/pdf_doc/sl3d/sl3d.pdf

[6] Aerolabs VisEngine: http://www.aerolabs.net

[7] Vires Simulationstechnik GmbH: http://www.vires.com/

Redundancies in Multibody Systems and Automatic Coupling of CATIA and Modelica

Hilding Elmqvist[1], Sven Erik Mattsson[1], Christophe Chapuis[2]
[1]Dassault Systèmes, Lund, Sweden (Dynasim)
[2]Dassault Systèmes, Velizy Villacoublay, France
{Hilding.Elmqvist, SvenErik.Mattsson, Christophe.Chapuis}@3ds.com

Abstract

Traditionally, multibody systems have been defined in Modelica by connecting bodies and joints in a model diagram. Additionally the user must enter values for parameters defining masses, inertias and three dimensional vectors of positions and orientations. More convenient definition of multibody systems can be made using a 3D editor available, for example, in CATIA from Dassault Systèmes with immediate 3D viewing.

A tool has been developed that translates a CATIA model to Modelica by traversing the internal CATIA structure to get information about parts and joints and how they are related. This information is then used to generate a corresponding Modelica model. The traversal provides information about the reference coordinate system, the center of mass in the local coordinate system, the mass, the inertia, the shape and color of the body exported in VRML format for animation purposes and the icon exported as a PNG file to be used in the Modelica diagrams.

The Modelica diagram layout is automatically generated and is based on the spanning tree structure of the mechanism. Models obtained in this way often contain redundant constraints. A new method has been developed for Dymola to facilitate simulation of such models, i.e. the model reduction is performed automatically.

An important property of the translated model is the possibility to use Modelica extends (inheritance) for adding controllers and other features of the model for dynamic simulation. For instance, the engine model can be extended by introducing models of the gas forces of the combustion acting on the cylindrical joints of the pistons. In that way, the translated model is separated and can be changed independently of the added models.

Keywords: MultiBody systems, Modelica, CATIA

1 Introduction

Traditionally, multibody systems have been defined in Modelica by connecting bodies and joints from the MultiBody library (Otter et.al., 2004) in a model diagram. More convenient definition of multibody systems can be made using a 3D editor, available, for example, in CATIA with immediate 3D viewing. See for example the CAD model with kinematic definition of a four cylinder engine in Figure 1.

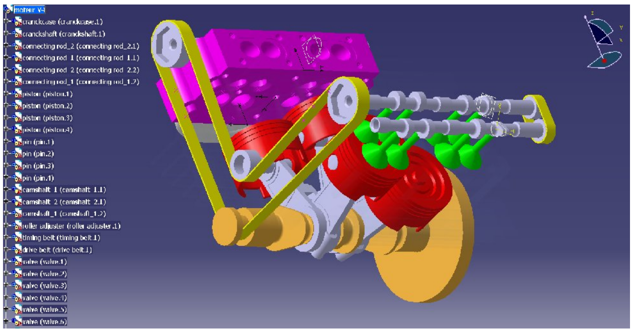

moteur V4
crankcase (crankcase.1)
crankshaft (crankshaft.1)
connecting rod_2 (connecting rod_2.1)
connecting rod 1 (connecting rod 1.1)
connecting rod 2 (connecting rod 2.2)
connecting rod_1 (connecting rod_1.2)
piston (piston.1)
piston (piston.2)
piston (piston.3)
piston (piston.4)
pin (pin.1)
pin (pin.2)
pin (pin.3)
pin (pin.4)
camshaft 1 (camshaft 1.1)
camshaft 2 (camshaft 2.1)
camshaft_1 (camshaft_1.2)
roller adjuster (roller adjuster.1)
timing belt (timing belt.1)
drive belt (drive belt.1)
valve (valve.1)
valve (valve.2)
valve (valve.3)
valve (valve.4)
valve (valve.5)
valve (valve.6)

Figure 1. CATIA V5 model of engine

This paper discusses how to *automatically* derive a corresponding Modelica model to enable dynamic simulation of the mechanism. The first part of the paper describes Modelica code generation, layout generation and how a mechanical model can be extended for multi-domain dynamic simulation. The last part discusses how redundancies in the kinematic definition are handled.

Engelson (2000) discusses automatic translation of SolidWorks models to an earlier version of the Modelica MultiBody library. However, this work did not consider automatic handling of kinematic loops, automatic selection of states nor elimination of redundancies in the kinematic definitions. Bowles et.al. (2000) present a converter from ADAMS to Modelica. This converter made automatic layout of the Modelica code in a similar way as to what is described in this paper. Cut joints were inserted auto-

DOI: 10.3384/ecp09430113

matically. However, planar loops and redundant joints were not handled. There are also other simulation software vendors that provide tools for conversion of CAD models to Modelica or similar formalisms. However, they require manual modification after the conversion to derive a valid model. This is a complicated and error prone task and also makes maintenance of models harder.

2 Modelica code generation

A tool has been developed that translates a CATIA V5 and V6 models to Modelica by traversing the internal CATIA structure to get information about parts and joints and how they are related. This information is then used to generate a corresponding Modelica model. Bodies and joints are mapped to Modelica models from a CATIA library written in Modelica. This library is a wrapper library that maps the models to Modelica.Mechanics.Multibody library or to specially written models.

The CATIA parts are mapped to a model called BodyShape. The traversal provides information about the reference coordinate system, the center of mass in the local coordinate system, the mass, the inertia, the shape and color of the body exported in VRML format for animation purposes and the icon exported as a PNG file to be used in the Modelica diagrams. The connector of the BodyShape body represents the center of mass of the body.

The joints are mapped to models in the CATIA.Joints package, see Figure 2.

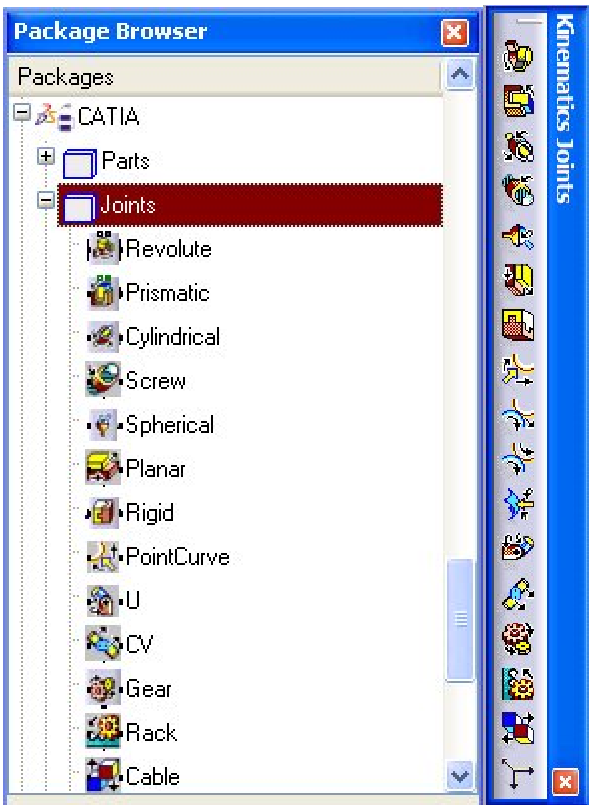
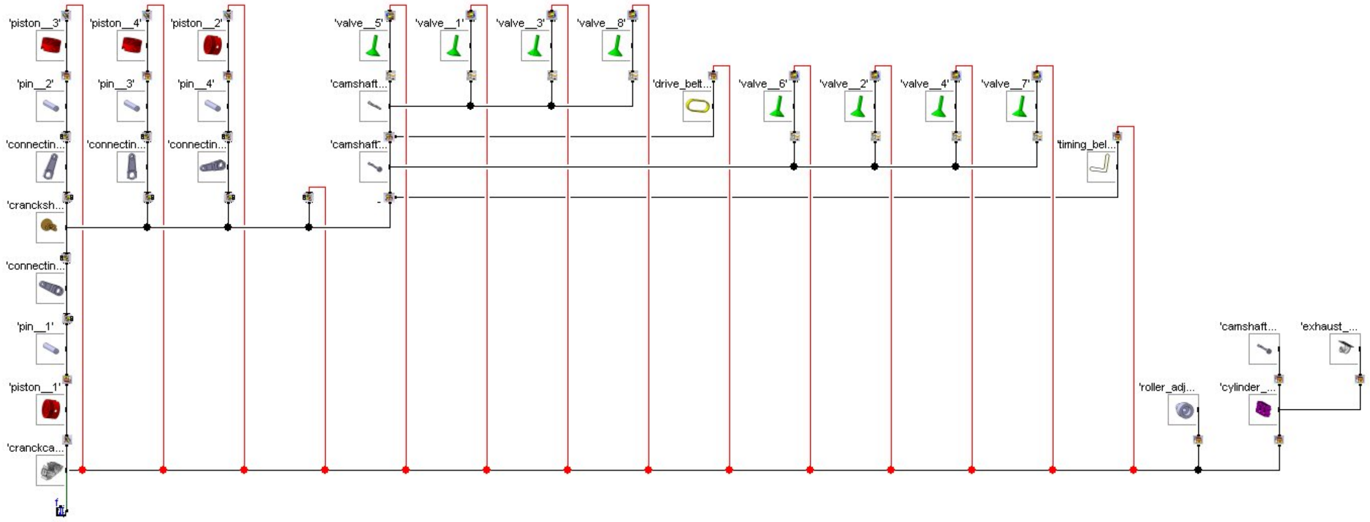


Figure 2. CATIA package structure and Kinematics Joints toolbar in CATIA V5.

There is a one to one mapping from the Modelica Joints to the corresponding Joints in CATIA. The joints include the necessary translations and rotations from the MultiBody joint frames to the center of mass of the respective bodies which the joint is attached to. In addition to the MultiBody joints, some new joint types such as point on curve were developed. The curve data are retrieved from CATIA and table lookup is used in the joint model.

The Modelica model corresponding to the CATIA model in Figure 1 is shown below. It consist of 30 bodies, 9 Revolute, 4 Cylindrical (one for each



cylinder), 8 Prismatic, 2 Gear (to drive the camshafts), 8 PointCurve (one for each valve) and 10 Rigid joints. In addition, the translator introduced 14 CutJoints, one for each kinematic loop. As an example, the Modelica code for the crankshaft is shown below:

```
CATIA.Parts.BodyShape 'crankshaft__1'(
  m = 0.861027543288454,
  r_0_start = {-0.182833605157492,
    0.00863258496368804,-0.0978102539095819},
  R_start=MBS.Frames.Orientation(
    T=[
      0,-0.847538988037033, 0.530733137986643;
      0.0199597528990819, -0.530627407596764,
        -0.84737014496106;
      0.999800784288643, 0.0105933028037161,
        0.0169166684516004],
    w={0,0,0}),
  r_CM = {-0.0584560176906743,0,0},
  I_11 = 0.0116473560180544,
  I_21 = 0.000299454247697959,
  I_31 = -0.000193283295897208,
  I_22 = 0.00484290739689144,
  I_32 = 0.00442125657996889,
  I_33 = 0.00911887589804025,
  shapeName = "2",
  iconName = "./crankshaft__1.png")
  annotation (Placement(transformation(extent=
    {{-120,460},{-60,520}})));
```

The intial position, r_0_start, and orientation, R_start, are retrieved for a consistent initial configuration of the entire mechanism. Mass, m, and inertia, I_11, …, I_33 are calculated for the density given in CATIA. The parameter shapeName refers to the name of a VRML file (2.wrl) exported by CATIA representing the shape and color of the body. The parameter iconName is the name of a PNG-file containing a 2D projection of the body. It is used for the icons in the bodies in the model diagram.

A joint instance is shown below. It contains the axis of rotation, n, and fixed translations, ra and rb, and rotations, fixedRotation_a and fixedRotation_b, from the joint to the center of mass of the connected bodies. In addition, the joint instance contains the initial configuration, phi_start, of the joint.

```
CATIA.Joints.Revolute 'Revolute__5'(
  n = {0,0,1},
  ra = {0,0,0.124000000000001},
  rb = {0,0,0},
  fixedRotation_a(
    R_rel=MBS.Frames.Orientation(
      T=[0,0.907781470137235,-0.41944344371495;
        0,0.419443443714955,0.907781470137245;
        0.999999999999986,0,0],
    w={0,0,0})),
  fixedRotation_b(
    R_rel=MBS.Frames.Orientation(
      T=[0,0.81704961687189,-0.576567362560056;
        0,0.57656736256006,0.817049616871915;
        0.999999999999979,0,0],
    w={0,0,0})),
    phi_start = 1.07387718370489e-008)
   annotation (Placement(transformation(extent=
     {{-70,660},{-50,680}}, rotation = 90)));
```

Figure 3 shows the CATIA V5 representation of kinematic joints, i.e. Revolute.5 is connected in between connecting rod_2.1 and pin.2.

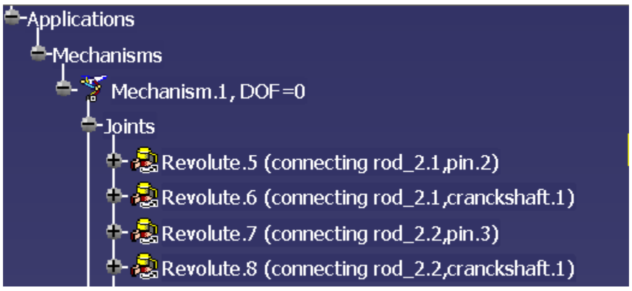

Applications
Mechanisms
Mechanism.1, DOF=0
Joints
Revolute.5 (connecting rod_2.1,pin.2)
Revolute.6 (connecting rod_2.1,cranckshaft.1)
Revolute.7 (connecting rod_2.2,pin.3)
Revolute.8 (connecting rod_2.2,cranckshaft.1)

Figure 3. CATIA V5 representation of kinematic joints in engine

The corresponding Modelica representation is two connections:

```
connect('pin__2'.frame_a,
  'Revolute__5'.frame_b );
connect('connecting_rod_2__1'.frame_a,
  'Revolute__5'.frame_a );
```

3 Layout Generation

The Modelica diagram layout is automatically generated and is based on the spanning tree structure of the mechanism obtained by the classical depth-first search algorithm of graph theory. See the layout of the engine four cylinders in the Modelica diagram for engine model above and in Figure 4 which shows the 4 cylinders zoomed in.

In this way, loops are generated and placed aside of the main branches. Notice that the closing of the loops are presented by red connections. For the Engine, it is possible to see the four cylinders and the longest possible loop is starting at the crank case, passing though a cylinder to the crank shaft and then back through another cylinder to the crank case. Then the other two cylinders are detected as loops from the crank shaft to the already-detected crank case and placed correspondingly. Other loops are the synchronization belts and valve shafts. The root of the tree is the fixed body present on the CATIA model.

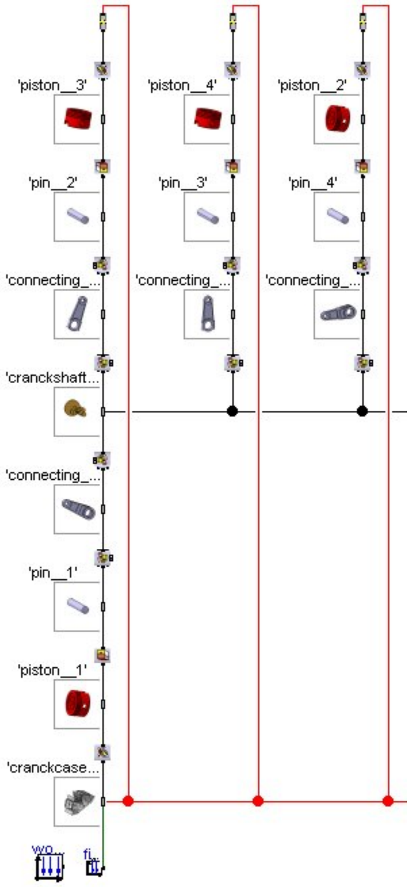


Figure 4. The four cylinders zoomed in.

4 Incorporating dynamics by extending

An important property of the translated model is the possibility to use Modelica extends (inheritance) for adding controllers and other features of the model for dynamic simulation. For instance, the engine model can be extended by introducing models of the gas forces of the combustion acting on the cylindrical joints of the pistons. In that way, the translated model is separated and can be changed independently of the added models.

CATIA models do not contain force aspects. This means that springs, dampers, friction and other force elements have to be added in the Modelica layer. Actuators such as electrical motors or hydraulic systems as well as sensors and control systems are also added to the Modelica layer.

5 Simulating engine

The possibly extended generated Modelica model is symbolically translated to ODE form in the usual way involving finding systems of simultaneous equations, making index reduction and automatic state variable selection. For details, see Otter et.al. (2007) and Mattsson et.al. (2000). The engine model contains point on curve joints. This means a constraint described by tabular data. The index reduction requires the constraints to be differentiated twice. The implication was that the tabular data was used for generation of splines which could be differentiated twice.

In addition, the engine model contains redundant joints. Special methods were developed to handle those, See section 6.

In fact the model has only one degree of freedom, the crank angle. The automatically generated model was extended and a constant torque generator was added to drive the crank shaft. Figure 5 shows plots of torque, acceleration and the position of each of the cylinders. It can be seen that the inertia is not constant since acceleration is time varying. In fact the amplitude of inertia variations increases for higher angular speeds.

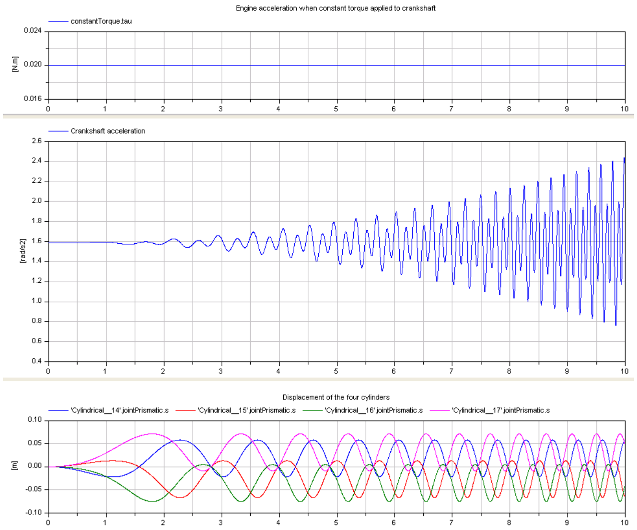


Figure 5. Simulations results.

The results can be animated since all bodies are represented by VRML. The animation in Dymola can be seen in Figure 6.

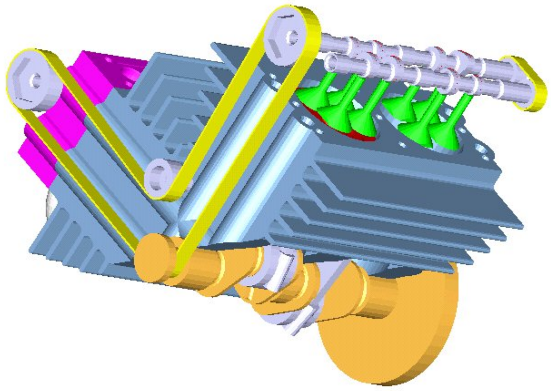


Figure 6. Dymola animation of dynamic behavior.

6 Redundancies in MultiBody Systems

The approximation or idealization that bodies of multibody systems are rigid implies, unfortunately, that it is not possible to determine forces acting on the members of any structure.

As a basic example, let us model a door with two hinges by the simple model Door1 in Figure 7.

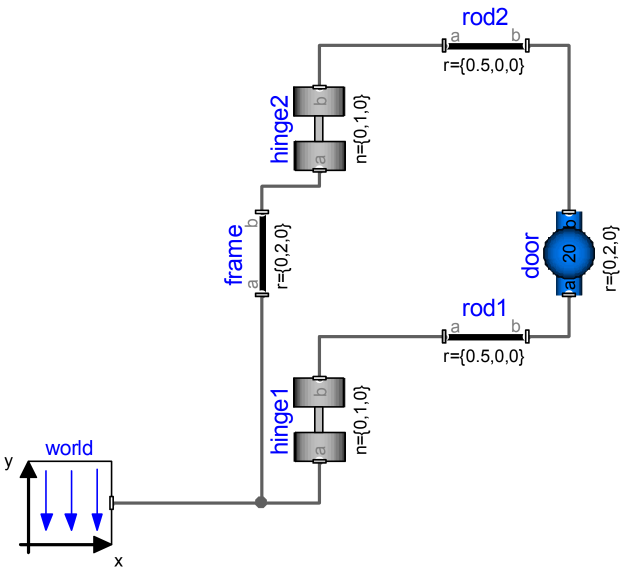


Figure 7. The model Door1

The object *door* represents the door as one rigid body. The house and the door frame are represented by the objects *world* and *frame*. Each of the two hinges is represented by a revolute allowing the door to rotate along a vertical axis, the y-axis. This model does not translate, because it is singular. The error diagnosis from Dymola contains the following message: “The model includes the following hints: All Forces cannot be uniquely calculated. The reason could be that the mechanism contains a planar loop or that joints constrain the same motion.”

It is true that joints constrain the same motion in the model. From a kinematics point of view this model is equivalent to the model Door1a in Figure 8.

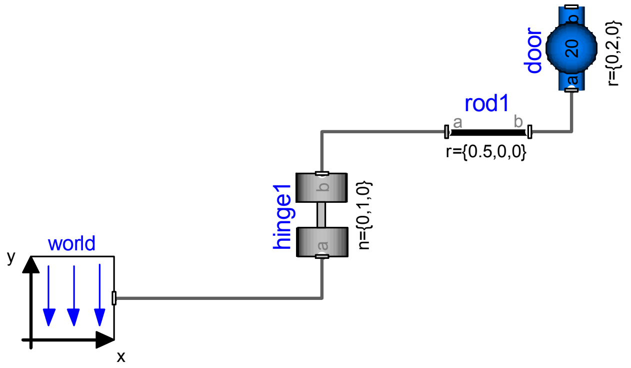


Figure 8. The model Door1a which is a kinematics equivalent to the model Door1.

In other words, the door in the two models can only rotate around the vertical axis, the y-axis. When it comes to forces, the two models are different. The model Door1a, requires hinge1 take all loads, while the model Door1 allows the load to be shared by two hinges.

Why do doors have at least two hinges? The hinges not only balance the vertical force due to gravity, but they need also balance the torque due to gravity. When having just one hinge, this hinge has to provide reactive torques in the x and y directions. There will always be some play in a real hinge and such a door would not behave as desired. When using two hinges, the balancing torque can be realized by a force pair. When having two hinges a more realistic model could be to use spherical joints to describe the hinge. A spherical joint does not produce any reaction torque and we can view as a way of taking the play into account. The balancing torque must be produced by a force pair from the two spherical hinges. Unfortunately, such a model is still singular. In real life, we know that it is important to mount the hinge parts in the frame in the door with good accuracy. If not, the vertical load will not be shared, but one of the hinges has to take all vertical loads. To make the model non-singular, one of the hinges has to include a prismatic joint in the vertical direction, which means that the other hinge has to take all vertical loads. You could think of letting both hinges have two actuated prismatic joints and use springs and dampers to split the vertical loads. However, if we just want to know how the door moves due to external forces, such a model would be unnecessary complex.

When the desire is to model the dynamic behavior of the door, it is just necessary to get the acceleration correct. For a rigid body, we need then to get the total forces and torques acting on it correct. We need not consider how the forces and torques acting on the body are distributed over the body. For the door, we need only the sum of the forces and torques acting on the body with respect to some common point, say, the center of inertia. We can in fact prescribe any value of the vertical reaction torque from hinge2, because, hinge1, will automatically counter balance it and the sum will be correct. The technique to relax redundant position or orientation constraints by prescribed forces and torques is to introduce cut joints.

Below we will introduce the concept of cut joints and how Dymola handles selection of constraints within cut joints automatically.

6.1 Introducing cut joints in kinematic loops

Redundancies in multibody systems can only appear when there are kinematic loops.

The model Door1 has the loop: frame, hinge2, rod2, door, hinge1, rod1 and back to frame. When traversing the CATIA model structure to generate the Modelica model, the tool keeps track of kinematic loops and introduces a cut joint in each kinematic loop. Thus the model of the door, call it Door2, will include a cut joint. Let us place it between hinge2 and rod2 as shown in Figure 9. Actuators have also been introduced for testing purposes.

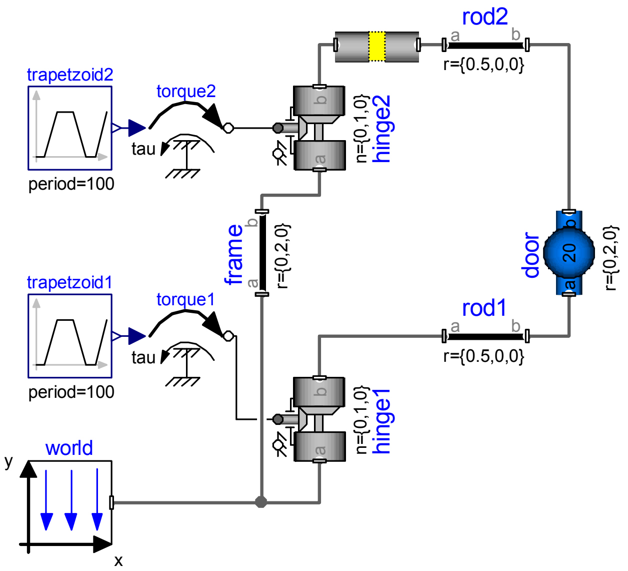


Figure 9. The model Door2.

The basic meaning of the cut-joint is that the position and orientation should be the same. However, it can handle the fact that these constraints might be redundant, i.e., they might be deduced without considering the cut-joint constraints. The cut joint does this by giving expressions for the relative positions and orientations of the two connector frames. The constraint saying that these should be zero is given by a special function with the interface

```
function conditionalConstraint(
  input  Boolean condition;
  input  Real u1;
  input  Real u2;
  output Real y;
  …
end conditionalConstraint;
```

The basic meaning is

```
  y = if condition then u1 else u2;
```

It is used vectorized in the cut joint as

```
  fill(0, 6)  =
  conditionalConstraint(
    conditions,
    {r_rel_a[1], r_rel_a[2], r_rel_a[3],
     phi[1], phi[2], phi[3]},
    {f_c[1], f_c[2], f_c[3],
     t_c[1], t_c[2], t_c[3]});
```

The first actual argument is a Boolean vector with 6 elements. The second argument includes the relative positions and rotations, while the third argument includes the forces and torques in the corresponding directions. A user could set elements of conditions to false to replace redundant position or orientation conditions by postulating corresponding forces or torques to zero. Unfortunately, realistic applications have many loops and since there 2^6 = 64 ways of setting the vector conditions for each loop, there is a combinatorial explosion. It is not feasible to set cut-joints manually. Dymola supports automatic selection of which of the constraints to use in order to have a well-posed problem. In simple cases this can be done statically at translation otherwise it is done dynamically at simulation.

Initialization of models with kinematics loop may be a non-trivial issue; in particular when there are conditional constraints to select. Fortunately, CATIA has a powerful kinematics solver, which means that the Modelica models generated from a CATIA model structure can be provided with consistent initial values, so there is no need for generating code for initialization.

Dymola allows plotting of the Boolean conditions vector in the usual way during and after simulation. Dymola also reports the final selection as new model extending from the model simulated with modifiers setting the condition vector of all conditionalConstraint functions:

```
model Door2_Fixed_Constraints
  extends Door2(cutJoint.conditions=
    {false, false, false,
     false, true, false});
end Door2_Fixed_Constraints;
```

Note that all elements are false but the fifth. It means that Dymola has really cut the loop. The fifth element represents rotation along the y-axis. In this respect the cut joint is in series with hinge1 that allow this rotation. Two revolute joints with the same axis of rotation without inertia in between imply ambiguity between the rotation angles of the two revolute joints and such a model would thus be singular.

Typically you need just to simulate the model to have it initialized, i.e., simulate with startTime = stopTime. A model obtained in this way can be used as a basis for further model development and simulation where Dymola's automatic selection is disabled. For example, initialization equations are now supported in usual way.

We have presented a general cut joint with conditional constraints. It can be placed anywhere in a loop with one important exception. It cannot be placed in series with a spherical joint if there is no inertia in between. A spherical joint puts no constraints on the orientation. It is has no Connections.branch(…) defined between its frame connectors to handle the over determined connectors describing the orientation. The same applies for the cut joint. It means that the part between the spherical joint and the cut joint has no root defined to handle the normalization of the overdetermined part of orientation description. To solve this problem, we have introduced a special spherical cut joint. Such a joint only has the position constraints conditional.

6.2 Example: The door model revisited

Let us consider the door model once more and put the cut joint at another place in the loop, namely between rod2 and door as shown in Figure 10.

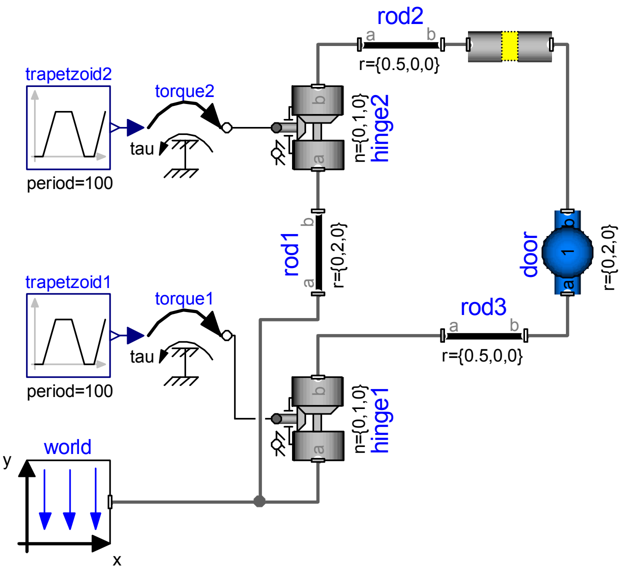


Figure 10. The model Door3.

Assume that the cut-joint would break the loop completely, i.e, all conditions are false. It means that hinge2 allows rod2 to rotate in a plane parallel to the x-z plane. It means that a cut joint cannot break the loop completely, but has to select one constraint. There are two possibilities. First, as for Door2, it can lock the rotation in the direction of the vertical axis. Call this angle φ and thus the active constraint will be $\varphi = 0$. Secondly, the motion can be prevented by requiring the relative displacement of the cut joint in the z direction, $\Delta z = |rod2.r|\cdot\sin\varphi$ to be zero, thus requiring $\varphi = 0$. The selection of constraints is based on analyzing the Jacobian of the constraints. For $\varphi \approx 0$, we have $\Delta z \approx |rod2.r|\cdot\varphi$. It means that if $|rod2.r| < 1$, the condition, $\varphi = 0$ will be selected. Otherwise the conditions $\Delta z = 0$ will be selected. The two will give the same accelerations and the door will move in exactly the same way. However, the torques and forces at the connector frames of the cut joint will be different. Choosing $\varphi = 0$ as the active constraint in the cut joint, will set all forces and torques of the cut joint to zero, except for the torque in the y direction, cutJoint.frame_a.t[2], which will propagate the torque due to the actuation of hinge2. Choosing $\Delta z = 0$ as the active constraint in the cut joint, will set all forces and torques of the cut joint to zero, except for the force in the z direction, cutJoint.frame_a.f[3], which will propagate the effects of the actuation of hinge2. Putting a cut joint close to hinge2 does not influence the possibilities to have actuation on it. This is a better alternative than removing it completely as done in model Door1a.

6.3 Planar kinematics loops

The door model is an example of a model having redundant joints, i.e., joints constraining the same motion. The error message for the model Door1 indicated another possibility, namely possibility of the loop being planar.

Consider the system PlanarLoop1 in Figure 11.

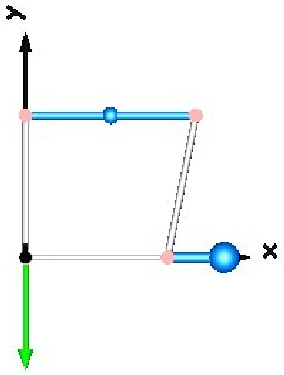


Figure 11. The system PlanarLoop1.

The system is built by bars lying in x-y plane and connected by revolute joints with their axes of rotation parallel with the z axis. A model is shown in Figure 12.

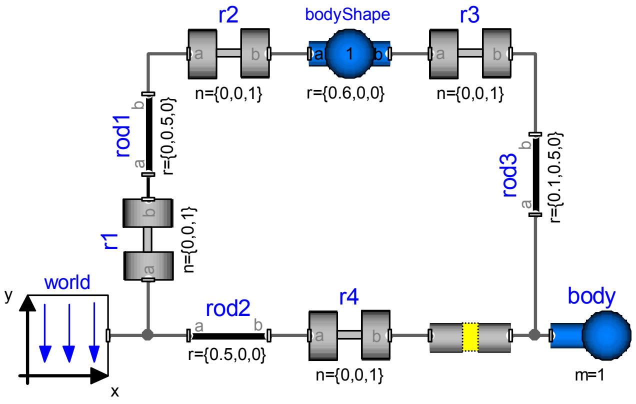


Figure 12. The model PlanarLoop1.

Dymola selects the constraints in the following way:

```
model PlanarLoop1_Fixed_Constraints
  extends PlanarLoop1(cutJoint.conditions=
  {true, true, false, false, false, true});
end PlanarLoop1_Fixed_Constraints;
```

Since the third argument is selected to false, the loop is cut in the z-direction, i.e., the position constraint in the z direction is redundant. First, the system cannot move in the z direction at all, because all joints are revolute joints with their axes of rotation parallel with the z axis. Starting from world going the path rod2, r4 we can deduce that the left end of the cut joint has always cutJoint.frame_a.r[3] = 0. Going the other way from world we find in similar way that cutJoint.frame_b.r[3]. This means that the position constraint in z-direction of the cutJoint is redundant. The components cannot rotate along any axes in the x-y plane. This explains why Dymola selected

```
cutJoint.conditions[4] = false;
cutJoint.conditions[5] = false;
```

The model PlanarLoop1 is built nicely in a coordinate plane z = 0. Far from all Modelica models generated from the CATIA models structure is that well-behaved. Dymola must be able to find any planar loop independent of which plane it exists in. Let us define the mechanism in the plane

-sin(a)*y + cos(a)*z = 0

It means a tilting of the mechanism along the x-axis an angle, a, as shown in Figure 13.

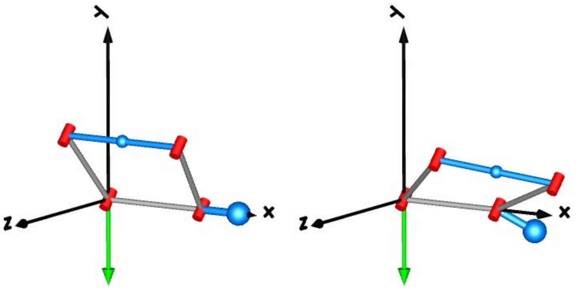


Figure 13. The system PlanarLoop2 in two positions.

The model is shown in Figure 14.

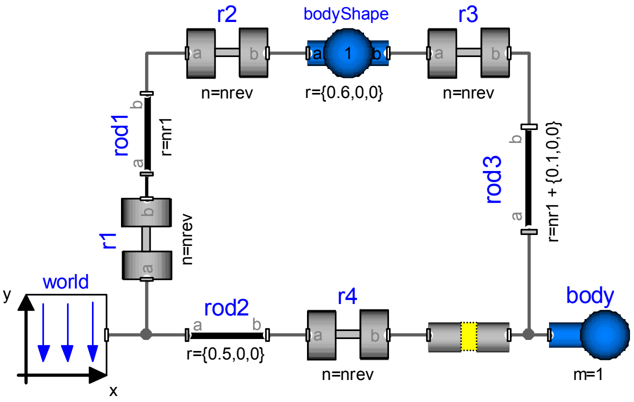


Figure 14. The model PlanarLoop2.

The model PlanarLoop2 is an extension of PlanarLoop1, introducing the parameters a, nrev = {0, -sin(a), cos(a)} and nr1 = 0.5*{0, cos(a), sin(a)}, and modifying the axes of rotations of the revolute joints and the axes of rod1 and rod3.

When the angle is small i.e., -45º<a <45º, Dymola selects the constraints as for the model PlanarLoop1, which is the special case a=0. For -90º<a <-45º or 45º<a <90º for Dymola selects the constraints in the following way:

```
model PlanarLoop2_Fixed_Constraints
  extends PlanarLoop1(cutJoint.conditions=
  {true, false, true, false, true, false});
end PlanarLoop2_Fixed_Constraints;
```

Simply said, you get the selection of the coordinate plane to which you have least tilt angle.

6.4 Example: A crank shaft, piston and cylinder mechanism.

Consider the model Engine1a, which is found in Modelica.Mechanics.MultiBody.Examples.Loops. see Figure 15 and 16..

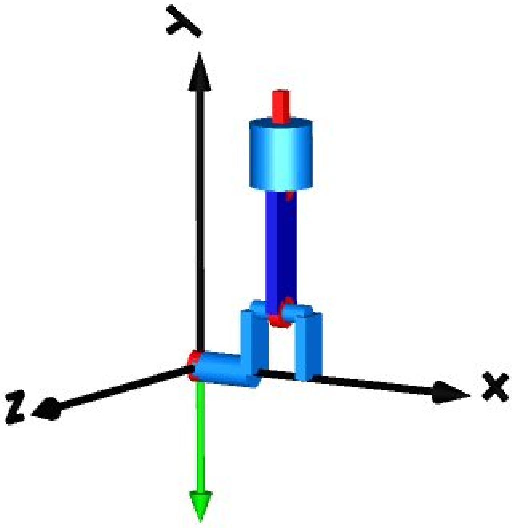


Figure 15. An animation of Engine1a.

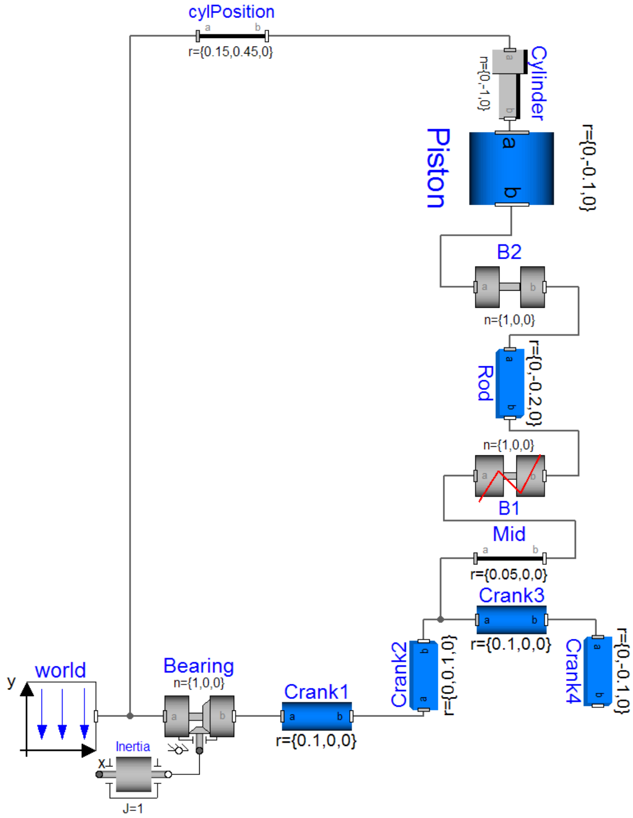


Figure 16. The model Engine1a.

The model is intricate. First, the revolute joint B1 is a revolute planar cut joint indicating that it is part of a planar loop. This is may be not that difficult to understand when looking at Figure 15. However, the component cylinder at the top is not a cylindrical joint as might be expected, but a prismatic joint. This joint does not allow the piston to rotate. If B1 had been a real revolute joint then the crank mechanism had prevented rotation of the piston along its length axis. Unfortunately, when B1 is a planar cut joint, then the piston can rotate. The component Cylinder being just a prismatic joint fixes that.

The new universal cut joint and the automatic constraint selection of Dymola relives the user from such complex “fixing” of the model. Let us make a more natural, straightforward model, call it Engine. It is similar to Engine1a, but B1 is an ordinary revolute joint and Cylinder is a cylindrical joint. Additional we add a cutJoint between cylPosition and Cylinder, because there is a kinematics loop. Dymola translates and simulates this model without any problems. Dymola reports the following selection of constraints for cutJoint:

```
model Engine_Fixed_Constraints
   extends Engine(cutJoint.conditions=
   {false, true, true, true, true, false});
end Engine_Fixed_Constraints;
```

Note, that the conditions vector has only two elements selected to be false. Reconsider the model Engine1a. The revolute cut-joint B1 means selecting the condition vector to be

```
   {false, true, true, true, false, false}
```

The fifth element is here false. It means that the revolute planar cut joint is cutting the loop a bit too much. If Cylinder in model Engine1a would have been a cylindrical joint, the part between B1 and Cylinder had been free to rotate along the y axis. Selecting Cylinder to be a prismatic joint prevents the rotation along the y-axis.

7 Conclusions

A method for coupling kinematic CAD models, in particular CATIA models, to Modelica has been described. Since such models often contain redundant joints, a new combined structural, symbolic and numerical technique has been developed for Dymola to handle such models. Details were given about typical such redundant kinematic definitions and how the new method handles those.

8 Acknowledgements

The authors would like to thank Martin Otter (DLR) and Hans Olsson (Dynasim) for their contributions.

References

Bowles P., Tiller M., Elmqvist H., Brûck D., Mattsson S.E., Möller A., Olsson H., Otter M (2000).: **Feasibility of Detailed Vehicle Modeling**. SAE World Congress 2000.

Dymola (2009). **Dymola Version 7.3**. Dassault Systèmes, Lund, Sweden (Dynasim). Homepage: www.dymola.com.

Engelson V. (2000): **Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing**. Department of Computer and Information Science, Linköping University, Sweden.

Mattsson S.E., Elmqvist H., and Olsson H. (2000): **Dynamic Selection of States in Dymola**. Proceedings of Modelica Workshop 2000, pp. 61-67. Download: http://www.modelica.org/events/workshop2000/proceedings/Mattsson.pdf

Otter M., Elmqvist H., Mattsson S.E. (2003): **The New Modelica MultiBody Library**. Proceedings of Modelica'2003, ed. P. Fritzson, pp. 311-330. Download: http://www.modelica.org/events/Conference2003/papers/h37_Otter_multibody.pdf

Otter M., Elmqvist H., Mattsson S.E. (2007): **Multidomain Modeling with Modelica.** Handbook of Dynamic System Modeling, Chapter 36, Ed. P. Fishwick, Taylor & Francis Group LCC .

Improvement of MSL Electrical Analog Library

Kristin Majetta Christoph Clauß Matthias Franke Peter Schneider
Fraunhofer-Institute for Integrated Circuits, Design Automation Division
Zeunerstraße 38, 01069 Dresden, Germany
{Kristin.Majetta, Christoph.Clauss, Matthias.Franke, Peter.Schneider}@eas.iis.fraunhofer.de

Abstract

The set of device models of the Modelica electrical analog library needs to be extended to support improved applications. Within the EUROSYSLIB project a multiple line model, a zener diode, a thyristor, an opamp and some switches with arc models, and a conditional heat port were added. The performance was improved generally. Furthermore, IGBT, TRIAC and converter models are planned.

Keywords: Modelica standard library, electrical analog library, EUROSYSLIB

1 Introduction

The electrical analog library of the Modelica Standard Library (MSL) has been developed for more than 10 years [1], [2]. It was widely used which caused suggestions for improving the library. The intention of the library development is to

- meet the user requirements for functionality
- give examples for device modeling
- meet a textbook like level of simplicity

Therefore, the library development is a long-lasting user-driven process. In this paper some single improvements are collected which arose on partner demand in the EUROSYSLIB project.

2 Improved performance due to smooth-operator

The smooth-operator is an event-related built-in intrinsic operator of Modelica [3]. It can be used to state the smootheness property of a Real expression. If an expression is expected to be discontinuous (e.g. due to an if-statement) usually an event has to be generated which interrupts the analog simulation algorithm. This interruption can be avoided if the smoothness property was stated using the smooth operator. Furthermore, the level of smoothness (how often the expression is continuously differentiable) can be used by the simulation tool to choose the appropriate simulation order.

The MSL Electrical Analog Library was checked with regard to smoothness. Mostly at semiconductor devices the smooth-operator could be used. This improved the performance in large circuit models considerably. As an example an adder chain was simulated using Dymola. It consists of 10 full adders built by MOSFET transistors which are modeled using the MSL Electrical.Analog.Semiconductor models. All in all 620 transistors are used. The improvement is shown in the following table 1.

Advanced. SmoothEvent	Smooth operator	F-Evaluations	steps	CPU Seconds
default (1)	not used	1994791	8575	1230
default (1)	used	1487274	6767	962
0	used	28284	1541	4.8

Tab. 1: Full adder chain statistics

The second line addresses the usage of the smooth-operator whereas all simulator parameters of Dymola are default.

To fully exploit the smoothness information given by the smooth-operator in Dymola the variable

```
Advanced.SmoothEvent = 0
```

is recommended to be set. This causes a drastic improvement shown in the third line of table 1. The SmoothEvent variable controls the event handling if the smooth operator is applied. By setting SmoothEvent to zero no events are generated which causes a high performance, but it must be taken into consideration that precision can be lost. Consequently, the smooth operator should be applied if further models will be added to the MSL. Otherwise, the smooth-operator should be improved to avoid the necessity of setting SmoothEvent.

3 Conditional heat port

In usual electrical circuit simulation a heat port is not necessary. Each device is simulated using its nomi-

DOI: 10.3384/ecp09430018

nal temperature which sometimes can be set via parameter. If otherwise a coupled electrical and thermal simulation is asked for a thermal heat port for the connection to a thermal network is desired. Therefore, a conditional heat port was introduced that can be switched on if needed. Moreover, the heat port is useful for a clear handling of energy conservation. The loss power which normally disappears in a pure electrical simulation is dissipated to the heat port, and the energy balance becomes correct.

As an example the new resistor model with conditional heat port is described. The Boolean parameter useHeatPort controls the presence of the heat port. Fig. 1 shows the resistor icon which is sensitive regarding to the useHeatPort parameter.

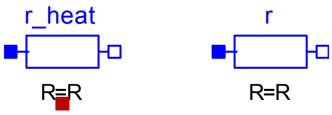


Fig. 1: Resistor icon with and without heat port

If the useHeatPort parameter is set to true, a heat port of the thermal heat transfer library exists the temperature of which is equal to an additional variable T_heatPort. The heat flow rate of the heat port is set to the variable LossPower, the value of which has to be calculated basing on electrical loss power. In this way the electrical loss power is transformed into an accompanying thermal network. Otherwise via T_heatPort the actual temperature can be used in the electrical component.

If the heat port is not active (useHeatPort false, default case) the actual temperature is constant by setting to a parameter T. Though the LossPower variable is calculated it is not used.

Since this functionality of the heat port is the same in all electrical components it was extracted to a partial model ConditionalHeatPort which can be extended from electrical components if a conditional heat port is necessary. The source code of the ConditionalHeatPort model is:

```
partial model ConditionalHeatPort
  import SI=Modelica.SIunits;
  import H=Modelica.Thermal.HeatTransfer;
  parameter Boolean useHeatPort = false
    “=true, if HeatPort is enabled”;
  parameter SI.Temperature T=293.15
    “Fixed device temperature
       if HeatPort is disabled”;
  H.Interfaces.HeatPort_a
    heatPort(T(start=T)=T_heatPort,
    Q_flow=-LossPower) if useHeatPort;
  SI.Power LossPower “Loss power leaving
    component via HeatPort”;
  SI.Temperature T_heatPort
    “Temperature of HeatPort”;
equation
  if not useHeatPort then
    T_heatPort = T;
  end if;
end ConditionalHeatPort;
```

Via inheritance the conditional heat port is used in the resistor component. To make the component temperature sensitive the parameters T_ref (refenrence temperature at which e.g. the resistance R was measured) and alpha (linear temperature coefficient) were introduced. The temperature T_heatPort which is either coming from the heat port or is set to T_ref if the heat port is not enabled is used to calculate the temperature dependent resistance R_actual:

```
  R_actual =R*(1+alpha*(T_heatPort-T_ref));
```

Using alpha = 0 (default) R_actual is identical to the resistance parameter R, and the resistor is not temperature dependent.

The electrical dissipation is simply calculated by

```
  LossPower = v*i;
```

The partial model ConditionalHeatPort connects the variable LossPower to the heat flow rate Q_flow of the heat port.

The source code of the improved resistor model is:

```
model Resistor
  import SI=Modelica.SIunits;
  import A=Modelica.Electrical.Analog;
  parameter SI.Resistance R(start=1)
  “Resistance at temperature T_ref”;
  parameter SI.Temperature T_ref=300.15
    “Reference temperature”;
  parameter SI.LinearTemperatureCoefficient
    alpha=0;
  extends A.Interfaces.OnePort;
  extends A.Interfaces.ConditionalHeatPort
    (T = T_ref);
  SI.Resistance R_actual;
equation
  R_actual =R*(1+alpha*(T_heatPort-T_ref));
  R_actual*i = v;
  LossPower = v*i;
end Resistor;
```

In this way a conditional heat port was added to many devices with electrical dissipation. The constant and the variable resistor and conductor as well as some semiconductor devices are temperature sensitive. Most of the ideal devices are not temperature sensitive, but the loss power is calculated. In many cases the temperature sensitive devices are still separated from not sensitive devices.

As an example a simple NOR circuit is regarded (fig. 2) with temperature sensitive bipolar transistors. Fig. 3 shows both the electrical and the thermal behavior. Since the thermal network is nonrealistic because of missing cooling devices, the temperature rises until the electrical function crashes (fig. 4).

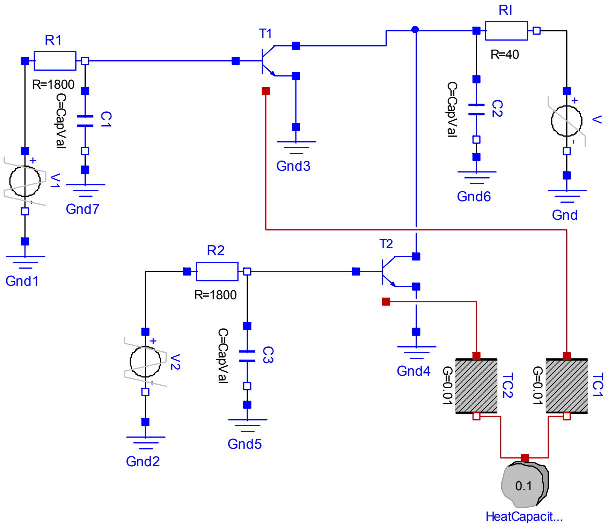


Fig. 2: Thermal electrical NOR circuit

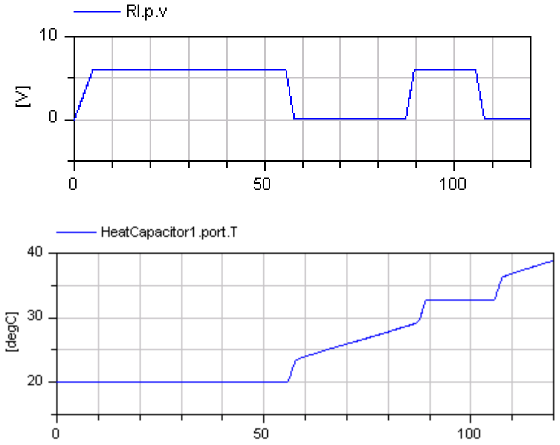


Fig. 3: Thermal and electrical behavior

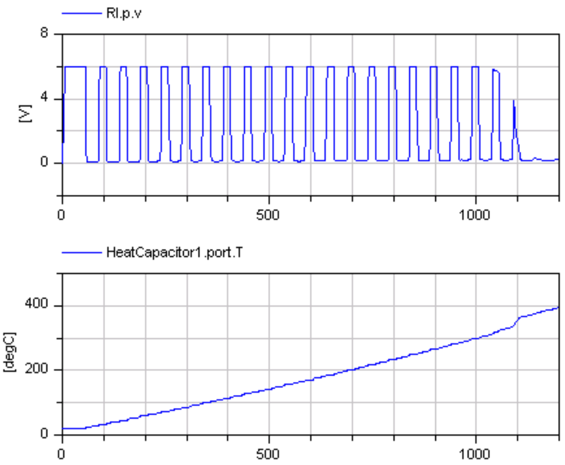


Fig. 4: Crash due to overheating

4 Multiple line model

For an improved modeling of electrical lines a multiple line model was added. It consists of a series of lumped RLGC segments according to fig. 5. Both the number of lines and the number of segments can be specified.

Further parameters are the length of line, and the matrices of the linear electric constants (r, l, g, c) which are used to calculate the segment parameters. E.g. the segment parameter R1 is calculates by

R1 = r1 * length / number of segments

The segment parameters do not vary along the line. Only the R and L parameters of the first segment are cut into halves. They are complemented to full values by a special segment at the other end of the line which consists of R and L devices only. This special arrangement is used for reasons of symmetry. Frequency dependent loss parameters [4] are not modeled.

Since the linear electric constants matrices are symmetrical only the upper triangular matrix has to be specified.

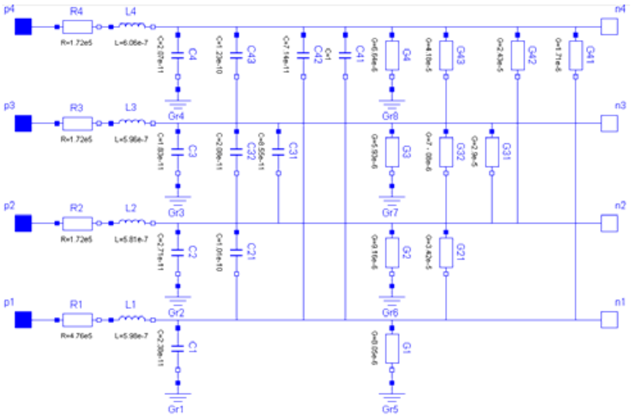


Fig. 5: RLGC segment

For the coupled inductors of the RLGC segment a special model **M_Transformer** was introduced (fig. 6). Added to the Basic subpackage it allows the modeling of multiple winding transformers. Its parameters are the number of inductors and the inductance matrix.

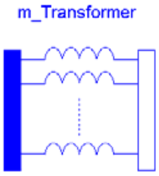


Fig. 6: Icon of the M_Transformer model

To demonstrate the multi line model behavior a four line model is connected with resistors and a voltage source according to fig. 7. Fig. 8 shows the expected behavior. The voltage rising of the first single line is temporarily coupled into the grounded single lines.

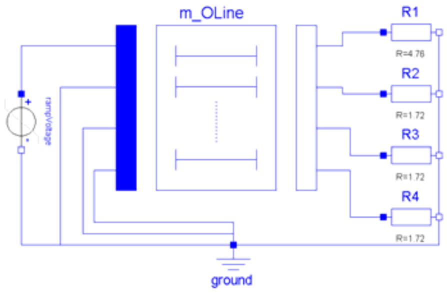


Fig. 7: Circuit containing a four line model

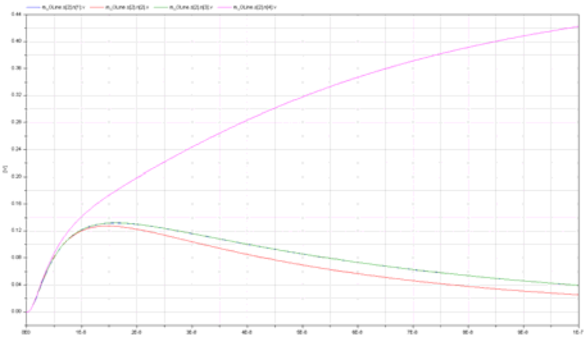


Fig. 8: Voltage coupling due to multiple line model

5 Zener diode

A simple Zener diode model [5] which is widely used for voltage stabilization was added to the semiconductor subpackage. Its equation combines the usual diode behavior with a breakthrough characteristic:

$$i = Ids(e^{v/Vt} - 1) - Ibv(e^{-(v+Bv)/(Nbv*Vt)})$$

Parameters are Ids (saturation current), Vt (voltage equivalent of temperature), Bv (Zener voltage), Ibv (breakthrough knee voltage), and Nbv (breakthrough emission coefficient). A parallel ohmic resistance is added. If an exponent reaches the limit Maxexp the characteristic is continued linearly to avoid overflow. Fig. 9 shows a voltage stabilization circuit with a sine input. The voltage stabilization reached due to Zener diodes is shown in fig. 10.

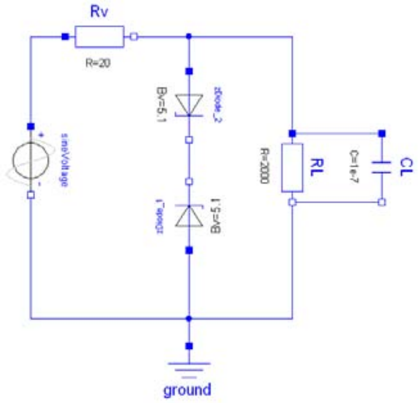


Fig. 9: Voltage stabilization circuit

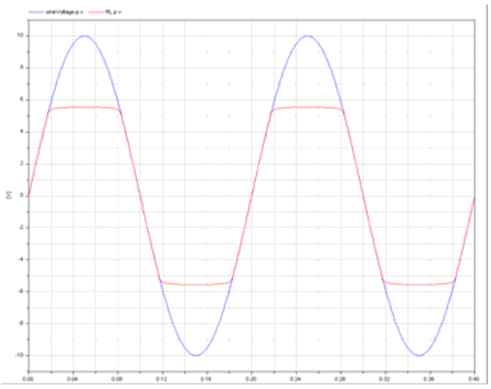


Fig. 10: Stabilized voltage due to Zener diodes

6 Thyristor

Since the ideal thyristor models are very simple ones a more realistic, but still simple thyristor model was added. It has the three pins Anode, Cathode and Gate.

If the thyristor is in the blocking mode the behavior is like a linear resistor. It changes into the conduction mode if either the voltage between cathode and anode exceeds a certain value or a positive gate current flows a certain time.

There is no possibility to switch off the thyristor via the gate. It stays in the conducting mode until the anode current falls below the holding current value which is a parameter. If the voltage between anode and cathode is negative the model represents a diode with reverse breakthrough voltage.

The following simple test circuit (fig. 11) shows the behavior of the thyristor model.

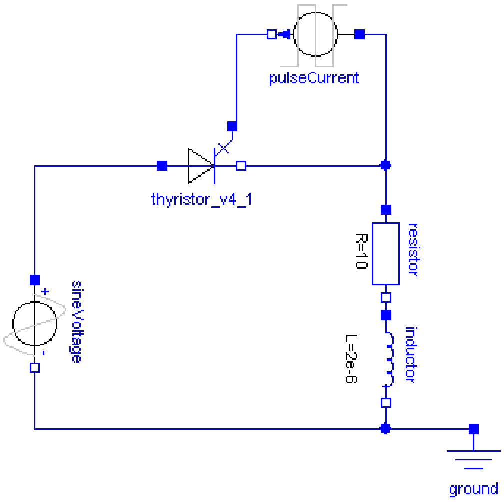


Fig. 11: Thyristor behavior test circuit

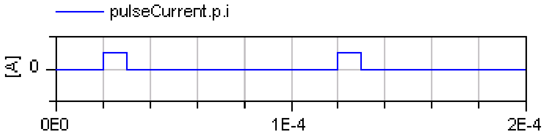


Fig. 12: Gate current of the thyristor test circuit

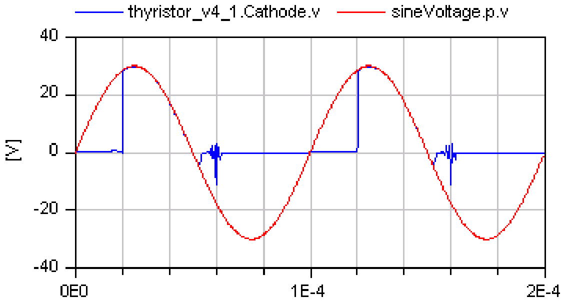


Fig. 13: Resulting behavior of the thyristor test

In fig. 13 the anode and cathode voltages are shown. If the gate current (fig. 11) rises, the thyristor becomes conducting and the voltage drop between anode and cathode is zero.

7 Operational Amplifier

In many cases the operational amplifier models of the MSL (IdealOpAmp, IdealOpAmpLimitted, OpAmp) are not sufficient. Therefore, a detailed model OpAmpDetailed was added, which is divided into five functional stages (input, frequency response, gain, slew rate, and output). Each stage contains a set of typical data sheet parameters, which are independent from parameters of the other stages. The model is described in detail in [6].

Fig. 14 shows an amplifier circuit using the OpAmpDetailed model. The results in fig. 15 show an amplification with voltage limitation.

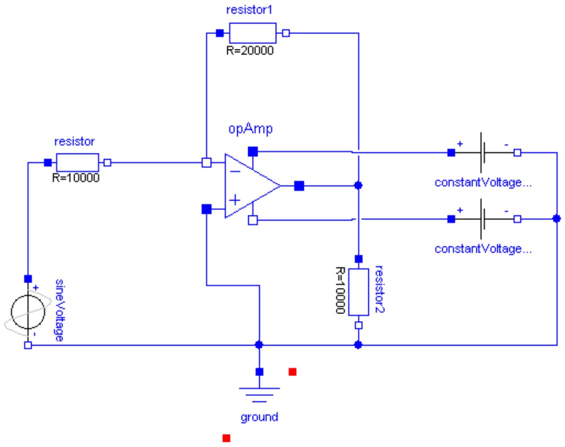


Fig. 14: Voltage amplifier circuit

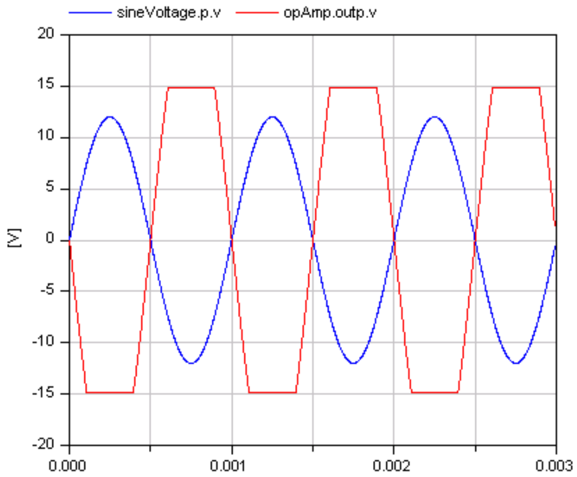


Fig. 15: Amplification with voltage limitation

8 Switches with Electric Arc

The ideal switch models of the Ideal subpackage interrupt the current flowing through the switch within an infinitesimal time span. This causes numerical difficulties if an inductive circuit is connected since that current has to be differentiated. The voltage across the switch is only limited by the numerical solution methods. To improve this switches a simple electric arc model was introduced.

When the switch opens a voltage across the opened switch is impressed. This voltage starts with the initial arc voltage (V0) and rises with the arc voltage slope (dVdt) until the maximum arc voltage (Vmax) is reached. V0, dVdt and Vmax are parameters. The arc quenches if the current once reaches that current that would flow in the off-state of the switch. Then the off-state is activated.

Both Boolean controlled and voltage controlled openers and closers with arc model are added to the standard library.

Fig. 16 shows a comparison circuit between switches without and with the arc model. The arc model avoids the sudden switching (fig. 17).

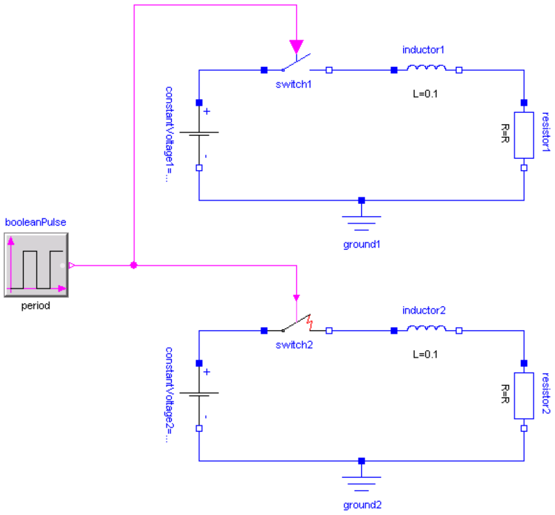


Fig. 16: Comparison circuit of switches without and with electrical arc

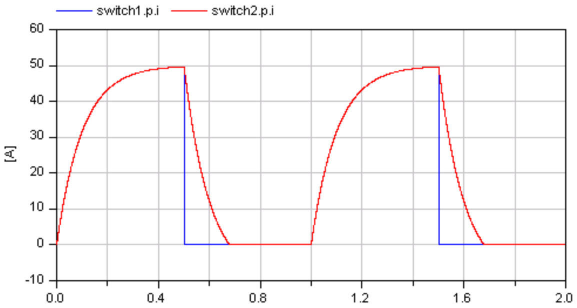


Fig. 17: Switch currents of the comparison circuit

9 Translational EMF

Similar to the EMF model of the MSL Electrical.Analog.Basic library which transforms electrical energy ideally into rotational mechanical energy a TranslationalEMF model was introduced that transforms the electrical energy into translational mechanical energy. Fig. 18 shows the icons:

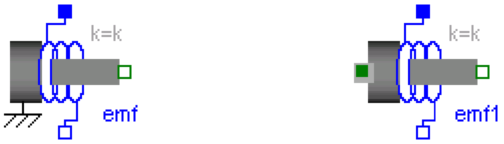


Fig. 18: Icons of the translational EMF model

If the parameter useSupport is set to false (default), the model is internally grounded. Therefore it has only one mechanical flange (left icon in fig. 18). Setting useSupport to true the model has two flanges without being internally grounded (right icon in fig. 18). With v being the voltage, i the current, vel the velocity and f the force the essential equations are:

```
k * vel = v;
f = - k * i;
```

k is the transformation coefficient.

10 Conclusions

The modelica electrical analog standard library was improved by a general usage of the smooth operator which increased the performance drastically. A conditional heat port allows thermal-electric simulation. A multiple line model, a multi inductor transformer model, a Zener diode model as well as thyristor and a detailed operational amplifier model, switches with an electric arc model and a translational EMF model were added.

Further models are planned to be added in future: IGBT, TRIAC and converter models. The MSL electrical analog library development is accompanied by the development of a library according to the Berkeley SPICE3 simulator devices.

Acknowledgement

This research was founded by the European ITEA2 project EUROSYSLIB. The authors thank A. Haumer, C. Kral (Arsenal), M. Otter (DLR), S. Wolf (Fraunhofer IIS), as well as both the Dymola and SimulationX support for either models or helping modeling.

References

[1] Clauß, C.; Leitner, T.; Schneider, A.; Schwarz, P.: *Modelling of electronic circuits with Modelica.* Proc. Modelica Workshop, Lund, Sweden, Oct. 2000, 3-11.

[2] http://www.modelica.org/libraries/Modelica

[3] Modelica – A unified object-oriented language for physical systems modeling. Language Specification 3.0, via http://www.modelica.org

[4] Smolyansky, D.; Klein, W.: Angewandte Simulation von verlustbehafteten digitalen Übertragungsleitungen. Test Kompendium 2004, 109-112

[5] Tietze, U.; Schenk, C.: Halbleiter-Schaltungstechnik. 12. Aufl., Springer-Verlag Berlin Heidelberg, 2002

[6] Conelly, J.A.; Choi, P.: Macromodelling with SPICE. Englewood Cliffs: Prentice Hall, 1992

SPICE3 Modelica Library

Kristin Majetta Sandra Böhme Christoph Clauß Peter Schneider
Fraunhofer-Institute for Integrated Circuits, Design Automation Division
Zeunerstraße 38, 01069 Dresden, Germany
{Kristin.Majetta, Sandra.Boehme, Christoph.Clauss, Peter.Schneider}@eas.iis.fraunhofer.de

Abstract

Since the very beginning of the Modelica development ambitions for electronic simulation exist. The electronic simulator SPICE, the SPICE models and the SPICE netlists grew to a quasi standard in electronics simulation for the last 30 years. That is why the wish arose to have SPICE models available in Modelica. This paper deals with modeling the SPICE3 models in Modelica directly extracted from the original SPICE3 source code. This courses the problem of transforming the sequential simulator-internal model descriptions of SPICE to the declarative description from Modelica. To solve this problem a way was developed and tested for some SPICE3 semiconductor models. The actual library is presented and further plans are shown.

Keywords: SPICE, Modelica, SPICE3 library for Modelica, Semiconductor models, Electronic circuit simulation

1 Motivation

With starting the development of Modelica, models for electrical circuits were taken into consideration [1]. Since SPICE and its derivatives grew to a quasi standard in electronics simulation the SPICE models should become available within Modelica.

Beyond the Modelica standard library (MSL) two SPICE libraries were developed [2], the SPICELib and the BondLib. The SPICELib [3], which covers different complex MOSFET models, is a standalone library with its own connectors. The BondLib [4] bases on bond graphs. It offers different levels of models related to HSpice.

The reason for developing this SPICE3 library is to provide both the original Berkeley SPICE3 models and the SPICE netlist approach. Furthermore, some additions will be prepared to cover PSPICE models.

Since the Berkeley SPICE3 simulator is the only known electric circuit simulation program with open source code it offers the opportunity to extract models for implementation in Modelica. The SPICE3 library uses that way for SPICE3 semiconductor models.

In this paper the modelling steps are considered which are done starting with a C++-model library which was extracted from SPICE3 formerly. The SPICE3 library structure is presented as well as a circuit example.

2 SPICE3 models and netlists

The Berkeley SPICE3 (latest versions e5 or f4) is a general-purpose circuit simulation program which has built-in models both for general devices (resistors, capacitors, inductors, dependent and independent sources) and semiconductor devices (Diode, MOSFET, BJT,…). Some models are a collection of different single models (levels). Instead of adding new models the user is able to choose a large variety of parameters. Only sometimes a new model is added by the developer. The set of SPICE models is like a standard in circuit simulation.

Via a netlist the SPICE3 models are composed to a circuit to be simulated. The netlist contains the model instances, their actual parameters, and the connection nodes. In more detail SPICE3 netlists are described in the SPICE3 user's manual [5]. For many electric and electronic devices SPICE3 netlists are available. For the following inverter circuit figure 2 shows the SPICE3 netlist.

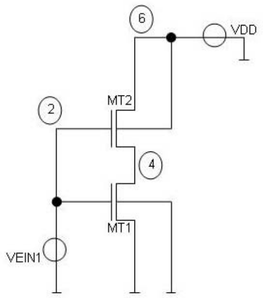


Fig. 1 MOSFET inverter circuit

DOI: 10.3384/ecp09430019

```
 Simulation inverter circuit
 MT1  4 2 0 0 Tran_NMOS L=2u  W=5u
 MT2  6 2 4 6 Tran_PMOS
 VDD  6 0 5.0
 VEIN1 2 0 dc=0 sin(0 5 0.5)
 .model  Tran_NMOS NMOS (VT0 =0.7
 tox=8n lambda=3e-2)
 .model  Tran_PMOS PMOS (VT0 =-0.7)
 .tran 0.001 10
 plot v(6) v(2) v(4)
.end
```

Fig. 2 SPICE3 netlist of the inverter circuit

Within the semiconductor devices SPICE3 differentiates between technology parameters and device parameters. Device parameters can be chosen for every single model instance, e.g. the channel length of a transistor. Technology parameters which are specified in a model card (.model) are adjustable for more than one element simultaneously, e.g. the type of transistors.

3 Model extraction out of SPICE3

The SPICE3 internal models were extracted from the SPICE3 source code, and stored in a (commercial) C++ library [6] [7]. This library was intensively tested by including it as external model code to SPICE3, so it was possible to test the C++ models and the original SPICE3 models in parallel.

The C++ library includes the whole model pool of the semiconductor elements of SPICE3. For each element both a C++ file (*.cpp) and a header file (*.h) exist. The header file of each semiconductor element contains classes with data (parameter and internal data) and declaration of methods. In the C++ file the methods are coded.

Due to the object-oriented principle, a class hierarchy of model components was created. Central base classes contain such values and their methods that, according to SPICE3, are needed in nearly every model, e.g. the nominal temperature. Via inheritance of the base classes their values are provided to other classes. Each functionality that is needed more than one time is coded in a separate function. Consequently, a strongly structured hierarchy of classes was developed.

To simulate a model with the C++ library a SPICE3 typical system of equations is generated (initialization phase) and for each solution step the current data are loaded (simulation phase). For each device of the circuit, model specific methods that are called according to different aspects are supplied. These methods can be disposed under functional aspects as follows:

- Methods to analyse the source code
- Methods to create the linear system of equations
- Methods for instantiation the models and parameters
- Methods to calculate values of the linear equation system
- Methods to insert values into the system of equations

For each model parameter a variable exists, that is called “parameterValue” which gets the value of the particular parameter. For some parameters it is important to know whether they were set by the user or their default values were used. Depending on which case comes into effect, different formulas are used in the further calculation. Even if the value set by the user is the same as the default value, the simulation results differ in some cases. The information if a parameter is set is stored in a Boolean value “IsGiven” (true, if the parameter is set). The “IsGiven” value is analysed by different methods.

The semiconductor devices are modelled by means of a substitute circuit. In this process the different physical effects are allocated at any one time to a component of this circuit. For each of this effects different methods exist, that insert the currents and conductances that are calculated for the actual voltages at the pins, into the linear system of equations. Also equations are arranged for the internal nodes of the substitute circuit. For the calculation also internal values of the integration method are used, e.g. the actual time step size and the history of the calculated values.

In summary the C++ library of the SPICE3 semiconductor elements can be characterized as follows:

- The complete library is according to the semiconductors structured in classes, which contain data and methods.
- For each device methods exist, that achieve the necessary calculations.

- For each element of the SPICE3 netlist, the according classes are instantiated. The needed methods are called for every instance.
- The aim of these calls is to create a linear system of equations to calculate the solution like in SPICE3.
- The parameter handling is special because of the calculations in the initial phase that uses so called “IsGiven” values.
- Internal values of the integration methods are used.

The C++ library which was thoroughly tested is the base for creating SPICE3 models in Modelica.

4 Modelling steps towards Modelica

The C++ functions are constructed to calculate the currents of an equivalent circuit starting with given node voltages. The currents are used inside SPICE3 for filling a linear system of equations.

Starting with the C++ equivalent circuit a Modelica top level model (figure 3) is constructed with electrical pins for connecting. The components of the top level model represent special (e.g. semiconductor) effects (e.g. channel current). Using the pin voltages the components call in their algorithm part typically a hierarchy of functions for the calculation of currents.

There are several steps of modelling semiconductor devices [8], which are described in the following:

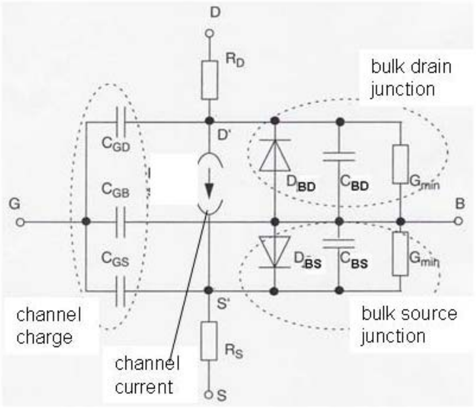


Fig. 3 MOS top level model

1. Construction of top level model

In Modelica every semiconductor device gets a so called top level model which calls the semiconductor functions and can be connected to other models via its connectors. This top level model is the semiconductor device component which will be applied by the user. As in SPICE3 the top level model is adjustable by choosing parameters. Within the top level model the branch currents are calculated using the existing voltages and parameters with the help of functions.

The physical values that are calculated in the C++ semiconductor devices are prepared to be inserted into the linear system of equations like in the SPICE3 simulator. Such a system of equations cannot be addressed in Modelica usually. Only the relation between voltage and current at the interfaces of the model is of interest (terminal behaviour [9]). The voltages at the pins, that are the results of the simulation algorithm, are gripped and given to functions that calculate currents and other values.

The top level model that can be connected and provided with parameters is extracted from the functionality in C++ (figure 3).

2. Parameter handling

The behaviour of a transistor is determined by its parameters significantly. Parameters are e.g. the physical dimension, the temperature or the oxide thickness. Before the simulation the Boolean value “IsGiven” is analysed, which gives the information whether a parameter was set by the user its default value is used.

In the C++ library the parameters are handled as a string. If a parameter is needed when calling a method, the string is searched for a value of the parameter. This way is also possible in Modelica, but it is not usual. In Modelica all parameters are provided in a parameter list, where the user can adjust the parameters. It is desirable that in the Modelica concept a possibility exists that decides whether a parameter is set by the user (“IsGiven”) or not. Unfortunately such a possibility does not exist yet. That is why a temporary solution was chosen. The default value of parameters whose “IsGiven” value is of importance, is set to a very big negative value (-1e40), because that values does make no sense as a normal parameter value. Afterwards it is checked if the value of a parameter is not equal to -1e40. In that case it is assumed that the parameter was given by the user and consequently “IsGiven” is true. Otherwise the parameter gets it default value. This solution is only preliminary and will be improved as soon as Modelica delivers the necessary possibilities. The example in figure 4 shows the parameter handling with the parameter phi.

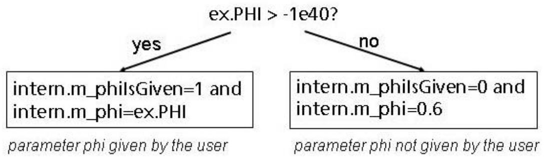

ex.PHI > -1e40?
yes
no
intern.m_philsGiven=1 and intern.m_phi=ex.PHI
intern.m_philsGiven=0 and intern.m_phi=0.6
parameter phi given by the user
parameter phi not given by the user

Fig. 4 parameter handling

As described in section two the parameters in SPICE3 are divided into two groups, device and technology parameters. In Modelica the device parameters are part of the semiconductor model. The technology parameters are collected in a record. This record is a parameter for all semiconductor devices. This courses that also the technology parameters can be adjusted in every single model separately which is not intended in SPICE. But in some cases it could make sense.

Furthermore the record with the technology parameters is available in the highest level of the circuit. Every semiconductor device gets the record as a parameter. So the components of the record can be adjusted in a global way for each device in the circuit. Another possibility to provide the technology record as global is to define a model in the circuit level that inherits the properties of the MOSFET where the desired parameters are unchangeably included. Both possibilities force the user to work within the source code. For untrained users it would be better to work in the graphical modus of Dymola and giving each single semiconductor device parameter its value by clicking on the device and inserting the parameter value in the prepared list.

3. Transformation of C++ library data structure

In the C++ source library the data are concentrated in classes and located in the according header file of the semiconductor elements. For each parameter a variable "parameterValue" exists that gets the particular value of the parameter. In Modelica the parameters are concentrated in records because these are the equivalent classes to the C++ classes with the parameters. Records were developed in Modelica to collect and administrate data and to instantiate data all at once. Inside the records the data get their default values. With a function call all data that are located inside a record can be accessed. Parameters that are needed for more than on model are collected in a higher level record which is inherited to the lower level records of the single models (figure 5).

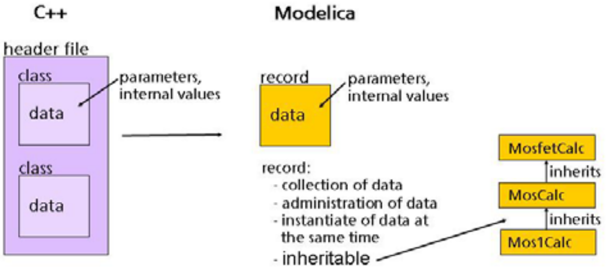


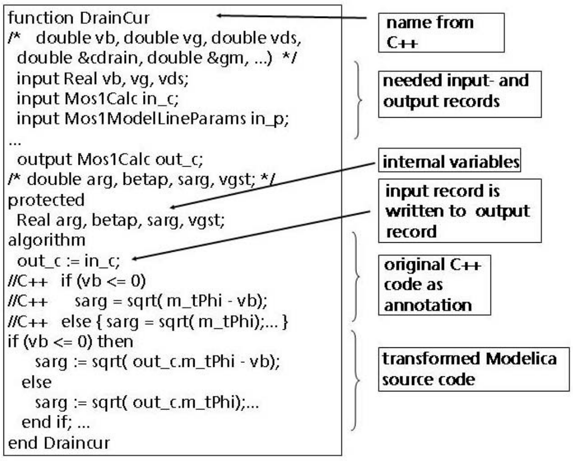
Fig. 5 Transformation of C++ data structure

4. Transformation of C++ library methods

The C++ library of the semiconductor elements of SPICE3 contains beyond parameters and variables that are concentrated in classes, also of a huge number of methods that need to be transformed. Within the transformation it is important, that the structure of the C++ library also remains in Modelica with the aim to recognise the C++ code.

Each semiconductor element in the C++ library becomes to a top level model in Modelica. Inside the top level model functions are called, that calculate both the parameters and the currents at the pins of the model. These functions need to be extracted from C++ and transformed to Modelica. In the C++ library a hierarchy of classes exists where often more than one method calculate one physical effect. Like in a tree structure one method calls another method that itself also calls another method and so on.

The transformation starts with the transfer of the name of the C++ method to the according Modelica function. That function has to be included into a package that has the name of the C++ class where the appropriate method came from. In the second step the parameters and values that are concentrated in classes in C++ are transformed to Modelica into records. In the third step the function text that changes the values in the classes respectively the records has to be directly red of the C++ code and transformed to Modelica where the original C++ names are used. Within that step the C++ code is included into the Modelica code as annotation to recognise the C++ code (figure 6).



```
function DrainCur
/*   double vb, double vg, double vds,
  double &cdrain, double &gm, ...)  */
  input Real vb, vg, vds;
  input Mos1Calc in_c;
  input Mos1ModelLineParams in_p;
...
  output Mos1Calc out_c;
/* double arg, betap, sarg, vgst; */
protected
  Real arg, betap, sarg, vgst;
algorithm
  out_c := in_c;
//C++   if (vb <= 0)
//C++      sarg = sqrt( m_tPhi - vb);
//C++   else { sarg = sqrt( m_tPhi);... }
if (vb <= 0) then
     sarg := sqrt( out_c.m_tPhi - vb);
   else
     sarg := sqrt( out_c.m_tPhi);...
   end if; ...
end Draincur
```

name from C++
needed input- and output records
internal variables
input record is written to output record
original C++ code as annotation
transformed Modelica source code

Fig. 6 Transformation of a function

5. Code revision

After a SPICE3 model was transformed into Modelica the source code is checked again with the aim to make is more effective. One point is to include the Modelica operator “smooth”. Within this all conditions (if) are checked to find out if it is continuous, also in higher derivations. In that case “smooth” avoids the not needed breaks of the analogue simulation algorithm. With this approach the performance of the simulation can be increased very much.

It also has to be checked if methods were transformed to Modelica that are actually not needed, to simplify the Modelica code.

The system of equations that is built in SPICE3/C++ is not used in Modelica as well as internal values of the integration method that is in connection to the SPICE3 solution algorithm. The calculation of the Jacobians that is done in SPICE3/C++ is also not used in Modelica. It was tried to ensure to transform only the functional aspects of the models to Modelica. In this way a mixture between model equations and numerical solution algorithms like in SPICE3 is avoided.

5 Structure of SPICE3 library

The current SPICE3 library contains the packages Basic, Interfaces, Semiconductors, Sources, Examples, Repository and Additionals (as can be seen in figure 7).

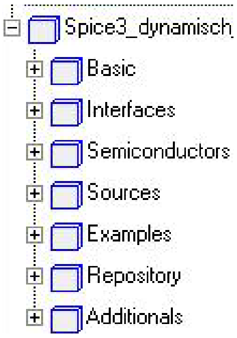
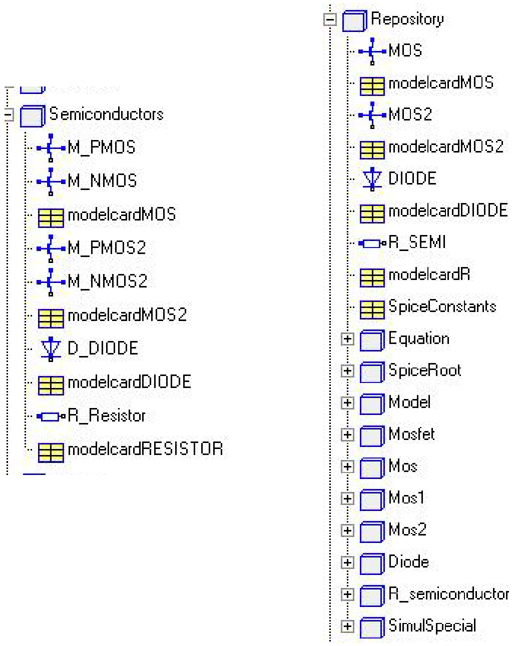


Fig. 7 SPICE3 library overview

The package Basic contains basic elements like resistor, capacitance, inductivity and controlled sources. In the package Sources there are the voltage- and current sources transformed from SPICE3. The package examples include some example circuits, to help the user getting a feeling of the behavior of the library and their elements.

Only the semiconductor models are written using the converted C++ library. The packages Semiconductor and Repository are related to each other very closely. In the package Repository the semiconductor devices and their model cards from SPICE3 are modeled. The necessary function and records are also in this package. This package is not for user access. The semiconductor package contains clearly arranged the offered semiconductor devices and their model card records for easy usage. The user should take the models out of this package. Via inheritance these models are connected to the repository. That’s why the user does not have to access to the repository directly.



- Semiconductors
  - M_PMOS
  - M_NMOS
  - modelcardMOS
  - M_PMOS2
  - M_NMOS2
  - modelcardMOS2
  - D_DIODE
  - modelcardDIODE
  - R_Resistor
  - modelcardRESISTOR

- Repository
  - MOS
  - modelcardMOS
  - MOS2
  - modelcardMOS2
  - DIODE
  - modelcardDIODE
  - R_SEMI
  - modelcardR
  - SpiceConstants
  - Equation
  - SpiceRoot
  - Model
  - Mosfet
  - Mos
  - Mos1
  - Mos2
  - Diode
  - R_semiconductor
  - SimulSpecial

Fig. 8 Packages Semiconductor and Repository

The package Additionals contains the polynomial sources like they are available in SPICE2 or PSPICE. Other models that are not from SPICE3 can be collected here.

6 Example

In this section a Modelica model of the inverter circuit shown in figure 1 is developed. The following two approaches are important.

Graphical composition

The SPICE3 library models are composed and connected with the graphical possibilities of the simulator. Figure 9 shows such a circuit (Dymola).

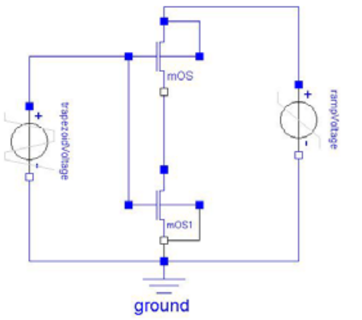


Figure 9: Graphically composed inverter circuit

Textual composition

Starting with the SPICE3 netlist (figure 2) the Modelica inverter model can be generated directly without using graphical information. This feature is important, because the SPICE3 netlists that exist for many circuits, modules and complex circuit elements, should also be available in Modelica. In the following example of two inverters, a way of transforming is shown. First of all the two source codes are opposed to each other.

SPICE3	Modelica

`inverter`	`model inverter`
	`  Spice3.Basic.Ground g;`

`Mp1 11 1 13 11`
`+ MPmos`
	`  Spice3…M`
	`Mp1(mtype=true,`
	`M(GAMMA=0.37));`

`Mn1 13 1 0 0`
`+ MNmos`
	`  Spice3…M`
	`Mn1(M(LAMBDA=0.02,`
	`GAMMA=0.37));`

`Vgate 1 0 PULSE`
`+      (0 5 2s 1s)`
	`  Spice3…V_pulse`
	`vgate(V1=0, V2=5, TD=2,`
	`TR=1);`

`Vdrain 11 0`
`+ PULSE(0 5 0s`
`+ 1s)`
	`  Spice3…V_pulse`
	`vdrain(V1=0, V2=5, TD=0,`
	`TR=1);`

	`  Spice3.Interfaces.Pin`
	`p_in, p_out;`

`.model MPmos PMOS`
`+ (gamma=0.37)`
	`protected`

`.model MNmos NMOS`
`+ (gamma=0.37`
`+ lambda=0.02)`
	`  Spice3.Interfaces.Pin`
	`n0, n1, n11, n13;`
	`equation`
	`  connect(p in, n1);`
	`  connect(p_out, n13);`

`.tran 0.01 5`	`  connect(g.p, n0);`

`.end`	`  connect(vdrain.n,n0);`
	`  connect(vdrain.p,n11);`

	`  connect(Mp1.NB,n11);`
	`  connect(Mp1.ND, n11);`
	`  connect(Mp1.NG, n1);`
	`  connect(Mp1.NS, n13);`
	`  connect(Mn1.NB,n0);`
	`  connect(Mn1.ND, n13);`
	`  connect(Mn1.NG, n1);`
	`  connect(Mn1.NS, n0);`

	`end inverter;`

Fig. 10 Inverter model in SPICE and Modelica

The creation of the Modelica texts requires the following steps:

1. The obligate name of the Modelica model can be derived from the first line in the SPICE3 netlist.

2. It is necessary to create entities of each circuit element of the SPICE3 netlist and to provide them with parameters, e.g. the SPICE3 line

```
Vdrain 11 0 PULSE(0 5 0 1)
```

is in Modelica

```
V_pulse vdrain(V1=0,V2=5,TD=0,TR=1);
```

3. For each node number in SPICE an internal pin has to be created in Modelica, e.g. for the node number 2 in SPICE, the Modelica line would be:

```
protected Spice3.Interfaces.Pin n2;
```

The "n" is necessary because in Modelica a single number is not a name.

4. According to the netlist the internal pins have to be connected to the circuit element, e.g.

```
connect (Mp1.ND, n11);
```

5. In the last step the external connectors have to be created and connected to the according internal connectors, e.g.

```
Spice3.Interfaces.Pin p_in, p_out;
connect(p_in, n1); connect(p_out, n2);
```

Concerning the semiconductor elements the model cards have to be transformed to Modelica. Two ways seem to be possible.

Separate record

The records of the technology parameters MPmos and MNmos are instances of the record model card in the model inverter for each transistor (Mp1, Mp2,…).

```
model inverter
  parameter …modelcardMOS Pmos(GAMMA=0.37);
  parameter …modelcardMOS Nmos(LAMBDA=0.02,
                               GAMMA=0.37);
  Spice3.Basic.Ground g;
  Spice3…MOS Mp1(mtype=1,modelcard=MPmos);
  Spice3…MOS Mp2(mtype=1,modelcard=MPmos);
  Spice3…MOS Mn1(modelcard=MNmos);
  Spice3…MOS Mn2(modelcard=MNmos);
  …
end inverter;
```

Extended model

For each technology parameter set a separate model is created. This model extends the transistor M that was defined in Modelica. Within this way the needed technology parameters are given.

```
model inverter
 model MPmos
  Spice3.Semiconductors.modelcardMOS M
              (GAMMA=0.37);
  extends Spice3…MOS(final type=1,
                           modelcard=M);
 end MPmos;
 model MNmos
  Spice3.Semiconductors.modelcardMOS M
            (LAMBDA=0.02, GAMMA=0.37);
  extends Spice3…MOS(final mtype=0,
                           modelcard=M);
 end MNmos;
  Spice3.Basic.Ground g;
  MPmos Mp1;
  MPmos Mp2;
  MNmos Mn1;
  MNmos Mn2;
  …
end inverter;
```

With the help of these two possibilities the user can give many transistors the same technology parameters like it can be done in SPICE3.

The textual composition could be done automatically by a special translator. The aim is to have such a translator in the future, maybe in the Modelica language.

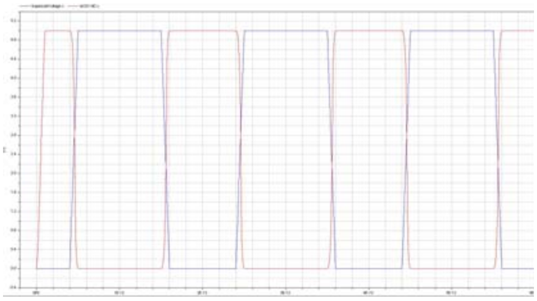


Fig. 11 Inverter simulation result

The result of the Dymola simulation of the inverter circuit is in accordance with the SPICE3 simulation result.

7 Test and Comparison

To verify the transformed models several different test steps were arranged. It is important that the Modelica library is in accordance with SPICE3. Since the C++ library was tested very intensively it can be assumed that it is correct. That is why SPICE3 as well as the C++ library are the base of the tests.

The C++ code was included to the Modelica code as comment. This allows the visual comparison of the source codes.

Single values of currents or other variables (e.g. capacitances) are compared between the Modelica simulation and the simulation of the C++ model library. This approach is very complex and time consuming. Therefore it is only done when the reason of known differences has to be found out.

The terminal behavior is compared between Modelica and SPICE3. Therefore single semiconductor devices are connected to voltage sources to calculate the current-voltage characteristics.

In the next step complex circuits are created with several semiconductor elements and the results are compared between SPICE3 and Modelica. Such circuits are the base for a collection of circuits for regression tests, which are maintained to ensure the correctness of the library in future.

A comparison between the Spice3 library for Modelica and the BondLib in Dymola showed that the two libraries have nearly the same results and performance. For the comparison three circuits were used (NAND, NOR, double Inverter). The following table 1 shows the results in detail:

Tab. 1 Comparison between BondLib and Spice3 Library for Modelica

	NAND		NOR		Inverter	
	SPICE3lib	BONDlib	SPICE3lib	BONDlib	Spice3lib	BONDlib
Before translating						
scalar unknowns	873	10.677	873	10.673	870	10.860
variables	1.157	12.136	1.157	12.132	1.152	12.315
After translating						
parameter depending	8	2.005	8	2032	8	2.030
time-varying variables	826	688	826	687	824	687

As it can be seen in the table 1, the Spice3 library has much less variables then the BondLib before translation of the model. After the model has been translated, the BondLib has little less variables than the Spice3 library. This shows that the simplification algorithms of Dymola work better for the BondLib.

For the double Inverter circuit the output voltage of the original SPICE3 simulator, the BondLib and the Spice3 library for Modelica are shown in the following figures 12/13.

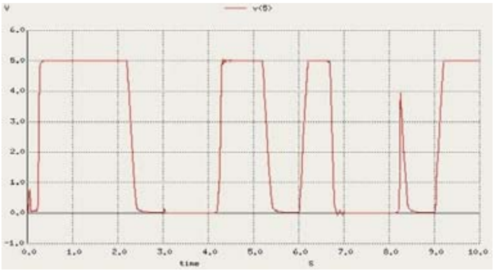


Fig. 12 Output voltage original SPICE3

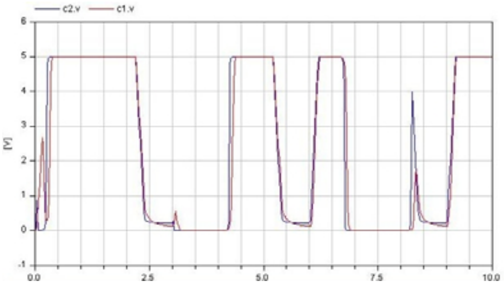


Fig 13 Output voltage BondLib and Spice3 library for Modelica

Each figure shows the output voltage of the second inverter. In figure 12 the result of the original SPICE3 simulator is shown. The results of the three simulators are nearly the same.

8 Conclusions

In this paper a concept was described to transform the procedural implemented SPICE3 models, which are directly extracted from the original SPICE3 source code, to declarative described models for Modelica. Therefore a list of modeling steps was elaborated and applied to transform several semiconductor devices from SPICE3 to Modelica whereas the parameter handling was focused on. The result is a SPICE3 library for Modelica which contains the general devices and first semiconductor devices.

A disadvantage of the Spice3 library compared with the Bondlib is that the Spice3 library has no heatport. At the moment it is possible to simulate with a fixed parameter “Temp”. It has to be figured out how this parameter can be made variable and time dependent in the future.

Further steps for the improvement of the SPICE3 library are:

- developing a method for automatically transforming SPICE3 netlists to Modelica
- increasing the performance of the Modelica models (e.g. application of the smooth operator)
- parameter treatment ("IsGiven") has to be simplified
- adding SI units to the Modelica models
- large number of tests
- testing large circuits (many devices)
- inclusion of further SPICE3 models
- intensively testing, comparison to SPICE3
- inclusion of some PSPICE model features
- comparison with existing electronic libraries
- adding a heatport

Acknowledgement

This research was founded by the European ITEA2, projects EUROSYSLIB and MODELISAR.

References

[1] Clauß, C.; Leitner, T.; Schneider, A.; Schwarz, P.: *Modelling of electronic circuits with Modelica.* Proc. Modelica Workshop, Lund, Sweden, Oct. 2000, 3-11.

[2] Cellier, F.E.; Clauß, C.; Urquia, A.: *Electronic circuit modeling and simulation in Modelica.* EUROSIM 2007, Ljubljana, Slovenia, 9.-13. Sept. 2007.

[3] Urquia, A.; Martin, C.; Dormido, S.: *Design of SPICELib: a Modelica Library for modeling and analysis of electric circuits.* Mathematical and Computer Modelling of Dynamical Systems, 11(1)2005, 43-60.

[4] Cellier, F.E.; Nebot, A.: *The Modelica bond graph library.* Proc. 4th Int. Modelica Conference, Hamburg-Harburg, Germany, 1, 2005, 57-65.

[5] SPICE Version 3e Users Manual, 1991

[6] Leitner, T.: *Entwicklung simulatorunabhängiger Modelle für Halbleiter-Bauelemente mit objektorientierten Methoden.* Chemnitz, Technische Universität, Diss., 1999.

[7] Leitner, T.: *A new approach for semiconductor models basing on SPICE model equations.* Proc. ECS'97, Bratislava, Slovakia, 4./5. 9. 1997, 119-123.

[8] Majetta, K.: *Entwicklung und prototypische Umsetzung eines Konzeptes für eine Modelica-Bibliothek von SPICE-Halbleiterbauelementen und Erarbeitung einer Teststrategie.* Dresden, Berufsakademie Sachsen, Dipl., 2008.

[9] Clauss, C.; Haase, J.; Kurth, G.; Schwarz, P.: *Extended Admittance Description of Nonlinear n-Poles.* AEÜ, Vol. 49 (1995) 2, 91-97.

Automatic modelling of Power Electronic Converter, Average model construction and Modelica model generation

Loig ALLAIN
LMS
La Cité Internationale, 84 quai Charles de Gaulle,F- 69006 LYON
loig.allain@lmsintl.com

Asma MERDASSI, Laurent GERBAUD
Seddik Bacha
G2ELAB
UMR 5269 INPG/UJF-CNRS
ENSE3, Domaine Universitaire, BP 46 SAINT MARTIN D'HERES F-38402,
Asma.Merdassi@g2elab.grenoble-inp.fr,
Laurent.Gerbaud@g2elab.grenoble-inp.fr,
Seddik.Bacha@g2elab.grenoble-inp.fr

Abstract

Development in semi-conductor production and the control of static converters leads to an increase use of these devices in classical electrical motor control application. But, the simulation of these elements in a complete system involves longer computation time in order to take into account the impact of switching on the global system. Using average models to simulate them brings a drastic decrease of computation time. Such models do not take into account switching. They are a good compromise between complexity, computing time and acceptable accuracy at a system level. Even if the method to build these models is well identified, it still requires a lot of manipulation to get from the very structure description to the simulation code. This paper deals with a method that automates the average modeling of power electronic converter. The associated tool prototype (AMG) that has been implemented is also presented and some simulation results are shown using the Imagine.Lab AMESim[1] and its Modelica Platform.

Keywords: code generation; average model; power electronics, Imagine.Lab AMESim

1 Introduction

Nowadays, numerous electrical actuators and drives are based on the association of power electronics and electrical motors driven by controller. Such association guarantees better time response and better efficiency for the whole actuator. Indeed, the control of electronic commutations allows a fine control of the power supply of motors.

Figure 1presents the most classical structure of an electromechanical actuator.

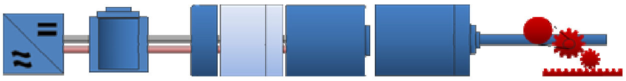


Figure 1: General structure for electro-mechanical actuator

The simulation of Control-Converter-Machine sets is a CPU time consuming task beacause it involves the complete simulation of power static converter and especially of power switch commutations. Indeed, classical PWM frequency may be about 500Hz to 50kHz, and has to be compared to the time response of the whole system that could be about minutes or more. So, it is important to find a formulation for power electronic converter that reduces the CPU time need.

The G2ELAB has a strong experience in this area, and is involved in the automatic analysis and simulation of power electronics converters [5][7]. Besides, a strong knowledge of engineer modeling experience and its specifities lead to the development of software tools to store and manage the engineer knowledge thanks to an object oriented language [9]. In this way, these developments allowed to explore new concepts about average modeling of power electronic structures. In an other direction, LMS-Imagine has a long experience in multiphysical system modeling, the code of his known software Imagine.Lab AMESim has been used world around to simulate some complex, multi-physical devices[4]. Using the facility of Imagine.Lab Modelica Platform, the G2ELAB has been able to develop and test its approach applied to generate Modelica models for power electronic converter.

Obviously, the software tools presented hereafter are not linked to the Imagine.Lab platform. As the pro-

DOI: 10.3384/ecp09430092

duced code is fully Modelica compliant, it may be used in any Modelica tool.

After, an introduction to average modeling in power electronics, the paper explores the way to automate code generation for average model in power electronics [7], [8]. The developed tool (namely AMG) deals with the following kinds of static converters: DC/DC (as in [2], [3], [4], [5]) but also DC/AC and AC/AC. The generated Modelica code is shortly presented and used to create a simulation model within the AMESim.Lab Modelica Platform. The generated code is presented for two static power converters: a three phase voltage inverter (DC/AC conversion) and a boost converter (DC/DC conversion). Every example aims at emphasizing interest point of static converter average models. Then, some numerical results are shown and discussed and the numerical efficiency and reliability are discussed with regards to classic representation. Eventually, advantages and drawbacks of average models are discussed as well as the interest of such an approach.

2 Average Model of power electronics converters

2.1 Electrical Actuator and Power Electronics

An electrical actuator aims at converting electrical energy into mechanical one. One of the eldest technologies used is the electrical motor. Using Direct Current (DC), or Alternating Current (AC) as an electrical power source, it produces a mechanical torque through electromagnetic transformation. Most common electrical motors can be driven through a control of the voltage of their power supply. This is achieved using power electronics switching to produce whether an equivalent voltage value or a given frequency [5]. Designing systems based on electronic power supply requires being able to simulate them. A well known drawback is that the classical simulation of power electronics must take into account the switching frequency and the transient phenomena of electronic switches. This requires a large amount of CPU time and may neither compliant with Real Time Simulation nor with controller design steps. In order to reduce the needed CPU time to simulate such complex associations, average model of power supply can be used.

Moreover, it is obvious for every electromechanical designer that, indirect conversion is a widely used technology to control both the amplitude and the frequency of applied voltage. As a consequence, one has to face the tricky problem of increasing the complexity of a whole system, and so the required simulation time.

2.2 Average Model of power Electronics converter

The purpose of the hereafter presented tool is not to question a well know process leading from a circuit description to an average model [2] [3] [4] [5] [6].

The paper aims at automating fully the process [7], [8] for DC/DC, DC/AC and AC/AC static converters. Please note that, in the paper, only controlled commutations are considered, so discontinuous conduction is not taken into account.

Here, a pure symbolic method (exact or topological model) is chosen to obtain average models.

3 Average model construction and code Generation

3.1 The generated models by AMG

3.1.1 Exact model

For an operating mode of a static converter, N possible configurations during the commutation period T are given.

The global state model for every topology (indexed by i) is:

$$\frac{dx}{dt}=\sum_{i=1}^{N}\left(A_i\cdot x+B_i\cdot e\right)\cdot f_i \quad (1)$$

Where:

- N: the number of topologies (configurations) that happen during the switching period T.
- Ai : the state matrix for the ith configuration.
- Bi : the input matrix.
- e : the external sources.

For each switching cell, an equivalent control function is assigned: $h_i(t)\in\{0,1\}\text{or}\{-1,1\}$ that depends on f_i (a configuration validation function of *i*):

$$f_i=\begin{cases}1 & for \quad t\in[\ t_{i-1},t_i[\ \ ;i=1,..,N\\ 0 & elsewhere\end{cases}$$

The global behavior of the static converter is formulated as equation (2):

$$\frac{dx}{dt}=Ax+\sum_{i=1}^{N}h_i(B_i x+b_i)+d \qquad (2)$$

Where:

- N: the number of topologies that happen during the commutation period.
- x : the state vector (dim[n]).
- A : the state matrix (dim[n,n]).
- B_i, b_i : the input matrixes (dim[n,n], dim[n,1]).
- d (dim[n,1]).

A, B_i, b_i are independent of x and the h_i.

3.1.2 Large Signal Average Model

The large signal average model transforms an exact model into an averaged one.
To calculate such an averaged model, the following properties are used for the continuous state:

$$\begin{cases} \left\langle \dfrac{dx}{dt} \right\rangle_0 = \dfrac{d}{dt}\langle x \rangle_0 \\ \langle x \cdot y \rangle_0 = \langle x \rangle_0 \cdot \langle y \rangle_0 \end{cases} \quad (3)$$

By applying equations (3) on equation (2), the equation of the large signal average model is deduced:

$$\frac{d}{dt}\langle x \rangle_0 = \langle A \rangle_0 \cdot \langle x \rangle_0 + \sum_{i=1}^{N} \left(\langle h_i \rangle_0 \cdot \langle B_i \rangle_0 \cdot \langle x \rangle_0 + \langle b_i \rangle_0 \cdot \langle h_i \rangle_0 \right) + \langle d \rangle_0 \quad (4)$$

In general, with linearization hypothesis, matrixes A, B_i and b_i are time invariant. So (4) can be simplified as (5):

$$\frac{d}{dt}\langle x \rangle_0 = A \cdot \langle x \rangle_0 + \sum_{i=1}^{N} \left(\langle h_i \rangle_0 \cdot B_i \cdot \langle x \rangle_0 + b_i \cdot \langle h_i \rangle_0 \right) + \langle d \rangle_0 \quad (5)$$

3.1.3 Small Signal Average Model

This model is also called the small signal state space model. It linearises the exact model and does not include any nonlinear elements.
This model is calculated by a linearization around the balance operating point (xe, ue) of the static converter.

This point is calculated for $\frac{dx}{dt} = 0$.

There, the modelling approach begins with equation (6).

$$\begin{cases} \dfrac{dx}{dt} = f(x,u) \\ y = g(x,u) \end{cases} \quad (6)$$

where:

- y : the output vector (dim[q])
- u : the input vector (dim[p]).

Equation (6) is transformed into the small signal average model (7):

$$\begin{cases} \dfrac{d\tilde{x}}{dt} = \tilde{A} \cdot \tilde{x} + \tilde{B} \cdot \tilde{u} \\ \tilde{y} = \tilde{C} \cdot \tilde{x} + \tilde{D} \cdot \tilde{u} \end{cases} \quad (7)$$

The matrixes of the small signal state space model are calculated by using the following expressions:

$$\tilde{A} = \left(\frac{\partial f(x,u)}{\partial x} \right)_{x_e,u_e} ; \tilde{B} = \left(\frac{\partial f(x,u)}{\partial u} \right)_{x_e,u_e}$$

$$\tilde{C} = \left(\frac{\partial g(x,u)}{\partial x} \right)_{x_e,u_e} ; \tilde{D} = \left(\frac{\partial}{\partial u} g(x,u) \right)_{x_e,u_e}$$

where:

- $\begin{cases} \tilde{u} = u - u_e \\ \tilde{x} = x - x_e \\ \tilde{y} = y - y_e \end{cases}$
- $\tilde{A}$: the state matrix (dim [n,n]).
- $\tilde{B}$: the input matrix (dim[n,p]).
- $\tilde{C}$: the output matrix (dim[q,n]).
- $\tilde{D}$: the feedforward matrix (dim[q,p]).

The transfer function is also deduced from (7):

$$\frac{y(s)}{u(s)} = \tilde{C} \cdot (s \cdot I - \tilde{A})^{-1} \cdot \tilde{B} + \tilde{D} \quad (8)$$

- I : The matrix identity (dim [n, n]).

3.1.4 Generalized Average Model

This model is applied on static converters which have one or several alternative variable states.
The general formulation of the state derivatives is:

$$\left\langle \frac{dx}{dt} \right\rangle_k = \frac{d\langle x \rangle_k}{dt} + j \cdot k \cdot \omega \cdot \langle x \rangle_k \quad (9)$$

where:

- k: The range of harmonic.

Equation (9) becomes:

- For alternative state variables (first harmonic, i.e.: k=1):

$$\left\langle \frac{dx}{dt} \right\rangle_1 = \frac{d\langle x \rangle_1}{dt} + j \cdot \omega \cdot \langle x \rangle_1 \qquad (10)$$

- For continuous states (k=0):

$$\left\langle \frac{dx}{dt} \right\rangle_0 = \frac{d\langle x \rangle_0}{dt} \qquad (11)$$

The exact model of equation (2) becomes:

$$\begin{aligned} &\frac{d}{dt}\langle x \rangle_k = -j \cdot k \cdot \omega \cdot \langle x \rangle_k + A \cdot \langle x \rangle_k \\ &+ \sum_{i=1}^{N} \left(B_i \langle x \cdot h_i \rangle_k + \langle b_i \cdot h_i \rangle_k \right) + \langle d \rangle_k \end{aligned} \qquad (12)$$

The development of its terms is made by using (13):

$$\begin{aligned} &\left\langle \prod_{i=1}^{N} a_i \right\rangle_k = \sum \left(\prod_{i=1}^{N} \langle a_i \rangle_{j_i} \right) \\ &\begin{cases} (j_i) \in \{-1,0,1\} \\ \sum^{N} j_i = k \end{cases} \end{aligned} \qquad (13)$$

The expressions of the alternative state variables are written under their complex shape (14):

$$\begin{cases} \langle a \rangle_1 = \mathrm{Re}(a) + j\,\mathrm{Im}(a) \\ \langle a \rangle_{-1} = \langle a \rangle_1^* = \mathrm{Re}(a) - j\,\mathrm{Im}(a) \end{cases} \qquad (14)$$

In general, the applications of such a model are limited to the fundamental (k=1) and averaged values (k=0).

The large signal average model is a specific application of the generalized average model.

3.1.5 The Equivalent Average Generator

The approach used to make such a model is described in Figure 2.

The equivalent average generator eliminates the discontinuous variables. The state variables are separated into continuous (slow) and alternatives (fast) variables.

The state vector is composed of two sub-vectors:

$$x = [x_s ; x_f]$$

where:

- x_s : the vector of slow state variables.
- x_f : the vector of fast state variables.

Now, the paper presents the steps of calculation:

- The behavior of the slow state variables given by equation (15):

$$\begin{aligned} &\frac{dx_s}{dt} = A_s^s \cdot x_s + A_s^f \cdot x_f \\ &+ \sum_{i=1}^{N} \left(B_{is}^s \cdot x_s + B_{is}^f \cdot x_f \right) \cdot h_i + b_{is} \cdot h_i + d_s \end{aligned} \qquad (15)$$

- The behavior of the slow state variables given by equation (16):

$$\begin{aligned} &\frac{dx_f}{dt} = A_f^s \cdot x_s + A_f^f \cdot x_f \\ &+ \sum_{i=1}^{N} \left(B_{if}^s \cdot x_s + B_{if}^f \cdot x_f \right) \cdot h_i + b_{if} \cdot h_i + d_f \end{aligned} \qquad (16)$$

- For the continuous state variables x_s, the large signal average model is applied to (15):

$$\begin{aligned} &\frac{d\langle x_s \rangle_0}{dt} = A_s^s \cdot \langle x_s \rangle_0 \\ &+ \sum_{i=1}^{N} \left(B_{is}^s \cdot \langle x_s \cdot h_i \rangle_0 + B_{is}^f \cdot \langle x_f \cdot h_i \rangle_0 \right) + b_{is} \cdot \langle h_i \rangle_0 + \langle d_s \rangle_0 \end{aligned} \qquad (17)$$

- For the alternative state variables x_f the generalized average model (16) (limited in this example to the first harmonic) is used and x_f is considered to be at its steady state:

$$\begin{aligned} &\frac{d\langle x_f \rangle_1}{dt} = -j\omega \cdot \langle x_f \rangle_1 + A_f^f \cdot \langle x_f \rangle_1 + A_f^s \cdot \langle x_s \rangle_1 \\ &+ \sum_{i=1}^{N} B_{if}^f \langle x_f \cdot h_i \rangle_1 + B_{if}^s \cdot \langle x_s \cdot h_i \rangle_1 + b_{if} \cdot \langle h_i \rangle_1 + \langle d_f \rangle_1 \end{aligned} \qquad (18)$$

- Equation (18) is solved by assuming that all the continuous state variables are constant.

The development of the terms: $\langle x_f \cdot h_i \rangle_0$, $\langle x_s \cdot h_i \rangle_0$, $\langle x_f \cdot h_i \rangle_1$ and $\langle x_s \cdot h_i \rangle_1$ is made by using (13).

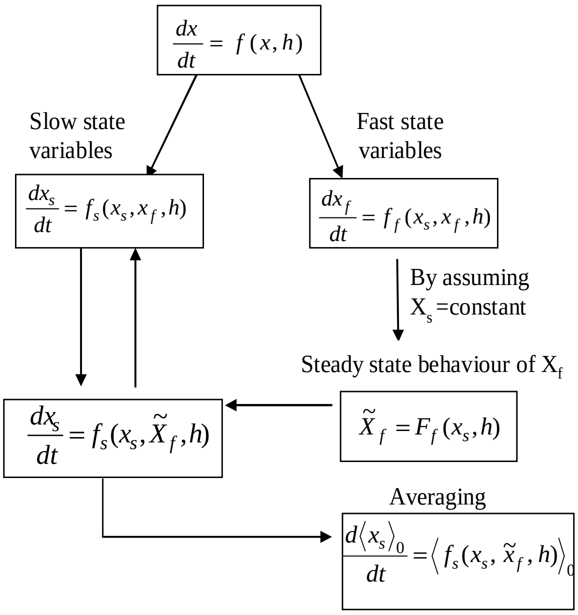

$\frac{dx}{dt} = f(x,h)$
Slow state variables
Fast state variables
$\frac{dx_s}{dt} = f_s(x_s, x_f, h)$
$\frac{dx_f}{dt} = f_f(x_s, x_f, h)$
By assuming X_s=constant
Steady state behaviour of X_f
$\frac{dx_s}{dt} = f_s(x_s, \tilde{X}_f, h)$
$\tilde{X}_f = F_f(x_s, h)$
Averaging
$\frac{d\langle x_s\rangle_0}{dt} = \langle f_s(x_s, \tilde{x}_f, h)\rangle_0$

Figure 2: Required operations to get an equivalent average generator

To obtain the equivalent average generator model, the steady state of the alternative variables x_f (in equation 18) is obtained when $\frac{d\langle x_f\rangle_1}{dt} = 0$ and replaced this steady state (stationary behavior) in equation (17).

3.2 Overview of code generation process

A large subset of the required operation as early presented can be automated; Figure 3 presents the general process that has been implemented in AMG generator.

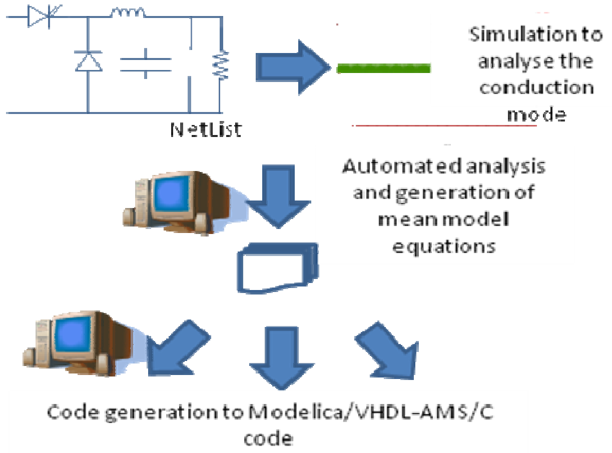

NetList
Simulation to analyse the conduction mode
Automated analysis and generation of mean model equations
Code generation to Modelica/VHDL-AMS/C code

Figure 3: Average model generation

Each model is entirely made in an automatic way by the generator AMG, from an initial description of a static converter structure:

- a netlist file: here the AMG user describes its structure in P-Spice by using a dedicated simplified component library); any tool giving a P-Spice netlist is also available.
- the switching mode, i.e. the cyclic sequence of switched configurations in an operating period,
- the switch control, mainly the control links of switches.

This description is given by the AMG user and induces a great "*a priori*" on the behaviour of the described static converter.

The average models are made from the state equations of every configuration of the converter operating mode.

3.3 Global model of static converters and causality

In order to automatically build the Average or exact model for static power converter, the kind of the connected electrical sources (or loads) must be defined. One could understand that a kind of causality is "de facto" imposed to the converter, but it should be noted that the resulting equations could be dealt in the same way that any other Modelica model. The point is that the limits of the converter have to be carefully chosen according to the kind of average model to generate:

- An exact model shall gather only the electronics switches and need a "classical" control law. This law shall be strictly identical to the one used for non average converter
- An average model aims at building the average value of state variable. As a consequence, one must cautious to take into account the component introducing state variable in the final description. Moreover the required input is also different as it should also be averaged. This is a fully explained with the boost converter example, below.

The model of the static converter is linked to the models of the other components of the system to simulate.

The causality on the pluging points is required **only to build the model**. This causality is cancelled by connecting the generated models with other models. This is useful to create the models of static converter that fed an electrical machine, without knowing the machine model that is generally not available in a circuit shape (see Figure 4). Eventually, the true causality is "set" by the Modelica compiler while it reorganizes mathematical relations to produce an executable code.

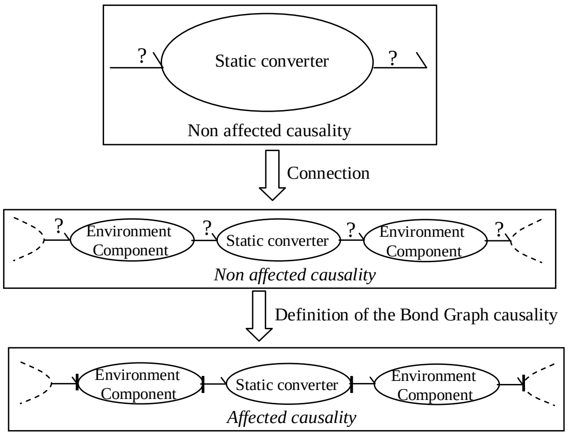


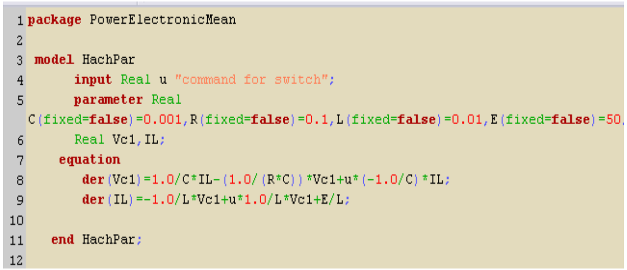
Figure 4 From a non oriented representation to an oriented representation: Bond Graph representation

One has to be cautious, while assessing the causality for electrical connectors, to respect the definition given in Bond Graph representation for the electrical field. In other words, the environment of a static converter must be translated into equivalent sources: a voltage source or a current source is imposed at every connection point.

3.4 Modelica Code and simulation

The process as explained above leads to the following code as it appears in Imagine.Lab Modelica Code Editor (Figure 5). This code has been generated for a Boost converter and includes both the source and load of static converter.

As a reminder, eventually this code shall be used either by Imagine.Lab platform or any other Modelica tool to create a simulation code. The Imagine.Lab platform allows to create a dedicated C fully compliant with the rest of the Imagine.Lab multi-physical libraries.



```
package PowerElectronicMean

 model HachPar
       input Real u "command for switch";
       parameter Real
C(fixed=false)=0.001,R(fixed=false)=0.1,L(fixed=false)=0.01,E(fixed=false)=50,
       Real Vc1,IL;
     equation
        der(Vc1)=1.0/C*IL-(1.0/(R*C))*Vc1+u*(-1.0/C)*IL;
        der(IL)=-1.0/L*Vc1+u*1.0/L*Vc1+E/L;

   end HachPar;

```

Figure 5: Boost Chopper Mean Model

4 Implementation

Every model is entirely made in an automatic way by the AMG software tool, from an initial description of a state converter structure (here a netlist file generated by PSpice), the commutation mode (cyclic sequence of switched configuration), the switch control sequence and the nature of state variables (slow or fast). Figure 6 introduced an overview of this process.

Firstly, the switch control and the operating mode are defined and the nature of state variables are deduced from the analyze of a simulation of the studied static converter, i.e. by using a power electronics simulation software.

Secondly, a topological description is carried out from a Netlist file extracted by PSpice.

This information is introduced into the software tool AMG and finally several models are generated according to the nature of the power conversion and the state variables.

The obtained result is a file which contains the desired models in an ASCII (text) shape.

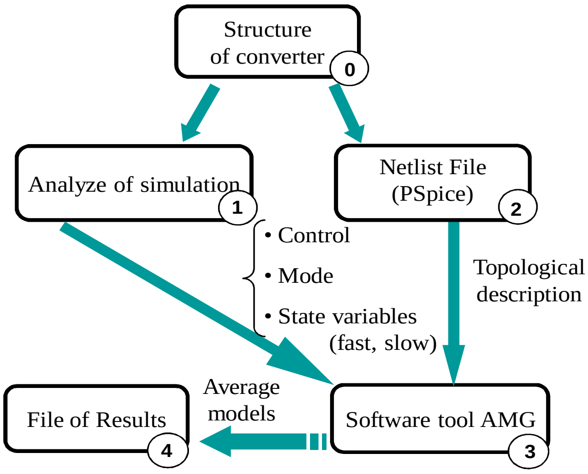


Figure 6: Steps of modeling

4.1 Exact model building

The first step is the analysis of the circuit NetList to carry out the incidence matrix of the whole circuit. Then, this matrix is reduced for each state, according to the conduction state of switches (state ON switches are considered as “short-circuits” or state OFF switches are considered as “open circuits”). For each configuration, the state matrixes are calculated from their simplified node equations. So, the global state space model is obtained by a weighted sum of the state matrixes of each mode state.

4.2 Implementation

In AMG, Java and Maple are combined to implement the previous approach. The building of the state matrixes for every configuration is developed in Java. The extraction of the averaged models from the symbolic approach is made in Maple as shown in Figure 7.

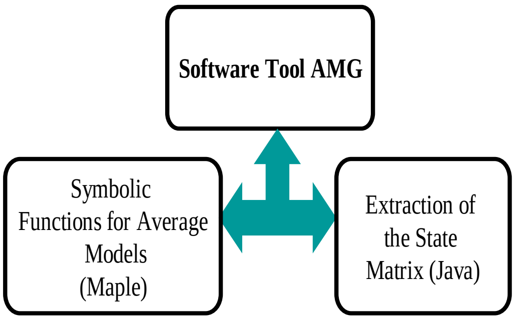

Software Tool AMG
Symbolic Functions for Average Models (Maple)
Extraction of the State Matrix (Java)

Figure 7: Software tool AMG

At the present time, AMG deals only with continuous conduction operating modes.

5 First application: Three phase inverter

5.1 Description of the three-phase voltage inverter

Let's have a look at the static converter known as three phase inverter and the assumption needed to build the average model. Figure 8 presents a three phase voltage inverter with a basic inductive load. Obviously, such an inductive load may even be an electrical motor.

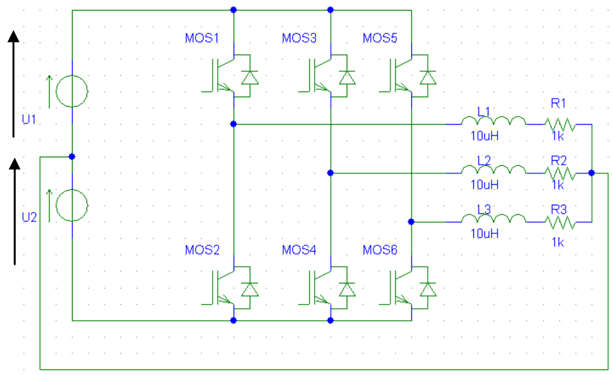


Figure 8: Three phase voltage inverter feeding a three phase electrical load

In this case, each commutation cell is an inverter legs. For, one leg when a switch is controlled to be on (i.e. passing) the other one is off. As a consequence, the inverter needs only three control signals corresponding to the upper switch of the arm (MOS1, MOS3 and MOS5). In the sequel, these signals will be known as h1, h2 and h3 respectively connected to MOS1, MOS3 and MOS5.

Each control signal is a full wave control one (not a pulse wave modulation, i.e. PWM). It is a square wave with a duty cycle of 50%. Moreover, to produce equilibrated three phase system, the three control signals are shifted of $2\pi/3$ rad. (see Figure 9).

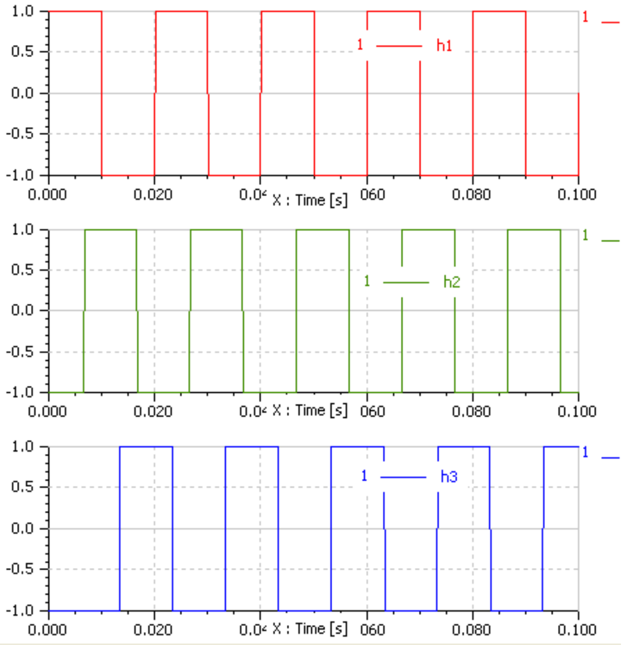


Figure 9: Full wave control for a three phase voltage inverter

5.2 Causality of the static converter

As explained above, AMG needs basic assumptions on the electrical sources to connect. For a voltage inverter, one cans obvious state that the power source is a voltage while the electrical load (inductive one) is a current source.

This intellectual building allows to extract the power static converter from its electrical neighborhood, as a consequence, it can be seen as presented on Figure 10.

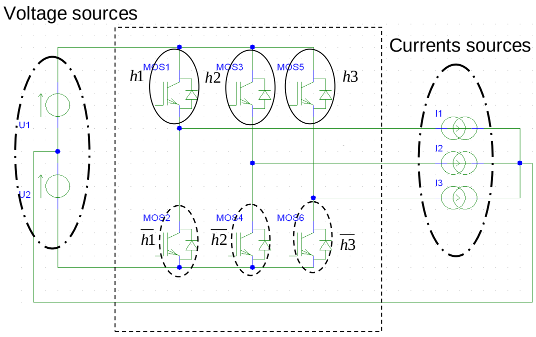
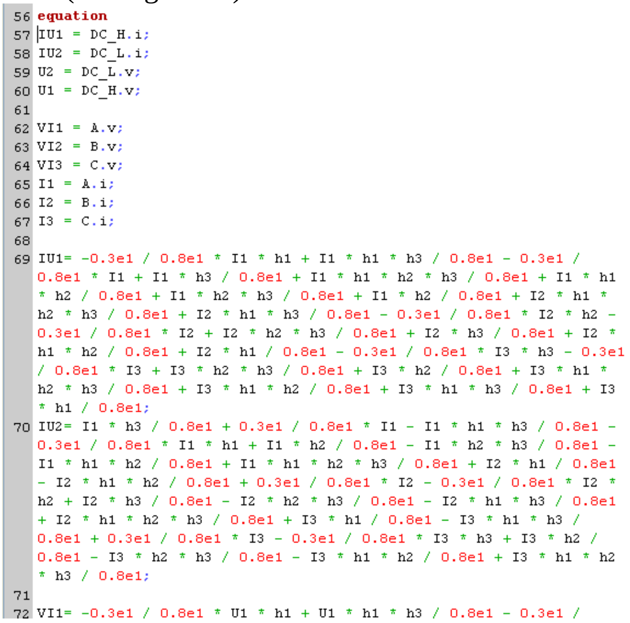


Figure 10: Causality for the three phase voltage inverter

5.3 Generation of the exact model

With the above assumptions on both electrical sources and load and on control laws, AMG is now able to generate the average model of the converter. Hereafter is a snapshot of the generated Modelica code (see Figure 11).



```
equation
IU1 = DC_H.i;
IU2 = DC_L.i;
U2 = DC_L.v;
U1 = DC_H.v;

VI1 = A.v;
VI2 = B.v;
VI3 = C.v;
I1 = A.i;
I2 = B.i;
I3 = C.i;

IU1= -0.3e1 / 0.8e1 * I1 * h1 + I1 * h1 * h3 / 0.8e1 - 0.3e1 /
0.8e1 * I1 + I1 * h3 / 0.8e1 + I1 * h1 * h2 * h3 / 0.8e1 + I1 * h1
* h2 / 0.8e1 + I1 * h2 * h3 / 0.8e1 + I1 * h2 / 0.8e1 + I2 * h1 *
h2 * h3 / 0.8e1 + I2 * h1 * h3 / 0.8e1 - 0.3e1 / 0.8e1 * I2 * h2 -
0.3e1 / 0.8e1 * I2 + I2 * h2 * h3 / 0.8e1 + I2 * h3 / 0.8e1 + I2 *
h1 * h2 / 0.8e1 + I2 * h1 / 0.8e1 - 0.3e1 / 0.8e1 * I3 * h3 - 0.3e1
/ 0.8e1 * I3 + I3 * h2 * h3 / 0.8e1 + I3 * h2 / 0.8e1 + I3 * h1 *
h2 * h3 / 0.8e1 + I3 * h1 * h2 / 0.8e1 + I3 * h1 * h3 / 0.8e1 + I3
* h1 / 0.8e1;
IU2= I1 * h3 / 0.8e1 + 0.3e1 / 0.8e1 * I1 - I1 * h1 * h3 / 0.8e1 -
0.3e1 / 0.8e1 * I1 * h1 + I1 * h2 / 0.8e1 - I1 * h2 * h3 / 0.8e1 -
I1 * h1 * h2 / 0.8e1 + I1 * h1 * h2 * h3 / 0.8e1 + I2 * h1 / 0.8e1
- I2 * h1 * h2 / 0.8e1 + 0.3e1 / 0.8e1 * I2 - 0.3e1 / 0.8e1 * I2 *
h2 + I2 * h3 / 0.8e1 - I2 * h2 * h3 / 0.8e1 - I2 * h1 * h3 / 0.8e1
+ I2 * h1 * h2 * h3 / 0.8e1 + I3 * h1 / 0.8e1 - I3 * h1 * h3 /
0.8e1 + 0.3e1 / 0.8e1 * I3 - 0.3e1 / 0.8e1 * I3 * h3 + I3 * h2 /
0.8e1 - I3 * h2 * h3 / 0.8e1 - I3 * h1 * h2 / 0.8e1 + I3 * h1 * h2
* h3 / 0.8e1;

VI1= -0.3e1 / 0.8e1 * U1 * h1 + U1 * h1 * h3 / 0.8e1 - 0.3e1 /
```

Figure 11: Exact model for a three phase voltage inverter generated by AMG

Analyzing the code, one can see the expression of DC bus current as a function of:

- Control signal: h1, h2, h3
- Current in phase I1, I2 and I3

In the same way, the voltage drop applied in the phase is a function of:

- The DC bus voltage (U1, U2)
- The control signal h1, h2, h3

The interest of the AMG tool becomes obvious by having a look at the equation generated for this very basic power converter structure. One can easily imagine the awful they may have for a multi-level converter.

5.4 Building of a complete application

Let us observe the numerical results obtained with a three phase inductive load. This can be achieved either by adding specific relation like presented in the Figure 12, or by connecting this component to electrical components from the Modelica Standard Library. (Figure 13)

```
/* Added relation for Three phase load*/

der(IL1)= (V1-(R4*I1))/L1;
der(IL2)= (V5-(R4*I2))/L2;
der(IL3)= (V3-(R4*I3))/L3;
```

Figure 12: Equations for a three phase RL load

Figure 13 presents the integration of the above model within a “complete circuit”.

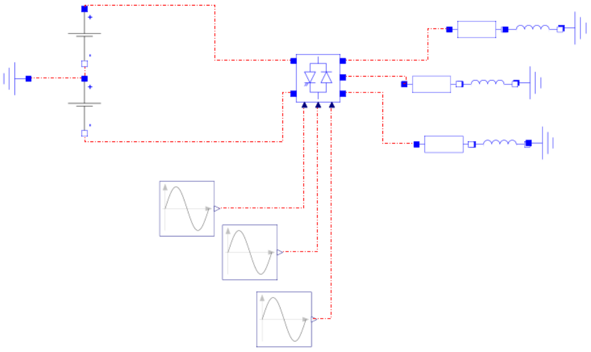


Figure 13: Three phase voltage inverter with electrical load

5.5 Numerical Results

Let us run a simulation with the following parameters:

R = 2 ohm; L=0.01H;E=320V;freq=50Hz

The simulated currents are represented on Figure 14.

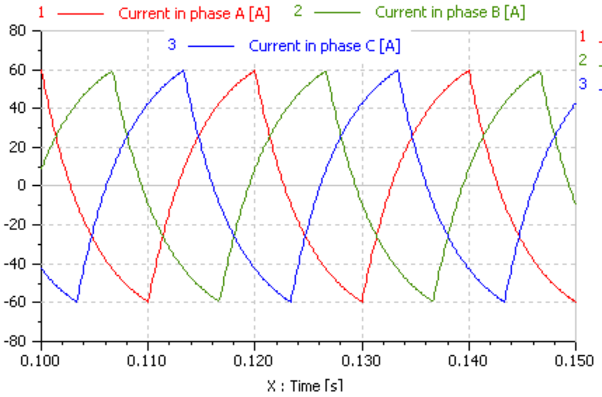


Figure 14: Simulation results: currents in the inductors

The influence of inductors may be observed on the following results.

One has to note that such numerical result is observable as there is no parasitic time constant due to inner resistance of electronic. In fact, in classical models like Ron/Roff ones, additional time constants are observed when switches are connected to inductors (or capacitors).

Then using exact model (as well as average one) brings the guaranty that no additional time constant is added to the whole electrical circuit.

6 Second application

6.1 Description of the DC-DC converter

The studied converter in this part is a Boost converter. The electrical conversion is DC/DC. Boost converter.

The static DC/DC boost converter allows increasing the voltage level. This simple example demonstrates both the capacity to analyze DC/DC converter and to generate both exact and average power converter models.

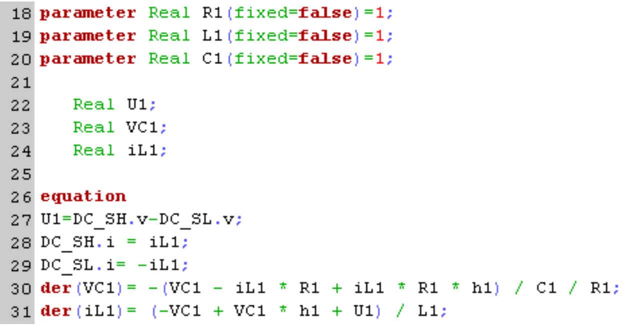
An exact model of a power converter allows to take into account the real control on electronic switches. Contrary average model aims at averaging the state variable of the circuit.

It should be noted that using an average model, the control law shall also be averaged:

- In case of DC/DC conversion, the input value is supposed to be the duty cycle
- In DC/AC conversion, average control is the sinus law for output voltage (or the shape is)

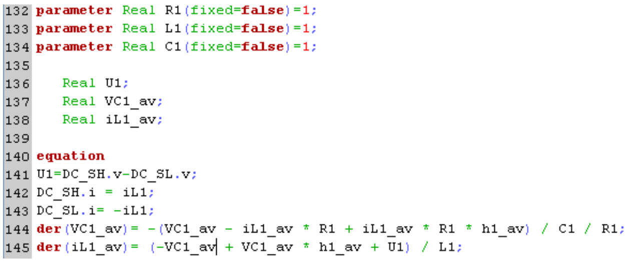
6.2 Average model of the static converter

Two kinds of model have been generated by AMG. Figure 15 presents the exact model and Figure 16 the average model (a large signal model).



```
parameter Real R1(fixed=false)=1;
parameter Real L1(fixed=false)=1;
parameter Real C1(fixed=false)=1;

   Real U1;
   Real VC1;
   Real iL1;

equation
U1=DC_SH.v-DC_SL.v;
DC_SH.i = iL1;
DC_SL.i= -iL1;
der(VC1)= -(VC1 - iL1 * R1 + iL1 * R1 * h1) / C1 / R1;
der(iL1)= (-VC1 + VC1 * h1 + U1) / L1;
```

Figure 15: Modelica exact model of the boost converter



```
parameter Real R1(fixed=false)=1;
parameter Real L1(fixed=false)=1;
parameter Real C1(fixed=false)=1;

   Real U1;
   Real VC1_av;
   Real iL1_av;

equation
U1=DC_SH.v-DC_SL.v;
DC_SH.i = iL1;
DC_SL.i= -iL1;
der(VC1_av)= -(VC1_av - iL1_av * R1 + iL1_av * R1 * h1_av) / C1 / R1;
der(iL1_av)= (-VC1_av + VC1_av * h1_av + U1) / L1;
```

Figure 16: Modelica large signal model of the boost converter

In the case of the boost converter, the average model and the exact model are similar. As it has explained above, the difference is the model of the control signal. For the exact model, the control signal is a square wave. For the average model, the control signal is the average value of the square wave control signal, i.e. the duty cycle on every switching period (α) (see Figure 17).

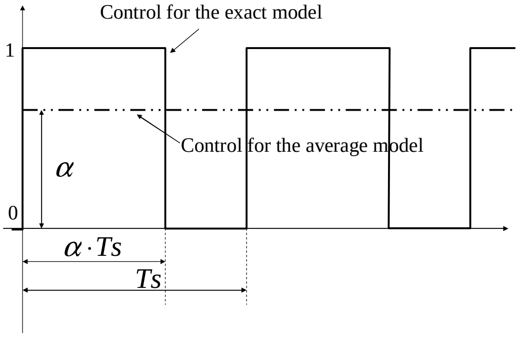


Figure 17: Control signal according to the model

6.3 Average model of the static converter

Numerical results for both of them are compared using AMESim.Lab. Figure 18 and 20 show these comparisons

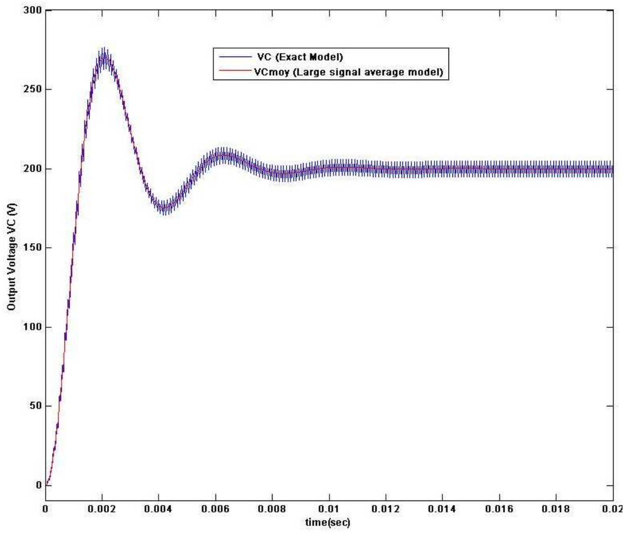


Figure 18: Capacitor Voltage [V] general overview

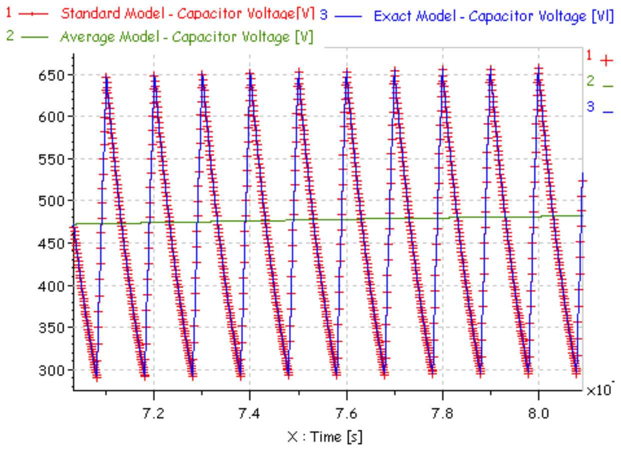


Figure 19: Capacitor Voltage [V] comparisons average/exact/standards models

The following array emphasizes the CPU time needed to perform the complete simulation for each kind of representation (see tables 1 and 2)

Parameter	**Value**
Time simulation	5 s
PWM frequency	10kHz
R	10 Ohm
L	0.001 H
C	1e-5 F
E	100 V
Duty cycle	0.8

Table 1: Parameters Value

Model	**CPU Time**
Standard representation	6.64s
Exact model	5.5s
Average model	0.016s

Table 2: CPU time

7 Advantages and drawbacks

Average models of static converter do not represent the commutations, so the simulation is faster. So, average models fit very well to system simulation or real time simulation.

The exact model represents switches by a short-circuit at on state and an open circuit at off state. Such a modeling that does not introduce parasitic time constants like the classical modeling with a binary resistance. For this last modeling, every switch is represented by a resistor which resistance is high at off state and low at on state.

However, for both average models and exact models, an a priori has to be carried out on the behavior of the studied static converter. This means that a change of the control mode may induce a new generation of a new dedicated model.

The use of Modelica as non causal modeling language allows achieving the model generation of power electronics converters by not being constrained by the causality. This aspect is very important and allows to deals with small power electronics structures as well as with large power electronics structures.

The use of Modelica also allows describing and generating the model of the static converter independently of the Modelica simulation environment.

In order to help the engineer to select the appropriate representation, we will also provide some guideline to select the model to be used.

8 Conclusions

In this paper, a methodology to get a fully functional Modelica code for a average model of power electronics converter has been presented. Its implementation within software has been briefly presented. Numerical tests for this code have also been introduced using the Imagine.Lab Modelica Platform. The efficiency of the proposed method has been discussed as well as some criteria to select the right appropriated sub-model according to the needs of engineer. A Modelica library of static converter controls (e.g. PWM), will be developed in future works as well as a guidelines for the modeling in discontinuous mode.

References

[1] http://www.lmsintl.com/imagine-amesim-1-d-multi-domain-system-simulation

[2] R.D. Middlebrook, S Cuk, "A general unified approach to modelling switching converter stages", IEEE Power Specialists Conf., pp 18-34, 1976.

[3] P. R. K. Chetty, "Current injected equivalent circuit approach to modelling and analysis of current programmed switching DC-to-DC converters (discontinuous inductor conduction mode)", IEEE Trans. on Industry Applications, Vol. IA-18, N°3, pp 295-299, 1982.

[4] S.R. Sanders, J. M. Noworolski, X. Z. Liu, G. Varghese, "Generalized averaging method for power conversion circuits", IEEE Power Electron. Specialists Conf. (PESC), Records 1990, pp 333-340.

[5] S. Bacha, M. Brunello, A. Hassan, "A general large signal model for DC-DC symmetric switching converters" Electric Machines and Power Systems, Vol 22, N° 4, July 1994 ; pp 493-510.

[6] Richard M.bass and Jian Sun, "Using symbolic computation for power electronics".Computers in Power Electronics, 1998. 6th Workshop, 19-22 July 1998, pp I IV.

[7] F.Verdiere, S.Bacha, L.Gerbaud, "Automatic modelling of static converter averaged models". *EPE* 2003- Toulouse .pp.1-9.

[8] Piero G.Maranesi, Marco Riva. "Automatic modelling of PWM DC-DC converters". IEEE Power Electron. Letters, Vol.1,N°4, December 2003.

[9] L. Allain, L. Gerbaud, C. Van Der Schaeghe, "Capitalisation and treatment of models for the optimisation of electric drives", in Optimisation and Inverse Problems in Electromagnetism, edited by M. Rudnicki and S. Wiak, Kluwer Academic Publishers, 2003, ISBN 1-4020-1506-2, pp 205-212 (8 pages)

Pre-designing an electronic card using a multi-domain models approach with DYMOLA

Behrouz Roumizadeh Jean Yves Choley Régis Plateaux Olivia Penas
Alain Riviere

SUPMECA
3 Rue Fernande Hainaut, Saint Ouen, France
behrouz.roumizadeh@supmeca.fr, jean-yves.choley@supmeca.fr, regis.plateaux@supmeca.fr,
olivia.penas@supmeca.fr, alain.riviere@supmeca.fr

Abstract

This paper presents a new method to model electronic circuits, considering not only the electrical aspect of the circuit but also its geometrical and multi-physical aspects via DYMOLA [1]. The new modeling method based on the representation of the electronic components and their environments, with the multi-domain models. This means combining the geometrical, electrical and multi-physicscal properties in one single mechatronical model. This kind of representation will allow us to simulate the electronic environments regarding their multi-domain aspects and relations. This new method will allows us to perform the pre-dimensioning and pre-placement of the components, through a simple modeling process in DYMOLA environment. We expect the result to be a more rapid pre-dimensioning procedure for the electrical circuits.

Keywords: Electrical circuit; multi-physics; multi domain modeling; DYMOLA; mechatronics.

1 Introduction

The design of automotive electrical circuits is a complicated procedure, which includes vast parts of science, from the pure electrical Eng., to the multi-physical simulations such as thermal simulations. This will result in the need for different and mostly unrelated modeling and simulation programs, in order to design and optimize the conception of electrical components. The result would be a complete mechatornic problem which includes electrical and mechanical problems. It is obvious that unrelativity of the different simulations, will result in the increase of the time and cost of the design and design process.

Generally, design of an electrical circuit begins with its schematic model (logical view). This stage of design consists of the electrical and logical connections between the components, and can be modeled and simulated by the different software such as CADENCE [2], ZUKEN [3], and the electrical library of DYMOLA.

Simulation of the electrical schematics will result in the electrical properties of the components and connections, which are needed for the multi-physical simulations. Although the electrical simulations at this level can be performed without taking into consideration the geometrical aspects of the circuit and its environment, but once the multi-physical simulations have been carried out, the circuit has to be remodeled and re-simulated due to changes originating from the electrical-multi-physical interactions.

Once the logical aspect of the circuit has been validated, components have to be placed on the surface on which they will operate. This step will be followed by a routing procedure.

Existing electrical software is only capable of performing the component placement and routing of the PCBs (Printed Circuit Boards) on the 2D surfaces [4]. There is also some new software [5] [6] and algorithms [4][7] which offer this possibility on the 3D surfaces, although existing software does not offer any solutions for some other electrical designs such as high current electronics and power electronics, for which the 3D MCAD(Mechanical CAD) systems are used as a design system.

The present design cycle is presented in fig.1. Once the electronic card has been designed, the multi-physical simulations, which are mostly the FE simulations, will be performed to validate the operability of the circuit, from the multi-physical point of view.

DOI: 10.3384/ecp09430117

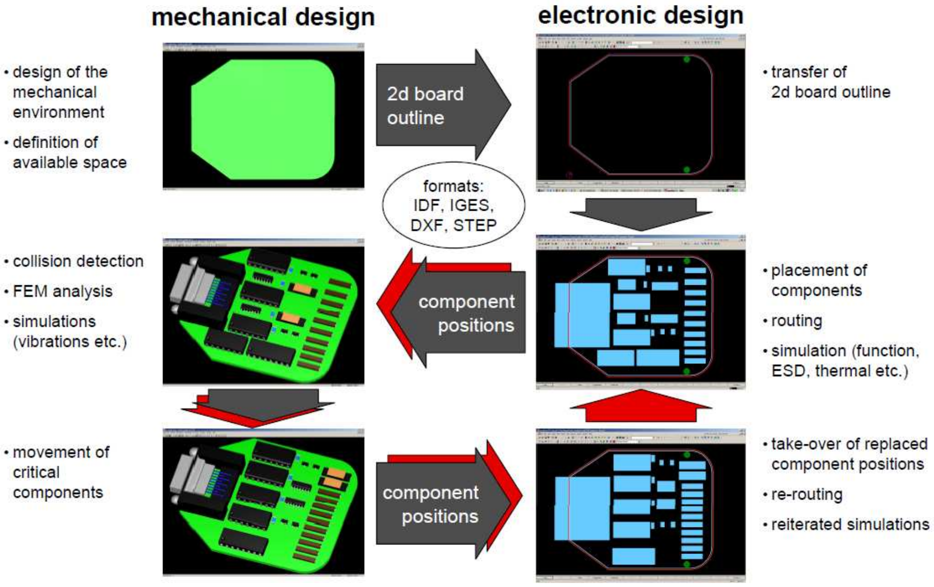

mechanical design
electronic design
• design of the mechanical environment
• definition of available space
2d board outline
formats: IDF, IGES, DXF, STEP
• transfer of 2d board outline
• collision detection
• FEM analysis
• simulations (vibrations etc.)
component positions
• placement of components
• routing
• simulation (function, ESD, thermal etc.)
• movement of critical components
component positions
• take-over of replaced component positions
• re-routing
• reiterated simulations

Figure 1: the actual electronic design cycle [8]

The two most important multi-physical simulations for electronics are the thermal and EMC (Electro Magnetic Compatibility) simulations. The circuit has to be simulated within its environments, from the thermal [9] and EMC [10] point of view, so that its functionality will be validated. As can be observed from the figure above, the position of the components may be changed once the thermal simulations for instance have been performed, and the board must be rerouted. It is obvious that this cycle and specially, the FE simulations are time consuming and costly specially because this conception cycle will continued till the optimized circuit would be achieved.

A solution for this problem would be a rapid prepositioning and pre-sizing procedure, which would result in a semi-optimized electronic card. We believe that this will reduce the need of FE simulations. This simulation process has to combine some multi-domain properties in one single model. DYMOLA proposes a good graphical environment which enables us to perform this multi-domain modeling.

As has been said above, DYMOLA proposes the ability to simulate electrical circuits and there are existing libraries to do so. It also proposes thermal models which enables thermal simulations, although a linked electro-thermal simulation which takes into account geometrical properties of the components and their placements, does not exist. To do so the designer has to calculate the geometrical effects on the thermal properties and with them perform the thermal simulations. Our proposition is to introduce a new modeling method so that it includes electrical, geometrical and thermal properties in a single model so that they are related. The representation of the geometrical and multi-physical properties such as thermal properties is simplified, so that the simulation will be performed rapidly and it will serve as a pre-dimensioning process so that it will reduce the need for the FE simulation and replacement cycle.

2 Mathematical multi-domain parameters relations

To perform the new modeling procedure we will need to have the mathematical relations of the different multi-physical properties, electrical properties and the geometrical properties with each other so that we could build the new multi domain models in DYMOLA. This is more essential in the sections that the different properties have the interactions with each other, such as the heating resistances or the surfaces which the multi-physical properties are related to the geometrical and electrical properties. [11] and [12] propose a method to extract such relations with the help of the topological diagrams. The capability of this has been shown by [13] which explained the need of the topological structure to model the complex systems. Fig.2 shows the algebraic diagram which by that we can associate the different properties of our model. On the first step the analysis will start from nodes on the Primal object. The nodes present the properties of the nodes which in our case are the coordination of the nodes (P), electrical potential of the nodes (V) and the temperature of each node (T). We will call the matrix which contains these properties matrix [N]. The incidence matrix [C], which builds up of the relations between different nodes will capable us to have the relations of the same properties between two nodes(Branches) such as the vector between two nodes, the potential differences between nodes and the temperature differences., this will be presented in matrix [B]. The dual object R_B for us will be the current and the heat flow in the branches. To arrive to the R_B we would need the admittance matrix Y_B which represents the relation between different properties.

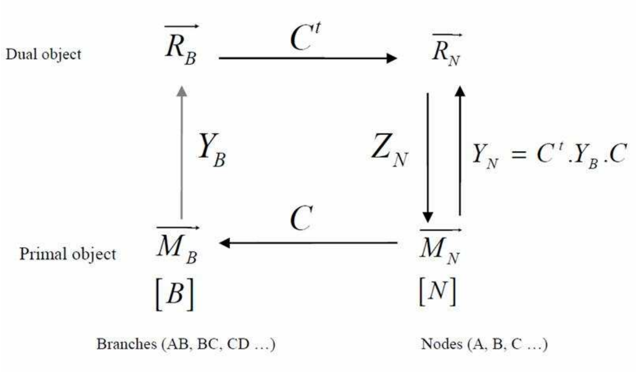


Figure 2: algebraic diagram of the relations between different properties of the model

As it can be seen, once we have the topological diagram of the model, from any property as the departure data we will be capable of calculating the other properties of the model. These relations will help us to perform our modeling and analysis. Although this method will respond to our need of knowing the relation between the components in the model but it would be hard to perform if the model is complicated. [14] had showed that by partitioning the matrices into smaller model and matrices we will be able to simplify the model. This is done by associate to each part of the model a topological diagram and incidence matrix [C] and by connecting the smaller parts together construct the bigger incidence matrix.

The possibility of partitioning the incidence matrix into smaller parts of the model will permit us to easily extract the equations of the models which do not exist in the standard DYMOLA library. As an example the relations of the multi-physical properties of a resistor are already exist in DYMOLA, and we just need to connect the existing multi domain models together by a new built connector which can support all the parameters of these models, but as the thermal relations on a surface with their interactions with the geometry does not exist, the relations between the different parameters of this new model has to be extracted by this method.

To calculate the thermal properties we have included the thermal resistance concept [15]. This will reduce the variables which are all included in the thermal resistance R_{th} and facilitate the calculations. Algebraic diagram in fig. 3 present the different variables in our calculations and their relations together. For the reason that the entire variable has to be calculated in the same time with the different interactions that they have with each other the relation between the variables which will be obtained by this diagram is quit complicated.

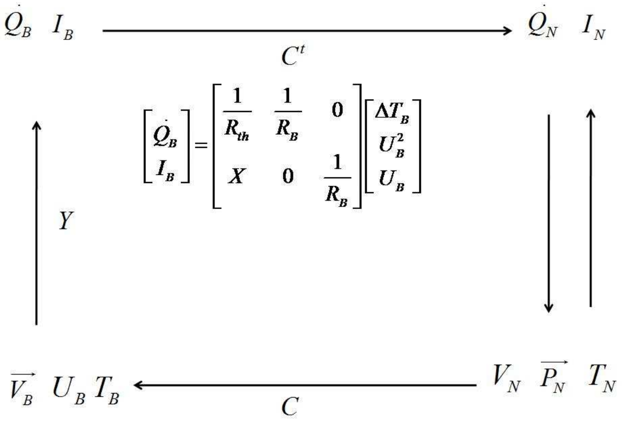

$\dot{Q}_B \quad I_B$
C^t
$\dot{Q}_N \quad I_N$
$\begin{bmatrix} \dot{Q}_B \\ I_B \end{bmatrix} = \begin{bmatrix} \frac{1}{R_{th}} & \frac{1}{R_B} & 0 \\ X & 0 & \frac{1}{R_B} \end{bmatrix} \begin{bmatrix} \Delta T_B \\ U_B^2 \\ U_B \end{bmatrix}$
Y
$\overrightarrow{V_B} \quad U_B \, T_B$
C
$V_N \quad \overrightarrow{P_N} \quad T_N$

Figure 3: multi-physical parameters relations

To reduce the complicity of the equations [12] offers a possibility of separating the diagram in more simple diagrams, solve these simpler diagrams and then calculate their interactions’ with each other.

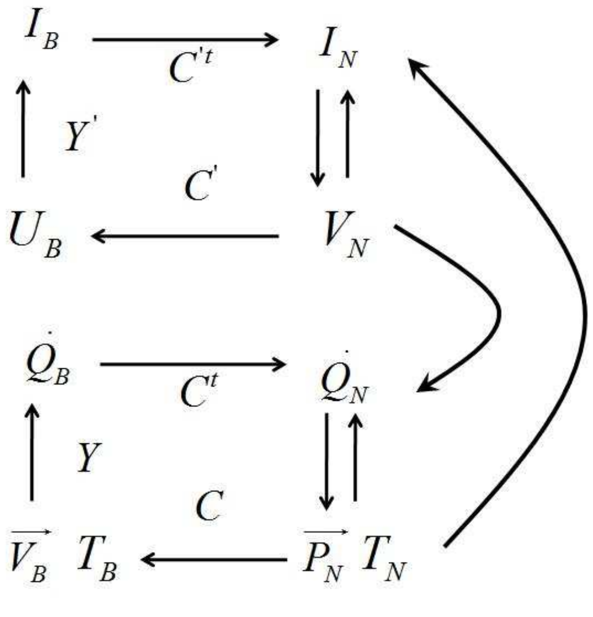


Figure 4: electrical and thermal algebraic diagrams and their interactions

This method will reduce the complexity of the equations and has the advantage that as the upper section (electrical diagram) which represent the electrical equations already exist in DYMOLA, it will give us the relation of these existing models with the newer parameters needed in the models.

2.1 Thermal characteristics of the electrical boards

In the electrical boards and circuits most of the heat passes through the board or surface of the circuit. This produces a need to include the heat transfer from the surfaces in the simulation. In the DYMOLA thermal library, there is no mean to calculate the heat transfer on a 2D surface. We have introduced two plane models with the ability to calculate the heat transfer between the connectors' positions in relation to the position of the connectors.
The first option to calculate the thermal properties on the plane and between different points which will be represented by the connectors, we will use the concept of the radial heat conduction on a disc [15] fig.5.

$$R_{th} = \frac{\ln(r_2 / r_1)}{2\pi L\lambda}$$

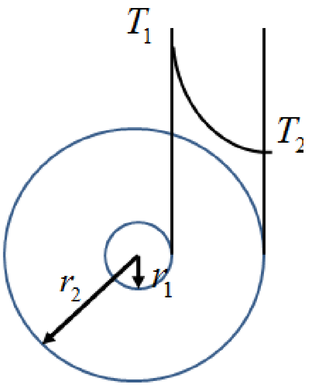


Figure 5: thermal resistance in a radial conduction on a plane

Although this method is simple to calculate and use, it would be applicable only if the surface could be supposed unlimited due to the dimensions of the r_1 and r_2. The other option is to use the equivalent electrical network [16] fig.6, which represent the system with the system with the thermal resistances and capacitors which can be simulated as an electrical network.

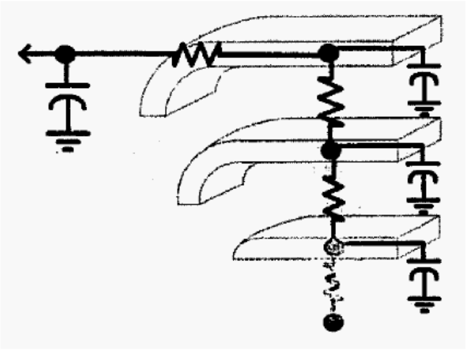


Figure 6: equivalent electrical network representation of a thermal system

To simplify the models in the first stage of modeling procedure we suppose the problem as a steady state situation so that we can neglect the C. This will simplify our plane thermal model to a model which has been divided by the smaller cubes which each will be represented by their thermal resistances. Fig. 7

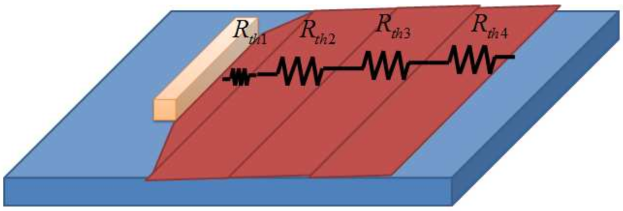


Figure 7: plane thermal model represented by the thermal resistance network concept

These two thermal models can be used for modeling the plane model including its geometrical equations in DYMOLA.

3 Modeling procedure

Once the mathematical equations and relations of the models have been extracted the models and the modeling procedure can be introduced. This new modeling method will need a new library which includes the electrical components presented with the new concept, and new components which will serve as the environment of the electronic circuit or the surfaces on which it will be installed. All of these new models ought to include the multi-domain models, properties and connectors inside them. Fig 8 shows a new internal representation for a resistance component.

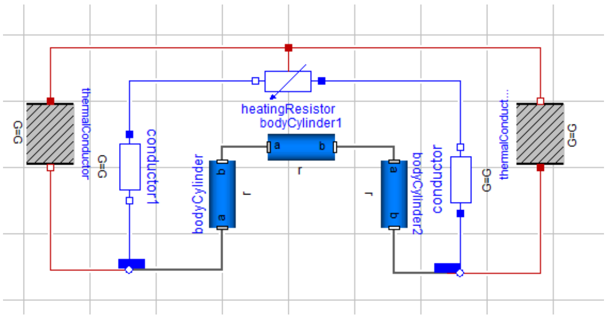
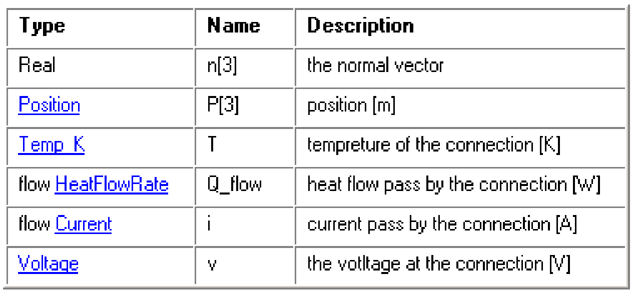


Figure 8: Internal multi-domain representation of an electrical resistance

It can be observed that the new model not only represents the electrical properties of the resistance but also its geometrical and thermal properties. This new model has been introduced by utilization of the existing DYMOLA library models with connecting them to a new connector. This new connector contains the combined properties needed in this model. This connector is needed to ensure the multi-domain modeling within a single model. Fig 9 shows the properties of the new connector used to connect new models build in DYMOLA.



Type	Name	Description
Real	n[3]	the normal vector
Position	P[3]	position [m]
Temp_K	T	tempreture of the connection [K]
flow HeatFlowRate	Q_flow	heat flow pass by the connection [W]
flow Current	i	current pass by the connection [A]
Voltage	v	the voltlage at the connection [V]

Figure 9: New connector characteristics

It is obvious that due to the need of taking into account the newer properties in the system such as the electromagnetic properties, other variables have to be introduced in the connector.

To perform the multi-physics simulations, we also need to integrate the environments of the circuits. To do so, we propose new geometrical models, such as plane model which is presented in fig 10, so that the other components can be installed on them, and they themselves can be connected to another environmental model.

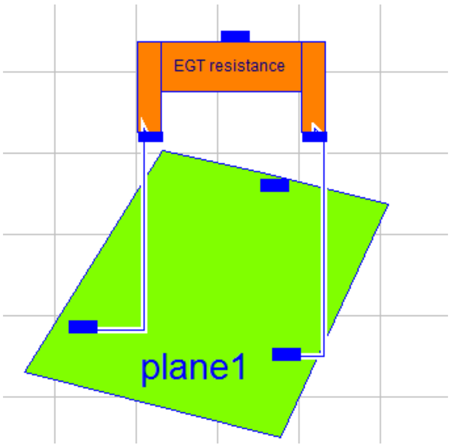


Figure 10: integration of the electrical components in their multi-domain environment e.g. plane

As the components are placed on the different places on the plane, we need to be capable of calculating the temperature (T), and the thermal flow (Q) between the different connector points. As it has been explained in section two, this is possible by using the thermal resistance concept. As the thermal resistance is related to the geometry, and the heat produced is related to the electrical properties of the circuit, the relations which would be achieved by the procedures in the section two would be used to build the new multi-domain models such as the plane model.

Using the multi-domain electrical and geometrical models to model the electrical circuits which are made by this method, we are be able to simulate the electrical circuits in DYMOLA not only due to their electrical properties but also geometrical and thermal properties in one single stage. The advantage would be that normally, the circuit has to be modeled and simulated due to one of these properties (electrical, geometrical and thermal) and the relations between each simulation has to be performed by the paperwork. For instance to perform the thermal simulations the electrical and geometrical variables of the circuits has to be transferred by hand from one simulation to the other.

4 Conclusion

We offer a new method to model and simulate electrical circuits, considering their geometrical and multi-physical properties in one single model. This new method will result in a pre-optimized electrical design and components placement, which will reduce the need of costly Finite Element simulations.

The possibility of exchange the geometrical properties with CATIA [17] will serves as a strong positive point for this new modeling method because the geometrical data and positioning of the components can be used in modeling a detailed geometry of the electrical circuits with CATIA. This and a procedure to perform an electrical routing on the 3D surfaces with the help of DYMOLA and CATIA would be subject of the future work.

References

[1] http://www.dynasim.se

[2] http://www.cadence.com

[3] http://www.zuken.com

[4] K. Feldmann, Y. Zhuo, C Alvarez, “3D Gridless routing for the design of Molded Interconnect Devices (MID)” Production engineering, XII-2 89-94. 2005.

[5] NEXTRA http://www.mecadtron.de

[6] E^3-SeriesZUKEN http://www.zuken.com/products/e3-series.aspx

[7] Y Zhuo, C. Alvarez, K. Feldmann, “An Integrated design system for Molded Interconnect Devices(3D-MID)”, Digital enterprise technology, 2007

[8] T. Krebs, “3D mechanical CAD collision detection with Allegro PCB editor and NEXTRA”, proceedings of EMEA conference, May 2007

[9] FLOTHERM http://www.mentor.com/products/mechanical/products/flotherm

[10] Q3D http://www.ansoft.com/products/si/q3d_extractor/

[11] Franklin H. Branin Jr. “The Algebraic-Topological Basis for network analogies and the vector calculus.” Symposium on generalized networks. 04/1966.

[12] Bjorke O., “Manufacturing systems theory”, 1995.

[13] R., Penas O., Riviere A., Choley J.Y., “A need for the definition of a topological structure for the complex systems modeling”., CPI2007.

[14] Elmqvist H., “A structured model language for large continuous systems”. PhD theses. LIT 1978.

[15] G.F. Hewitt, G. L. Shires and T. R. Bott, “Process heat transfer”, CRC press, 1993.

[16] J.M. Ortizo-Rodriguez, D. Berning, M. Hernandez, “Lumped-parameter Thermal Modeling of an IPEM using Thermal Component Models” IEEE 2004

[17] P. Bhattacharya, N. S. Welakwe and R. Makanaboyina, “Integration of CATIA with Modelica”, proceedings of Modelica conference, 2006

Modelica Libraries for Linear Control Systems

Marcus Baur, Martin Otter, Bernhard Thiele
German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Germany
Marcus.Baur@DLR.de, Martin.Otter@DLR.de, Bernhard.Thiele@DLR.de

Abstract

This article presents and describes the new LinearSystems and Controller libraries which are developed to enhance analysis, design and simulation of linear control systems in Modelica. The LinearSystems library contains basic functions for linear system analysis and controller design for state-space, transfer-function, and zeros-and-poles representation. The library utilizes the operator overloading technique from Modelica 3.1. The Controller library provides input/output blocks for these basic system descriptions and allows to quickly switch between a continuous and a discrete representation.

Keywords: linear systems; control design; system control; sampled systems

1 Introduction

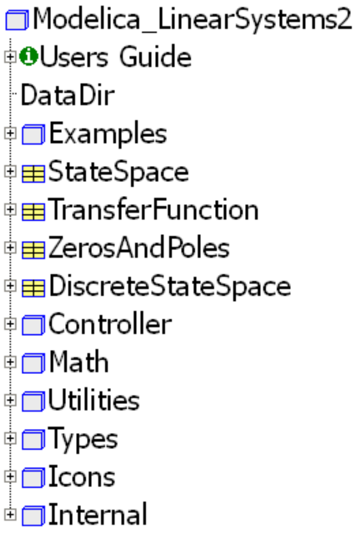
This article gives an overview of two new, open source Modelica libraries to enhance the analysis, design and simulation of linear control systems in Modelica, i.e. the LinearSystems library and the Controller library. The LinearSystems library contains about 180 Modelica functions for the analysis and design of linear control systems in different description forms. The Controller library contains about 30 controller blocks where it is easy to switch between a continuous and a discrete representation of the blocks. The Controller library is based on the description forms provided by the LinearSystems library.

All numerical functions of the LinearSystems library are natively implemented in Modelica, with exception of linear algebra functions (e.g. solving linear systems of equations) that use the LAPACK library [6]. Therefore, the functions of the LinearSystems library can be used in the production code of a controller, e.g., to design at every sample instant a linear observer for a non-linear plant model that is linearized around the actual operating point. The goal is to use this functionality in combination with the Modelica_EmbeddedSystems library [4] for advanced embedded control systems, where non-linear inverse plant models are present in a controller.

Parts of the functions and blocks of the two libraries are based on the Modelica_LinearSystems library described in [9]. It is planned that both libraries will be included in one of the next versions of the Modelica Standard Library. Currently, they are again collected in one library called Modelica_LinearSystems2 for the LinearSystems library and the sublibrary Modelica_LinearSystems2.Controller for the Controller library.

2 LinearSystems library

The LinearSystems library contains currently about 180 functions, whereas the previous version had about 20 functions. A screen shot of the first hierarchical level of the library is shown in Figure 1. Most important are the four records that contain the basic data structures and functions for linear control systems according to the following mathematical description forms:



Modelica_LinearSystems2
- Users Guide
- DataDir
- Examples
- StateSpace
- TransferFunction
- ZerosAndPoles
- DiscreteStateSpace
- Controller
- Math
- Utilities
- Types
- Icons
- Internal

Figure 1: LinearSystems library

DOI: 10.3384/ecp09430068

- StateSpace

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{Ax}(t)+\mathbf{Bu}(t)\\ \mathbf{y}(t) &= \mathbf{Cx}(t)+\mathbf{Du}(t)\end{aligned}$$

- TransferFunction

$$y=\frac{n(s)}{d(s)}\cdot u$$

- ZerosAndPoles

$$y=k\cdot\frac{\prod(s+n_{1i})\cdot\prod(s^2+n_{2j}s+n_{3j})}{\prod(s+d_{1k})\cdot\prod(s^2+d_{2l}s+d_{3l})}\cdot u$$

- DiscreteStateSpace

$$\begin{aligned}\mathbf{x}_d(t_{k+1}) &= \mathbf{Ax}_d(t_k)+\mathbf{Bu}(t_k)\\ \mathbf{y}(t_k) &= \mathbf{Cx}_d(t_k)+\mathbf{Du}(t_k)\\ \mathbf{x}(t_k) &= \mathbf{x}_d(t_k)+\mathbf{B}_2(t_k)\end{aligned}$$

with the Laplace variable s. Note, that the matrix $\mathbf{B}_2$ to derive the continuous state space vector $\mathbf{x}(t_k)$ from discrete $\mathbf{x}_d(t_k)$ depends on the linearization method. The users view of the ZerosAndPoles data structure are the zeros, poles and the gain of the transfer function. Internally, the transfer function is represented by the real coefficients of first and second order polynomials, so that the operations on this representation results in transfer functions with real coefficients. If complex poles and zeros would be used, inaccuracies in computational calculation could result in systems with complex poles without the conjugated complex counterpart.

The data of the mathematical description forms are stored in the respective record, e.g., the matrices **A**, **B**, **C**, **D** are the data of the state space representation stored in the StateSpace record. Additionally, the signal names can be optionally stored as well. When linearizing a Modelica model, e.g., with `StateSpace.Import.fromModel(..)`, the full signal names of the original model are automatically included in the linear system record.

StateSpace-, TransferFunction-, and ZerosAndPoles-records have the same basic structure. As an example, the first hierarchical level of the StateSpace record consisting of several sublibraries is shown in Figure 2.

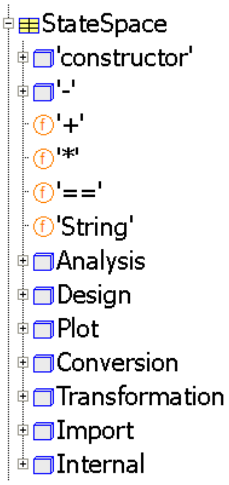


Figure 2: StateSpace record

The first few elements, coated with quotes, are made for the usage of operator overloading, see [8], in order that the elementary operations $+,-,*,==$, String(..) on the data structure can be conveniently carried out.

For example, the following transfer functions

$$G_0(s)=\frac{s+1}{s^2+3s-2}$$

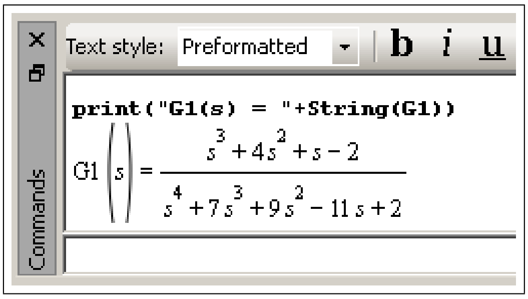
and

$$G_1(s)=\frac{G0}{1+G0}$$

can be easily defined in the interactive environment of Dymola [3] in the following way:

```
import tf =
Modelica_LinearSystems2.TransferFunction;
s  = tf.s();
G0 = (s+1)/(s^2+3*s-2);
G1 = G0 / (1 + G0);
G1
  // = "(s^3 + 4*s^2 + s - 2)/
  //    (s^4 + 7*s^3 + 9*s^2 - 11*s + 2)"
```

The new Command window of Dymola shows such results appropriately rendered, e.g. Figure 3.



```
print("G1(s) = "+String(G1))
```

$$\mathrm{G1}\left(\left|s\right|\right)=\frac{s^3+4s^2+s-2}{s^4+7s^3+9s^2-11s+2}$$

Figure 3: Dymola Command window

The further sublibraries structure the available functions:

- **Analysis** contains functions to compute eigenvalues, poles, zeros, or properties like controllability.
- **Design** contains functions to design control systems, e.g., with the pole placement or the Riccati method.

- **Plot** contains functions to compute and plot poles and zeros, frequency responses, step responses etc.
- **Conversion** contains functions to convert from one data structure to another description form, e.g., from StateSpace to TransferFunction.
- **Transformation** contains functions to perform a similarity transformation, e.g., to transform to controllability form.
- **Import** contains functions to import the data structure from a model (by linearization) or from a file.

All functions of the library are Modelica functions. For basic linear algebra computations, like solutions of linear systems of equations, or eigenvalue computations, the standard numerical library LAPACK [6] is used. Besides LAPACK, no other external functions are utilized.

In the following sections, some of the above quoted sublibraries are described in more detail for StateSpace systems.

2.1 StateSpace.Analysis

The Analysis package of StateSpace contains functions to compute eigenvalues, invariant zeros, and various system properties.

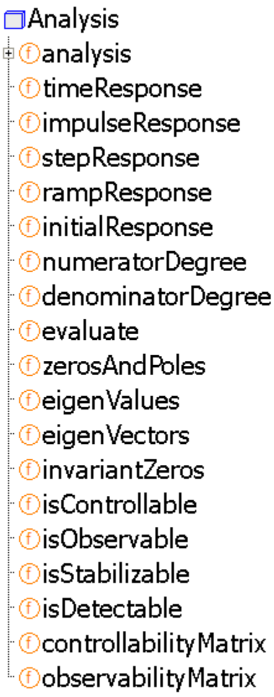

Analysis
analysis
timeResponse
impulseResponse
stepResponse
rampResponse
initialResponse
numeratorDegree
denominatorDegree
evaluate
zerosAndPoles
eigenValues
eigenVectors
invariantZeros
isControllable
isObservable
isStabilizable
isDetectable
controllabilityMatrix
observabilityMatrix

Figure 4: Package StateSpace.Analysis

The eigenvalues of a state space system, which are the eigenvalues of the system matrix $\mathbf{A}$, are calculated with the LAPACK function dggevx, i.e., using the basic eigenvalue computation with an additional balancing transformation to improve the conditioning of the eigenvalues. The principle of the algorithm is to reduce matrix $\mathbf{A}$ to an upper Hessenberg form first. The QR algorithm is then used to further reduce the matrix to a real Schur form (RSF) from which the eigenvalues are easily computed.

Beside the eigenvalues, also the invariant zeros play an important role for linear dynamic systems. They identify those exponential input signals that are completely blocked by the system. A complex number $s = z_k$ is an invariant zero of a state space system

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{Ax} + \mathbf{Bu} \\ \mathbf{y} &= \mathbf{Cx} + \mathbf{Du} \end{aligned}$$

if the Rosenbrock matrix

$$\mathbf{P}(s) = \begin{pmatrix} s\mathbf{I} - \mathbf{A} & \mathbf{B} \\ -\mathbf{C} & \mathbf{D} \end{pmatrix}$$

is rank deficient for $s = z_k$, i.e.

$$rank(\mathbf{P}(z_k)) < \max_s (rank(\mathbf{P}(s))) \qquad (1)$$

(note, that the modification of the signs in the matrix do not change the rank). If the system has the same number of inputs and outputs, the invariant zeros are the generalized eigenvalues of the pair of square matrices $(\mathbf{L}, \mathbf{M})$ with

$$\mathbf{L} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}, \quad \mathbf{M} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}.$$

In this case the generalized eigenvalues, i.e., the invariant zeros, could be computed with a standard QZ-algorithm, e.g., with LAPACK function dggev. However, this algorithm fails if the system has not the same number of inputs and outputs or if one of the matrices $\mathbf{B}$ or $\mathbf{C}$ does not have full column or full row rank respectively. For this reason, in the LinearSystems library the algorithm from [5] is used to calculate the invariant zeros of arbitrary StateSpace systems, i.e., with arbitrary numbers of inputs and outputs and rank deficient matrices. The approach is to compress the matrices $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ with QR-decompositions to $(\mathbf{A}_r, \mathbf{B}_r, \mathbf{C}_r, \mathbf{D}_r)$ such that a reduced order system matrix

$$\mathbf{P}_r(s) = \begin{pmatrix} s\mathbf{I} - \mathbf{A}_r & \mathbf{B}_r \\ -\mathbf{C}_r & \mathbf{D}_r \end{pmatrix}$$

with invertible matrix $\mathbf{D}_r$ have the same zeros as $\mathbf{P}(s)$. After another transformation that compresses the columns of $[\mathbf{C}_r, \mathbf{D}_r]$ to $[\mathbf{0}, \mathbf{D}_f]$ such that

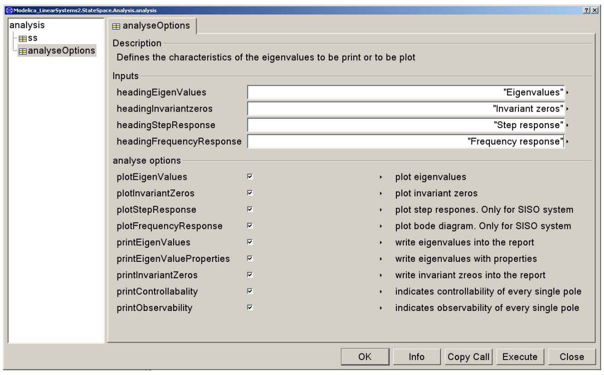
$$\begin{pmatrix} \mathbf{A}_f & * \\ \mathbf{0} & \mathbf{D}_f \end{pmatrix} = \begin{pmatrix} \mathbf{A}_r & \mathbf{B}_r \\ \mathbf{C}_r & \mathbf{D}_r \end{pmatrix} \mathbf{V}$$

and

$$\begin{pmatrix} \mathbf{B}_f & * \\ \mathbf{0} & \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \mathbf{V}$$

the QZ-algorithm is applied to calculate the generalized eigenvalues of the pair (A_f, B_f). These generalized eigenvalues are the invariant zeros of the system.

Controllability, observability, stability, detectability, and stabilizability are computed with numerically effective staircase algorithms. All properties can be computed with the convenience function "analyze" that calls all functions from the Analysis package and presents the results in a nice layout in html format. The requested results are selected by a menu (see Figure 5). Furthermore, the dynamics of the states are being analyzed in relation to the dynamics of the modal states, i.e., the states of the corresponding similar uncoupled modal system representation. This helps to understand which eigenvalue and/or eigen response is associated with which variable.



analyseOptions

Description

Defines the characteristics of the eigenvalues to be print or to be plot

Inputs

headingEigenValues	"Eigenvalues"
headingInvariantzeros	"Invariant zeros"
headingStepResponse	"Step response"
headingFrequencyResponse	"Frequency response"

analyse options

plotEigenValues	☑	plot eigenvalues
plotInvariantZeros	☑	plot invariant zeros
plotStepResponse	☑	plot step respones. Only for SISO system
plotFrequencyResponse	☑	plot bode diagram. Only for SISO system
printEigenValues	☑	write eigenvalues into the report
printEigenValueProperties	☑	write eigenvalues with properties
printInvariantZeros	☑	write invariant zreos into the report
printControllabality	☑	indicates controllability of every single pole
printObservability	☑	indicates observability of every single pole

Figure 5: Menu of Analysis.analysis function

The following example illustrates this relation: The state space system with the matrices

$$A = \begin{pmatrix} -3 & 2 & -3 & 4 \\ 0 & 6 & 7 & 8 \\ 0 & 13 & 34 & 0 \\ 0 & -17 & 0 & 0 \end{pmatrix}, \; B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \qquad D = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

has the four eigenvalues

$$\lambda_1 = -3, \;\lambda_2 = 36.65, \;\lambda_{3,4} = 1.67 \pm 11.11i.$$

The system analysis function reports that this system is neither stable nor observable but it is controllable and therefore it is stabilizable and it is detectable. In particular:

	characteristics
λ_1	stable, controllable, not observable
λ_2	not stable, stabilizable, detectable
$\lambda_{3,4}$	not stable, stabilizable, detectable

The contribution of the modal states $z_i(t)$ to the system states $x_i(t)$ and the characteristics of the dynamic behavior of the modal states is given by the following tables on the result file in html format:

i	λ_i	T[s]	z[i] contributes
1	-3	0.333	to x[1] with 100 %
2	36.65	0.273	to x[3] with 72.8 % to x[2] with 14.8 %

i	λ_i	z[i] contributes
3/4	1.67 ± i 11.11	to x[4] with 43.9 % to x[2] with 29 %

i	frequency[Hz]	damping
3/4	1.7877	-0.1492

The relation of the system states to the modal states, i.e., the composition of the system states is shown in the next table:

System state	is composed of
x[1]	60.6 % by z[1] 34.9 % by z[3/4]
x[2]	83.9 % by z[3/4] 16.1 % by z[2]
x[3]	71.2 % by z[2] 28.8 % by z[3/4]]
x[4]	94.4 % by z[3/4] 5.6 % by z[2]

The meaning of contribution and composition is explained subsequently. The idea of this approach is, that the zero input response

$$\mathbf{x}_h(t) = e^{\mathbf{A}t}\mathbf{x}_0$$

of a state space system can be represented by a transformation

$$\mathbf{x}_h(t) = \tilde{\mathbf{V}}\mathbf{z}_h(t) \tag{2}$$

of the decoupled zero input responses $\mathbf{z}_h(t)$ of the corresponding modal system

$$\dot{\mathbf{z}} = \tilde{\mathbf{V}}^{-1}\mathbf{A}\tilde{\mathbf{V}}\mathbf{z} = \tilde{\mathbf{\Lambda}}\mathbf{z}$$

with

$$z_k(t) = e^{\lambda_i t} z_{k0}$$

for a single real eigenvalue λ_k or

$$\begin{pmatrix} z_k(t) \\ z_{k+1}(t) \end{pmatrix} = e^{\boldsymbol{\Gamma}_k t} \begin{pmatrix} z_{k0} \\ z_{k+1_0} \end{pmatrix}$$

for a single pair of conjugated complex eigenvalues $(\lambda_k, \lambda_{k+1}) = \alpha_k \pm i\beta_k$ respectively. The matrix

$$\boldsymbol{\Gamma}_k = \mathbf{T}^{-1}\boldsymbol{\Lambda}_k\mathbf{T} = \begin{pmatrix} \alpha_k & \beta_k \\ -\beta_k & \alpha_k \end{pmatrix}$$

with

$$\mathbf{T} = \frac{1}{2}\begin{pmatrix} 1 & -j \\ 1 & j \end{pmatrix} \text{ and } \boldsymbol{\Lambda}_k = \begin{pmatrix} \alpha_k + j\beta_k & 0 \\ 0 & \alpha_k - i\beta_k \end{pmatrix}$$

results from a transformation of the complex eigenvalue matrix $\boldsymbol{\Lambda}_k$ into the corresponding real form. The matrix $\tilde{\mathbf{V}}$ is given by

$$\tilde{\mathbf{V}} = \mathbf{V}\begin{pmatrix} \hat{\mathbf{T}} & \mathbf{0} \\ 0 & \mathbf{I} \end{pmatrix}$$

where $\hat{\mathbf{T}}$ is a block diagonal matrix of order $2r$ with r matrices $\mathbf{T}$ in its diagonal. Note, that the matrix $\boldsymbol{\Lambda} = \text{diag}\,(\lambda_k)$ is assumed to be appropriately sorted, i.e. the first $2r$ elements of the diagonal are the r conjugated complex pole pairs of the system. Considering (2), each element $x_{h_k}(t)$ of $\mathbf{x}_h(t)$ is represented by a linear combination $\tilde{\mathbf{v}}_k^T\mathbf{z}_h$ of the zero input responses $z_{h_k}(t)$, where $\tilde{\mathbf{v}}_k^T$ indicates the k'th row of matrix $\tilde{\mathbf{V}}$. Furthermore, x_{h_k} is composed by $p_{kl} = 100 \cdot \left(|\tilde{v}_{kl}|/|\tilde{\mathbf{v}}_k^T|\right)\%$ by the element z_{h_l}. On the other hand, equation (2) can be written as

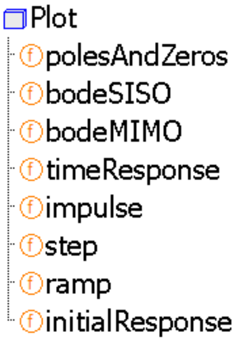
$$\mathbf{x} = \tilde{\mathbf{V}}\mathbf{z} = \sum_{k=1}^{n} z_k\tilde{\mathbf{v}}_k,$$

i.e., for each modal state z_{h_k} of $\mathbf{z}_h$ the elements $q_{lk} = 100 \cdot |\tilde{v}_{lk}|/|\tilde{\mathbf{v}}_k|\%$ of the corresponding vector $\tilde{\mathbf{v}}_k$ indicates the proportion of z_{h_k} that is contributed to the state x_{h_l}. Since the state x_k is assigned to a system variable, which can probably be assigned to a physical component or a certain subsystem, then p_{kl} and q_{lk} help to indicate the influence of this subsystem to the dynamical behavior of the system which, mathematically, is composed of the dynamics of the modal states $\mathbf{z}_k$ associated to the eigenvalues λ_k.

Currently, the poles are only considered as real poles or pole pairs with multiplicity 1. Systems with eigenvalues of higher multiplicity are processed as multiple eigenvalues of multiplicity 1. This should be improved in the future.

2.2 StateSpace.Plot

The sublibrary Plot contains functions to plot poles, zeros, frequency responses and various time responses. For the plotting, a separate small package is provided that must be adapted to the features of the used Modelica tool, since plotting is not standardized in Modelica. Currently, the plot package is only provided for Dymola.



Plot
polesAndZeros
bodeSISO
bodeMIMO
timeResponse
impulse
step
ramp
initialResponse

Figure 6: Package StateSpace.Plot

For example, the following function call computes the step responses of a state space system from every input to every output and plots the corresponding curves:

```
StateSpace.Plot.step(
    ss=ss, dt=0.05,tSpan=10);
```

Figure 7 shows a step response of a SISO state space system with the matrices

$$\begin{aligned}\mathbf{A} &= \begin{pmatrix} 0 & 1 \\ -25 & -1.5 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 0 \\ 25 \end{pmatrix}, \\ \mathbf{C} &= \begin{pmatrix} 1 & -0.5 \end{pmatrix}, \quad \mathbf{D} = \mathbf{0}.\end{aligned}$$

Figure 8 shows the corresponding bode diagram generated with `StateSpace.Plot.bodeSISO(ss=ss);`

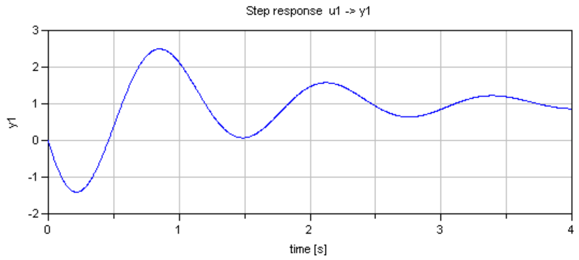


Figure 7: Step response

2.3 StateSpace.Design

The Design sublibrary (see Figure 9) provides standard controller design methods for linear systems. The pole assignment design function `StateSpace.assignPolesMI()` for StateSpace systems with one or more inputs is based on the

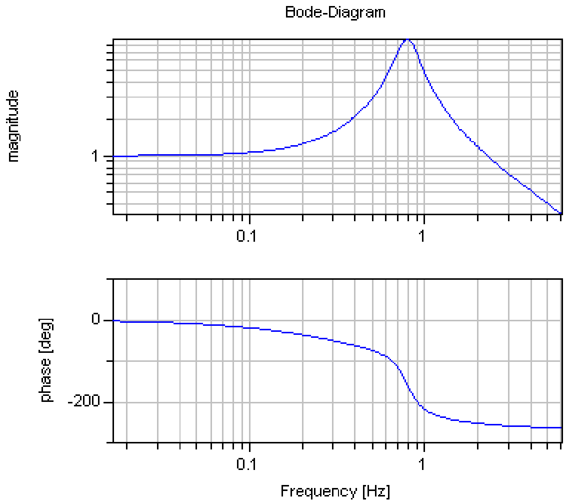


Figure 8: Bode diagram

approach described in [11], where the system matrix $\mathbf{A}$ is reduced to a real Schur form that allows sequential and/or partial eigenvalue assignment.

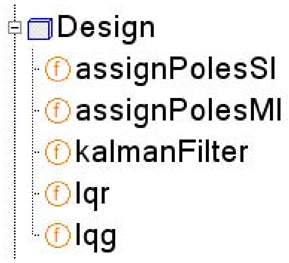

Design
assignPolesSI
assignPolesMI
kalmanFilter
lqr
lqg

Figure 9: Package StateSpace.Design

The standard design method of a ”linear quadratic optimal controller“ computes the matrix $\mathbf{K}$ of the state-feedback law $\mathbf{u} = -\mathbf{K}\mathbf{x}$ in such a form that a quadratic cost function with symmetric weighting matrices $\mathbf{Q}$ and $\mathbf{R}$

$$J(\mathbf{x}) = \int_0^\infty \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\,\mathbf{u}\,dt \qquad (3)$$

is minimized. The feedback matrix $\mathbf{K}$ is obtained from the solution $\mathbf{X}$ of the algebraic Riccati equation

$$\mathbf{Q} + \mathbf{A}^T\mathbf{X} + \mathbf{X}\mathbf{A} - \mathbf{X}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{X} = \mathbf{0}. \qquad (4)$$

In the LinearSystems library the linear quadratic optimal controller design is performed with the function call `(K, sslqr, X, evlqr) = StateSpace.lqr(ss, Q, R)`. The first output of the function is the optimal and stabilizing gain matrix $\mathbf{K}$. It is calculated from the solution $\mathbf{X}$ of (4) by $\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{X}$. Additionally, the complex output vector `evlqr` contains the closed loop system eigenvalues, i.e., the eigenvalues of the matrix $\mathbf{A} - \mathbf{B}\mathbf{K}$. The record `sslqr` contains the system representation of the closed loop system

$$\begin{aligned}\dot{\mathbf{x}} &= (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} + \mathbf{B}\mathbf{u}\\ \mathbf{y} &= (\mathbf{C} - \mathbf{D}\mathbf{K})\mathbf{x} + \mathbf{D}\mathbf{u}.\end{aligned}$$

The corresponding control system structure for the state feedback control is depicted in Figure 11 and is available as a model template in the Controller library. Alternatively, a feedback control structure with observer, as shown in Figure 12, is also available.

Kalman filter design is related to the LQR-problem. Actually, the corresponding design function makes use of `StateSpace.Design.lqr()` but with a dual system for the observer problem. Kalman filters are computed with `(L, sskf)=StateSpace.kalmanFilter(ss, Q, R)`. Beside the filter matrix $\mathbf{L}$ the output `sskf` represents the system together with the Kalman filter, i.e.

$$\begin{aligned}\dot{\mathbf{x}} &= (\mathbf{A} - \mathbf{L}\mathbf{C})\mathbf{x} + \begin{pmatrix}\mathbf{B} - \mathbf{L}\mathbf{D}, & \mathbf{L}\end{pmatrix}\begin{pmatrix}\mathbf{u}\\ \mathbf{y}\end{pmatrix}\\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \begin{pmatrix}\mathbf{D}, & \mathbf{0}\end{pmatrix}\begin{pmatrix}\mathbf{u}\\ \mathbf{y}\end{pmatrix}.\end{aligned}$$

LQG design determines the matrices $\mathbf{K}_c$ and $\mathbf{K}_f$ for linear quadratic gaussian problems (LQG), i.e., the minimization of the expected value of a cost function under the assumption of stochastically disturbed states and outputs. Therefore, it is a combination of LQR design and Kalman filter design. The function `(Kc, Kf, sslqg)=Design.lqg(ss, Q, R, V, W)` returns the controller matrix $\mathbf{K}_c$ and the filter matrix $\mathbf{K}_f$. Again, the matrices $\mathbf{Q}$ and $\mathbf{R}$ are weighting matrices in the cost function of the controller. The matrices $\mathbf{V}$ and $\mathbf{W}$ are assumed to be the covariance matrices of the disturbances $\mathbf{v}$ and $\mathbf{w}$ respectively but due to the lack of deeper insight are usually treated as weighting matrices. The output record `sslqg` represents the estimated system

$$\begin{aligned}\dot{\hat{\mathbf{x}}} &= (\mathbf{A} - \mathbf{K}_f\mathbf{C} - \mathbf{B}\mathbf{K}_c + \mathbf{K}_f\mathbf{D}\mathbf{K}_c)\hat{\mathbf{x}} + \mathbf{K}_f\mathbf{y}\\ \hat{\mathbf{y}} &= (\mathbf{C} - \mathbf{D}\mathbf{K}_c)\hat{\mathbf{x}}\end{aligned}$$

with $\mathbf{y}(t)$, the output of the original system, as input.

The solution of the Riccati equation (4) is provided by `X = Modelica_LinearSystms2.Math.Matrices.care(A, B, R, Q)`, which computes the solution with a Schur vector method for the continuous algebraic Riccati equation (CARE) [2, 7]. The approach is to form the $2n \times 2n$ Hamilton matrix (note that matrix $\mathbf{A}$ has order n)

$$\mathbf{H} = \begin{pmatrix}\mathbf{A} & -\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\\ -\mathbf{Q} & -\mathbf{A}^T\end{pmatrix}$$

and to transform it into an ordered real Schur form

$$\mathbf{T} = \begin{pmatrix} \mathbf{T}_{11} & \mathbf{T}_{12} \\ \mathbf{0} & \mathbf{T}_{22} \end{pmatrix} = \mathbf{U}^T\mathbf{H}\mathbf{U}$$

where $\mathbf{T}_{11}$ contains the n eigenvalues of $\mathbf{H}$ with negative real parts [2]. Considering an appropriate partitioning of matrix $\mathbf{U}$

$$U = \begin{pmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ \mathbf{U}_{21} & \mathbf{U}_{22} \end{pmatrix},$$

with $n \times n$ matrices $\mathbf{U}_{ij}$, the solution $\mathbf{X}$ can be computed by solving

$$\mathbf{X}\mathbf{U}_{11} = \mathbf{U}_{21}.$$

Additionally, an optional refinement function based on Newton's Method with exact line search is provided [1].

3 Controller library

The former beta version of the library Modelica_LinearSystems [9] contained a sub-package *Sampled* with input/output blocks for continuous and discrete linear systems simulation. This package was extended and was adapted to the Modelica_LinearSystems2 library sketched in the previous section. Furthermore, the package was renamed and is now called ”Modelica_LinearSystems2.Controller“. The Controller library contains input/output blocks for StateSpace, TransferFunction and ZerosAndPoles systems, as well as PI, PID, FirstOrder, SecondOrder, Integrator, Filter blocks etc. Every block is available in a continuous and a discrete (sampled) representation, where the representation can be chosen by a Boolean parameter. By specifying a discretization method and a sample time, the discrete representation is automatically derived from the continuous form.

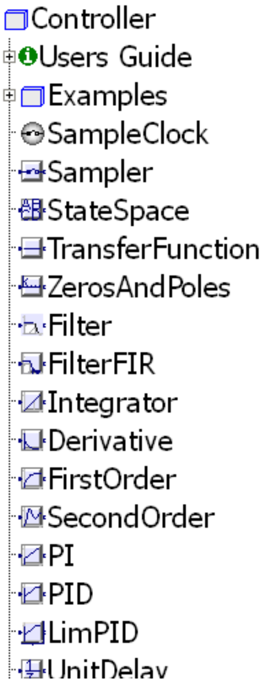

Controller
Users Guide
Examples
SampleClock
Sampler
StateSpace
TransferFunction
ZerosAndPoles
Filter
FilterFIR
Integrator
Derivative
FirstOrder
SecondOrder
PI
PID
LimPID
UnitDelay

Figure 10: Controller library

Besides standard input/output blocks, especially for the data structures of the LinearSystems library, a “Template” sublibrary is present which provides standard controller structures with replaceable components. As an example, a state feedback control is shown in Figure 11, and state feedback control based on estimated states using an observer is shown in Figure 12.

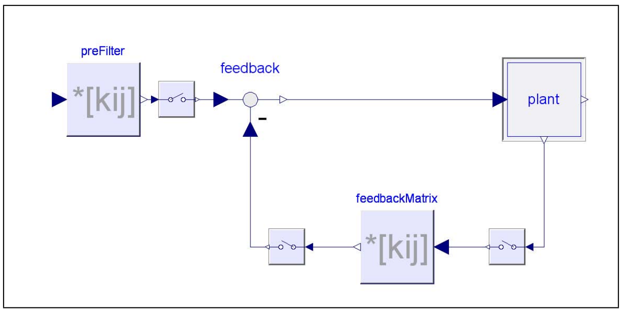


Figure 11: Template for state feedback control

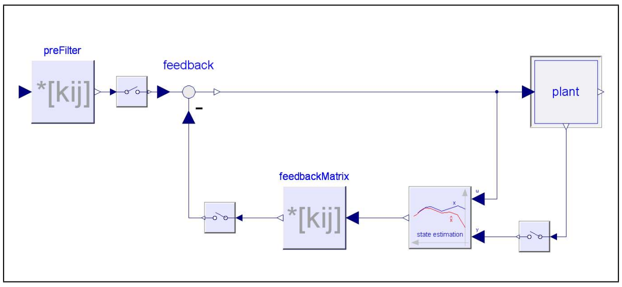


Figure 12: Template for state feedback control with observer

The observer structure is also provided as a template (Figure 13) which automatically adapts the dimensions to the loaded system.

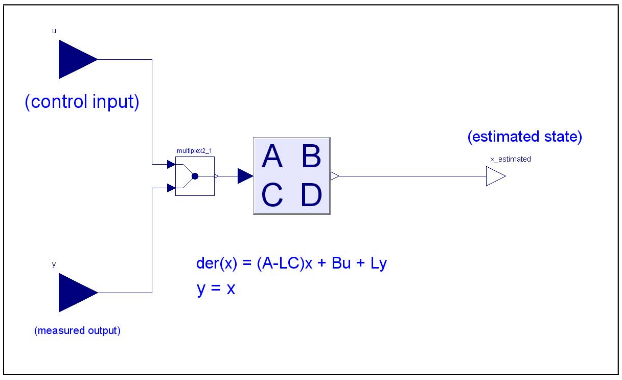


Figure 13: Template for observer

Figure 14 shows a two degree of freedom controller template with a (usually non-linear) inverse system model in the feed forward loop. How to utilize non-linear inverse (Modelica) models in controllers is described in [10]. The templates are provided to support quick implementations of controllers by simple redeclarations of the replaceable components.

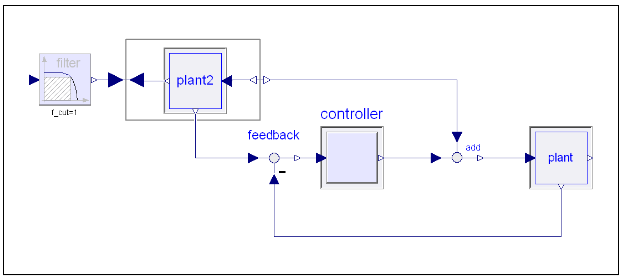


Figure 14: Template for a two degree of freedom controller.

4 Example

Figure 15 shows the multi-body model of a simple inverse double pendulum and Figure 16 shows the corresponding animation of the simulated system.

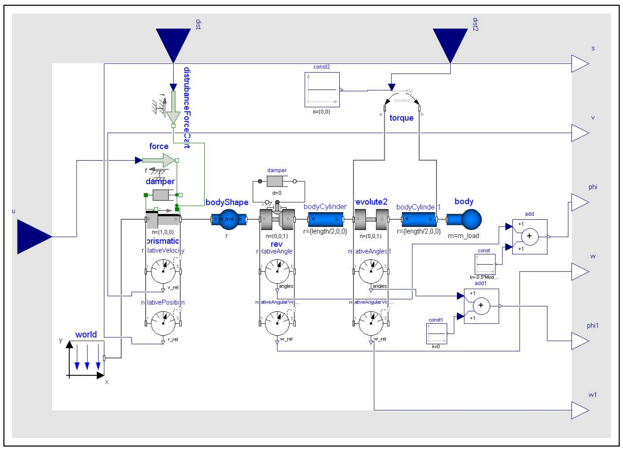


Figure 15: Multi-body model of inverse double pendulum

The input of the system is a one-dimensional horizontal force to move the cart. It is assumed that the position and the angle between the lower rod and the cart are measurable. Hence, the velocity, the angle of the upper joint, and the angular velocities of the joints have to be estimated by an observer. Disturbances can be conditionally activated and can be added to the horizontal force input and as additional torque at the upper joint applied at the upper rod. The control task is to track the cart to a given (time varying) position without dropping the pendulum.

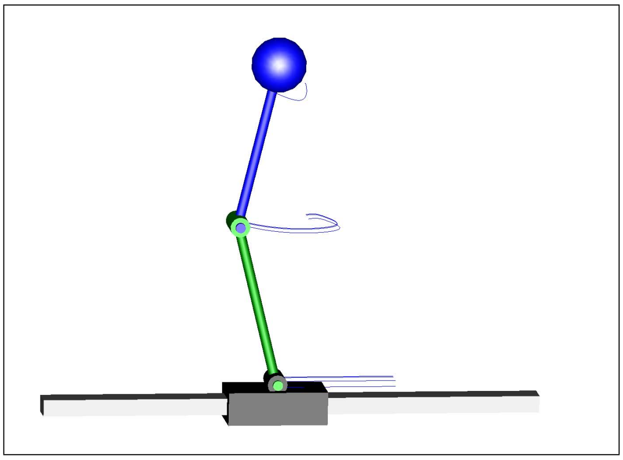


Figure 16: Screenshot of animated simulation

4.1 Controller design

Linearization of the system around the vertical position of the double pendulum is performed numerically with function `StateSpace.Import.fromModel(modelName)`, which returns an appropriate instance of the StateSpace record. Both, the controller and the observer are designed by pole assignment. With the state space system `ss` of the linearized model and the desired poles p_c the feedback matrix K_{p_c} is calculated by calling `K_pc := assignPolesMI(ss, pc);`

Due to the duality of controllability and observability, function `assignPolesMI()`can also be used to design the observer feedback matrix $\mathbf{K}_{p_o}$. The inputs are the dual state space system $\left(\mathbf{A}^T, \mathbf{C}^T, \mathbf{B}^T, \mathbf{D}^T\right)$. Finally, the pre filter, see Figure 12, is designed such that the model would follow a constant input in case of steady state behavior:

```
M_pa := -inv(ss.C*Matrices.solve2(
  ss.A - ss.B*K_pc, ss.B));
```

4.2 Simulation

The model of the controlled system (Figure 17) is extended from the corresponding template shown in Figure 12. To fit the template, the physical system model must be put into an appropriate form as depicted in Figure 18. The data of the controller are directly copied into the controller

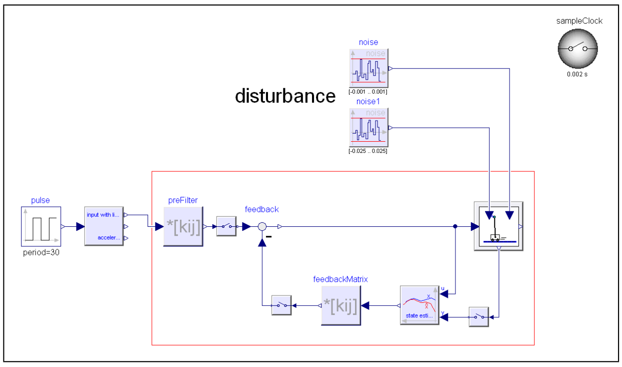


Figure 17: Block diagram of the controlled system

matrix or loaded from a file.

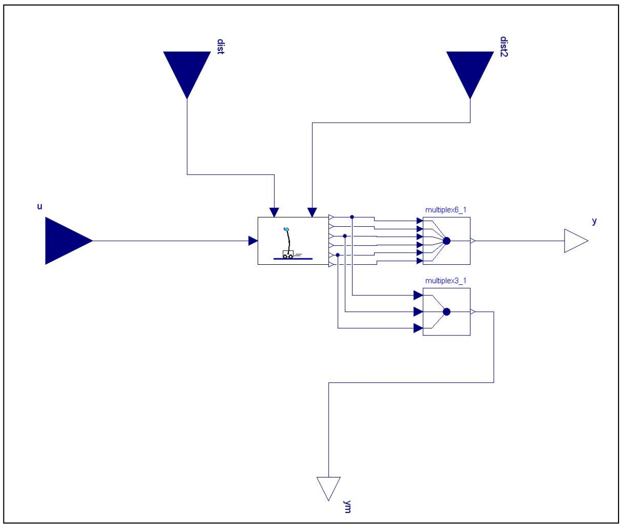


Figure 18: Model to fit the physical model of the template shape

The characteristics of the undisturbed controlled continuous system is shown in Figure 19. In the upper picture, the position of the inverted pendulum and its set point are depicted. The lower picture shows the corresponding input force. Figure 20 shows the same signals of a disturbed system with a discrete controller.

The disturbances have of course much impact on the angle of the upper joint. By comparison, Figure 21 shows the first 15 seconds of this angle and the corresponding estimated value for the undisturbed system (upper picture) and the disturbed system (lower picture).

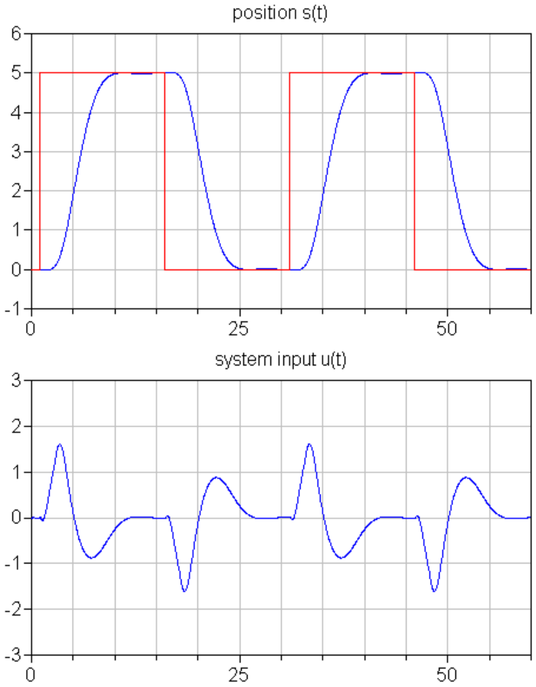


Figure 19: Controlled system without disturbances and with continuous controllers. Position of the inverted pendulum and its setpoint (above) and the input force (below)

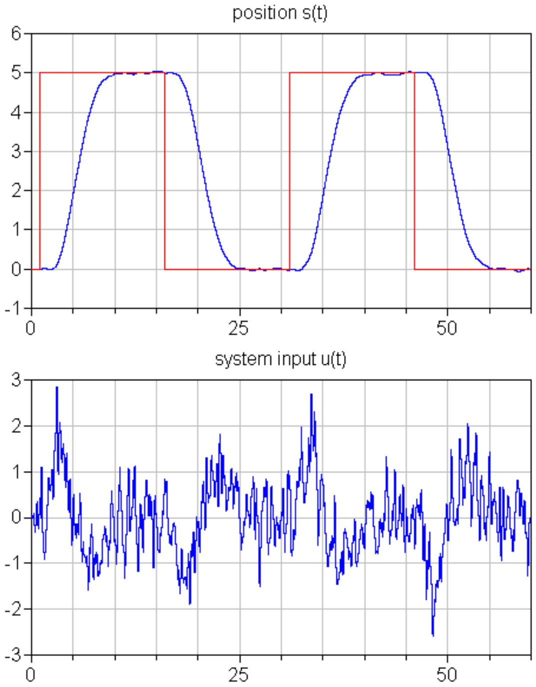


Figure 20: Controlled system with disturbances and with discrete controllers

5 Conclusions

This paper has sketched the Modelica libraries LinearSystems and Controller, i.e. a library for linear systems analysis and synthesis and a library to model continuous and discrete linear controller blocks. Compared with the former version

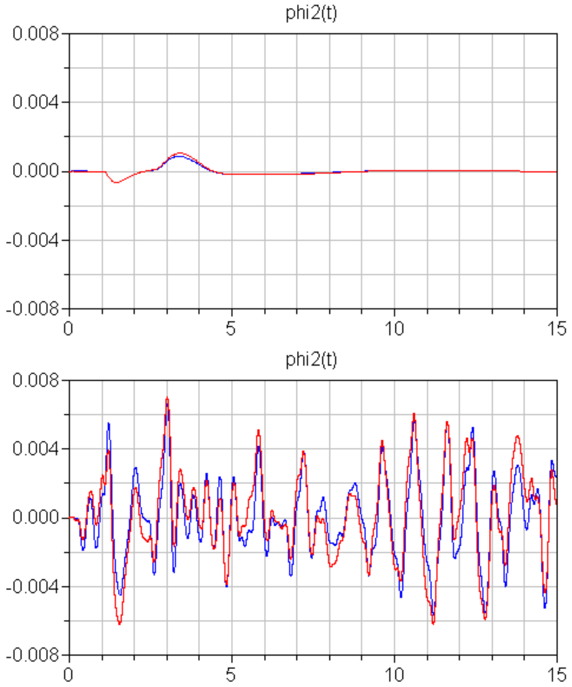


Figure 21: Angle φ_2 (upper joint) and estimated value (red line) for undisturbed (above) and disturbed (below) system

of the LinearSystems library, the current version has been extensively restructured and new functions have been added. The most important functions of this library have been described for the case of state space systems and the mathematical approaches have been outlined.

Concerning the Controller library, the new feature to use templates for common control system structures have been introduced. An example of the design of a controller for a inverse double pendulum demonstrates the usage of the libraries. It is planned to include the libraries in the Modelica Standard library. Currently, they are available from http://www.Modelica.org/libraries.

Acknowledgements

Partial financial support of DLR by BMBF (BMBF Förderkennzeichen: 01IS07022F) for this work within the ITEA project EUROSYSLIB (http://www.itea2.org/public/project_leaflets/ EUROSYSLIB_profile_oct-07.pdf) is highly appreciated.

We would also like to thank Hilding Elmqvist and Hans Olsson from Dassault Systèmes (Dynasim) for their support and for discussions especially related to operator overloading.

References

[1] Benner P. and Byers, R. (1998): **An Exact Line Search Method for Solving Generalized Continuous-Time Algebraic Riccati Equations**. IEEE Transactions on Automatic Control, vol 43, pp. 101-107

[2] Datta B. N. (2004): **Numerical Methods for Linear Control Systems**. Elsevier Academic Press.

[3] Dymola (2009): **Dymola Version 7.3**. Dassault Systèmes, Lund, Sweden (Dynasim). http://www.dymola.com.

[4] Elmqvist H., Otter M., Henriksson D., Thiele B., and Mattsson S. E. (2009): **Modelica for Embedded Systems**. In: Proc. of the 7th Modelica Conference 2009, Como, Italy, Sept. 20-22. http://www.modelica.org/events/modelica2009.

[5] Emami-Naeini, A. and Van Dooren, P. (1982): **Computations of zeros of linear multivariable systems**, Automatica 26, pp. 415-430

[6] LAPACK (2009): http://www.netlib.org/lapack/.

[7] Laub A. J. (1979): **A Schur Method for Solving Algebraic Riccati equations**. IEEE Trans. Auto. Contr., vol 24, pp. 913-921.

[8] Olsson H., Otter M., Elmqvist H., and Brück D. (2009): **Operator Overloading in Modelica 3.1**. In: Proc. of the 7th Modelica Conference 2009, Como, Italy, Sept. 20-22.

[9] Otter M. (2006): **The LinearSystems library for continuous and discrete control systems**. In: Proc. of the 5th Modelica Conference 2006, Wien, Austria, Sept. 4-5. http://www.modelica.org/events/modelica2006/-Proceedings/sessions/Session5c1.pdf

[10] Thümmel M., Looye G., Kurze M., Otter M., and Bals J. (2005): **Nonlinear Inverse Models for Control**. In: Proc. of the 4th Int. Modelica Conference 2005, Hamburg, March 7-8. http://www.modelica.org/events/Conference2005/-online_proceedings/Session3/Session3c3.pdf

[11] Varga A. (1981): **A Schur method for pole assignment**. IEEE Trans. Autom. Control, Vol. AC-26, pp. 517-519.

Dymola and Modelica_EmbeddedSystems in Teaching – Experiences from a Project Course

Johan Åkesson[ab] Ulf Nordström[c] Hilding Elmqvist[c]
[a]Department of Automatic Control, Lund University, Sweden
[b]Modelon AB, Lund, Sweden
[c]Dassault Systèmes, Lund, Sweden (Dynasim)
Johan.Akesson@control.lth.se Ulf.Nordstrom@3ds.com Hilding.Elmqvist@3ds.com

Abstract

This contribution presents experiences from a master level project course where the Modelica-based tool Dymola, supporting embedded control system design, has been used. In a recent initiative, the Modelica language has been enhanced to support modeling of embedded systems and code generation targeted at micro processors.[1] The new specification is supported by Dymola and enables wide range of design tasks to be performed in a unified framework. Such tasks include software in the loop simulation to test controller code in simulation, hardware in the loop simulation, and final deployment on the target. In the context of teaching, the new features of Modelica/Dymola enable universities to offer a realistic environment providing students with hands on experiences from model-based control system development.

Keywords: Modelica; Dymola; Embedded Control Systems; Teaching

1 Introduction

Much effort is devoted to studies of analysis and synthesis methods in engineering programs oriented towards systems and control. Often, the course material is mainly of theoretical nature, sometimes complemented with laboratory sessions. To further strengthen the practical skills of the students, a project course, “Projects in Automatic Control” is offered by the Department of Automatic Control, Lund University. The main themes of the course are practical application of theoretical skills acquired in previous courses and working in teams.

This contribution describes two projects that were part of the course of 2009, where Dymola and Modelica_EmbeddedSystems was used to develop control systems for two-wheel robots, Figure 1, built using the Lego Mindstorms NXT platform.

The paper is outlined as follows. In Section 2, an overview is given over different approaches to teaching embedded systems and control, and in Section 3 the Project in Automatic Control course is described. Section 4 and 5 describes, respectively, the Modelica_EmbeddedSystems library and the LEGO_Mindstorms library. In Section 6, some common usage scenarios are discussed and in Section 7 the fixed point code generation module of Dymola is outlined. The paper ends with a review of the course results in Section 8 and a summary in Section 9.

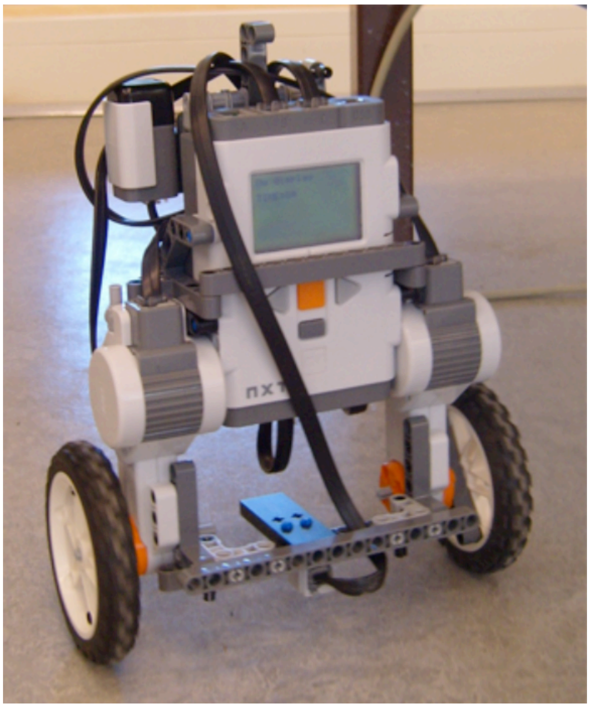


Figure 1. Lego robot

[1] This effort has been performed within the EUROSYSLIB project.

DOI: 10.3384/ecp09430086

2 Background

Teaching of embedded control systems requires insights into different disciplines, including mathematical modeling, control system design, and computer science. In the latter case, real-time systems are particularly important. Embedded control systems are distinguished by the complex interplay between the behaviors of the controller to be implemented, typically designed in continuous time, the discretization method used in order to obtain a discrete time approximation of the controller, and the properties of the execution environment. In order to analyze the closed loop behavior of the controlled system, all three aspects need to be attended to.

Teaching of embedded systems can be approached in several ways, using different levels of abstraction. At the lowest level, control systems are encoded in C, or even assembly. The control system is then typically run without an operating system and periodic processes, or tasks, are mapped onto timer interrupts. Using this approach, the modeling and control systems design is typically done prior to the encoding phase, using different tools and methodologies. It is also common that controllers need to be translated into fixed-point arithmetics. From a pedagogical perspective, this method has distinct advantages and disadvantages. Coding an embedded control system in a low level language, perhaps including manual fixed-point conversion, does indeed promote understanding of the tasks involved. Also, mapping of periodic tasks onto hardware interrupts further strengthens the student’s understanding of the methods involved. On the other hand, modeling and control system design is disjoint from the actual encoding and execution of the control system. Debugging is often further complicated by limited means to log signals in the embedded control application.

At the next level of abstraction, a high-level language, relative to C or assembly, can be used for implementation. For example, Java offers suitable abstractions for creating periodic tasks, e.g., threads and synchronization. Also, there are platforms providing Java support, including Lego Mindstorms NXT. Modeling and control system design, proceeds, however, as with the previous approach, and is typically disjoint from the actual implementation. Never the less, this approach captures important aspects of embedded control system, such as multi-threaded applications and the consequences thereof.

In order to promote joint modeling, control systems design and implementation, tools like Real-Time Workshop for Matlab/Simulink are available. A similar tool is Scilab/Scicos. Such tools offer strong support for block-based modeling, which is well suited for development of control systems. Real-Time Workshop may then translate the block-oriented graphical Simulink model into executable C code, which in turn can be compiled and downloaded to the target processor. Using the simulation capabilities of Simulink, the control system can be simulated together with a model of a physical plant in order to assess the closed loop behavior prior to deployment. There is also a toolbox for fixed-point arithmetics available for Simulink as well as a freely available toolbox for simulation of the temporal behavior of embedded kernels and computer networks, TrueTime [1].

The approach taken in this paper is similar to that of Matlab/Simulink and Real-Time Workshop. The simulation software Dymola is used for physical modeling as well as development of the control system. Modelica is used as modeling and implementation language. As compared to Matlab/Simulink, Modelica offers stronger support for physical modeling, and supports advanced modeling concepts such as object orientation, equations, and acausal connections between components. An additional advantage of Modelica is that the code is available to the user, which adds to the transparency of the method. Using novel features of Dymola and additions to the Modelica language explored in the Modelica_Embedded library, it is possible to generate C code, automatically translated to fixed-point if desired, corresponding to the control system. The generated C code may then be either compiled and downloaded to the target or compiled and linked with a simulation executable. The latter case enables detailed study, in simulation, of the closed loop behavior of the system prior to deployment.

The method of automatic code generation from a high-level description is a novel addition to the course portfolio of Automatic Control. Joint control system design and implementation on embedded platforms has been a long-standing theme of the department, both in research and in teaching, but so far, C and Java (and previously also Modula-2) has been used as implementation languages. Dymola and Modelica therefore offer an appealing complement for providing the students with experiences from a different environment.

3 Project in Automatic Control

The Department of Automatic Control has a long tradition of laboratory work in control education. Laboratory sessions are included in all theoretically oriented courses and some courses also offer small projects. In order to further strengthen the practical and experimental skills of the students, a dedicated project course is offered to master's level students. The course gives 7.5 ECTS units and is categorized as advanced level. The syllabus of the course changes each year depending on the number of students and the availability of interesting projects, usually with connection to research or industrial applications. The projects are typically set up so that the students need to go through several steps in the design cycle, including mathematical modelling, parameter identification based on measurement data, control design, control system implementation, user interaction and testing. The examination of the course consists of weekly meetings with a teacher, a written report and an oral presentation.

The students in the course have in most cases taken several control courses covering topics such as linear and non-linear control system design, multivariable control, sampled systems and real-time systems. For a list of courses offered by the Department of Automatic Control, see [2]. In the project course, the students need to apply their knowledge from previous courses in order to solve a larger design problem in a team consisting of three to five students. More often than not, the course helps the students to put their theoretical knowledge into a practical perspective where sensors and actuation, unit conversions, and limited computing resources play important roles.

For the course as of spring 2009, the Lego Mindstorms NXT [6] platform was selected as a basis for the course projects. The platform features several possibilities for sensors and actuations, also from third party manufacturers, and the embedded micro processor can be programmed in several ways using e.g., C/C++, NXC or Java. Out of 22 students in total, two teams of five students in each were selected to perform projects where the Dymola software was used for modeling, control design and embedded code generation.

3.1 Project infrastructure support

In order to emphasize and support the collaborative character of the projects, a version control repository and a web-based tool for project planning were made available for each project group. As for version control, Subversion [3] was used and Trac [4] was used as project planning platform. The objective of introducing these tools in the course was to add an additional element of industrial realism to the projects. Also, the students were required to prepare each weekly meeting with their teacher by updating the Trac site to reflect the current status of the project.

Throughout the projects, the students had access to a lab where the Lego sets and computers for development were available.

3.2 Tutorials

All students in the course were offered a tutorial on how to use Trac and Subversion, since few had any experience of such tools. The students participating in the Dymola projects were offered additional tutorial lectures in order to get started with the course work. A basic tutorial on how to operate the Lego Mindstorms NXT hardware was offered in the beginning of the course, with the objective of introducing the students to basic operation such as reading from sensors, compilation of programs, and downloading and running programs. Since the students lacked previous experience with Modelica, an introductory lecture was given. The tutorial covered basic Modelica features, including textual and graphical modeling, as well as an introduction to Dymola. Finally, a lecture on advanced Modelica and multibody modeling was offered, covering also the animation features of Dymola. The final lecture was given by personel from Dynasim, whereas the three first were given by personel from the Department of Automatic Control.

The initial tutorial lectures given early in the course provided the students with sufficient information to get started with Modelica and Dymola. However, some additional support in the form of informal tutorials in front of the computer was also needed, especially in order for the students to learn how to use the new advanced features related to Modelica_EmbeddedSystems and code generation.

3.3 Project task

The task for the students to solve was to construct a two-wheel robot, see Figure 1, and to develop a model-based stabilizing control scheme using Modelica and Dymola. Dynamic modeling of the robot was done using the multi-body library in Dymola. While modeling of the mechanical parts is fairly straightforward, the Lego servos pose a challenge. In order to obtain a good model for these, identification experiments need to be performed. This was made possible by the data-logging feature of Dymola; a small Modelica test program was downloaded and the resulting signals were logged back to Dymola over the BlueTooth communication link. Having constructed a dynamic model, a linearized approximation can be derived and exported from Dymola. Both groups opted to use a state feedback controller designed using Control Systems Toolbox in Matlab.

Given the controller, sensors and sensor processing needs to be considered. The Lego servos have built-in angular measurements, and in addition, one rate gyro and one accelerometer were available to each group. In the final step, the control system was designed using blocks from the Modelica Standard Library, and the details of the embedded platform were set up. As a parallel task, animation of the robot was set up in Dymola.

4 Modelica_EmbeddedSystems

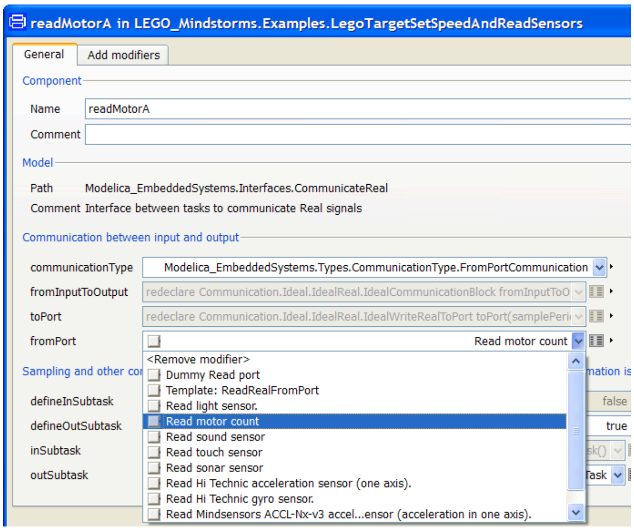
The Modelica_EmbeddedSystems library [5] was used to set up the models for use with embedded systems. Using components from the library, target configuration records and communication points are inserted in the models containing properties of the target system and computational tasks.

4.1 Communication points

One of the key components of the library is the `CommunicateReal` block. It is used to set up communication with external I/O ports of the target system or to model the interface.

In the LEGO_Mindstorms Modelica library, described in a later section, I/O communications blocks were implemented such that they fit in the framework set by the `CommunicateReal` block. The design allows for straight forward use of the GUI (parameter dialog) to enable access to the external I/O blocks by a simple pull-down menu, depicted in Figure 2.

A user could thus implement new I/O blocks that would end up in the same dialog for selection.



readMotorA in LEGO_Mindstorms.Examples.LegoTargetSetSpeedAndReadSensors

Component

Name: readMotorA

Comment:

Model

Path: Modelica_EmbeddedSystems.Interfaces.CommunicateReal

Comment: Interface between tasks to communicate Real signals

Communication between input and output

communicationType: Modelica_EmbeddedSystems.Types.CommunicationType.FromPortCommunication

fromInputToOutput: redeclare Communication.Ideal.IdealReal.IdealCommunicationBlock fromInputToO

toPort: redeclare Communication.Ideal.IdealReal.IdealWriteRealToPort toPort(samplePeri

fromPort: □ Read motor count

Sampling and other co

defineInSubtask: false

defineOutSubtask: true

inSubtask: sk()

outSubtask: Task

<Remove modifier>
□ Dummy Read port
□ Template: ReadRealFromPort
□ Read light sensor.
□ Read motor count
□ Read sound sensor
□ Read touch sensor
□ Read sonar sensor
□ Read Hi Technic acceleration sensor (one axis).
□ Read Hi Technic gyro sensor.
□ Read Mindsensors ACCL-Nx-v3 accel...ensor (acceleration in one axis).

Figure 2. Lego I/O blocks in the CommunicateReal parameter dialog

4.2 Configuration records

Another key component of the library is the configuration record that is used to configure the models with respect to the target platform and task partitioning. A configuration record is a user configurable nested record (record containing records). Depending on the problem, the record could contain one or more targets, tasks and subtask. A simple example is depicted below, Figure 3, where there is just one target, one task and one subtask. The additional block with a Bluetooth icon is from the LEGO_Mindstorms library and is used to select virtual COM ports for Bluetooth communication.

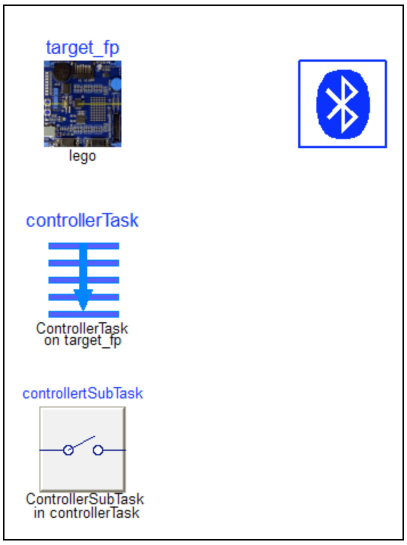

target_fp
lego
controllerTask
ControllerTask on target_fp
controllertSubTask
ControllerSubTask in controllerTask

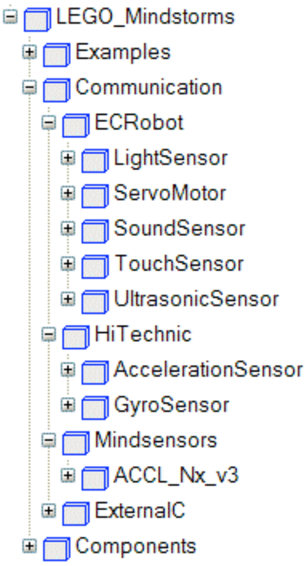
Figure 3. Example of configuration record

5 Dymola Lego Mindstorms API

The Lego Mindstorms NXT device can run under several operating systems. For this project course the nxtOSEK [7] open source real-time operating system was selected due to its openness and well documented C API for sensors, motors and other devices (including some third party sensors). It provides a C programming environment using a GCC tool chain and comes with an extensive set of samples that help the students to get a throughout understanding of the platform and interaction with sensors and actuators. Based on these samples a main program was developed as a wrapper to the Dymola generated model code and variable declarations. The main program handles initialization and termination of sensors and Bluetooth communication (invoking the hook routines described in the nxtOSEK C API Reference [7]) and mapping of system time to fixed-point time while the Dymola generated code that is included performs all the computations.

5.1 LEGO_Mindstorms library

The LEGO_Mindstroms library, Figure 4, has been developed for education and implements communication blocks and a small set of examples and additional components.



- LEGO_Mindstorms
  - Examples
  - Communication
    - ECRobot
      - LightSensor
      - ServoMotor
      - SoundSensor
      - TouchSensor
      - UltrasonicSensor
    - HiTechnic
      - AccelerationSensor
      - GyroSensor
    - Mindsensors
      - ACCL_Nx_v3
    - ExternalC
  - Components

Figure 4. LEGO _Mindstorms library

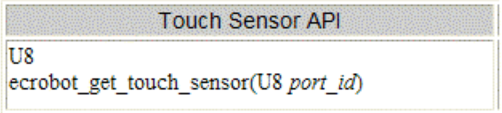
The communication blocks can be used in models to map Modelica variables to low-level C functions on the Lego Mindstorms NXT device. An example would be to map the output of a speed controller to the servo motors and to feed the same controller with data from the ultrasonic sensor for obstacle detection.

The communication blocks provide the mapping to selected function of the API for interaction with sensors and actuators. Also included are some third party sensors from HiTechnic [8] and Mindsensors [9]. The design extends from the Modelica_EmbeddedSystems architecture in such a way that the various blocks can be conveniently selected from a drop down list in the parameter dialog of the `CommunicateReal` block. This enables the students to easily configure the interaction with sensors and actuators in their models.

In addition to the standard sensors of the NXT device the students had access to third party sensors, some included in the C API for nxtOSEK and some not included. Currently the following sensors and actuators are supported:

- ECRobot
  - Light sensor
  - Servo sensor
  - Sound sensor
  - Touch sensor
  - Ultrasonic sensor
- HiTechnic
  - Acceleration sensor (NAC1040)
  - Gyro sensor (NGY1044)
- Mindsensors
  - Acceleration sensor (ACCL-Nx-v3)

The `ECRobot` sub package contains the interface blocks for the standard Lego Mindstorms I/O devises. The blocks contain a mapping to the corresponding nxtOSEK C API functions and utilises the Modelica external function concept. Below is a simple example using the Touch Sensor. As can be seen in Figure 5 the Touch Sensor API takes an U8 (unsigned 8-bit integer) as argument and returns an U8.

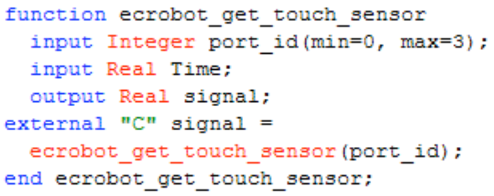


Touch Sensor API

U8
ecrobot_get_touch_sensor(U8 *port_id*)

Figure 5. Touch Sensor API (in nxtOSEK)

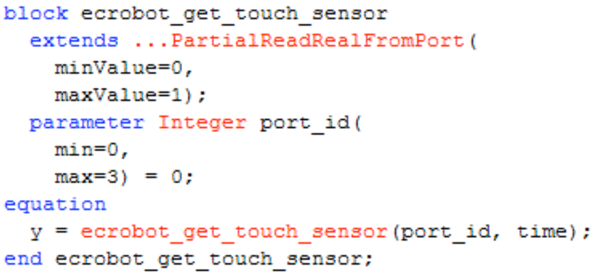
The corresponding function in Modelica that maps to this is depicted below in Figure 6.



```
function ecrobot_get_touch_sensor
  input Integer port_id(min=0, max=3);
  input Real Time;
  output Real signal;
external "C" signal =
  ecrobot_get_touch_sensor(port_id);
end ecrobot_get_touch_sensor;
```

Figure 6. Modelica function mapping

A block that can be used in the `CommunicateReal` block of Modelica_EmbeddedSystems is constructed by extending from the appropriate base class and calling the mapping function, see Figure 7.

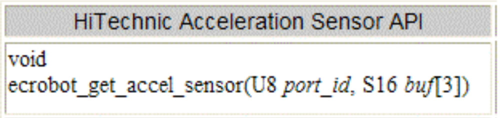


```
block ecrobot_get_touch_sensor
  extends ...PartialReadRealFromPort(
    minValue=0,
    maxValue=1);
  parameter Integer port_id(
    min=0,
    max=3) = 0;
equation
  y = ecrobot_get_touch_sensor(port_id, time);
end ecrobot_get_touch_sensor;
```

Figure 7. Block calling the mapping function (paths have been shortened to fit in the picture)

Note that in this first implementation the return type of the Modelica function is `Real` even though the C function returns an integer (U8). This was done to simplify usage for the students but should be redesigned for a final version of the library. The type conversions are handled by Dymola and the C compiler automatically. All of the sensors and actuators in this sub package are standard Lego sensors and they are all included in the nxtOSEK C API.

The `HiTechnic` sub package contains the interface blocks to two third party sensors from HiTechnic, a gyro sensor and an acceleration sensor. Both sensors are available in the C API which make the Modelica implementation straight forward with one exception. The API for the acceleration sensor, Figure 8 below, differs in that it takes an integer array to store the results in.



HiTechnic Acceleration Sensor API

void
ecrobot_get_accel_sensor(U8 *port_id*, S16 *buf*[3])

Figure 8. Acceleration sensor API (in nxtOSEK)

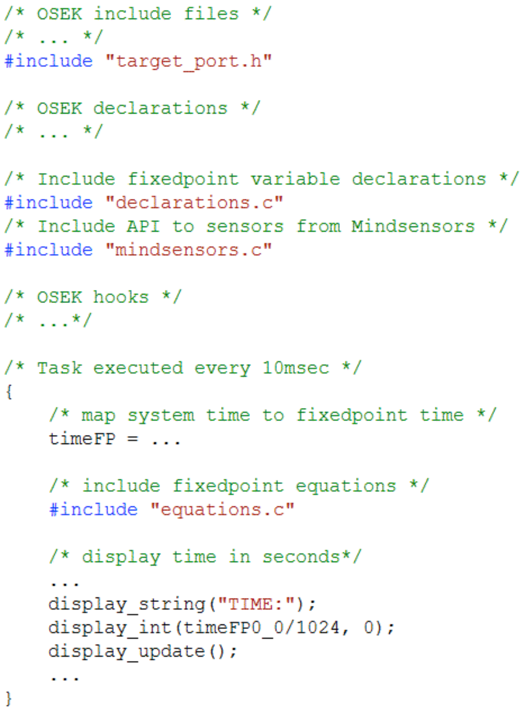
For this sensor a special wrapper has been written in C to extract only one of the elements since the `CommunicateReal` block in Modelica_EmbeddedSystems currently only supports scalars. In Modelica you then choose with a parameter which axis to read from. The drawback is that you need three blocks to read all axes compared to just one function call if using the API as it is.

The `Mindsensors` sub package contains an additional third party sensor: the ACCL-Nx-v3 acceleration sensor from Mindsensors. This sensor can be used either as a tilt sensor or to measure acceleration in any of the x-, y- or z-axis. It is more sensitive than the acceleration sensor from HiTechnic and returns the measured acceleration in units of milli-G, where G is the gravitational unit. This sensor was not represented in the nxtOSEK C API so API functions had to be written manually and supplied to the students.

The `Components.BlueTooth` sub package contains a block that is used to set up Bluetooth communication from the Lego NXT to dymosim (the standard Dymola simulator). It is designed and tested only for Windows and uses virtual COM ports for Bluetooth communication.

5.2 Main program

The main program is based on the sample programs from the nxtOSEK distribution and acts as a wrapper to the model code generated by Dymola. It also handles mapping of system clock to fixed-point time (currently hard coded to 10 fractional bits) and provides some wrappers and API functions for third party sensors. Below in Figure 9 the main program is outlined in pseudo code.



```
/* OSEK include files */
/* ... */
#include "target_port.h"

/* OSEK declarations */
/* ... */

/* Include fixedpoint variable declarations */
#include "declarations.c"
/* Include API to sensors from Mindsensors */
#include "mindsensors.c"

/* OSEK hooks */
/* ...*/

/* Task executed every 10msec */
{
    /* map system time to fixedpoint time */
    timeFP = ...

    /* include fixedpoint equations */
    #include "equations.c"

    /* display time in seconds*/
    ...
    display_string("TIME:");
    display_int(timeFP0_0/1024, 0);
    display_update();
    ...
}
```

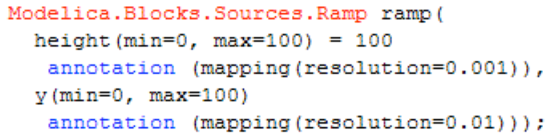
Figure 9. Main program pseudo code

The students could easily modify the program for more advanced use of the display, reconfiguring, adding or removing sensors etc. It is also possible to access all the fixed-point variables for online debugging etc. using their fixed-point representation (integer values used to store the signals). For more convenient debugging the variables can be sent to Dymola using the Bluetooth connection.

6 Dymola and code generation

6.1 Configuring the model for fixed-point

The Lego Mindstorms NXT device does not have hardware support for floating-point arithmetic's and in order to avoid using computationally heavy and memory consuming floating point math libraries, fixed-point code is preferred. In order to use the fixed-point code generation capabilities of Dymola the model must be annotated with additional information. This is done using the *min* and *max* attributes to specify the range of a variable and the relative resolution with newly introduced experimental annotation, *annotation(mapping(resolution=0.001))*. In Figure 10 an example of setting the resolution for two variables of a component is shown. Note that this experimental annotation can be set as a modifier. The information is then used during translation to allocate integer and fractional bits for the fixed-point variables.

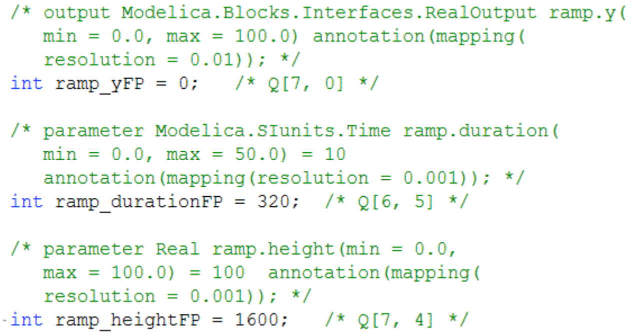


```
Modelica.Blocks.Sources.Ramp ramp(
  height(min=0, max=100) = 100
   annotation (mapping(resolution=0.001)),
  y(min=0, max=100)
   annotation (mapping(resolution=0.01)));
```

Figure 10. Fixed-point annotated component

6.2 Code output

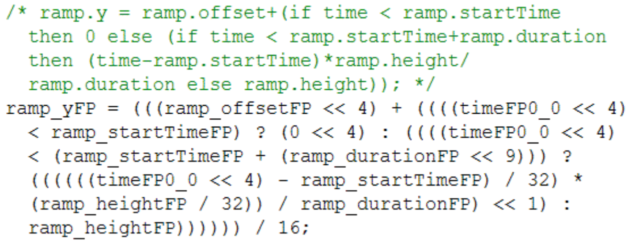
When configured for external code generation, Dymola generates two files, namely declaration.c and equations.c with fixed-point code to be included in the main program. The code is well documented and includes the original Modelica code and the assigned fixed-point format in Q-notation. An example of declatation.c can be seen in Figure 11, note the full Modelica declaration from where the variable originates and the Q-notation indicating the number of integer- and fractional bits.



```
/* output Modelica.Blocks.Interfaces.RealOutput ramp.y(
   min = 0.0, max = 100.0) annotation(mapping(
   resolution = 0.01)); */
int ramp_yFP = 0;   /* Q[7, 0] */

/* parameter Modelica.SIunits.Time ramp.duration(
   min = 0.0, max = 50.0) = 10
   annotation(mapping(resolution = 0.001)); */
int ramp_durationFP = 320;  /* Q[6, 5] */

/* parameter Real ramp.height(min = 0.0,
   max = 100.0) = 100  annotation(mapping(
   resolution = 0.001)); */
int ramp_heightFP = 1600;   /* Q[7, 4] */
```

Figure 11. Declaration of fixed-point variables

All computations are collected in the file equations.c. Just as for the declarations, the eqation file includes the original Modelica eqation as a comment for traceability. Below in Figure 12 is an example of generated fixed-point code for a ramp-function.



```
/* ramp.y = ramp.offset+(if time < ramp.startTime
  then 0 else (if time < ramp.startTime+ramp.duration
  then (time-ramp.startTime)*ramp.height/
  ramp.duration else ramp.height)); */
ramp_yFP = (((ramp_offsetFP << 4) + ((((timeFP0_0 << 4)
  < ramp_startTimeFP) ? (0 << 4) : ((((timeFP0_0 << 4)
  < (ramp_startTimeFP + (ramp_durationFP << 9))) ?
  ((((((timeFP0_0 << 4) - ramp_startTimeFP) / 32) *
  (ramp_heightFP / 32)) / ramp_durationFP) << 1) :
  ramp_heightFP)))))) / 16;
```

Figure 12. Fixed-point code for an equation

6.3 Bluetooth data logging

It can be a very hard task to debug code in embedded systems. To make debugging easier, Dymola generates code (for the Lego Mindstorms NXT target) to send the internal variables of the target in fixed-point representation to Dymola using Bluetooth. The received values are automatically re-scaled to their corresponding Real (SIunit) values. This enables real-time plotting of the internal variables of the target as well as storing the data.

7 Scenarios

In this course, Dymola and Modelica_EmbeddedSystems were used in several of the scenarios the students were faced with. Typical such scenarios are plant modelling and controller design including development and tuning using Model-in-the-Loop (MIL) simulation and Software-in-the-Loop (SIL) simulation. Also for final production code generation and deployment Dymola was used (in combination with other tools; Cygwin, GCC to name the most important).

7.1 Model in the Loop simulation

MIL simulations were performed to test the control strategy with the student's model of the robot. These simulations are typically done with continuous time (ideal) controllers without taking into account effects of sample, communication delays, fixed-point arithmetic's etc. It serves as a foundation, to validate that the control strategy is feasible.

To set up the model for MIL simulation one uses communication blocks from Modelica_Embedded-Systems. These blocks are inserted between different parts of the model, for example controller and plant, to define the border between different tasks.

7.2 Software in the Loop simulation

The next logical step after MIL simulation is SIL simulation where more detail is included in the controller (non-ideal), in this course, the effects of fixed-point arithmetic's in particular.

The model is prepared for SIL simulation by using the Modelica extends mechanism (inheritance) together with a modifier with another configuration record to indicate that the target of the control task does not have a floating-point arithmetic unit. This means that the original model is not changed which is a great benefit in larger projects. The reconfiguring described above is a very simple modification of the model assuming that the model was correctly partitioned for MIL simulation and that the configuration records was set up containing all necessary details. Below, in Figure 13, is an example plot showing the effects of fixed-point arithmetic's on a

PI controller with low resolution driving a simple drive train.

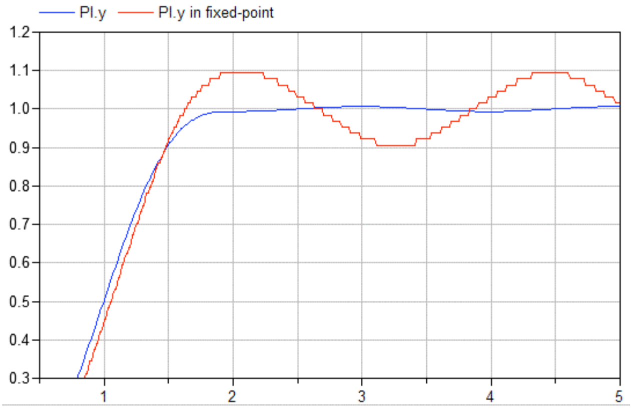


Figure 13. Plot of control signal for a system in closed loop in floating-point vs. low resolution fixed-point

7.3 Production code

Dymola was used for this final stage to generate fixed-point C code for the model equations. This code combined with the main program described in an earlier section can be downloaded to the Lego Mindstorms NXT device and run.

As for the case above, SIL simulation, the reconfiguration is very simple to do. Again the Modelica extends mechanism is used together with a modifier to change the configuration record. This new configuration record specifies the target to be a Lego Mindstroms NXT unit without a floating-point arithmetic support. Dymola could then recognize this target and generate code to fit with the main program. Code is also generated for dymosim which is running in parallel with the Lego controller to collect variable data and convert them for logging, plotting, animation and debugging using Bluetooth, more on this in Section 1. Below in Figure 14 an animation of the Lego robot can be seen.

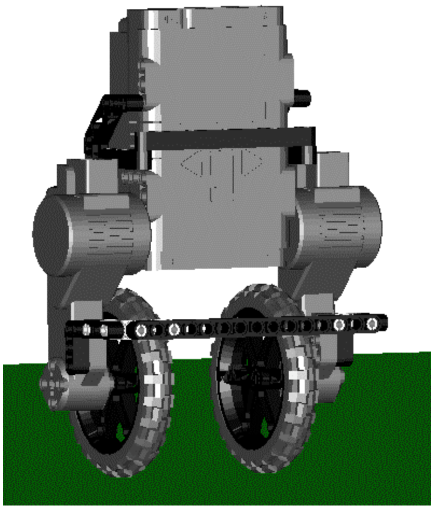


Figure 14. Dymola animation of Lego robot

8 Student results and experiences

Both project groups working with Dymola reached their goal of designing a stabilizing controller based on their multi-body models. The approach was very similar in both cases, and followed largely the steps outlined in Section 3.3. However, the students ran into numerous problems on their way, which needed attention.

While the students quickly constructed mechanical models for their robots, the servos posed a challenge, both in terms of unknown dynamics and in terms of how to connect such a model once available to the mechanical parts. Much time was devoted to solve this problem. The diagram layer for a mechanical model constructed by one of the student groups is shown in Figure 15.

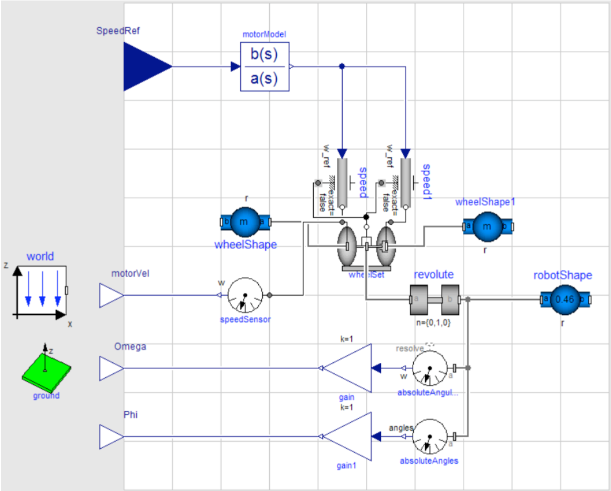


Figure 15. A Modelica model for a two-wheel robot constructed by one of the student groups.

Once a complete model for the robot had been constructed, linearizations were computed to use in the control design. In initial attempts, the linearized models were of high order, in some cases due to high-order servo models derived by means of black-box systems identification. While the high-order models did not impair the possibility to design controllers, problems arose in the controller implementation phase where the availability of measurement signals was limited. In order to solve this problem, simpler models were derived, in particular by simplifying the servo models, and increased attention was given to the available sensors. In the end, the complexity of the controllers was matched to the available measurement signals, but without compromising the model-based approach.

The students experienced some problems with specification of the mapping of controllers onto hardware and the fixed-point code generation in Dymola. Most of the problems were a result of the beta-status of these features at the time of the course. The problems where, however, quickly solved and did not significantly hinder the students in their work.

The reactions from the students were overly positive: *"great to apply knowledge from previous courses in practice"* and *"appreciated the opportunity to work with an industrially relevant tool like Dymola"* were some of the comments. While the students in some cases were a bit disappointed by implementing only stabilization but not remote control the general opinion seems to be that they learnt a lot. Not the least to put their theoretical knowledge into a practical perspective.

9 Summary and conclusions

In this paper, we have reported an application of Modelica in education. Modelica, Dymola, and in particular Modelica_EmbeddedSystems have been used in a master's level course; Project in Automatic Control. The experiences are very encouraging and the tools and methods used in the course of 2009 will be used also in the next year's course.

References

[1] Cervin, A., Henriksson, D., Lincoln, B., Eker, J., Årzén, K-E.: How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime. *IEEE Control Systems Magazine,* **23:3** pp. 16-30, June 2003.

[2] Courses at Automatic Control: http://www.control.lth.se/education/civing.html

[3] Pilato, C., Collins-Sussman, B., Fitzpatrick, B. (2008): *Version Control with Subversion.* O'Reilly Media, Inc.

[4] Trac webpage: http://trac.edgewall.org/

[5] Elmqvist, H., Otter, M., H.,Henriksson, D., Thiele, B.,Mattson, S.E.: Modelica for Embedded Systems, Modelica Conference 2009.

[6] Lego Mindstorms webpage: http://mindstorms.lego.com/

[7] NxtOSEK webpage: http://lejos-osek.sourceforge.net/

[8] HiTechnic webpage: http://www.hitechnic.com/

[9] Mindsensors webpage: http://www.mindsensors.com/

Towards Unified System Modeling and Simulation with ModelicaML: Modeling of Executable Behavior Using Graphical Notations

Wladimir Schamai[1], Peter Fritzson[2], Chris Paredis[3], Adrian Pop[2]
[1]EADS Innovation Works, Hamburg, Germany
[2]PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
[3]Georgia Institute of Technology, Atlanta, USA
wladimir.schamai@eads.net, chris.paredis@me.gatech.edu, {petfr, adrpo}@ida.liu.se

Abstract

This paper is a further step towards application of the Model-Based Systems Engineering (MBSE) paradigm, using standardized, graphical, and executable system modeling languages. It presents further development of Modelica graphical Modeling Language (ModelicaML), a UML Profile for Modelica, which enables an integrated modeling and simulation of system requirements and design (for systems including both hardware and software). This approach combines the power of the OMG UML/SysML standardized graphical notation for system and software modeling, and the modeling and simulation power of Modelica. It facilitates the creation of executable system-specification and analysis models that can simulate time-discrete (or event-based) and time-continuous system behavior.

Keywords: Modelica, ModelicaML, UML, SysML, graphical modeling, system requirements, system design.

1 Introduction

UML/SysML [2],[4] and Modelica [1] are object-oriented modeling languages. Both provide means to represent a system as objects and to describe its internal structure and behavior. SysML is a UML profile for systems modeling. It facilitates efficient capturing of relevant system requirements, design, or test data by means of graphical formalisms, crosscutting constructs and views (diagrams) on the model-data. Modelica is defined as a textual language with standardized graphical annotations for model icons, and is designed for efficient simulation of system dynamic behavior.

1.1 Paper Structure

This paper first presents the motivation and previous work done on the integration of UML/SysML and Modelica, followed by a brief description of UML/SysML, Modelica, and ModelicaML languages. Section 4 summarizes the basic mapping between UML and Modelica, which results in the ModelicaML profile, and provides examples of applications. Section 5 discusses graphical notations for Modelica behavioral concepts. Sections 6 and 7 discuss ModelicaML concepts not present in Modelica. Sections 8, 9 and 10 address the supporting modeling, code generation and simulation environment.

2 Motivation

By integrating Modelica and UML/SysML the UML/SysML's strength in graphical and descriptive modeling is complemented with Modelica's formal executable modeling for analyses and trade studies. Vice versa, Modelica will benefit from using the selected subset of the UML/SysML graphical notation (visual formalisms) for editing, reading and maintaining Modelica models.

Graphical modeling, as promoted by the OMG [13], promises to be more effective and efficient, regarding editing, human reader perception of models, and maintaining models compared to a traditional textual representation. A unified, standardized graphical notation for systems modeling and simulation will facilitate the common understanding of models for parties involved in the development of systems (i.e., system-engineers, designers, and testers; software-developers, customers or stakeholder).

Existing UML/SysML formalisms are typically translated into (and limited to) the time-discrete or event-based simulation of a system or software. This limitation disappears when Modelica comes into play. UML/SysML models will then be of a higher expressiveness and correctness, because they will become executable while covering simulation of hardware and software, with integrated continuous-time and event-based or time-discrete behavior.

DOI: 10.3384/ecp09430081

3 Background and Related Work

Some research work previously done has already identified the need for integrating UML/SysML and Modelica, and has addressed integration issues to some extent. For example, [7] has identified the basic mapping of the structural constructs of Modelica to SysML. It also pointed out that the SysML Parametrics concept is not sufficient for modeling the equation-based behavior of a class. By contrast, [9] leverages the SysML Parametrics concept for the integration of continuous-time behavior into SysML models. [8] presents a concept to use SysML for integrating models of continuous-time dynamic system behavior with SysML information models representing systems engineering problems, and provides rules for graph-based bidirectional transformation of SysML and Modelica models.

The main focus of this paper is the representation of Modelica behavioral constructs using graphical notations and formalisms that are based on a subset of UML, which can be translated into executable Modelica code.

3.1 OMG Systems Modeling Language (SysML)

SysML [4] is a UML profile[1] and a general-purpose systems modeling language that enables systems engineers to create and manage models of engineered systems using graphical notations. SysML reuses a subset of UML 2.1 [2] constructs and extends them by adding new modeling elements and two new diagram types. Through these extensions, SysML is capable of representing the specification, analysis, design, verification, and validation of any engineered system.

MBSE promotes the usage of models as primary engineering artifacts. However, textual requirements are still the main vehicle for communicating and agreeing on system specification in a system development process. SysML provides mechanisms to include textual requirements into models. In doing so, traceability of textual requirements to design artifacts and test cases is facilitated.

The logical behavior of systems is captured in SysML through a combination of activity diagrams, state machine diagrams, and/or interaction diagrams. In addition, SysML includes Parametrics to support the execution of constraint-based behavior such as continuous-time dynamics in terms of energy flow. However, the syntax and semantics of such behavioral descriptions in Parametrics have been left unspecified to interoperate with other simulation and analysis modeling capabilities.

3.2 The Modelica Language

Modelica is an object-oriented equation-based modeling language primarily aimed at physical systems. The model behavior is based on ordinary and differential algebraic equation (OAE and DAE) systems combined with discrete events, so-called hybrid DAEs. Such models are ideally suited for representing physical behavior and the exchange of energy, signals, or other continuous-time or discrete-time interactions between system components.

Modelica models are similar in structure to UML/SysML models in the sense that Modelica models consist of compositions of sub-models connected by ports that represent energy flow (undirected) or signal flow (directed). The models are acausal, equation-based, and declarative. The Modelica language is defined and maintained by the Modelica Association [1] which publishes a formal specification but also provides an extensive Modelica Standard Library that includes a broad foundation of essential models covering domains ranging from (analog and digital) electrical systems, mechanical motion and thermal systems, to block diagrams for control. Finally, it is worth noting that there are several efforts within the Modelica community to develop open-source solvers, such as in the OpenModelica project [12].

3.3 ModelicaML

This paper presents the further development of the Modelica graphical Modeling Language (ModelicaML), a UML profile for Modelica. The main purpose of ModelicaML is to enable an efficient and effective way to create, read or understand, and maintain Modelica models reusing notations that are also used for software modeling. ModelicaML is defined as a graphical notation that facilitates different views (composition, inheritance, behavior) on system models. It is based on a subset of the OMG Unified Modeling Language (UML) and reuses concepts from the OMG Systems Modeling Language (SysML). ModelicaML is designed towards the generation of Modelica code from graphical models. Since the ModelicaML profile is an extension of the UML meta-model it can be used for both: Modeling with standard UML and with SysML[2].

UML/SysML provide the modeler with powerful descriptive constructs at the expense of loosely defined

[1] UML profiles allow domain-specific extensions of UML by means of stereotypes.

[2] SysML itself is also a UML Profile. All stereotypes that extend UML meta-classes are also applicable to the corresponding SysML elements.

semantics that are marked as "semantic variation points" in the UML/SysML specifications. The intention of ModelicaML is to provide the modeler with powerful executable constructs and precise execution semantics that are based on the Modelica language.

Therefore, ModelicaML uses a limited set of the UML, extends the UML meta-model (using the UML profiling mechanism) with new constructs in order to introduce missing Modelica concepts, and reuses concepts from the SysML. However, like UML and SysML, ModelicaML is only a graphical notation. ModelicaML models are eventually translated into Modelica code. Hence, the execution semantics are defined by the Modelica language and ultimately by a Modelica compiler that will translate the generated Modelica code into an executable form.

4 Representing Modelica Structural Constructs in ModelicaML

The class concept is the basic structural unit in Modelica. Classes provide the structure for objects and contain equations, which ultimately serves as the basis for the executable simulation code. The most general kind of class is "model". Specialized categories of classes such as "record", "type", "block", "package", "function" and "connector" have most of the properties of a "model" but with restrictions and sometimes enhancements.

In UML the "Class" is the main structural unit which can have behavior. A non-behavioral concept is the "DataType".

The following table summarizes the mapping of the structural Modelica constructs to UML. The details of the associated properties of the Modelica constructs are left out.

Table 1: Mapping of Modelica structural constructs to UML

Modelica	UML
package	UML::Package
model, block	UML::Class
connector, record, type	UML::DataType
component of type connector	UML::Port
variable, component	UML::Property
extends relation	UML::Generalization
connection clause	UML::Connector

The mapping listed above is specified by [11] and has been implemented as a UML profile in a the Eclipse-based open-source tool Papyrus UML [10]. Modelica constructs are represented using stereotypes (extensions of the UML meta-model) with required properties (attributes) that are specific to Modelica.

It is subject to the current implementation work of the ModelicaML editor to reflect the Modelica language wording, so that the Modelica modeler will not be forced to work with UML/SysML wording. Based on this mapping it is also possible to import existing Modelica models (or libraries) into ModelicaML models, to represent them using graphical notations and to reuse them the same way as is done in Modelica tools.

The following figures present examples of tank systems inspired from [3], sections 12.2.3, 12.2.4 and 12.2.5. The only means to represent Modelica code graphically is the Modelica *connection diagram* (see the two tanks example on the Figure 1). A Connection Diagram shows Modelica class components (typically depicted as domain specific icons with connectors) of the class and their interconnection (connect clauses) as depicted in the figure below. The graphical notation is defined by the Modelica modeler (e.g. the developer of a library) and is not standardized by the language specification; it is usually specific to the domain of application.

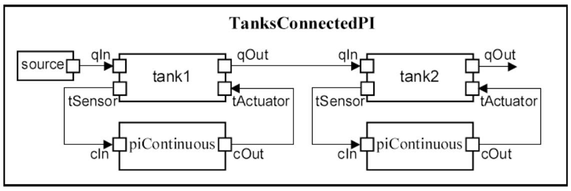


Figure 1. Two Tanks System example, [3] page 391.

The corresponding ModelicaML notation is based on the UML Composite Diagram as illustrated in Figure 2.

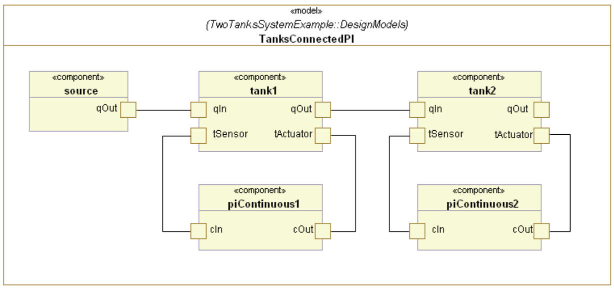


Figure 2. Example of ModelicaML notation (connections)

By contrast, UML defines different types of diagrams, which enable different visual views on the model data, such as inheritance, classes that are nested, the composition of a class or interconnection of components of a class or its defined behavior.

Moreover, the graphical notation is not specific to a domain (although it is possible to include domain specific icons into the class compartment). It is abstracted from the domain. Thanks to such an abstracted, unified notation, engineers from different domains or disciplines will share a common understanding of the model.

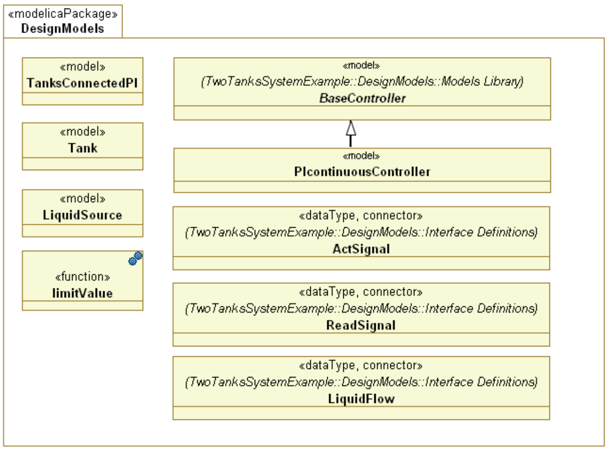


Figure 3. Example of ModelicaML notation (packages, classes)

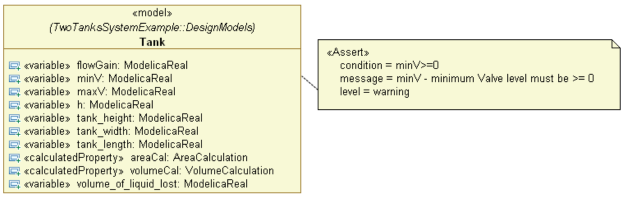

«model»
(TwoTanksSystemExample::DesignModels)
Tank
«variable» flowGain: ModelicaReal
«variable» minV: ModelicaReal
«variable» maxV: ModelicaReal
«variable» h: ModelicaReal
«variable» tank_height: ModelicaReal
«variable» tank_width: ModelicaReal
«variable» tank_length: ModelicaReal
«calculatedProperty» areaCal: AreaCalculation
«calculatedProperty» volumeCal: VolumeCalculation
«variable» volume_of_liquid_lost: ModelicaReal
«Assert»
condition = minV>=0
message = minV - minimum Valve level must be >= 0
level = warning

Figure 4. Example of ModelicaML notation (class, components, asserts)

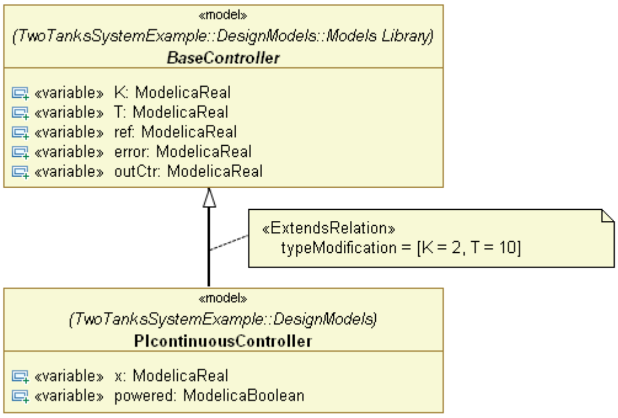

«model»
(TwoTanksSystemExample::DesignModels::Models Library)
BaseController
«variable» K: ModelicaReal
«variable» T: ModelicaReal
«variable» ref: ModelicaReal
«variable» error: ModelicaReal
«variable» outCtr: ModelicaReal
«ExtendsRelation»
typeModification = [K = 2, T = 10]
«model»
(TwoTanksSystemExample::DesignModels)
PIcontinuousController
«variable» x: ModelicaReal
«variable» powered: ModelicaBoolean

Figure 5. Example of ModelicaML notation (inheritance)

In particular the inheritance (extension) graphical representation (Figure 5) is useful if there are multiple levels of inheritance.

5 Representing Modelica Behavioral Constructs in ModelicaML

Modelica does not define any graphical notation for representing the behavior of a class. Instead, the behavior of a Modelica Class is specified by its equations or algorithm statements (including all conditional constructs) which are provided as text.

In addition to basic equations or statements Modelica defines conditional constructs, which are allowed in both equation and algorithm sections, and can have nested constructs or not.

A good match for representing conditional constructs in UML is the Activity Diagrams notation including decision nodes and conditional control flow constructs. The following figures present notations that is used for representing Modelica conditional "if-statement". This notation is used for both "if/when" statements and "if/when" equations. The execution semantics of such Activity Diagrams are the same as for the conditional statements or equations in Modelica. The conditions are evaluated at each time instance. The actions, presented on the diagram are not time-consuming activities; their execution does not take any simulation time.

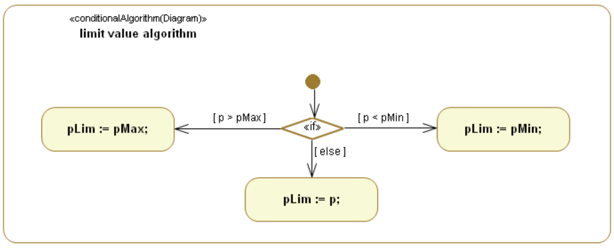


Figure 6. Conditional "if-statement" in ModelicaML

Modelica is a specific language in the context of UML/SysML. For the capturing code of specific languages UML provides opaque constructs which are defined as "A behavior with implementation-specific semantics." (see [2], p.446). In UML, any opaque construct has an attribute "language" (in our case it will be set to "Modelica") indicating how to interpret the code that is entered into the further attribute "body".

Since the UML is an object-oriented modeling language (encapsulating data and behavior), the UML meta-model defines that a classifier can have ownedBehavior (0..*). A behavior in UML can be represented by: State Machine, Activity, Interaction or OpaqueBehavior (see Figure 7).

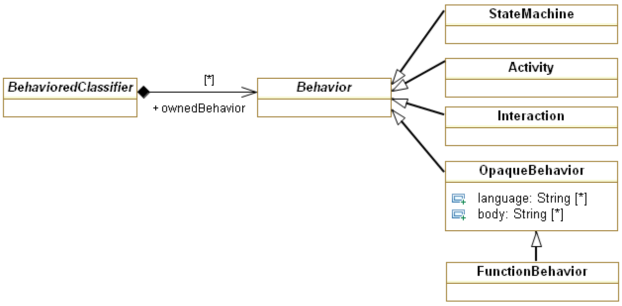
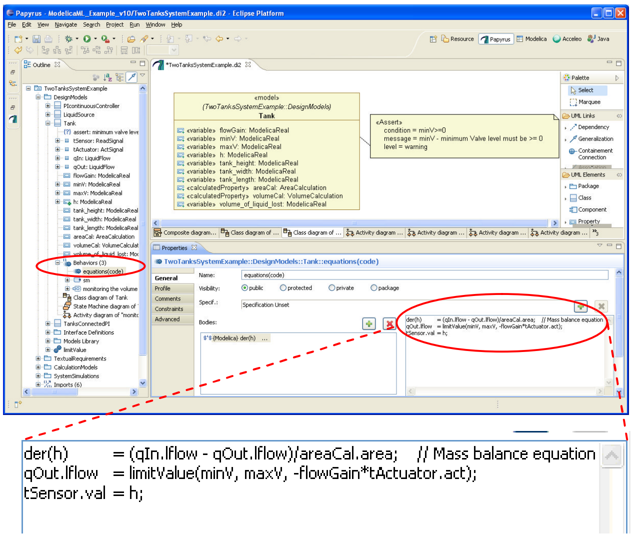


Figure 7. Extract from the UML meta model, from [2] page 426.

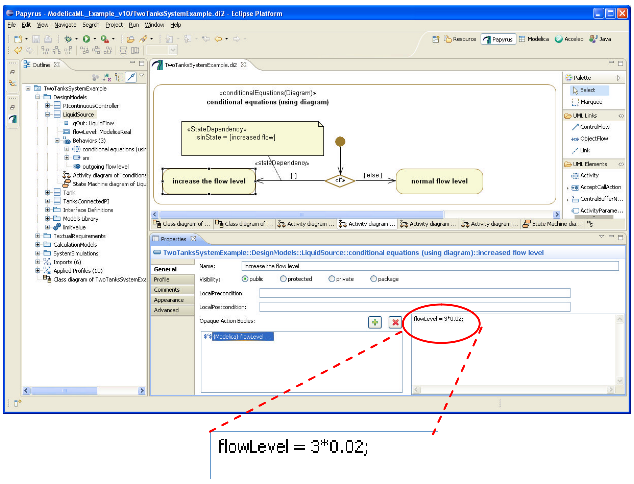
A Modelica model can have (0..*) (from zero to any number) of equation or algorithm sections, which corresponds to the ownedBehavior associations of a Classifier in UML. Conditional equation or algorithm statements can be modeled using a subset of the UML Activity Diagram as illustrated above. Alternatively, the modeler may use the OpaqueBehavior for capturing pure textual Modelica code as illustrated in the following figure.



```
der(h)        = (qIn.lflow - qOut.lflow)/areaCal.area;   // Mass balance equation
qOut.lflow    = limitValue(minV, maxV, -flowGain*tActuator.act);
tSensor.val = h;
```

Figure 8. Modelica textual code in ModelicaML models

If conditional equations or algorithm statements are modeled using UML Activity Diagrams, the actual equations or statements are captured using UML OpaqueAction as depicted in the following figure.



```
flowLevel = 3*0.02;
```

Figure 9. Modelica code in ModelicaML diagrams

[11] summarizes the mapping of the Modelica behavioral constructs to the UML in detail.

6 ModelicaML Concepts Not Provided by Modelica

UML State Machines are typically used for modeling the reactive (event-based) behavior of objects. In ModelicaML the State Machines are used for modeling explicit states or modes of a system or its components. The behavior defined by a State Machine is translated into Modelica algorithm code. Following the principles of a synchronous language the following restrictions are imposed on the semantic of the State Machines as used in ModelicaML:

- The system is at any time in a defined state (note, that the state machines include composite and parallel states, which means that it can be in multiple sub-states at the same time)
- Events and transitions between states take no simulation time. For that reason the effect actions on transitions are not allowed.
- Any behavior that is executed when the state is entered or exited takes no simulation time as well.
- Even though the system will stay in certain states for a time the Do-behavior of a state is also not time-consuming.

Consider the State Machine defined for the tank. Depending on the level of liquid in the tank (represented by the variable "h") we can define that the tank is empty, partially filled or even in an overflow state.

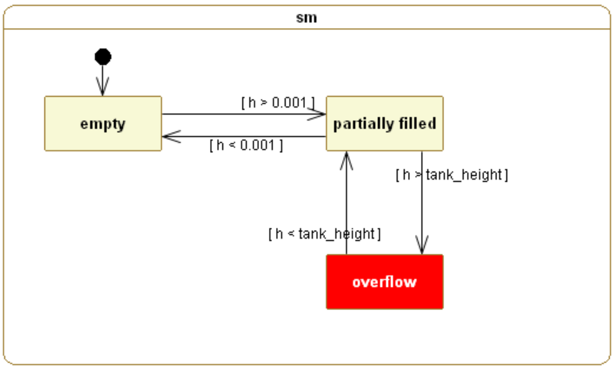


Figure 10. State Machine of the Tank

The next State Machine specifies the behavior of the controller. It shows that only if the controller is in the state “on” it will monitor or control the level of liquid in tank depending on the sensor values received.

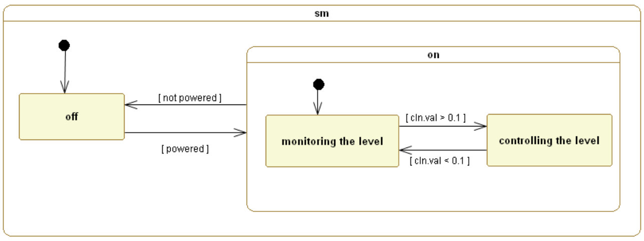


Figure 11. State Machine of the Controller

Any other behavior defined for a system can be defined as being dependent on a specific explicit state of the system. For example, the following shows how conditional equations are modeled including the dependence on the defined states. Depending on if the controller is in the state ”controlling the level” it will activate (the equation is not visible on the diagram, it is: cOut.act = outCtr;) or deactivate (cOut.act = 0;) the actuator signal.

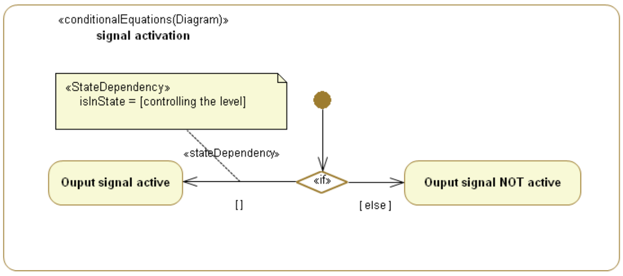
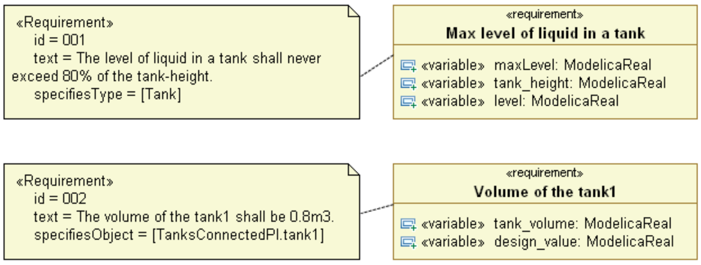


Figure 12. Example of state-dependent equations

The generation of Modelica code from StateCharts was already investigated previously, for example in [5]. Furthermore, [6] introduced the State Graph library for Modelica, which has similar power compared to StateCharts, although it has a slightly different graphical notation. ModelicaML takes a similar approach. In addition to the limitation listed above, the current version of the ModelicaML code generator does not support compound transitions (transition which cross state hierarchy borders), History, Fork/Joins, Entry/ExitPoints and ConnectionPointReference. The limitation and formal definition of the semantics for the State Machines and the Activity Diagrams (including time-consuming activities) are subject to the current ModelicaML research work.

7 Further Concepts (under investigation by ModelicaML)

Inspired by the SysML, ModelicaML reuses the concept of textual requirements within models. As in the SysML it is possible to include textual requirements into ModelicaML models and link requirements to model artifacts. This enables traceability between textual requirements and design artifacts, and supports impact analysis when requirements and/or the model change. Figure 13 illustrates how textual requirements appears graphically on diagrams in ModelicaML.



«Requirement»
id = 001
text = The level of liquid in a tank shall never exceed 80% of the tank-height.
specifiesType = [Tank]

«requirement»
Max level of liquid in a tank
«variable» maxLevel: ModelicaReal
«variable» tank_height: ModelicaReal
«variable» level: ModelicaReal

«Requirement»
id = 002
text = The volume of the tank1 shall be 0.8m3.
specifiesObject = [TanksConnectedPI.tank1]

«requirement»
Volume of the tank1
«variable» tank_volume: ModelicaReal
«variable» design_value: ModelicaReal

Figure 13. Example of textual requirements in ModelicaML

In contrast to SysML, requirement is defined in ModelicaML as a sub-class of the UML Class which can have behavior. It is possible to define properties and behavior (e.g. assertions) for requirements. In doing so it is possible to evaluate if a requirement is violated or not during system simulation. Our current research in this field aims at finding ways to formalize different types of requirements and to find a flexible way to associate requirements with design models. The following examples present some ideas.

Assume the following requirements to be imposed on the two tanks system:

Req. 001: The level of liquid in a tank shall never exceed 80% of the tank-height.

Req. 002: The volume of tank1 shall be 0.8 m^3.

The first requirement specifies a type: Tank in this case. In order to establish the traceability between the textual requirement and the design artifact the class Tank is referenced from the requirement inside the model using the requirement property “specifiesType”. It implies that any instance of a tank must meet this requirement. In contrast, the second requirement is a design requirement defining the required volume of tank 1. This requirement is imposed only on a particular instance of the type Tank. Therefore, the dot-notation in the requirement property “specifiesObject” is used to reference the respective instance. The “specifies…” - relations are descriptive only. They are not translated into Modelica code and do not impact the simulation.

In order to be able to evaluate these requirements during the system simulation requirements need to be formalized. In the following one possible way to do so is presented.

From the textual statement of the requirement 001 we can identify measurable properties such as: level (current level in a tank), maxLevel (80 % max. allowed level), tank_height (the height of a tank). Moreover, we can define a property indicating if the requirement is violated or not by evaluating: `level > maxLevel * tank_height`. Consider the following state machine specifying if the requirement 001 is violated or not. The second requirement is modeled in a similar way; it is not presented here.

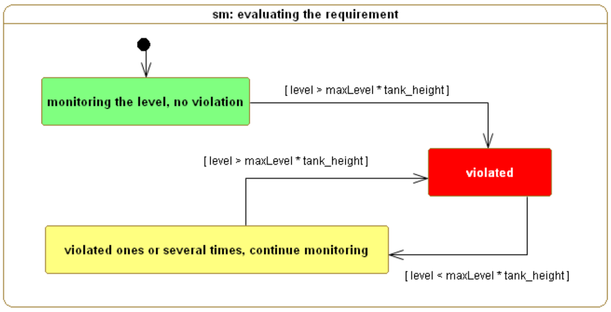
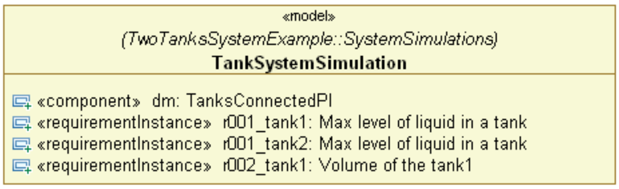

sm: evaluating the requirement
monitoring the level, no violation
[level > maxLevel * tank_height]
violated
[level > maxLevel * tank_height]
violated ones or several times, continue monitoring
[level < maxLevel * tank_height]

Figure 14: Example of requirements behavior

The modeled requirements can now be instantiated and their properties can be bound to the values within the corresponding design model (TanksConnectedPI in that case). In this example, the declarations for the r001_tank2 (Figure 14) are:

- `level = dm.tank1.h`
- `tank_height = dm.tank1.tank_height`


«model»
(TwoTanksSystemExample::SystemSimulations)
TankSystemSimulation

«component» dm: TanksConnectedPI
«requirementInstance» r001_tank1: Max level of liquid in a tank
«requirementInstance» r001_tank2: Max level of liquid in a tank
«requirementInstance» r002_tank1: Volume of the tank1

Figure 15: Instantiated design model and associated requirements

Note that requirement 001, which specifies the type Tank, is instantiated two times (because there are two tanks in the system).

Figure 16 shows the results of the evaluation (the tank_height is 0.6m in this example). The requirement 001 evaluated for the tank2 (r001_tank2) was violated two times during the simulation.

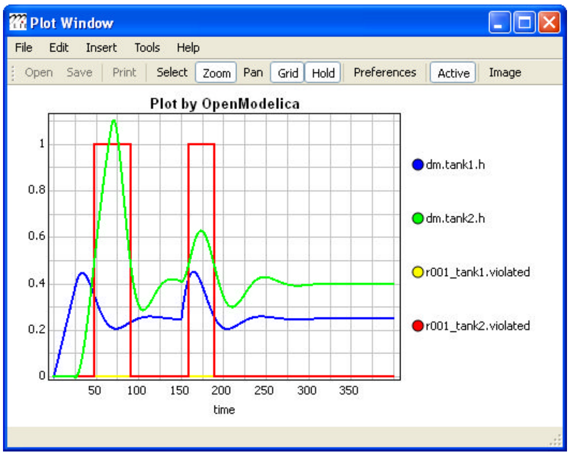


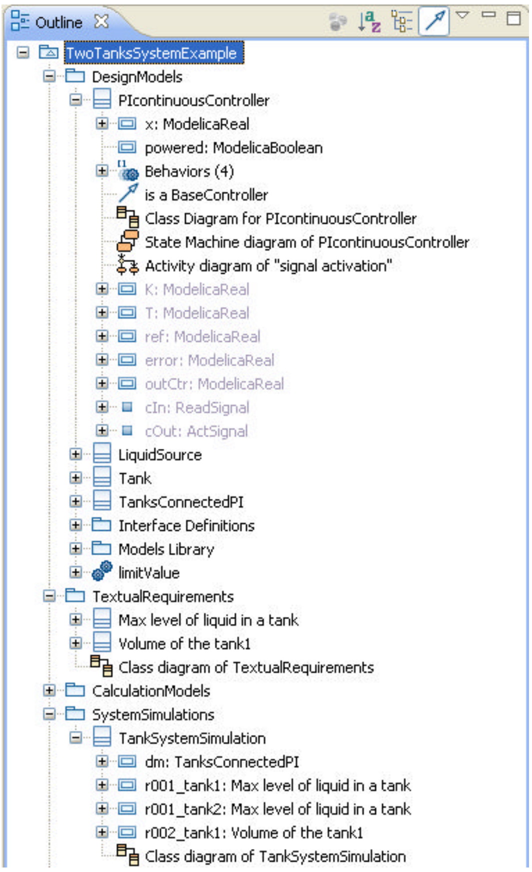
Figure 16: Example of requirements evaluation during system simulation

Similar to the concept of textual requirements, the modeller can define measures of effectiveness of models, which are used to record dedicated, measurable properties of system models during simulations and can compare them according to predefined metrics, for example, in order to select the best potential design alternative.

Our future ModelicaML research aims at developing a flexible association of requirements to multiple design alternatives in a way that requirement models can be instantiated automatically together with the associated design models in order to be evaluated during system simulation.

8 Modeling Support

Usually, when using a UML modeling tool, the model elements can be created either directly in the model browser (a tree-like representation of the classes, etc.) or using diagrams. In both cases the model data is stored in the model repository (see Figure 17).



- TwoTanksSystemExample
  - DesignModels
    - PIcontinuousController
      - x: ModelicaReal
      - powered: ModelicaBoolean
      - Behaviors (4)
      - is a BaseController
      - Class Diagram for PIcontinuousController
      - State Machine diagram of PIcontinuousController
      - Activity diagram of "signal activation"
      - K: ModelicaReal
      - T: ModelicaReal
      - ref: ModelicaReal
      - error: ModelicaReal
      - outCtr: ModelicaReal
      - cIn: ReadSignal
      - cOut: ActSignal
    - LiquidSource
    - Tank
    - TanksConnectedPI
    - Interface Definitions
    - Models Library
    - limitValue
  - TextualRequirements
    - Max level of liquid in a tank
    - Volume of the tank1
    - Class diagram of TextualRequirements
  - CalculationModels
  - SystemSimulations
    - TankSystemSimulation
      - dm: TanksConnectedPI
      - r001_tank1: Max level of liquid in a tank
      - r001_tank2: Max level of liquid in a tank
      - r002_tank1: Volume of the tank1
      - Class diagram of TankSystemSimulation

Figure 17: Example of a ModelicaML model browser

Diagrams only provide a view on a selected part of the model data. Diagrams can be used only for modeling (i.e., capturing the data), and might be deleted[3] after the data is captured. In some cases the modeler may decide to leave some diagrams for documentation or communication purposes. In this case, the modeler will need to select the data that should appear on dedicated diagrams (depending on which data can be displayed on a specific diagram type). An appropriate partitioning of the model data into different diagram and diagram types is essential in order to improve readability and to support the modeler by automatic generation and layout of diagrams. For example, the diagrams in figures Figure 3, Figure 4, Figure 5, Figure 13 or Figure 15 would not need to be modeled (and arranged visually). These can be generated from the model data.

This will prove rather difficult for the diagrams in Figure 2, Figure 10, Figure 11 or Figure 14. Those diagrams will need to be modeled (arranged visually) by the modeler. This is a good indicator to see if value is added by spending time on a diagram.

9 Model Validation and Code Generation

The ModelicaML code generator that generates Modelica code from the ModelicaML models is implemented using the Acceleo Eclipse Plug-In [16], which follows the MDA approach and the model-to-text recommendations of the OMG.

Presently, ModelicaML is implemented as a UML Profile that can be used in any (Eclipse-based) UML2 tool. This way the modeler needs to first create a UML element and then apply a stereotype, defined in the ModelicaML profile, in order to represent a specific concept or to introduce (or to specify) the semantics. The a*dvantage* of this approach is: it allows creating or reading ModelicaML models using any UML2 tool. *The disadvantage is*: the modeling tool GUI does not directly reflect the Modelica wording. The modeler needs to have a basic knowledge of the UML in order to know which stereotypes of the ModelicaML profile should be applied to which UML elements. Moreover, all limitations, constraints and possible inconsistencies will have to be checked and resolved before the Modelica code generation. Therefore, the ModelicaML code generator includes a validator that checks the model and informs the modeler about inconsistencies before the Modelica code is generated.

10 Simulation Support (Using OpenModelica Environment)

In addition to the convenient way of simulating a Modelica model from startTime to stopTime, in the frame of the ModelicaML research and implementation the OpenModelica Environment [12] was enhanced by interaction simulation capabilities (similar to the Interaction Library in Dymola [15] Modelica tool). It is possible to generate Modelica code directly from the ModelicaML models and to pass it to the OMC. A dedicated simulation GUI has been implemented providing the user with possibilities to interact with the Modelica model (i.e., to change parameters at runtime) and to observe the reaction of the system immediately on plots. Moreover, any additional GUI (with domain specific animations or widgets) can be implemented and connected to the simulation using the implemented OMC interactive simulation interface. This feature will support model debugging as well as the communicating and discussing of the modeled system behavior to and with any parties involved in the system development process.

[3] Of course, any diagram can be recreated from the model data.

11 Conclusion

This paper presents a step towards, and a proof of concept for, a unified executable system modeling language and environment using open-source UML modeling (Papyrus UML) and simulation (OpenModelica) tools.

One of our main future research activities in the field of ModelicaML will be dedicated to developing graphical notations for modeling any kind of equations or statements, as well as other constructs (e.g. type- or instance-modification) that are now captured using strings. This will avoid the refactoring of models and enable semantic analysis of the ModelicaML models.

In conclusion, UML/SysML and Modelica are complementary languages supported by two active communities. By integrating UML/SysML and Modelica into ModelicaML, we combine the very expressive, formal language for differential algebraic equations and discrete events of Modelica with the expressive UML/SysML graphical notation for requirements, structural decomposition, logical behavior, and corresponding cross-cutting constructs.

In addition, the two communities are expected to benefit from the exchange of multi-domain model libraries and the potential for improved and expanded commercial and open-source tool support.

12 Acknowledgements

This work has been supported by EADS Innovation Works, by Swedish Vinnova in the ITEA2 OPENPROD project and by the Swedish Research Council (VR).

References

[1] Modelica Association. Modelica: A Unified Object-Oriented Language for Physical Systems Modeling: Language Specification Version 3.0, Sept 2007. www.modelica.org

[2] OMG. OMG Unified Modeling Language ™ (OMG UML). Superstructure Version 2.2, February 2009.

[3] Fritzson P. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEEE Press, 2004.

[4] OMG. OMG Systems Modeling Language (OMG SysML™), Version 1.1, November 2008.

[5] Ferreira J. A. and Estima de Oliveira J. P., Department of Mechanical Engineering, University of Aveiro, 3810 Aveiro (PORTUGAL), Department of Electronic Engineering, University of Aveiro, 3810 Aveiro (PORTUGAL), MODELLING HYBRID SYSTEMS USING STATECHARTS AND MODELICA, J. A.

[6] M. Otter, K.-E. Arz´en, I. Dressler. StateGraph-A Modelica Library for Hierarchical State Machines. DLR Oberpfaenhofen, Germany; Lund Institute of Technology, Sweden. Proceedings of the 4th International Modelica Conference, Hamburg. March 7-8, 200.

[7] Pop, A., and Akhvlediani, D., and Fritzson, P. Towards Unified Systems Modeling with the ModelicaML UML Profile. *International Workshop on Equation-Based Object-Oriented Languages and Tools. Berlin, Germany, Linköping University Electronic Press, 2007*

[8] Peak, R., McGinnis, L., Paredis, C. Integrating System Design with Simulation and Analysis Using SysML – Phase 1 Final Report. 2008

[9] Johnson, T. A. Integrating Models and Simulations of Continuous Dynamic System Behavior into SysML. M.S. Thesis, G.W. Wood-ruff School of Mechanical Engineering, Georgia Institute of Technology. Atlanta, GA. 2008

[10] Papyrus UML, www.papyrusuml.org

[11] Schamai W.. Modelica Modeling Language (ModelicaML) A UML Profile for Modelica, technical report 2009:5, EADS IW, Germany, Linkoping University, Sweden, 2009

[12] The OpenModelica Project www.ida.liu.se/labs/pelab/modelica/OpenModelica.html

[13] Object Management Group (OMG). www.omg.org

[14] Modelica Association. www.modelica.org

[15] Dymola (Dynamic Modeling Laboratory), Dynamism. www.dymola.com

[16] Acceleo, Eclipse Plug-In. www.acceleo.org/pages/home/en

Appendix: Modelica Example Code

```
connector ActSignal "Signal to actuator for setting
valve position"
  Real act;
end ActSignal;

connector ReadSignal "Reading fluid level"
  Real val(unit = "m");
end ReadSignal;

connector LiquidFlow  "Liquid flow at inlets or
outlets"
  Real lflow(unit = "m3/s");
end LiquidFlow;

partial model BaseController
  parameter Real K = 2 "Gain";
  parameter Real T(unit = "s") = 10 "Time constant";
  ReadSignal cIn "Input sensor level, connector";
  ActSignal  cOut "Control to actuator, connector";
  parameter Real ref "Reference level";
  Real error "Deviation from reference
level";
  Real outCtr "Output control signal";
equation
  error = ref - cIn.val;
  cOut.act = outCtr;
end BaseController;

function limitValue
  input  Real pMin;
  input  Real pMax;
  input  Real p;
  output Real pLim;
 algorithm
  pLim := if p>pMax then pMax
          else if p<pMin then pMin
          else p;
end limitValue;

model LiquidSource
  LiquidFlow qOut;
  parameter Real flowLevel = 0.02;
equation
  qOut.lflow = if time > 150 then 3*flowLevel else
flowLevel;
end LiquidSource;
```

```
model PIcontinuousController
  extends BaseController(K = 2, T = 10);
  Real  x  "State variable of continuous PI
controller";
equation
  der(x) = error/T;
  outCtr = K*(error + x);
end PIcontinuousController;

model Tank
  ReadSignal tSensor "Connector, sensor reading tank
level (m)";
  ActSignal tActuator "Connector, actuator controlling
input flow";
  LiquidFlow qIn "Connector, flow (m3/s) through input
valve";
  LiquidFlow  qOut "Connector, flow (m3/s) through
output valve";
  parameter Real area(unit = "m2") =  0.5;
  parameter Real flowGain(unit = "m2/s") = 0.05;
  parameter Real minV= 0, maxV = 10; // Limits for
output valve flow
  Real h(start = 0.0, unit = "m") "Tank level";
 equation
  assert(minV>=0,"minV - minimum Valve level must be
>= 0 ");
  der(h) = (qIn.lflow - qOut.lflow)/area; // Mass
balance equation
  qOut.lflow  = limitValue(minV, maxV, -
    flowGain*tActuator.act);
  tSensor.val = h;
end Tank;

model TanksConnectedPI
  LiquidSource  source(flowLevel = 0.02);
  Tank          tank1(area = 1);
  Tank          tank2(area = 1.3);
  PIcontinuousController piContinuous1(ref = 0.25);
  PIcontinuousController piContinuous2(ref = 0.4);
 equation
  connect(source.qOut,tank1.qIn);
  connect(tank1.tActuator,piContinuous1.cOut);
  connect(tank1.tSensor,piContinuous1.cIn);
  connect(tank1.qOut,tank2.qIn);
  connect(tank2.tActuator,piContinuous2.cOut);
  connect(tank2.tSensor,piContinuous2.cIn);
end TanksConnectedPI;
```

Object-oriented simulation of preemptive feedback process schedulers

Martina Maggio*, Alberto Leva
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy
{maggio,leva}@elet.polimi.it
*PhD student at the Dipartimento di Elettronica e Informazione

Abstract

Based on recent research, very simple discrete-time control structures can be used to synthesise preemptive process schedulers for multitasking systems within a rigorous system-theoretical formalism. Doing so virtually eliminates any heuristics, and allows for a methodologically grounded analysis and assessment of the achieved performances. This paper introduces a Modelica library for the above purpose, at present still under development, and illustrates its use with some tests.

Keywords: Feedback scheduling; Multitasking systems; Preemptive systems.

1 Introduction

Many problems related to computing systems are being recognised and tackled as *control* problems [4]. A notable example is that of process scheduling in multitasking (not necessarily real-time) computing systems. The role of the process scheduler in such systems is to allocate the CPU usage to the running processes, so as to guarantee properties like fairness, responsiveness, and so forth [6]. Feedback-based techniques have been applied to the scheduling problem [2, 3, 8] to deal with uncertainty and disturbances, such as the behaviour of the processes, and the availability of the resources they may require.

In virtually the totality of the feedback scheduling literature, however, the idea is concisely to "close some loop around an *existing* scheduler". Since the object to be controlled (the "plant" to stick to the standard terminology) includes said scheduler, in the above context the term "actuators" takes the specific meaning of "having the feedback controller assign the values of some scheduling parameters" like queue lengths, priority variations, and so on, while the term "sensors" refers to measurements of the required properties, such as the processes' CPU utilisation [1].

Moreover, existing schedulers are conceived by their designers in terms of algorithms and data structures (i.e., the way *computer* scientists think of the word "model") and not of equations (i.e., the way the same word is thought of by *control* scientists). Modeling those schedulers is thus generally complex, but above all it is highly unnatural with formalisms that allow for powerful and simple analysis and synthesis tools. Some attempts were made to devise a model for such a scheduler (as well as other hardware and software components) in Modelica [9]; that work however does not include feedback policies.

This work is part of a wider research that takes completely different an attitude. Instead of acting on the scheduler already present in the considered system, the idea here is to replace that scheduler completely. And correspondingly, instead of writing a model to reflect the scheduling algorithm, the *modus operandi* is to have that algorithm emerge from the digital realisation of a controller model—a perspective shift indeed.

In other words, in this research schedulers are designed exactly in the same way as a feedback controller is synthesised, so as to allow expressing the specifications by means of the usual concepts of set point tracking, load disturbance rejection, and so on. Thanks to the adopted formalism—that of discrete-time *linear* dynamic systems, as will be briefly justified later on—the above concepts can be given a *quantitative* meaning, which is a novelty of this research with respect to other approaches to the same problem, where qualitativeness and heuristics play a central (albeit often tedious for the designer) role.

The aim of this manuscript, within the mentioned research, is to present a small Modelica library (at present in its first version and under continuous development) aimed at helping the designer of a scheduling

DOI: 10.3384/ecp09430033

policy assess the behaviour of said policy by means of simulations representing the *on line* system behaviour under suitably chosen load conditions. This is another peculiarity of this work, since the great majority of the available literature concentrates on *off line* schedulability analysis issues [5].

2 The modelling formalism

To set up a process scheduler in the same way as a feedback regulator is designed it is first necessary to select a control-theoretical formalism that admits a clear separation between the "plant" and the "controller". The necessity of such a separation *de facto* rules out the use of discrete-event models. In fact, in the discrete-event control context, defining an open-loop process model almost inherently calls for specifying the desired behaviour in terms of constraints only—think for example of the supervisory control framework—while if said behaviour is more naturally expressed (also) as a desired sequence of events, then only top/down approaches (where the model of the plant and the controller live jointly right from the beginning, however) allow to guarantee some formal property for the closed-loop system.

In addition, in the formalism to be chosen here, the modelled objects have to admit a direct, *non ambiguous* realisation as algorithms. This is a further argument to avoid discrete-event frameworks such as queue networks and Petri nets for our purposes (a promising work on the matter however is [7]), because their inherently asynchronous nature requires to specify "something else" in order to turn a model into an algorithm—think, for example, of evolution rules or similar ideas.

The next step is to define a correct partition between the plant and the controller. The matter is addressed, in this work, in the context of a single-processor system, and of negligible context switch durations; both hypotheses can be relaxed at an acceptable cost, but keeping them in for now eases the treatise. In this context, a physical separation between controller and plant is extremely cumbersome to figure out, but a time-based partition between them is on the contrary very natural. In fact, the (one) CPU is either executing some of the scheduled processes, or the scheduling algorithm. The complete system can thus be viewed as a discrete-time (not sampled-signals, however) one, with a time index being related to the scheduler interventions.

It is now necessary to state what is to be meant for (the model of) "the plant in open loop". In the classical applications of the control theory, think for example of the process or motion control domains, doing so is trivial (at least conceptually). The plant (model) is a system of differential and/or algebraic equations, stemming essentially from the underlying physics, where the inputs (the controller actions) are thought of as exogenous signals. Here, the time-based model partition comes into play: the model of the plant in open loop is a discrete-time system that receives as inputs the results of the "controller" algorithm execution, and returns the results of the "plant" algorithm execution.

The last step, and another peculiarity of the chosen modelling formalism, is that *anything else* but the scheduler action is treated here as an *exogenous* disturbance. This may appear to be a limitation, since for example a resource request is not exogenous at all for the computing system composed of the running processes (meaning that it can be somehow predicted, for example). However, although not exogenous for the running processes (the plant), such a fact is exogenous *for the scheduler* (the controller). Adopting such an attitude, we can take profit of the extremely powerful idea of "disturbance" as thought of in the control theory, i.e., as one of the fundamental reasons for the necessity of feedback.

Consider a single-processor multitasking system with a preemptive scheduler; let N be the number of processes to schedule, that we assume for now constant (some words on the matter will be spent later on). Let the column vectors $\tau_p(k) \in \Re^N$, $\rho_p(k) \in \Re^N$, $b(k) \in \Re^{n(k)}$ and $\delta b(k) \in \Re^{n(k)}$, $1 \leq n(k) \leq N \forall k$ represent, respectively,

- the CPU times allocated to the processes in the beginning k-th scheduling round, thus defining (as anticipated) the meaning of k,
- the times to completion at the beginning of the k-th scheduling round for the processes that have a duration assigned (elements corresponding to processes without an assigned duration will be $+\infty$, therefore allowing for the presence of both batch processes and interactive ones),
- the bursts assigned by the scheduler to the processes at the k-th scheduling round,
- the disturbances possibly acting on the scheduling action during the k-th scheduling round,

where $n(k)$ is the number of processes that the scheduler considers at each round (traditionally constant and equal to one—an example of aprioristic constraint that in principle can be relaxed). Denoting by t the total time actually elapsed from the system initialisation, the simplest plant model one can conceive is then

$$\left\{\begin{array}{rcl} \tau_p(k) & = & S_\sigma b(k-1)+\delta b(k-1) \\ t(k) & = & t(k-1)+r_1\tau_p(k-1) \\ \rho_p(k) & = & \max\left(\rho_p(k-1)-S_\sigma b(k-1)-\delta b(k-1),0\right) \end{array}\right. \quad (1)$$

where r_1 is a row vector of length N with unit elements, and $S_\sigma \in \Sigma$ a $N \times n(k)$ switching matrix with elements equal to 0 or 1 and only one 1 per column, assigning the elements of $b(k)$ to the correct processes. Notice that, being $n(k)$ bounded, set Σ is finite for any given N.

As for variations of the process pool, a newly arriving one simply requires to increase N by one, add one zero at the end of τ_p and one duration (or one $+\infty$) at the end of ρ_p, and add one row to S_σ. Incidentally, the row position orders the processes by arrival time, which is of interest for some existing and already used scheduling policies. Similarly, altering the "grouping" of the process pool in sub-pools, for example in view of a multilevel scheduling, means acting on $n(k)$.

In synthesis, under the sole limitation (to be possibly relaxed in the future) that n be constant, adding (and obviously removing) processes from the pool simply means formulating a new model in the form (1), that is initialised from the last state of the previous one in a straightforward way. Since process arrivals or terminations are events that occur on a time scale much longer than that of the scheduling task, we concentrate in this manuscript on the constant pool case.

Notice that the first two equations in (1) form a linear, switching discrete-time dynamic system, apart from the obvious input saturation constraint given by the impossibility of negative bursts. The third one is conversely nonlinear, but given the role of disturbances in the adopted framework, a process reaching termination before exhausting its burst is simply modelled as a negative disturbance element on that burst—then the process is of course removed from the pool, see above. Recalling that the relevant fact is here that disturbances are exogenous to the scheduler only, one can therefore safely treat the model as the linear switching one

$$\left\{\begin{array}{rcl} \tau_p(k) & = & S_\sigma b(k-1)+\delta b(k-1) \\ t(k) & = & t(k-1)+r_1\tau_p(k-1) \\ \rho_p(k) & = & \rho_p(k-1)-S_\sigma b(k-1)-\delta b(k-1) \end{array}\right. \quad (2)$$

apart from the mentioned saturation issue.

3 Schedulers as controllers

3.1 Classical scheduling policies

If conditions are imposed to n and/or S_σ, some very common existing scheduling policies are represented by the chosen formalism entirely. For example

- $n = 1$ and a N-periodic S_σ produce all the possible Round Robin (RR) policies having the (scalar) $b(k)$ as the only control input, and obviously the pure round robin if $b(k)$ is kept constant,
- $n = 1$ and a S_σ chosen so as to assign the CPU to the process with the minimum row index and a ρ_p greater than zero produces the First Come First Served (FCFS) policy,
- $n = 1$ and a S_σ giving control to the process with the minimum ρ_p yields the Shortest Remaining Time First (SRTF) policy,
- $n = 1$ and a S_σ that switches according to the increasing order of the initial ρ_p vector produces the Shortest Job First (SJF) policy (notice that this is the same as SRTF if no change to the process pool occurs).

It is important to observe that in the list above the classical "actuators" of the previous works for the mentioned policies are evidenced, but here as entities in a neat system-theoretical framework. Also the state variables of the scheduler, that in the previous works is part of the plant, are clearly defined (examples are that required to switch S_σ in a periodic manner, or to store the initial ρ_p). The chosen formalism hence allows to include also rules that appear very far from it, e.g. owing to the presence of queues. A notable example is the so called selfish round robin (SRR), that is represented by $n = 1$ and S_σ depending on a controller state variable representing the time spent by the corresponding process waiting for the CPU.

However, it is worth further stressing that in evidencing the actuators, constraints had to be imposed on the "pure" plant model, therefore *including in that model something that is actually control, not plant.* Moreover, the resulting "plant including the scheduler" (model) is not only switching but apparently nonlinear, which is *not* true for (2) (we mention the input saturation issue here for the last time, as there are plenty of methods to deal with it while reasoning for the control synthesis in a linear context).

Removing the above mentioned constraints yields therefore two benefits. First, it is a generalisation of current scheduling policies, in a sense that is now characterised. Second, it does not put into the plant model any element that is *de facto* control, and as a result leads to a plant model that can be treated as linear—and in any case, regarding also possible futire extensions, is in nature much simpler than any other one aiming at represent also partd of the control.

3.2 New policies within the same framework

So far, it was shown that the simple modelling framework adopted here can represent classical, well known scheduling policies as variations (better, *restrictions* or specialisations) of a *single* discrete-time dynamic system. The question is then immediately what can be done if *different* specialisations are imposed to the general model. In the opinion of the authors, such an exploration opens a way toward the design of schedulers as dynamic systems, the mentioned "specialisations" qualifying (sub)classes of schedulers in a system-theoretical (not algorithmic) taxonomy.

To illustrate the idea at the present state of the research, two control strategies will be explored in the following, that use a specialisation of the model (2) different from the ones proposed above to replicate existing policies, and by the way introduce control structures that are widely used and very well assessed in other contexts than scheduling.

Let $\tau_r(k) \in \Re$ represent the *actual* duration of the k-th round and let the model not take into account the CPU time required by each process, dealing with process termination and insertion with re-initialisation as will be explained later on. Furthermore, let the S_σ switching signal be an ordered sequence that starts from the first process and proceeds to last one, and consistently be $n = N$. To deal with the possibility of non activating a process (e.g. because the scheduler knows that it has not all the needed resources to execute) the given burst can be zero. In other words, let the "specialisation" affect only the switching signal, so that the result be an LTI discrete-time system that can be controlled by an LTI control structure designed with well assessed methods—maybe the most interesting class of schedulers to study, for sure the simplest from the control-theoretical standpoint. Doing so the model becomes

$$\left\{ \begin{array}{rcl} \tau_p(k) & = & b(k-1) + \delta b(k-1) \\ \tau_r(k) & = & r_1 \tau_p(k-1) \\ t(k) & = & t(k-1) + r_1 \tau_p(k-1) \end{array} \right. \qquad (3)$$

and some remarks are in order.

- Model (3) is linear and time-invariant. Of course negative bursts are not allowed, but that is simply a matter of input saturation, and can be tackled with a number of techniques. Crudely speaking, the control literature has been managing controller design in that way with linear models for decades, so it can be assumed that there is no need here for anything more complex.
- Similarly, each $b + \delta b$ element cannot be negative. This rigorously means that the disturbance is bounded in a time-varying manner, which is however irrelevant for the controller design as far as the disturbance is taken as totally exogenous (i.e., assuming that $b + \delta b$ *could in principle* be negative one considers a set of disturbances wider than the real one, to the apparent validity confirmation of the devised solutions).
- The scheduler is acting not at each process activation, but only once per round. Clearly some b elements can be zero, meaning that not all the process will actually run. Since the proposed scheme allows to control the round duration, system responsiveness issues are implicitly addressed.

The proposed control scheme has the nested loop structure of figure 1. Let τ_r° be the required scheduling round duration, and let

$$\theta_p^\circ \in \Re^N, \qquad \theta_{p,i}^\circ \geq 0, \quad \sum_{i=1}^{N} \theta_{p,i}^\circ = 1 \qquad (4)$$

be the vector containing the required CPU time fractions to be allotted to each process.

First, consider the closed-loop system (CL_1 in figure 1) having as set point the desired CPU times consumed by each process in the current round, i.e., $\tau_p^\circ(k)$, and as controlled variable the CPU times actually consumed in the same round, i.e., $\tau_p(k)$; the control variable is the burst vector $b(k)$, while $\delta b(k)$ is a (vector) load disturbance.

By choosing R_p as a diagonal integral regulator with gain k_{pi}, i.e., $A_{R_p} = C_{R_p} = I_N$, $B_{R_p} = k_{pi} I_N$, $D_{R_P} = 0_{N\times N}$, one makes CL_1 a (diagonal) system the $2N$ eigenvalues of which are N times the couple $0.5 \mp \sqrt{0.25 - k_{pi}}$. More control on those eigenvalues could be achieved with slightly more complex a structure for R_p, but the choice adopted here is adequate for this work, where the scheduling algorithm complexity needs keeping to a minimum.

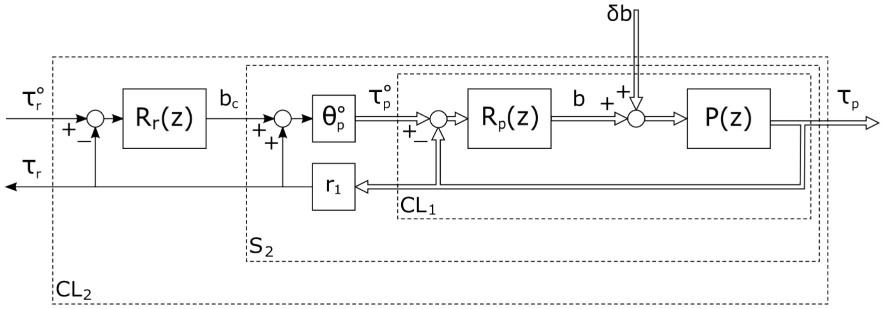


Figure 1: The proposed scheme.

If τ_p° were chosen as $\theta_p^\circ \tau_r^\circ$, then CL_1 would control both the CPU distribution and the round duration, but there would be two problems. First, the dynamics of those two controls would be ruled by the same eigenvalues, which can be inadequate in some cases: for example, one may want the CPU distribution to move smoothly from one situation to another, but the round duration to respond very quickly to its set point, e.g. because a change of that set point means that a greater system responsiveness is needed immediately. Second, and more serious, should some process be blocked, the round duration set point could not be attained.

Consider therefore the system denoted in figure 1 by S_2. Its dynamic matrix is

$$A_{S_2} = \begin{bmatrix} 0_{N\times N} & C_{R_p} \\ \theta_p^\circ r_1 - I_N & A_{R_p} \end{bmatrix} \tag{5}$$

With the chosen R_p, the $2N$ eigenvalues of (5) are 0 and 1 (with multiplicity 1 each) and $N-1$ times the couple $0.5 \mp \sqrt{0.25 - k_{pi}}$. Notice that said eigenvalues do not depend on θ_p°. The SISO system with input b_c and output τ_r seen by R_r in figure 1 has thus the transfer function

$$\frac{T_r(z)}{B_c(z)} = \frac{k_{pi}}{z(z-1)} \tag{6}$$

and in the following two choices are proposed for R_r.

The overall system (CL_2 in figure 1) has the dynamic matrix

$$A_{CL_2} = \begin{bmatrix} D_{R_p}(\theta_p^\circ(r_1 - D_{R_r} r_1) - I_N) & C_{R_p} & D_{R_p}\theta_p^\circ C_{R_r} \\ B_{R_p}(\theta_p^\circ(r_1 - D_{R_r} r_1) - I_N) & A_{R_p} & B_{R_p}\theta_p^\circ C_{R_r} \\ -B_{R_r} r_1 & 0_{N\times N} & A_{R_r} \end{bmatrix} \tag{7}$$

The simplest idea is to choose $R_r(z)$ as a purely proportional controller, i.e., $R_r(z) = k_{rp}$. In this case the $2N$ eigenvalues of A_{CL_2} are those of A_{S_2} with couple (0,1) replaced by $0.5 \mp \sqrt{0.25 - k_{pi}k_{rp}}$. This choice will be termed the "I+P" one from now on, with evident meaning.

In the case of a constant required CPU distribution, the I+P scheme can assign the dynamics of τ_p and τ_r in a (partially) independent manner, having R_r act by means of an additive correction b_c to the round CPU times set point as computed based on the actual round duration. If, conversely, a variable CPU distribution is to be considered, the same scheme can be viewed as a linear switching system with switch signal θ_p°. However, its eigenvalues do not depend on the switching signal, and in force of well known results there surely exists a finite dwell time ensuring the exponential stability of the scheme as switching system.

The I+P scheme is apparently the computationally lightest choice, allows for the simple stability statement above, also permits to give the CPU distribution and the round duration controls different dynamics, but still has the (only) drawback that the round duration control is lost if a process stays blocked (the following examples illustrate the fact). In this scheme the closed-loop transfer function from τ_r° to τ_r is

$$\frac{T_r(z)}{T_r^\circ(z)} = \frac{k_{pi}k_{rp}}{z^2 - z + k_{pi}k_{rp}} \tag{8}$$

thus allowing for a simple choice of the parameters (see the examples later on).

If one cannot guarantee that no persistent process blockings arise, it is advisable to select for R_r a PI structure, i.e., $R_r(z) = k_{rr}(z - z_{rr})/(z-1)$, that can recover such a situation, and leads to what will be termed the "I+PI" scheme. In this case the $2N+1$ eigenvalues of CL_2 have a long expression omitted for brevity, but still do not depend on the switching signal if a variable required CPU distribution has to be assumed. Hence, the same stability considerations above apply. In the I+PI case the closed-loop transfer function from τ_r° to τ_r is

$$\frac{T_r(z)}{T_r^\circ(z)} = \frac{k_{pi}k_{rr}(z - z_{rr})}{z^3 - 2z^2 + (1 + k_{pi}k_{rr})z - k_{pi}k_{rr}z_{rr}} \tag{9}$$

and the parameter choice is just slightly more articulated (again, refer to the following examples).

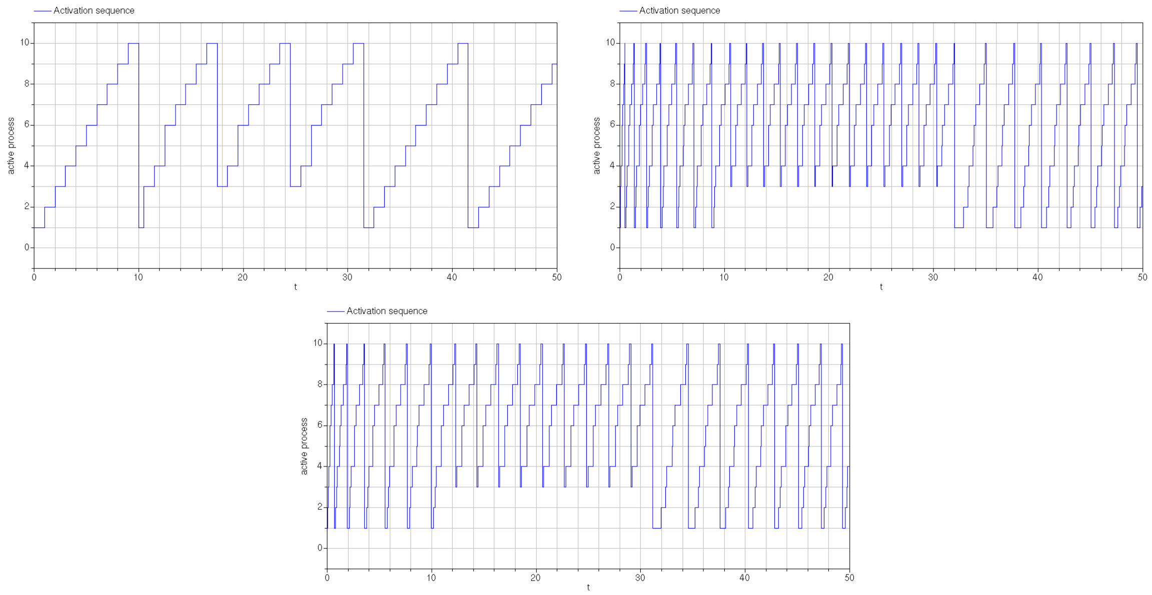


Figure 2: Activation sequence, in the first figure the round robin sequence is presented, in the second one the I+P and in the third one the I+PI.

4 The SkedSim library

The objects to be modelled are the `process`, the `pool` of processes, the `scheduler`, and the complete system (the `world` in our terminology). Since dynamic allocation of objects is not permitted, the pool will simply be a vector of processes, i.e., a fixed number of processes *possibly* being scheduled is allocated, and each of them can be in the scheduling round or not at any given instant.

The process does not *de facto* accomplish anything for the purpose of this research, that is centered on the scheduler's operation; therefore the process is simply a time delay realised by a convenient use of the modelica `when` clause, taking control of the CPU for a time equal to the allotted burst plus a certain variability, and counting the elapsed time by trivially setting the derivative of a convenient continuous variable to one or zero depending on the process having or not the CPU.

The scheduler is in fact the object determining how the switching signal is managed, hence the place where different "actuation" policies are realised. In this manuscript a "round robin" scheduler suffices for all the presented policies, the difference between the standard round robin and the I+P or I+PI schemes residing only in the way the bursts are computed at the beginning of each round.

Thus, each type of scheduler can further specialise its operation by calling a particular *scheduling function*, that implements the specific way of determining the quantities needed to turn an actuation policy into a complete scheduling mechanism. Such a partition between "actuation" and scheduling is consistent with the proposed way of classifying the policies, which in turn corresponds to the system-theoretical idea of viewing them as particular cases of a switching signal, the taxonomy residing precisely in how the switching signal is managed.

Given all the above, the library organisation is very simple and intuitive. To avoid a lengthy treatise, in the following an excerpt of the `process` modelica code is reported.

```
model Process
  ProcessPort p;             // Process/scheduler I/F
  input Boolean blocked;     // Variable for blockings
  discrete Real startTime;   // Last activation time
  Real gCpuPerc;             // Global cpu percentage
  Real gCpuTime;             // Global cpu time
  Real rCpuTime;             // Last round cpu time
algorithm
  when edge(p.activation) then
    p.running := true;
    startTime := time;
    reinit(rCpuTime, 0);
  end when;
  when time>=startTime+p.burst or
       p.burst<=0 or (pre(p.running) and p.blocked)
       then
       p.running := false;
  end when;
equation
  blocked = p.blocked;
  gCpuPerc = if time<=0 then 0 else gCpuTime/time;
  der(gCpuTime) = if p.running then 1 else 0;
  der(rCpuTime) = if p.running then 1 else 0;
  p.gCpuTime = gCpuTime;
  p.rCpuTime = rCpuTime;
end Process;
```

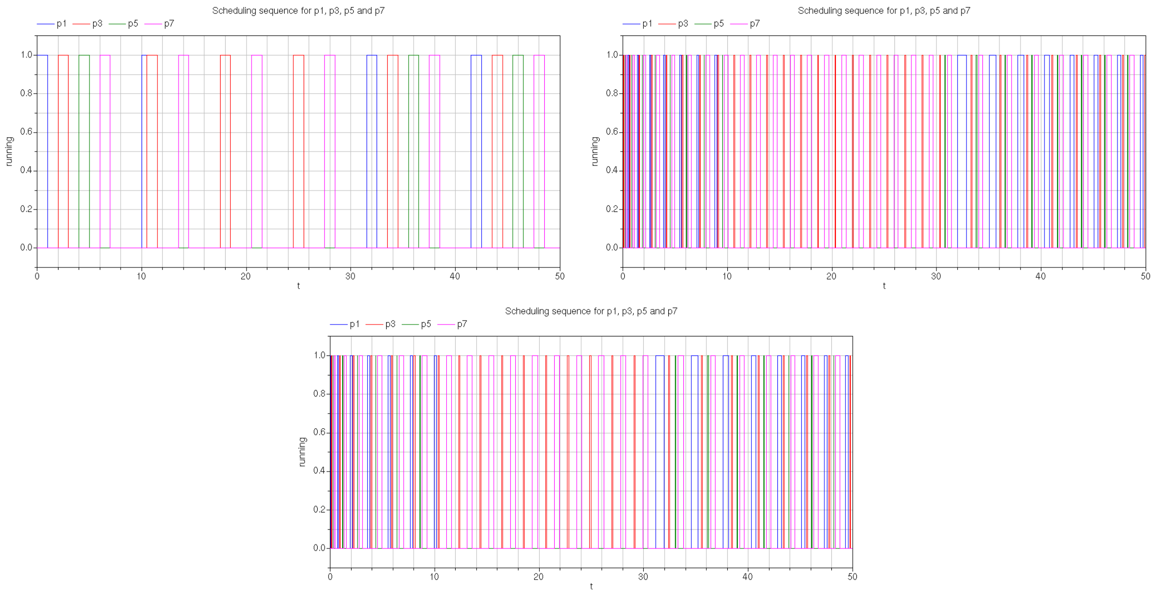


Figure 3: Running time for processes 1, 3, 5 and 7, in the first figure the round robin sequence is presented, in the second one the I+P and in the third one the I+PI.

Moreover,the relevant (algorithm and equation) section of the `scheduler` is shown below.

```
algorithm
 when edge(not_running[previous]) or initial() then
      activation[previous]:=false;
      if previous==nProcesses then
         bursts := Schedulers.SchedulerFunctions
                   .IplusP
                   (nProcesses,SP_Tr,alfa,Tp,bursts);
         // ...or any other scheduling function
         previous := 1;
         roundDuration := time - startTime;
         startTime := time;
         for i in 1:nProcesses loop
           CPUPercPerRound[i]:=if roundDuration<=0
           then 0
           else Tp[i]/roundDuration;
         end for;
       else
         previous:=previous+1;
       end if;
       while bursts[previous]<=0 loop
         previous:=previous+1;
         if previous>nProcesses then
            bursts := Schedulers.SchedulerFunctions
                      .IplusP
                      (nProcesses,SP_Tr,alfa,Tp,bursts);
            // ...same as above
            previous:=1;
            roundDuration := time - startTime;
            startTime := time;
            for i in 1:nProcesses loop
                CPUPercPerRound[i]:=if roundDuration<=0
                then 0
                else Tp[i]/roundDuration;
            end for;
         end if;
       end while;
 activation[previous]:=true;
 end when;

equation
```

```
for i in 1:nProcesses loop
    not_running[i] = not pre(running[i]);
end for;
```

As can be seen, the library organisation and the meaning of the various models are consistent with the specific way of addressing the scheduling problem proposed in this research, and take profit of the possibility (typical of modelica) of mixing algorithmic modeling and asynchronous events. The SkedSim library will be made available to the scientific community under the terms of the GPL license as soon as possible.

5 Simulation examples

This section presents some simulation examples, comparing a standard round robin policy with the two proposed feedback ones, that share the actuation scheme but encompass a more powerful burst computation mechanism. The simulations are conducted with ten running processes. Processes 1, 2 and 5 block from time 10.5 to time 30.5. The round duration set point is 2 for the I+P and I+PI schemes (there is no equivalent in the standard, open-loop one) while the desired percentage distribution is $\{0.1, 0.05, 0.05, 0.2, 0.01, 0.09, 0.2, 0.2, 0.05, 0.05\}$. The quantum for the round robin scheduling is 1 time unit.

Figure 2 shows the activation sequence (i.e., the order in which the scheduler makes the processes run).

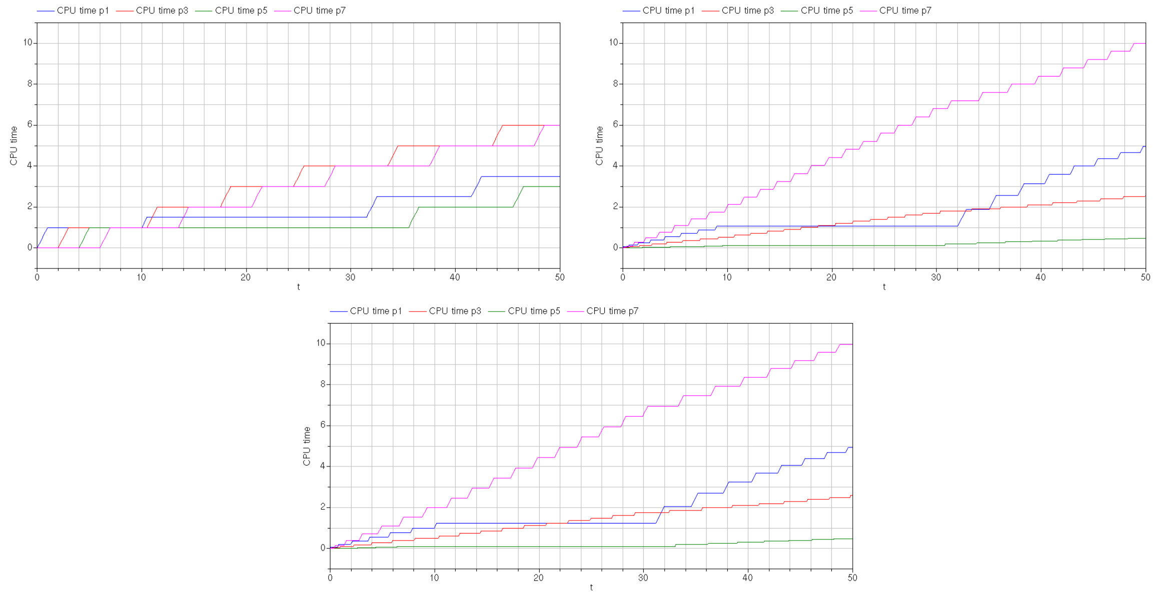


Figure 4: CPU time allotted to processes 1, 3, 5 and 7, in the first figure the round robin sequence is presented, in the second one the I+P and in the third one the I+PI.

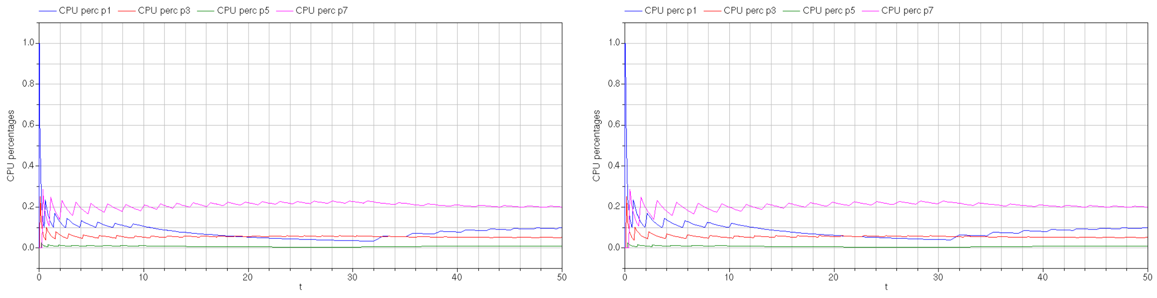


Figure 5: CPU percentages to processes 1, 3, 5 and 7, in the first figure the I+P sequence is presented, in the second one the I+PI.

Figure 3 shows the `running` signal taken from the processes. One can see that in the round robin example the first process was activated at time 10 with a quantum of a time unit but at time 10.5 blocks, releasing therefore the CPU, that is given to process three (the second is blocked too). In figure 4 the times allotted to processes 1, 3, 5 and 7 is shown; the comparison between the round robin and the feedback policies is self-explanatory (the feedback policies allows for a “controlled” distribution, while in the round robin policy no adaptation is possible).

Finally, figures 5 and 6 depict the CPU percentage distribution. The first one shows the distribution over time, while the second one reports the distribution within the current round. It can be seen that even if the percentage per round when the processes are blocked is zero, in the long run the gap is filled thanks to the feedback strategy.

6 Conclusions and future work

In this work, very simple discrete-time control structures were used to synthesise preemptive process schedulers for multitasking systems within a rigorous system-theoretical formalism. In particular, in this paper a Modelica library for the above purpose, at present still under development, was presented, and its use was illustrated with some tests.

Based on the work done until now, the Modelica language has shown some advantages when it comes to modelling computing systems. First of all, it allows to integrate discrete dynamics and events (like signals from the system components) with continuous evolution (i.e., the processes’ execution seen in the real world time). Along the same line, thinking of future extensions, not only scheduling-related control strategies can be seamlessly implemented, but it is also possible to test their validity in the case they have

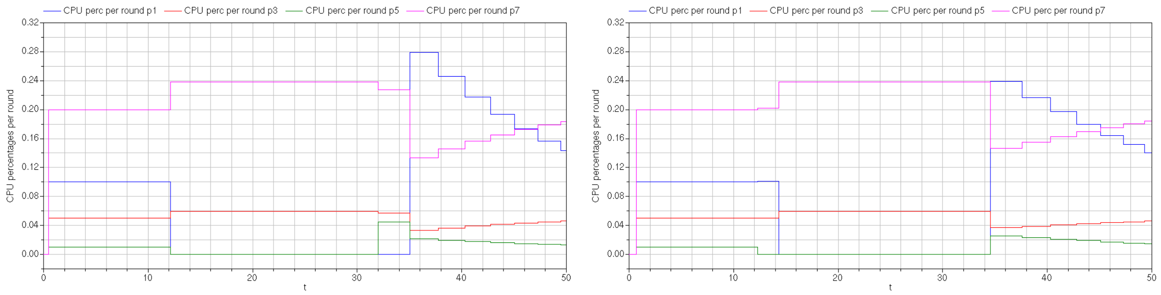


Figure 6: CPU percentages per round to processes 1, 3, 5 and 7, in the first figure the I+P sequence is presented, in the second one the I+PI.

to be used for realising control systems connected to the physical world (as is typically the case for embedded real-time ones, a field that has been devoted much research but is in some sense lateral with respect to this work).

However, the present research has also highlighted some limitations of the Modelica use for the chosen purpose. For example, it would be of great interest in this work to investigate the scheduling algorithm and time performance, analysing how much time is spent for the scheduler execution, and therefore having a clue of the expectable system performances. For well known motivations the Modelica language is not conceived for such a kind of analysis, however.

For the reasons summarised above, future directions for the presented research still need some reasoning and discussion to be envisaged clearly, but given the positive remarks above, other attempts will certainly be done to overcome the mentioned limitations and employ Modelica for the purpose sketched out herein.

References

[1] T.F. Abdelzaher, J.A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23, 2003.

[2] Luca Abeni, Luigi Palopoli, Giuseppe Lipari, and Jonathan Walpole. Analysis of a reservation-based feedback scheduler. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 71–80, 2002.

[3] Ashvin Goel, Molly H. Shor, Jonathan Walpole, David Steere, and Calton Pu. Using feedback control for a network and cpu resource management application. In *Proceedings of the 2001 American Control Conference*, volume 4, pages 2974–2980, Arlington, VA, USA, 2001.

[4] J.L. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury. *Feedback Control of Computing Systems*. Wiley, September 2004.

[5] J.C. Palencia and M. González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *In Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 26–37, 1998.

[6] M. Pinedo. *Scheduling Theory, Algorithms, and Systems*. Springer, third edition edition, July 2008.

[7] O.H. Roux and A.M. Déplanche. A t-time petri net extension for real time-task scheduling modeling. *European Journal of Automation*, 36, 2002.

[8] David C. Steere, Molly H. Shor, Ashvin Goel, Jonathan Walpole, and Calton Pu. Control and modeling issues in computer operating systems: resource management for real-rate computer applications. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 3, pages 2212–2221, Sydney, NSW, Australia, 2000.

[9] F. Wagner, L. Liu, and G. Frey. Simulation of distributed automation systems in modelica. In *Proceedings of the 6th International Modelica Conference*, Bielefeld, Germany, 2008.

Building Modelica Tools using the Modelica SDK

Peter Harman
deltatheta uk ltd.
The Technocentre, Puma Way
Coventry, CV1 2TT, UK
peter.harman@deltatheta.com

Michael Tiller
Emmeskay Inc.
47119 Five Mile Road
Plymouth, MI 48170, USA
mtiller@emmeskay.com

Abstract

Modelica provides numerous opportunities for the engineering industry to promote the reuse and exchange of simulation models by providing a clear standard, open libraries and metadata support via annotations. This opportunity is often underutilized because full Modelica support could not be easily incorporated into software tools without requiring considerable resources.

This paper presents a software development kit, the Modelica SDK, designed specifically to assist developers with integrating Modelica support into any software tool. The philosophy behind this library is to provide maximum extensibility to power users so they can fully utilize the features of the Modelica language and integrate them into their engineering processes for maximum benefit.

The mechanisms provided for a developer to integrate or extend the functionality of the tool into their own software are discussed in detail and examples of the extension points available and their uses are shown.

Keywords: Modelica translator, Java, SDK, API

1 Introduction

This paper presents Modelica SDK, an implementation of Modelica available as a Java library. The Modelica SDK is suitable for adding Modelica support to existing tools as well as developing new Modelica tools and utilities.

Similar tools have been developed with specific goals, such as for style checking and version control [1] as well as translating models for us in an optimization framework [2]. Open source tools [2,3,4] do exist which allow the developer to modify the code to build custom tools. However this not only requires detailed understanding of the underlying software architecture in order to make such modifications but the licensing terms may also be incompatible with the intended purpose of the tool. The aim of the Modelica SDK is to cover all these use cases and allow a broad range of applications, without burdening the developer with creating and maintaining their own implementation of Modelica and/or hindering the developer with undesirable licensing terms.

2 Applications

The aim of this library is to enable the use of Modelica in a wide range of tools and processes and, as such, it is not possible to cover all the possible uses. Instead, we will focus on a number of areas where Modelica could be used in existing tools or where the capabilities of Modelica can be extended using new tools.

2.1 Custom Simulation Platforms

Modelica is first and foremost a modeling language not a simulation platform. Depending on the intended application, the simulation requirements may vary greatly.

For example, in a hardware-in-the-loop (HiL) application it is reasonable to sacrifice a certain degree of accuracy in order to achieve real-time performance as compared to desktop simulation. On the other hand, for detailed design studies, the use of variable time step integrators and careful event handling may dictate greater focus on accuracy at the expense of computational performance. Furthermore, depending on the simulation platform (e.g. HiL hardware, grid computing) I/O requirements may vary (e.g. file system access, IPC support). Finally, domain specific simulation tools, such as engine simulation tools, multi-body tools and/or control system design tools, often have an API for integration of user defined models. In such cases, it may be necessary for the code generation process to support third-party code provided in C, C++, Fortran or some other custom format.

DOI: 10.3384/ecp09430046

For this reason, the Modelica SDK library exposes a code generation API which allows the developer great flexibility in controlling the code generation process. In this way, the developer can choose to target a wide variety of applications (e.g. Hardware-in-the-loop, desktop simulation), control how I/O and integration of third party code is handled and choose in what particular programming language the target code will be written (e.g. C, assembly, Java).

Some code generation schemes use templates or an intermediate form [5] to provide such flexibility. However these require an extra stage of parsing and also may expose code the developer wishes to remain concealed. For this reason, the Modelica SDK provides a set of Java interfaces allowing developers to implement their own code generator (or subclass the SDK provided ones) to suit their specific needs.

2.2 Parameter Management

It is often desirable to store parameters externally to a model, either for the convenience of editing in spreadsheets or other tools, or simply to comply with company policies requiring the use of standardized formats or a central database [6, 7]. By translating these to and from Modelica records, or providing a method of loading records from this data as described with the path and file system package later, the data can be referenced directly from within Modelica editors by using Modelica dot-notation.

2.3 Parameter Studies and Optimization

Whether using a dedicated optimization package or a spreadsheet, the first step of a parameter study or optimization with a Modelica model is to identify the parameters in the model, their default values and units, and their maximum and minimum values if defined. This can be obtained using the “query” package described later.

Whether a calculated variable depends on a particular parameter is often determined by running multiple simulations in a sensitivity study. However, using the Modelica SDK this information can be determined easily from just the flattened system of equations using API calls (described later in the paper) rather than relying on a brute force numerical approach.

Sensitivities may also be handled during a single simulation with modified ODE solvers which require additional code to be generated by customizing the code generation process using the code generation interfaces.

2.4 Issue Tracking and Version Control

The Modelica SDK also provides the foundation for developing utilities for issue tracking and version control. For example, Modelica aware versions of diff and merge tools could be developed that allow visualization of differences in the model structure rather than just focusing on comparing lines of code. Also, as annotations can be added, queried, modified and removed using the SDK, these capabilities finally make the valuable metadata stored in Modelica accessible for storing additional information about the model, e.g. relating issues (in an issue tracking database) to particular model versions.

2.5 Plug-ins to Vertex

As the Modelica SDK is the Modelica implementation used in the Vertex [8] simulation tool, extensions developed for the Modelica SDK also act as plug-ins to the Vertex tool. In this way, any enhancements or features added through Modelica SDK extension points are therefore also accessible from Vertex. Such extensions might include adding the ability to define new simulation targets, new checking rules or new sources to load Modelica models from (e.g. network servers, version control systems).

3 Interfaces

As a software development kit the ease of integration with different software platforms is essential. Different means of integration have been included for different applications.

3.1 Java

The Modelica SDK is developed in the Java language. This gives advantages of platform-neutrality, convenient packaging as “jar” archive files, and “javadoc” documentation (automatically generated from the Java source code).

The examples given in this paper were all implemented in Java.

3.2 Scripting Languages

All the operations accessible to Java are also accessible by scripting languages running on the Java Virtual Machine (JVM), creating a powerful environment for experimentation. Users of Python (Jython), Ruby (JRuby), Groovy, Beanshell and MATLAB can *directly* access the SDK classes and methods.

3.3 C++ and .NET

The platform neutrality of Java is important for a tool such as the Modelica SDK. However, many applications are developed using other programming languages and platforms. For this reason the key methods in the SDK can be exposed to C++ or .NET applications. Note that in such circumstances the SDK still runs in the Java Virtual Machine, and extensions must still be written in Java.

3.4 Web Services

For complete portability a web services interface has been developed using SOAP. SOAP clients libraries are available in most programming languages. The SOAP services were implemented using Axis2 [9], which comes in both Java and C versions.

4 Capabilities

The SDK provides a Modelica 3.1 compatible parser and object model as well as a translator implementing most features of Modelica 3.1. In this section, we will discuss the various features in the SDK and how they impact both users and developers.

4.1 File and Path Management

For developers, the SDK handles all loading and saving of files. Classes are automatically loaded to memory when required thereby freeing the developer from the tedious tasks of managing memory or specifying which files to load.

4.2 Parsing, Object Model and Manipulation

An important design feature of the SDK is that loaded classes are treated as a set of “live” objects, which can be manipulated and queried in memory (similar to how the DOM object model is handled for XML objects). Finding of a particular component, annotation value or equation, is performed by traversing this object-graph. Every object, whether it represents a Modelica class, component, modification or annotation, can have listeners added to it which are notified of any changes. This makes the SDK ideal for editors and other interactive tools.

Modelica classes represented using the SDK's object model can be transformed into other representations, e.g. written back out in Modelica syntax, using the PrintTarget class. By supplying a custom PrintTarget object, the developer can control the appearance and format of the output, e.g. producing XML, HTML or other formats.

4.3 Querying

A package is included for defining and executing queries on Modelica classes or components. This allows selection of components or classes which match a specified predicate. The Apache Commons Collections [10] library defines an interface called Predicate with one method to evaluate whether an object should be accepted or not. In addition to the interface itself, a number of useful predicates are also provided, such as selecting a component by type or variability, or selecting a class by classes it extends from.

For selecting components there is an additional Boolean flag which controls whether or not to iterate through the hierarchy of the model. If this flag is false, only components at the top-level of the class are returned. If true, components from any level of the class are considered. When selecting classes, a similar additional argument controls whether to restrict queries to only models that are in memory or to iterate over all classes found in the path (thus triggering these classes to be automatically loaded as needed). The latter is, of course, quite slow as all relevant files will be loaded and processed, but it can be useful to consider all classes in the path for some applications.

As an example, the following code selects all the parameters from a specified class:

```
Predicate<ModelicaComponent> predicate =
     new ComponentVariabilityPredicate(
     Variability.PARAMETER );
List<ModelicaComponent> parameters =
     Query.selectComponents(myClass,
     predicate, true);
```

Similarly, the following code selects all loaded classes that extend from Real:

```
Predicate<ModelicaClass> classPredicate
     = new SubclassOfPredicate(“Real”);
List<ModelicaClass> realTypes =
     Query.selectClasses(classPredicate,
     false);
```

4.4 Checking

The “checking” package provides convenient methods for validating a model against the Modelica specification and reporting any issues found. The rules provided check for a variety of different types of issues but the real power in the checking capabilities comes from the ability of developers to *add* their own rules to the system (as discussed later in this paper).

When a developer performs a check on a class, the result is a `CheckResultSet` object which contains the set of all results for rules the class has triggered. Each result has a severity and the set as a whole can be queried as to the highest severity result in the set.

There are two powerful features available once a `CheckResultSet` has been obtained. First, it is possible to add a listener to be notified when a result changes. In this way, the system can automatically update these results as users make changes. Second, each result provides a set of actions (extending `javax.swing.Action`) which can be performed to correct the error. This feature is exploited in Vertex to provide a “one-click” auto-correction mechanism where applicable.

```
CheckResultSet crs =
      CheckManager.checkClass(myClass);
if(Severity.ERROR==crs.getSeverity()) {
      // handle an error
}
```

4.5 Flattening and Symbolic Processing

“Flattening” of a class, from hierarchical form to a flat list of equations and variables, is divided into two steps, “instantiation” and “elaboration”. The instantiation process flattens the hierarchy to a set of variables, equations, connectors and connections, each using dot-notation. The elaboration process performs the symbolic processing.

The symbolic processing handles over-constrained connections [11], generates connection equations, and divides the system by the variability of equations. For each level of variability it then assigns the causality of the equations, and for continuous equations reduces the DAE index by differentiating selected equations.

The following code fetches a class from the Modelica Standard Library, instantiates it, elaborates it and requests the total list of equations:

```
ModelicaClass cc =
    ModelicaClassLoader.findClass(
    “Modelica.Mechanics.Rotational.Examp
    les.CoupledClutches”);
Model model =
    DefaultInstantiator.instantiate(cc);
model.elaborate();
List<Equation> equations =
     model.getEquations();
```

4.6 Code Generation

As mentioned previously, the SDK includes methods to translate a model to code. The flexible code generation architecture is provided through extension points which are discussed in greater detail shortly.

5 Limitations

The design objective for the Modelica SDK is to cover as much of the Modelica specification [12] as possible. The parser and object model cover the full 3.1 specification, so operations which manipulate, query, check or flatten classes, as discussed earlier, operate on Modelica 3.1 code, these are tested on the Modelica Standard Library 3.1. The current limitations are in the symbolic processing and code generation sections of the tool. The current limitations are:

- Overloaded operators are not yet supported
- Expandable connections do not yet generate equations
- Stream connectors do not yet generate equations
- Subtasks are not yet generated and `Subtask.decouple(x)` defaults to `x`
- `semiLinear` is not optimized
- MultiBody extensions are supported but not optimized, e.g. `rooted(x)` defaults to `true`
- Inverse annotations are not yet supported, currently symbolic solutions of equations can be defined as described later
- Reducing systems of equations via methods such as tearing is not currently performed, though this is in development and some other optimizations are currently performed.

6 Extension Points

The SDK provides a number of operations which the user can replace or add to with their own code. These are referred to as extension points. Some of these will be described here. In order to understand how extension points work some knowledge of the Modelica translation process as well as the relationship between Modelica classes and files is required. The extension mechanism makes use of the Java `ServiceLoader` mechanism [13] which eases the development of modular applications. The extension points are defined in the SDK, but extensions can be provided in separate modules on the Java classpath.

6.1 Modelica Path

The Modelica Specification describes the relationship between Modelica classes and files on the filesystem. It further describes the operation of the Modelica path lookup mechanism for locating files which uses URLs of the form “modelica://…” to identify the location of files relative to Modelica definitions. The SDK contains interfaces which represent the generalized storage model (for both source files and other types of data contained in the package hierarchy, such as images, data, etc). Implementations of these interfaces are provided for two methods for two such storage schemes. The first is the traditional file system based approach. The second supports accessing code and data from an archive file.

The archive file implementation is based upon a proposal for Modelica 3.1 which allows an entire package (or collection of packages) to be bundled together as a single file. An additional part of that same proposal, the ability to resolve “modelica://” URL’s to locate resources such as image or data files from within the archive file itself, is also supported.

This is one of the most powerful features of the SDK because Modelica code, data, images, etc. can all be bundled into a single archive and easily distributed (rather than stored as an elaborate directory structure on the file system). Furthermore, this feature can be used to create an extensible "virtual package hierarchy" where additional resources are mapped to a Modelica package structure and can thus be referenced directly in Modelica code.

An example use of this feature would be the loading of parameter data from a database. By defining two Java classes a database table could be expressed as a package with a series of records contained within it, and these could be referenced within other Modelica code.

6.2 Checking Rules

An important feature of the SDK is the ability to evaluate rules to check a class for validity and find errors. A number of rules are built-in for checking compatibility of a model to the Modelica Specification. By providing an extension of the class `com.deltatheta.modelica.check.CheckRule` additional rules can be added.

A check rule has a single method, to check a class. This method is passed a `CheckResultSet` object, and is expected to add a `CheckResult` instance to the set if the class fails to pass the rule.

```
public class UseOfPartialClassCheck
     implements CheckRule {


  public void checkClass(CheckResultSet
     results, ModelicaClass clazz) {
      for (ModelicaComponent component :
     clazz.getComponents()) {
        try {
          ModelicaClass type =
     component.getModelicaClass();
          if (type.isPartial()) {
              addResult(results,
     component, type, clazz);
          }
      } catch
     (ClassDefinitionNotFoundException
     ex) {           }
    }
  }


  private void addResult(CheckResultSet
     results, final ModelicaComponent
     comp, final ModelicaClass type,
     final ModelicaClass clazz) {
        results.add(new
     AbstractCheckResult() {
        public String getDescription() {
                return ("Component " +
     comp.getName() + " in class" +
     clazz.getPath() + " is of type " +
     type.getPath() + " which is
     partial");
          }
        public Severity getSeverity() {
                return (Severity.ERROR);
        }
      });
    }
}
```

6.3 Symbolic Rules

The SDK has the facility to add rules for solution of scalar equations. This is useful for cases where a particular format of equation is known to appear regularly in a modeling domain but is not solved by the SDK, or where a user defined function has a particular solution. For example, the following code adds the solution of a simple linear equation:

```
Solution.defineRule(“&y=&m*&x+&c”,
     ”&x:=(&y-&c)/&m”);
```

6.4 Code Generation

Where the SDK is being used to generate simulation code there is a code generation stage in the translation process. In order to support the diverse range of applications proposed this code generation must be able to be language and platform independent. This is achieved by the code generator itself being an extension matching a specified interface. Note there is no built-in code generator provided in the SDK. Furthermore, the specification of the code generation interface is commercially sensitive and therefore only available to licensed developers.

7 Conclusions

Enabling a product to edit, simulate, import or extract data from Modelica models is no longer a lengthy process. With the availability of the Modelica SDK such capabilities can now be rapidly developed or integrated into existing tools. This dramatically eases the adoption of Modelica as a standard for systems model development and exchange. Furthermore, it creates many opportunities for exploiting currently underutilized features in Modelica (e.g. annotations) and integrating Modelica into the engineering process.

References

[1] Tiller M. *“Parsing and Semantic Analysis of Modelica Code for Non-Simulation Applications”*. Modelica 2003.

[2] Åkesson J., Hedin G., Ekman T., *“Development of a Modelica Compiler using JastAdd”*, Seventh Workshop on Language Descriptions, Tools and Applications. 2007.

[3] Najafi, M., Nikoukhah, R., Steer, S., Furic, S. *“New features and new challenges in modeling and simulation in Scicos”*. 2005 IEEE Conference on Control Applications

[4] Fritzson, P., Aronsson, P., Lundvall, H., Nyström, K., Pop, A., Saldamli, L., Broman, D. *“The OpenModelica Modeling, Simulation, and Development Environment”*. SIMS 2005.

[5] Jonas Larsson and Peter Fritzson. *“A Modelica-based Format for Flexible Modelica Code Generation and Causal Model Transformations”*. In Proceedings of the 5th International Modelica Conference (Modelica'2006), Vienna, Austria, Sept. 4-5, 2006.

[6] Tiller M. *“Implementation of a Generic Data Retrieval API for Modelica”*. Modelica 2005.

[7] Koehler J., Banerjee A. *“Usage of Modelica for transmission simulation at ZF”*. Modelica 2005

[8] deltatheta Vertex. [online] http://www.deltatheta.com/products/vertex/

[9] Apache Software Foundation. Apache Axis2. [online] http://ws.apache.org/axis2/

[10] Apache Software Foundation. Commons Collections. [online] http://commons.apache.org/collections/

[11] Otter M., Elmqvist H., Mattsson SE. *“The New Modelica MultiBody Library”*. Modelica 2003.

[12] Modelica 3.1 Specification. 2009.

[13] O’Conner J. *“Creating Extensible Applications with the Java Platform”*. [online] http://java.sun.com/developer/technicalArticles/javase/extensible/. 2007.

The Role of Modelica in a Robust Engineering Process

Peter Harman
deltatheta uk limited
The Technocentre, Puma Way, Coventry, CV1 2TT, UK
peter.harman@deltatheta.com

Abstract

An engineering process comprises stages of design, analysis and optimization, often with specialist tools and engineers in each of these areas. We look at how the interaction and communication between these stages can be tailored to maximize the robustness of the overall process, and what the role of Modelica is in achieving this. This is described for industrial examples. We then discuss the additional requirements of Modelica tools and libraries for such a process.

Keywords: Engineering process; simulation strategy; parameter management; Product Lifecycle Management (PLM)

1 Introduction

The requirements of physical modeling tools in terms of their overall structure and capabilities [1] and the history of their development [2] have been discussed with the common conclusion that a correctly developed tool can greatly simplify the job of the user by the use of features such as symbolic manipulation, drag-and-drop modeling and version control. These however prompt the further question of what can additionally be done for a physical modeling tool to have a greater influence on the engineering process as a whole. This paper attempts to study this question with industrial examples and the tool requirements these create.

2 Robust Engineering Processes

2.1 A Definition

Rather than robustness of the developed product we define a robust engineering process as one where:

- Errors due to different engineers using different sets of data are not made
- Engineers do not spend time performing calculations just to move a design from one toolset to another, such as CAD to mathematical model or vice versa
- The true optimized design is found and the fact that assumptions and objectives change during the process is both acknowledged and handled

Although this definition sounds like purely policy changes, it is really a matter of communication and data management. Figure 1 shows a comparison between two engineering processes. A serial process where stages of design and analysis follow each other, with data translations between the two, and a concurrent process where design and analysis engineers draw data from the same source, which is updated with any changes made by each, and where translation is automated. This concurrent process is the robust process we aim to achieve, and although in this case it may be a small improvement, considering a case where more areas, such as control systems development or test engineering, are involved in the project, the efficiency is greatly improved.

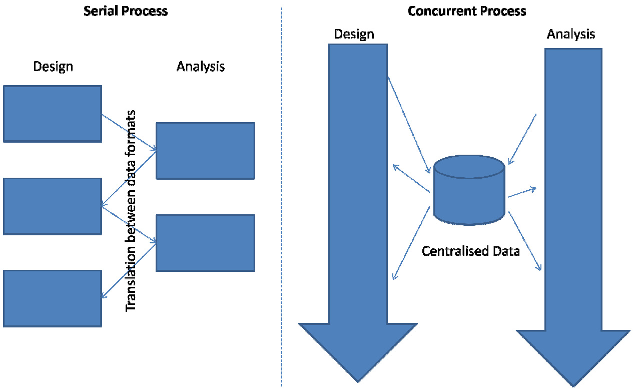


Figure 1: Serial and Concurrent Processes

It is important to stress that this process does not involve engineers blindly using whatever data is currently available, management of versions is critical and as with CAD data or software version control there must be a process of publishing / releasing fi-

DOI: 10.3384/ecp09430045

nished work and monitoring dependencies between versions.

2.2 Modelica and PLM

An analogy can be drawn between successful introduction of Modelica into an engineering process and the introduction of Product Lifecycle Management [3] (PLM). Modelica aims to maximize model reuse and therefore Modelica users within a company will be working from the same set of libraries. In the same PLM allows all engineers in the company to work from the same data, whether in design, manufacturing or purchasing.

3 Industrial Examples

We will look at two industrial examples, one where speed of development is critical, and one where development is shared between different organizations.

3.1 Formula 1

In Formula 1 timescales are compressed and commercial success can only be gained from the most optimized design. While a small industry, it is a clear example of the benefits of efficient and robust design processes and has been an example often used in the introduction of PLM [4].

In this environment simulation is critical and is typically applied in different areas across the engineering departments within a team. This may include CFD of the aerodynamics, multibody modeling of the vehicle dynamics, lap simulation, powertrain simulation, hydraulic system simulation, control system development and driver training simulation.

Each of these areas might use different simulation tools and hence different models. There is a cost motivation to share models between departments in order to save work, and a technical motivation in order to ensure all departments are working from the same assumptions. The reason for the different tools is that the requirements of the models in these simulations are very different, and even where the domain of the model is related, such as between a vehicle dynamics simulation and a lap simulation, the detail level varies widely.

Modelica can be used to unify the modeling and simulation across the organization as long as:

1. The Modelica environment used can generate simulation code for all the required platforms.
2. The models make use of replaceable components to achieve different detail levels with the same parameter data.
3. The models can be parameterized directly from design data and car setup data.

Achieving this reduces the time from a design change or car setup change to having a running simulation, and hence the time for simulation engineers to feedback recommendations based on results. Since the start of 2009 track testing during the season has been banned making simulation more critical, and it is desirable that all design changes are successfully analyzed before racing so such a reduction in the required time to do this is important.

3.2 Automotive Tier 1

Tier 1 suppliers in the automotive industry have ever growing responsibility for the delivery of complete systems for the vehicle. Whilst this is fully within the engineering capabilities of the suppliers, defining the responsibilities for system integration often causes issues. This becomes especially important when a system integration issue arises during production creating a situation of assigning blame and responsibility.

By sharing systems engineering data, the Tier 1 and OEM can speed up system integration during development, and speed up location of faults, and hence responsibility, with failures during production. Virtual testing of the full vehicle can occur early in development using models developed by those responsible for the component development.

As an open-standard, Modelica is ideal for such sharing of models. This is achievable as long as:

1. Design versions are managed and a model can be selected that corresponds to a particular design of the system.
2. Models can easily be communicated, either by packaging models and resources in a single file, or by using standardized protocols e.g. a PLM system or a Subversion version control system.
3. Supplier and OEM intellectual property can be protected without hiding the intention of and operation of models.

Achieving this reduces duplication of modeling work in both parties, ensures models are developed by those closest to the design, and creates a collaborative development.

4 Requirements for Tools and Libraries

The concepts discussed here are entirely feasible within the existing Modelica tools available. However in order to achieve the levels of efficiency, data management and integration required to achieve the processes mentioned there are a number of recommendations for both Modelica tools and libraries.

4.1 Integration of tools

In the F1 example above one requirement was to parameterize models from design data and car setup data. This is a wider issue where many users require parameters sourced from spreadsheets, CAD software and company specific software.

Much data is stored in spreadsheets as a convenient form for viewing and performing calculations. Rather than copying data from the spreadsheet a tool should allow model parameters to be linked to spreadsheet cells, and updated as the spreadsheet file is updated.

Translators from CAD to Modelica multibody models have been written [5] however to successfully connect the two the Modelica tool would be able to monitor the CAD data for changes and the model would be continually updated accordingly. The CAD data contains more than just the structure of the assembly and mass and inertia data. The model should be able to request other parameters from CAD features e.g. the area of a piston for a hydraulic actuator.

Most engineering companies will have data stored in a company specific manner. Following the examples above, an F1 team will have data for setup of the car by race engineers which will ideally be possible to load into a simulation, an automotive tier 1 might supply multiple customers and have data sets for each customer which they want to keep separately to the model itself. Tools should have an API to allow parameterization from any data source.

4.2 Protection of models

In the Automotive Tier 1 example discussed above one issue was protection of IPR. There are three main solutions in this area, each with advantages, which one is preferable will always be an area of debate.

1. Encryption: The Modelica text is encrypted and the tool does not allow access to it. This has the advantage that the model structure is preserved and optimizations can be performed when it is simulated, however tracing bugs when the tool will not allow access to equations or variable names can cause problems, and the encryption is tool-specific and not part of the Modelica specification.
2. Compilation: The model is compiled to a dynamic linked library (dll) that can be used at simulation time. The MODELISAR [6] project will define a standard for this, so the issue of compatibility is reduced, however compiled code is platform specific, and no further optimizations or analysis of the model structure can be performed.
3. Obfuscation: This is a means of hiding the details of interpreted languages, most widely used examples being Javascript code in web pages, by changing the names of variables and methods and using complex syntax to make the code unreadable but still functional. This has not been widely explored with Modelica but may provide a tool-neutral alternative to encryption.

4.3 Communication of models

The Modelica specification defines a filesystem structure for storage of Modelica libraries and a mapping between files and classes. In addition to the Modelica code a library often requires resources such as image files and HTML documentation. There is a proposal for Modelica 3.2 to allow an entire library and associated resources to be stored within an archive file similar to .zip or .jar files. This would ease communication of models as only 1 file is required for an entire library.

An alternative approach for ensuring libraries can be shared and transmitted would be for Modelica libraries to be loaded directly from a central source e.g. a version control or PLM system.

4.4 Modelica compliance

Release of new features in the Modelica specification not surprisingly precedes the implementation of them in tools. Also some features may not be relevant to the focus of a particular tool and are not implemented. Communication of models between different companies who might use different software could be complicated by whether one Modelica tool supports the same Modelica features as another. At the moment there is no standard description of compliance to the language specification. It would be very useful to companies using or considering using Modelica to be able to compare compliance to the standard, this could be illustrated with a table of features against whether the tool in question supports them.

4.5 Discretization of models

Parameterization of a model from CAD and other design data is made easier if the discretisation of the model, that is the division of the model into individual components, matches the physical system e.g. one physical part, and hence one CAD part, corresponds to one Modelica component.

5 Conclusions

Successful adoption of Modelica can influence more than just model development and generate improvements in the engineering process as a whole. There are actions required by engineering companies adopting Modelica in order to achieve this, but also actions on Modelica tool and library suppliers to aid this. Tool integration and communication of models are important features which must be considered. To ease compatibility of libraries to Modelica tools, a matrix of tool compliance to individual features of the Modelica specification should be published.

References

[1] Tiller, M. *"An Analysis of Physical Modeling Requirements and Techniques"*, SAE 2009

[2] Breitenecker, F. *"Software for Modelling and Simulation - History, Developments, Trends and Challenges"*, Simulation and Visualization 2006

[3] Stark, J. *"Product Lifecycle Management (PLM): 21st Century Paradigm for Product Realisation"*

[4] Jeffreys, M. *"The Challenge of Introducing Product Lifecycle Management in Formula 1 and lessons for other organizations and sectors"*. Matthew Jeffreys Consulting Ltd, 2007.

[5] Engelson, V. *"Tools for Design, Interactive-Simulation, and Visualization of Object-Oriented Models in Scientific Computing"*, Linköpings Universitet 2000

[6] *"MODELISAR Project Profile"* 2008 [online] http://www.itea2.org/public/project_leaflets/MODELISAR_profile_oct-08.pdf

Towards a full integration of Modelica models in the Scicos environment

Ramine Nikoukhah[1]   Sébastien Furic[2]
[1]INRIA Rocquencourt, France
ramine.nikoukhah@inria.fr
[2]LMS Imagine, France
sebastien.furic@lmsintl.com

Abstract

In this paper, we show that, provided the Modelica language specification is enriched with the notion of *external connector*, interesting applications follow, where Modelica can be used to describe *synchronous event-activated* blocks that communicate efficiently with the “outside world”. Such blocks described in Modelica and compiled separately can still be connected together to form new blocks. Scicos (www.scicos.org), a freely available modeling environment, can make efficient use of those blocks, that in addition enable seamlessly integration of Modelica into a causal environment.

Keywords: external connector; events; synchronization

1 Introduction

Scicos is a software tool for modeling and simulation of dynamical systems [1]. It provides a hierarchical graphical editor for the construction of complex dynamical systems, a simulator and a code generator. Scicos is distributed with the free scientific software package ScicosLab (www.scicoslab.org).

Currently Modelica is partially supported by Scicos. In particular it is possible to use component models in Scicos diagrams where the dynamics of the component has been described in Modelica [2]. This means that, in the same diagram, sections of the model are designed using standard Scicos blocks and others using Modelica blocks. So far, the Modelica section has only been used to model piecewise continuous-time, leaving all discrete-time behaviors to the Scicos-block section [3]. This configuration, in which Scicos and Modelica blocks co-exist in the same environment, has been available in Scicos for a number of years and is very similar to the Simulink/ Simscape product recently released by the Mathworks.

To go further in the integration of Modelica within the Scicos environment, the question of discrete dynamics in Modelica and its synchronization with the outside world needs to be addressed. There are two difficulties in extending the current integration:

- Modelica is conceived to be used to model the whole system. Except external functions and I/O routines with very limited functionalities and under restricted conditions, a Modelica model is written entirely in the Modelica language
- the event synchronization properties in Modelica itself are not unambiguously specified [4].

In this paper, we use the extensions proposed in [4] to propose a solution for the full integration of Modelica components in Scicos or in any other causal simulation environment with synchronous block activation semantics.

2 Current Scicos/Modelica interaction

2.1 Mixed Scicos diagrams

In a *mixed Scicos diagram*, the interaction between standard Scicos and Modelica blocks is made through *mixed type blocks*. These blocks have both regular Scicos input and/or output ports (i.e., *causal ports*) and Modelica connectors (i.e., *acausal ports*). In the example given below, we can see that Scicos ports are drawn as arrow heads whereas Modelica connectors are drawn as plain rectangles. Clearly it does not make sense to connect directly a causal port to an acausal one, and the Scicos editor forbids it.

DOI: 10.3384/ecp09430024

The interaction between the controller part of this diagram, which is designed using standard Scicos blocks, and the hydraulic Modelica model is done through two mixed-type blocks: the controlled valve and the reservoir. In the case of the reservoir, for instance, the regular output port corresponds to a level sensor.

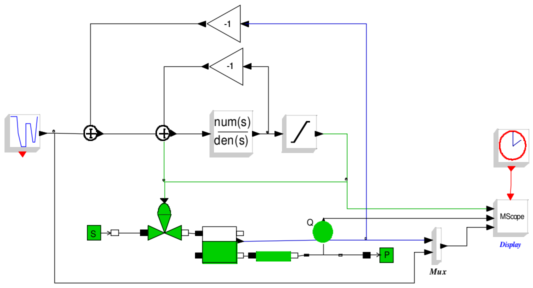


Figure 1: A mixed Scicos diagram including standard Scicos and Modelica blocks

The way Scicos compiler deals with mixed diagrams is to group all the Modelica blocks (including mixed type blocks, which are written in Modelica[1]) and generate the corresponding Modelica program. This program is then compiled with Modelicac (the free Modelica compiler distributed with Scicos), generating a C code that is transparently compiled and linked with Scicos so that, at the end, the whole Modelica section becomes a regular Scicos block where its (now causal) ports correspond to the input/output ports of the mixed block types in the original diagrams.

Even though in Modelica the whole system should normally be designed using the language itself, omitting the description of the outside world (by only specifying "external" input/output ports) would not cause major difficulty, as already proven by pure Scicos models already, which already enable separate compilation. Since the Modelica language specification does not explicitly define interaction with outside world, we propose here to fill that lack of specification by defining *external connectors* and by giving the rules that govern their use.

3 Extension of the Scicos/Modelica interaction

3.1 Activation signals

In Scicos, *activation signals* are used explicitly to specify control actions. For example on Figure 1, the time instants when the Scope block samples its input values and displays them is specified by the activation signal (sequence of periodic events in this case generated by the event clock) it receives through its activation input port placed on top of the block. Basic operations on activation signals in Scicos are used to provide periodic and non periodic clocks, sub-sampling, conditioning, etc. Such functionalities in Modelica are provided through special language constructs such as **sample** and **when**, not through signals that can be communicated through connectors. Actually, Modelica is not designed in the same spirit synchronous languages such as Scicos-the-language are. Consider for example the simple model below:

```
block Counter
  input Boolean activated;
  output Integer count(start=0);
equation
  when activated then
    count = pre(count) + 1;
  end when;
end Counter;
```

Modelica semantics currently impose instantaneous equations to be activated by the *detection of an edge* in Boolean conditions of "when" equations, with the notable exception of equations guarded by the **sample** operator[2]. As a consequence, it is not possible to separately compile a counter that increments its value when the environment into which it will be put simply decides it should do: testing the Boolean expression is still necessary to eventually activate the equations. The purpose of activation signals proposed by Scicos is precisely to obviate that problem: indeed, activation signals only take one value that gives the time instant *when* the model should be activated. The rest of the time, the signal is simply *absent* (i.e., no value is present at the model's input port). If we make an analogy with booleans, we can say that activation signals can only take the `true` value, *when they have a value*[3].

[1] Despite Modelica the lack of a way to express true "external" connectors. Typically, top-level input/output declarations are specifically treated by compiler to yield external connectors as a final result. That has also been used in Dymola for generating Simulink blocks.

[2] Hence the necessity to define it as a primitive of the language.

Going back to our example, the optimal code typically generated for the "when" equation would be (suppose the target language is C):

```
if (activated) count = count + 1;
```

Each time the value of `activated` is updated, the simulation environment has to run the code above, which performs the test, as explained above, and then update `count` if required.

Now suppose we have an `Event` type (see [4]) in Modelica to represent the type of activation signals. Our counter could then be written:

```
block Counter
  input Event activated;
  output Integer count(start=0);
equation
  when activated then
    count = pre(count) + 1;
  end when;
end Counter;
```

We have only changed the type of the input. But the optimal code typically generated for the "when" equation now becomes:

```
count = count + 1;
```

No test is required since the only possible value for `activated` is equivalent to the Boolean value `true`: incrementing the counter is just a matter of calling the above code, which is the responsibility of the outer environment. Imagine a model that would contain hundreds of (deeply nested) sub-models: the use of activation signals would result in significant increase in performance.

3.2 Activation ports in Modelica

Since the lack of activation signals in Modelica makes the interaction with Scicos (and potentially any other external causal environment) difficult, we simply propose to define external connectors capable of listening/emitting *events*.

An interesting solution would have been to reuse the **external** keyword as prefix to the declaration of external connectors as in:

```
external connector EventInput
  Event e;
end EventInput;
```

But since a compliant Modelica compiler is required to compile code for interaction with the outer world as soon as "toplevel" declarations of variables with prefix **input** or **output** exist, we will simply omit the keyword **external**, to remain compatible with the current language specification. So the only addition we propose to the language is the `Event` type (with accompanying operations). Interfacing with the outer world is just a matter of special interpretation of toplevel declarations, as usual.

For example, the following model, once compiled at toplevel, yields a Scicos block with two input activation events and one output activation event:

```
block B
  input Event e1, e2;
  output Event e3;
equation
  ...
end B;
```

Of course, the ability to declare activation ports is far from sufficient to enable seamless integration of Modelica into the Scicos environment. In particular, Scicos has to know where the events are generated and which input/output activation ports depend on which, in order to ensure global synchronization of the whole model

After presenting in the next section the `Event` type in more details and operations that apply to its members, we explain the principle of model abstraction in presence of activation events.

3.3 The `Event` type

The `Event` type is the type of all *events*, that is, abstract objects which sole purpose is to represent *activations times* in hybrid models. Notice that by "activation time" we do not speak about normal Modelica values that would represent measures of time, as provided by the pseudo-variable `time`: we are talking about the **domain** of signals which is implicit in Modelica, where programs only manipulate values of signals, i.e., elements of their codomains.

As already said in previous sections, events only have one possible value (but different domains!). Contrary to the design choice made in the Signal language, we do not want compatibility of `Event` with `Boolean` (by declaring the former to be a subtype of the latter). Instead of that, we prefer explicit conversion in expressions when required, to avoid unnecessary complications and user confusion. Only "when" equations accept both events and booleans, mainly for compatibility with existing practice.

[3] In the synchronous language Signal, the value of activation signals is denoted *true* and is compatible with the Boolean type.

Since `Event` is such a degenerate type, the only operations that make sense for events involve in reality subsets of the time line, i.e., their domain[4].

Creation of events corresponding to edge detection requires the following syntax:

```
e = event(x >= 0.0);
```

The primitive `event()` creates an event activated whenever its Boolean expression argument becomes `true`. It should not be confused with the primitive `edge()` that, in Modelica, does not generate impulses (see [4]).

It is also possible to program events in the future by specifying a delay from current time:

```
e1 = event(x >= 0.0);
when e1 then
  e2 = e1 + 10.0;
end when;
```

The equation inside the "when" reads: "`e2` occurs ten units of time after `e1`".

Event synchronization, subsampling, etc., can simply be expressed by means of "when" equations, provided their semantics are revised so that tractable syntactic methods would be used to detect clock dependency[5], as explained in [6].

Modelica is now ready for true separate compilation, as explained in the next section.

3.4 Abstraction of models having activation ports

With the new connectors introduced here, an isolated Modelica compiled model can be represented as follows:

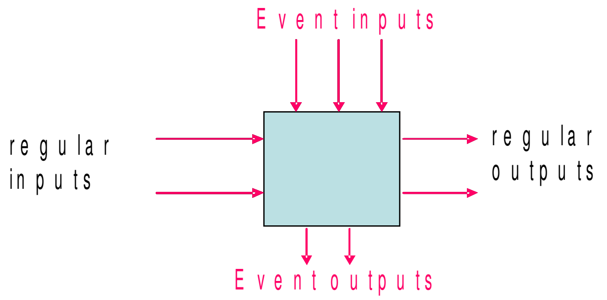


Figure 2: an abstracted separately compiled Modelica model with activation ports

This compiled model can be inserted and used as an ordinary Scicos block provided an additional constraint is fulfilled: Scicos imposes that, in basic blocks, output events are never synchronous with input events. Fortunately, this property can be checked at model compilation time: indeed, as mentioned above, synchronicity can be determined syntactically.

In Scicos, input events can be synchronized. This synchronization is imposed from the outside of the module (so outside of the Modelica code) depending where the input events come from. If two input events in a model are such that, say, one of them is a subsample of the other, then clearly the synchronicity of the two events must be taken into account within the Modelica program. For example, in the case of the module depicted in Figure 2 above, the activation can come from the first event, the second, event, the third event, the simultaneous first and second events, the simultaneous first and third events, etc. In [4], the "switchwhen" clause was introduced specifically to deal with this problem: that clause simply generalizes the Modelica "when" clause by providing a way to express constraints depending on explicit synchronicity conditions. We propose in this paper a more "Modelica-like" version of that construct, that for clarity however requires some rudimentary pattern matching feature[6]. Here is an example:

```
match {e1, e2, e3} with
  case {-, -, *} then
    <eq1>
    <eq2>
    ...
  end case;
  case {*, -, *} then
    <eq3>
    <eq4>
    ...
  end case;
  case {*, *, * } then
    <eq5>
    <eq6>
  end case;
```

[4] Which is fortunate since events have been precisely introduced for that purpose...

[5] In other words, clock calculus would become typing.

[6] Pattern matching has already been proposed in other Modelica-like languages: the MetaModelica language used in the OpenModelica project features pattern matching. See http://www.ida.liu.se/~pelab/modelica/OpenModelica.html

```
end match;
```

The content of one and only one case is activated depending on what combination of events `e1`, `e2` and `e3` is generated. A "case" clause is activated whenever the expression after **`match`** matches one of the *patterns* following each **`case`**. In a pattern, "`-`" and "`*`" denote respectively the absence and the presence of a signal[7]: the first symbol corresponds to the first event (here `c1`) and so on. For example if events `c1` and `c3` are simultaneously generated but not `c2`, then the second case is activated. For this to happen, `c1` and `c2` must be synchronous. By using this clause, the Modelica code inside the module can anticipate every possible combination of input events activating it.

3.5 Composition of separately compiled models

An interesting property of separately compiled models is that they do *compose*. In other words, it is still possible to connect such compiled models to build new models without recompilation. More importantly, models that only define discrete-time signals (by means of "when" clauses) compose without significant loss of efficiency. This is not so surprising: since they are undistinguishable from ordinary Scicos blocks, they share the same properties (and Scicos blocks do compose efficiently!).

4 Conclusion

The Modelica extensions we have proposed allow us to integrate a larger class of Modelica models into Scicos blocks. This functionality should be made available in future versions of Scicos. Also, any causal simulation environment capable of handling activation signals (that, at runtime, only consists in calling blocks in the right order by ensuring synchronization of inputs if required) would benefit of such extensions.

Another important consequence of our proposal is that, implemented by tool vendors, it would lead to a compilation scheme for pure discrete-time **open models** that, for example, could be used by component providers to produce efficient versions of library models to their clients while still preserving IP.

[7] Remember that the only useful information carried out by an event is its presence.

References

[1] S.L. Campbell, J. Chancelier, R. Nikoukhah, ***Modeling and Simulation in Scilab/Scicos***, Springer 2005 - Chinese edition, BuptPress, Beijing, 2007

[2] M. Najafi, S. Furic, R. Nikoukhah, ***Scicos: a general purpose modeling and simulation environment***, Proc. of the 4th International Modelica Conference, Hamburg, 2005

[3] M. Najafi and R. Nikoukhah, ***On the Integration of Modelica in the Modeling and Simulation Software Scicos***, Proc. of Modelling, Identification and Control, 2009

[4] R. Nikoukhah, ***Extensions to Modelica for efficient code generation and separate compilation***, in Proc. EOOLT Workshop at ECOOP'07, Berlin, 2007

[5] R. Nikoukhah, ***Hybrid dynamics in Modelica: Should all events be considered synchronous***, in Proc. EOOLT Workshop at ECOOP'07, Berlin, 2007

[6] R. Nikoukhah, S. Furic, ***Synchronous and Asynchronous Events in Modelica: Proposal for an Improved Hybrid Model***, in Proc. Modelica Conference, Bielefeld, 2008

Linear Analysis Approach for Modelica Models

Loig Allain Stéphane Neyrat Antoine Viel
LMS Imagine
La Cité Internationale, 84 quai Charles de Gaulle, F-69006 LYON

Abstract

This paper is dedicated to the Linear Analysis of dynamic systems represented by Modelica models. The simulation of dynamic systems helps the engineer in his design activities giving him a better understanding of the system he is in charge. Additionally to time-domain simulations, frequency domain analysis provides a complementary view of the systems. This paper gives a general overview of the Linear Analysis methods that can be performed on Modelica models using the existing facilities in LMS Imagine.Lab platform. It highlights the use of Eigenvalues, Modals Shapes, Transfer Functions or Root Locus to fully understand the intrinsic dynamic behaviors of the systems, including non-linear systems. Different multi-physics examples show how these tools can be used practically.

Keywords: Modelica; linear analysis; eigenvalues; modal shapes; transfer function; root locus

1 Introduction

The responsibility of any engineer is to find a technical solution to a given problem. Most issues appearing around dynamics systems are related to vibrations or instabilities of the control laws used to pilot these systems. In order to analyze the response of their systems, the engineers can use the computer efficiency. With virtual prototypes, engineers can explore different design options such as evaluating the influence of parameters variations in time-domain simulations. However, more powerful approaches such as frequency-domain analysis can reveal the intrinsic dynamic properties of these systems, independently from their time-domain excitations, and with very few computation times. Finally, the full understanding of the in-depth dynamic behavior of the systems allows making the most appropriate design choices with an important efficiency in the design process (fewer iterations, then reduced development time).

In order to improve the behaviour of the system, several possibilities could be tested:

Some engineers could:

- perform several batch runs in the simulation tool, changing a set of parameters and looking at the corresponding time-responses, with quite long CPU-times.
- launch Design Exploration tools such as Optimization, Design Of Experiments, Monte-Carlo, Pareto Plots, … with quite long CPU-times.

Or, others engineers would prefer to:

- analyze the root locus of the system as function of each parameter, and to check if the eigenvalues (real and imaginary parts, that provide natural frequency [Hz] and damping ratio [null]) move towards more stable areas.
- analyze the modal shapes in order to understand how each mode contributes to the motion of each output observer variable.
- analyze the transfer functions of the system, in particular the one linking some output observer variables to some input control variables (O(s) / I(s)), and to check which are the conditions of resonance (peaks in Bode plots for example), when natural frequencies can be excited by the inputs.

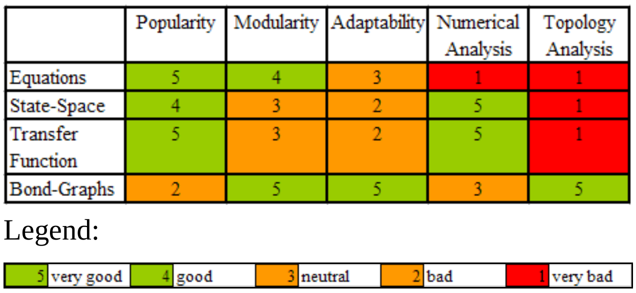
This paper emphasizes the Linear Analysis approach as a very powerful method. State space representation and the Linearization process used in the LMS Imagine.Lab AMESim tool supporting Modelica models are presented. The second section shows some practical applications on multi-domains systems such as mechanical, thermal and electrical models. Finally, the Linear Analysis approach is applied to a Diesel Common Rail Injection System. This general approach is discussed to conclude on its benefits in system modeling.

DOI: 10.3384/ecp09430097

2 Linearization Process of Nonlinear Systems

2.1 Level and Formalism for physical modeling

Various formalisms exist to represent physical models. For example, Vangheluwe [1] proposed a classification with a multi-criteria analysis that shows that each formalism has its advantages and its flaws.



	Popularity	Modularity	Adaptability	Numerical Analysis	Topology Analysis
Equations	5	4	3	1	1
State-Space	4	3	2	5	1
Transfer Function	5	3	2	5	1
Bond-Graphs	2	5	5	3	5

Legend:

5 very good	4 good	3 neutral	2 bad	1 very bad

Figure 1: Comparison of formalisms/languages, extract of some criterias from [1]

With the comparison of the formalisms, it then becomes clear that Modelica, with its high level of abstract for the models description, can help the *Equations* formalism to fulfill non-good criteria such as Numerical Analysis or Topology Analysis, to bring *Equations* formalism combined with Modelica to the highest values.

However, it may still be interesting to use formalisms such as *State-Space equations* or *Transfer Functions* commonly used in the Control community, to add valuable information about the system's dynamics (Otter [2]). Since these ways of representation are only usable in linear cases, engineers could think this would strongly limit their applicability. But it is not actually the case since non-linear systems can be linearized around some operating points. It is also very useful to make analogies on complex physical systems to represent them as simpler equivalent linear mass-springs models. Additionally, the utilities developed by control-specialists on the basis of these formalisms are numerous and extremely useful (e.g., the study of dynamics and stability on Bode, Nyquist or Black-Nichols charts, the location of Evans poles, etc.), which largely justifies their use in the context of process control problems.

2.2 Numerical linearization of the system

One of the principal techniques for the analysis of nonlinear system is to approximate them with a proper linear system and then to use the linear system theory, which is fully established since about three decades (Russell [3]).

The very interesting point is that whereas engineering systems are never linear, they can practically be approximated as linear, and it appears that such a linear system description is sufficient enough around selected operating points (N [rpm], P [bar], T [°C], …). It is particularly true when the amplitude and frequency of the excitation signal (the inputs) are kept in between certain limits, which define the domain of linearity.

The linearization process needs the determination of the Jacobian matrix of the system at the desired operating point.

The evaluation of the [A, B, C, D] matrices is quite equivalent to considering the linear behaviour tangent to the nonlinear system in correspondence with an operating point which has also to be an equilibrium point.

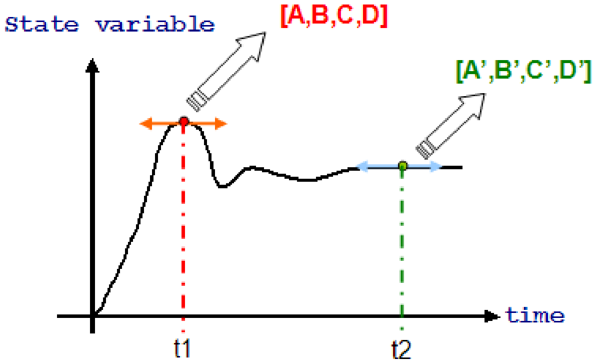


Figure 2 : linearization of a nonlinear system around two different local maximum points

Finally, a full description of the non-linear system around various operating points can be obtained considering several times the linear description of the non-linear system around several equilibrium operating points. Ones could imagine it would lead to a large number of linear analysis to be done, but in practice, only the extreme operating points, such as an injector fully opened and an injector fully closed, are considered.

The non-linear system may be written as:

$$\frac{dx}{dt} = f(x,t)$$

The linearization of these differential equations gives the Jacobian matrix A:

$$A = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n} \right]$$

Each term of the ith vector is evaluated, applying a numerical perturbation to the state variable x_i:

$$\frac{\partial f}{\partial x_i}=\begin{bmatrix}\frac{\partial f_1}{\partial x_i}\\ \vdots\\ \frac{\partial f_2}{\partial x_i}\\ \vdots\\ \frac{\partial f_n}{\partial x_i}\end{bmatrix}=\begin{bmatrix}\frac{f_1(x_1,\cdots,x_i+\delta,\cdots,x_N,t)-f_1(x_1,\cdots,x_i-\delta,\cdots,x_N,t)}{2\delta}\\ \cdots\\ \frac{f_n(x_1,\cdots,x_i+\delta,\cdots,x_N,t)-f_n(x_1,\cdots,x_i-\delta,\cdots,x_N,t)}{2\delta}\end{bmatrix}$$

The other B, C and D matrices are determined in the same way. For an ordinary differential equation system, the linearized system is represented under the common A, B, C, D state-space representation:

$$\begin{cases}\dfrac{dx}{dt}=Ax+Bu\\ y=Cx+Du\end{cases}$$

Where:

- u is the vector of the control variables
- x is the vector of the state variables
- y is the vector of the observer variables

From this control representation, it is possible to apply linear algebra algorithms to compute the eigenvalues (that are included in A matrix) representing the system natural modes, the modal shapes of the frequencies (eigenvectors from matrixes A and C) representing the distribution of a frequency all along the system, the transfer functions (linking A, B, C, D matrices) that represent the frequency response in magnitude and phase of an observer variable due to the excitation of another control variable, and the root locus (from A matrix) that represent the evolutions in frequency and damping ratio of the natural modes to some parameters changes in a real/imaginary plot.

For differential algebraic equation (DAE) systems, we first consider a semi-explicit DAE, as obtained by topologically sorting the constitutive equations in LMS Imagine.Lab AMESim:

$$\frac{dx}{dt}=f(x,z,u,t)$$
$$0=g(x,z,u,t)$$
$$y=h(x,z,u,t)$$

x (resp. z) being the differential (resp. algebraic) state variables, in addition to the already defined input and observer variables.

This system may be linearized around an equilibrium point:

$$\frac{dx}{dt}=\nabla_x f\cdot x+\nabla_z f\cdot z+\nabla_u f\cdot u \qquad (1)$$

$$0=\nabla_x g\cdot\frac{dx}{dt}+\nabla_z g\cdot\frac{dz}{dt}+\nabla_u g\cdot\frac{du}{dt} \qquad (2)$$

$$y=\nabla_x h\cdot x+\nabla_z h\cdot z+\nabla_u h\cdot u \qquad (3)$$

If the system index is one, then the partial Jacobian matrix $\nabla_z g$ is non-singular. Taking the Laplace transform of (1)-(3), and introducing the augmented state $\phi=(X(s),Z(s))^T$, the state-space equations are given by:

$$s\phi(s)=A\phi(s)+B_0U(s)+B_1sU(s)$$

with

$$A=\begin{bmatrix}\nabla_x f & \nabla_z f\\ -(\nabla_z g)^{-1}\nabla_x g\nabla_x f & -(\nabla_z g)^{-1}\nabla_x g\nabla_z f\end{bmatrix}$$

$$B_0=\begin{bmatrix}\nabla_u f\\ 0\end{bmatrix}$$

$$B_1=\begin{bmatrix}0\\ -(\nabla_z g)^{-1}(\nabla_x f\nabla_u f+\nabla_u g)\end{bmatrix}$$

As the output of the system is

$$Y(s)=C\phi(s)+D_0U(s)+D_1sU(s)$$

with the following additional matrix definitions

$$C=[\nabla_x h \quad \nabla_z h]$$
$$D_0=\nabla_u h$$
$$D_1=0$$

the transfer matrix of the system is finally given by:

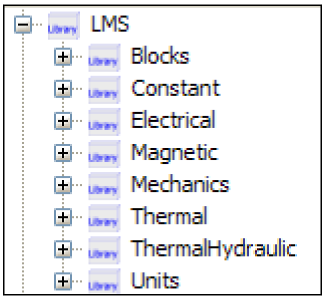
$$Y(s)=\sum_{i=0}^{1}\left(C(sI-A)^{-1}B_i+D_i\right)s^iU(s) \qquad (4)$$

The above reasoning can be extended to higher index systems. In this case, the equation (2) is differentiated up to the index of the DAE. In the resulting linearization process, higher order derivatives of the input variables are retained, and thus the transfer matrix defined by (4) is generalized by extending the summation order up to the system index.

Finally, the linear analysis tools used for differential algebraic equation system are the same as the ones used for ordinary differential equation systems.

3 LMS Modelica libraries overview

The LMS Modelica libraries are an in-house library that presents several packages as shown below for the first hierarchical level:



- LMS
  - Blocks
  - Constant
  - Electrical
  - Magnetic
  - Mechanics
  - Thermal
  - ThermalHydraulic
  - Units

Figure 3: screen shot of some Imagine.Lab libraries in the package browser

The packages are structured in functional sub-packages (Tummescheit [7]). Some of them include quite simple models. The intention there was mainly to support the development of the Modelica compiler embedded in Imagine.Lab AMESim.

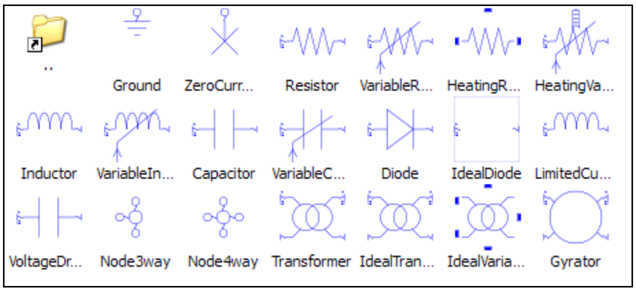


Figure 4: example of Imagine.Lab library for Electrical / Basics components

Other libraries offer more complex models. These LMS Modelica models will be used to illustrate the Linear Analysis approach described in the next sections.

4 Linear analysis in various physical domains

The Linear Analysis approach can be applied to any physical system modeled with Modelica models, whatever are the involved domains of physics (Karnopp [4]).

4.1 Mechanical systems

Let's consider the example of the longitudinal vibrations of a rod in fixed conditions. We sample the continuous system in distributed parameters nodes with 7 masses and 8 stiffnesses with dampers:

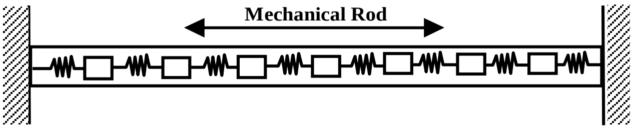


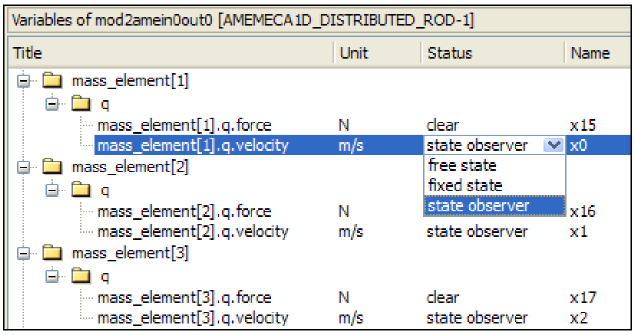
Figure 5: distributed rod in fixed conditions

We consider the Modelica model of the system, built by connecting together the simple Modelica models of mass elements with spring and damper elements:

```
model distributed_rod
  parameter Integer n(fixed=true) = 7;
  LMS.Mechanics.Translational.Mass mass_element[n](each
    m=1.290569);
  LMS.Mechanics.Translational.Spring
    spring_element[n+1](each k=12692085.0);
  LMS.Mechanics.Translational.Damper
    damper_element[n+1](each d=1000.0);
  LMS.Mechanics.Translational.Ground ground1;
  LMS.Mechanics.Translational.Ground ground2;
equation
  connect(ground1.q, spring_element[1].q1);
  connect(ground1.q, damper_element[1].q1);
  for i in 1:n loop
    connect(spring_element[i].q2, mass_element[i].q);
    connect(mass_element[i].q, spring_element[i+1].q1);
    connect(damper_element[i].q2, mass_element[i].q);
    connect(mass_element[i].q, damper_element[i+1].q1);
  end for;
  connect(ground2.q, spring_element[n+1].q2);
  connect(ground2.q, damper_element[n+1].q2);
end distributed_rod;
```

Figure 6: Modelica text for the rod model

To get the modal shapes of the system, we need first to set the mass velocities [m/s] as state observers. This is done through the simulation tool GUI. Note that no modification of the Modelica models within the sketch is needed for accessing the linear analysis settings, neither any specific library with added blocks on the sketch. A selection list is directly available at the tool level, being in the Linear Analysis mode:

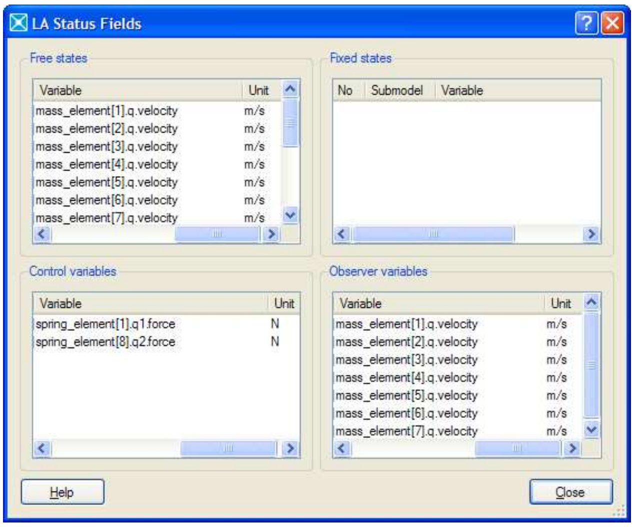


Variables of mod2amein0out0 [AMEMECA1D_DISTRIBUTED_ROD-1]

Title	Unit	Status	Name
mass_element[1]			
q			
mass_element[1].q.force	N	clear	x15
mass_element[1].q.velocity	m/s	state observer	x0
mass_element[2]		free state	
q		fixed state	
mass_element[2].q.force	N	state observer	x16
mass_element[2].q.velocity	m/s	state observer	x1
mass_element[3]			
q			
mass_element[3].q.force	N	clear	x17
mass_element[3].q.velocity	m/s	state observer	x2

Figure 7: selection list for observer variables in Linear Analysis mode

An additional Linear Status window summarizes all the status (observer / control / …) of the variables in the Modelica model:



LA Status Fields

Free states

Variable	Unit
mass_element[1].q.velocity	m/s
mass_element[2].q.velocity	m/s
mass_element[3].q.velocity	m/s
mass_element[4].q.velocity	m/s
mass_element[5].q.velocity	m/s
mass_element[6].q.velocity	m/s
mass_element[7].q.velocity	m/s

Fixed states

No	Submodel	Variable

Control variables

Variable	Unit
spring_element[1].q1.force	N
spring_element[8].q2.force	N

Observer variables

Variable	Unit
mass_element[1].q.velocity	m/s
mass_element[2].q.velocity	m/s
mass_element[3].q.velocity	m/s
mass_element[4].q.velocity	m/s
mass_element[5].q.velocity	m/s
mass_element[6].q.velocity	m/s
mass_element[7].q.velocity	m/s

Figure 8: Linearization Status window

These modal shapes provide the distribution of the natural mode along the different observed parts. It is easy to recognize here the well-known normal modal shapes of the continuous rod system (dashed line):

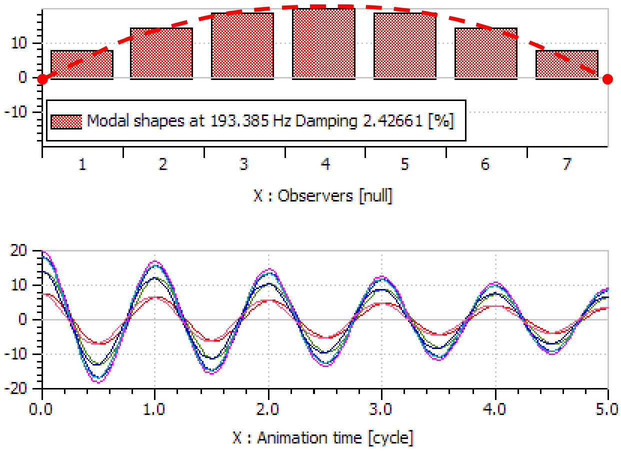


Figure 9: first modal shape (f_1=193 Hz, ζ=2.4%) of a mechanical rod with fixed-fixed boundary conditions

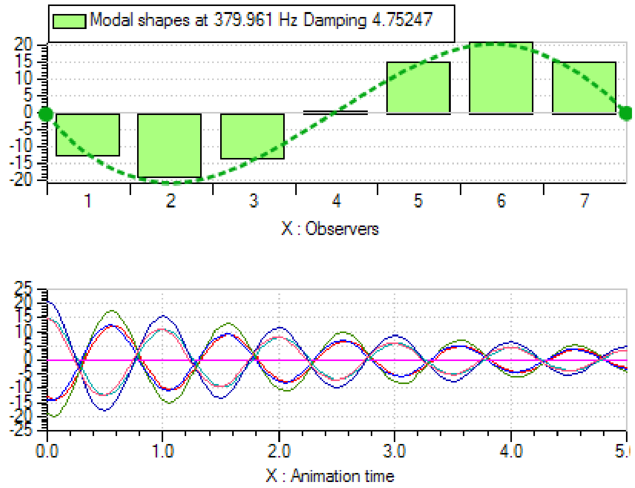


Figure 10: second modal shape (f_2=379 Hz, ζ=4.7%) of a mechanical rod with fixed-fixed boundary conditions

This approach greatly helps the engineer understanding which parts of the systems are involved in the frequencies present on experimentally measured oscillations.

4.2 Thermal systems

We start here from a yet existing Modelica model and we will reuse it to determine the eigenvalues of the system:

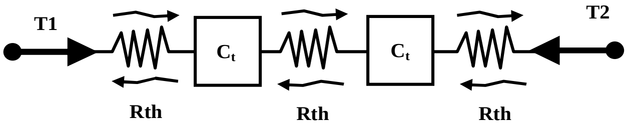


Figure 11: two thermal capacities with conductance

The Modelica model was built as an AMESim supercomponent to be able to mix the Modelica model with non-Modelica (standard C) models.

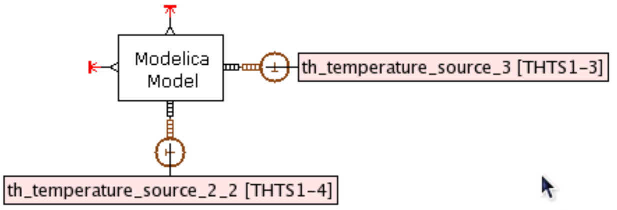

Modelica Model
th_temperature_source_3 [THTS1-3]
th_temperature_source_2_2 [THTS1-4]

Figure 12: simple Modelica thermal model connected to non-Modelica thermal sources and signal

The Modelica text is quite simple, just calling the Modelica components in libraries to connect them together to get the assembled circuit of the thermal system.

The time-response of the temperature in the thermal capacities are the following:

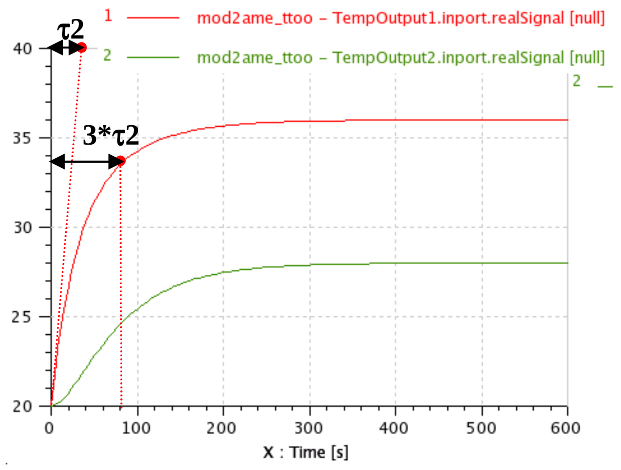
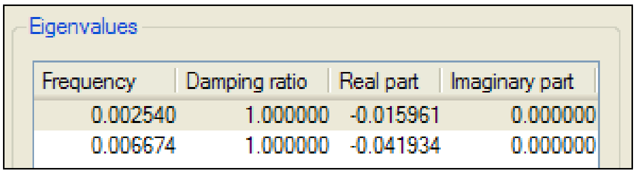


Figure 13: evolution of the temperatures Temp [degree] in the two masses in time-domain

Looking at the eigenvalues shows that there are two time constants τ:


Eigenvalues

Frequency	Damping ratio	Real part	Imaginary part
0.002540	1.000000	-0.015961	0.000000
0.006674	1.000000	-0.041934	0.000000

Figure 14: eigenvalues of the thermal system

Then:

- τ1 = 1/(2π 0.002540) = 62.6 s
- τ2 = 1/(2π 0.006674) = 23.8 s

It is clear that these two time constants τ1 and τ2 directly represent the system dynamics. It is typical to get slow dynamics for thermal systems (range is several [s]).

Alternatively, the analysis of the eigenvalues of this thermal model could help the engineer finding out the best sampling for distributed thermal capacities in his thermal model, in order to reduce the number of states variables, still being sure that the system dynamics are preserved and not altered by the model reduction. This is practically very helpful.

4.3 Electromechanical systems

We now show the interest of eigenvalues analysis for electromechanical systems. We consider there the electrical motor of a tailgate motorized opening system.

An electrical motor is loaded by the rotor inertia with an external torque applied as input:

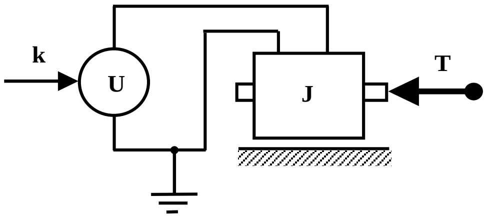


Figure 15: electrical motor with inertia and torque

The Modelica text for the electrical model is presented below:

```
model PermanentMagnetDCMotorWithLoad
  LMS.Electrical.ElectricMotors.PermanentMagnetDCMotor
    motor;
  LMS.Electrical.Sources.SignalVoltage Source;
  LMS.Mechanics.Rotational.Inertia J;
  LMS.Mechanics.Rotational.SignalTorque Tau;
  LMS.Blocks.Sources.ConstantSig torque(k(fixed=false)=0.33);
  LMS.Blocks.Sources.ConstantSig voltage(k(fixed=false)=13);
  LMS.Electrical.Ground g;
equation
   connect(g.p, Source.n);
   connect(g.p, motor.n);
   connect(Source.p, motor.p);
   connect(motor.rotor, J.flange_b);
   connect(J.flange_a, Tau.flange);
   connect(Tau.inputTorque, torque.outport);
   connect(Source.inputVoltage, voltage.outport);
end PermanentMagnetDCMotorWithLoad;
```

Figure 16: Modelica text for the model of electrical motor with inertia and torque

When the U = 13 V voltage source is activated, the inertia starts increasing its shaft speed up to its steady state value. Depending on the inertia J, we can see that the transient behavior completely differs:

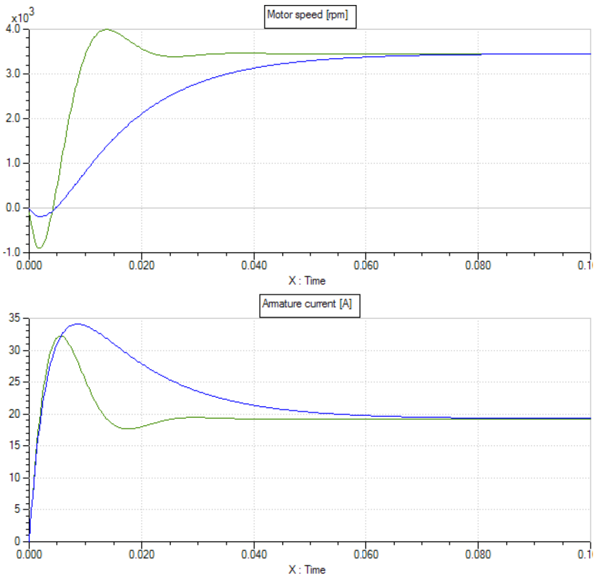
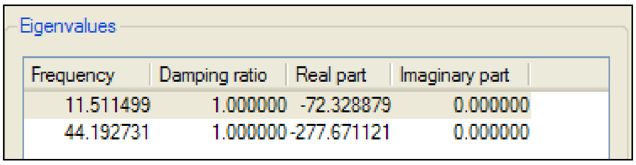


Figure 17: evolution of the motor speed N [rev/min] (top) and the armature current I [A] (bottom) in time-

domain – J=3·10^{-6} kg.m² (green curve) and J=15·10^{-6} kg.m² (blue curve)

It is commonly considered that such a system has an electrical time-response τ1 and a mechanical time-constant τ2. Actually, the eigenvalues reveal that there are two separated time-constants τ1 and τ2:



Eigenvalues

Frequency	Damping ratio	Real part	Imaginary part
11.511499	1.000000	-72.328879	0.000000
44.192731	1.000000	-277.671121	0.000000

Figure 18: eigenvalues of the electrical system - J=15·10^{-6} kg.m²

But these considerations of time-constants being separated are quite abusive since these two time-constants are linked together (it is a second order system). These two time-constants are therefore not uncoupled. For example, considering a change in the moment of inertia from 3·10^{-6} kg.m² to 15·10^{-6} kg.m² would reveal that the dynamic behavior shifts from an oscillatory mode around f = 50 Hz, well damped ζ = 55%, to the two separated time-constants observed previously. The best way to follow this change of dynamics is to plot a Root Locus after making batch runs with modified inertia values J [kg.m²]:

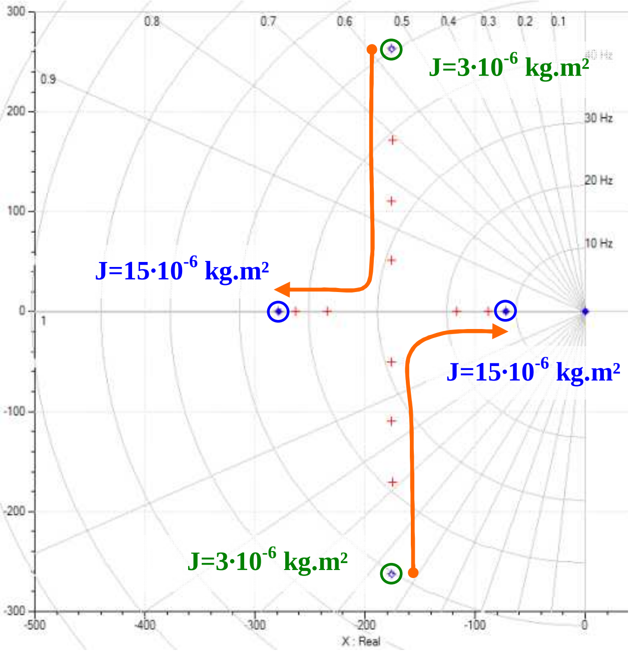


Figure 19: Root-Locus of the electrical motor with inertia - J =3·10^{-6} kg.m² to 15·10^{-6} kg.m²

One has to remember the corresponding time-responses associated to the location of the eigenvalues:

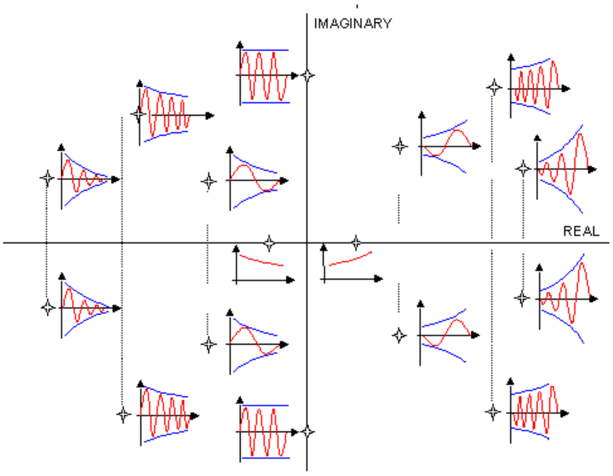


Figure 20: root locus and typical equivalent time-responses

It is finally easy to follow the evolution of the natural modes (frequency f [Hz] and damping ratio ζ [% or null]) to any change of parameters. Root Locus analysis are very appropriate for solving optimal design issues.

5 Case Study: Diesel Common Rail Injection System

Diesel engines need to meet reduced fuel consumption and pollutants emissions. In order to reach these targets, Diesel Common Rail injection systems have been introduced some years ago, with various evolutions of their design and architecture. Below is presented a typical Common Rail injection system with a Bosch CP3 pump that delivers the required amount of fuel to be injected by the injectors, depending on the operating conditions (N_{engine} [rpm], P_{rail} [bar], T_{rail} [°C], …).

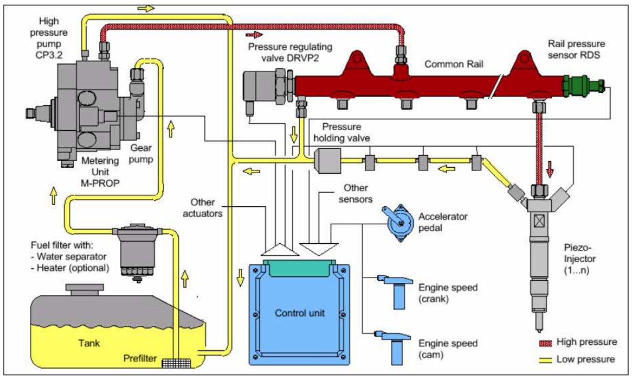


Figure 21: Diesel Common Rail Injection System – Bosch GmbH - Second generation with CP3 pump

Nowadays, the control strategies of injection include multiple-injections to be able to pilot the injected flow rate Q [mm^3/ms] that directly impacts the combustion in the engine cylinder:

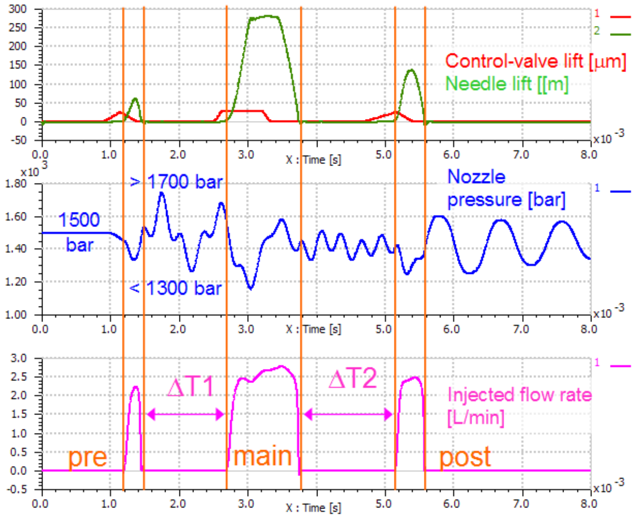


Figure 22: Typical multiple-injection with pre/main/post-injections

The fuel quantities [mg/stroke] to be injected are very small with low deviations admitted from stroke to stroke and from cylinder to cylinder. Therefore, any pressure oscillations as the ones appearing during the injection (typically from 800 Hz to 1000 Hz) have to be understood and controlled, if not damped enough. We propose below to use the Linear Analysis approach on a simple injection system to demonstrate where these frequencies come from. It provides a complementary view to time-domain analysis such as the one proposed by Corno, Casella and All [5] for Gasoline injection system in Modelica.

5.1 Coupling between L4 cylinders

We propose to start with a L4 engine.

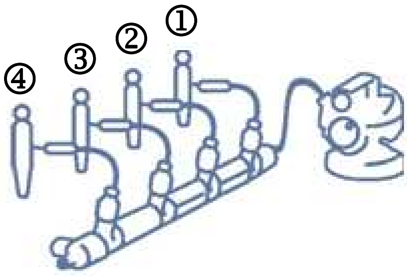


Figure 23: schematics of a L4 injection system: pump with common rail and 4 injectors

The injector is detailed below. Note that an inner pipe is located around the injector needle, which runs from the filter at one end up to the nozzle at the other end. The dimensions of this inner injector line are usually quite similar to the dimensions of the connecting line from the rail to the injector (∅2.4 mm – L= 150 mm to 200 mm):

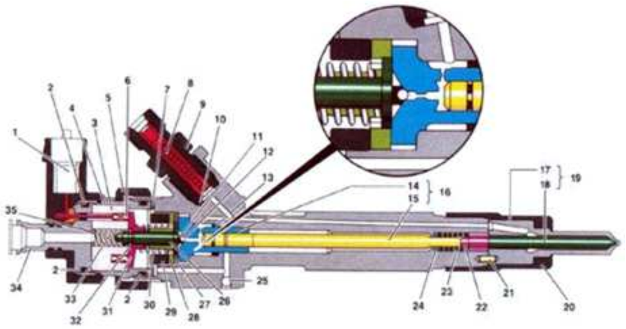


Figure 24: Common Rail Injector– cross-sectional view

The injector ends by the nozzle that delivers the amount of fuel in the cylinders. As a first approach, the nozzle can be modeled as a simple orifice.

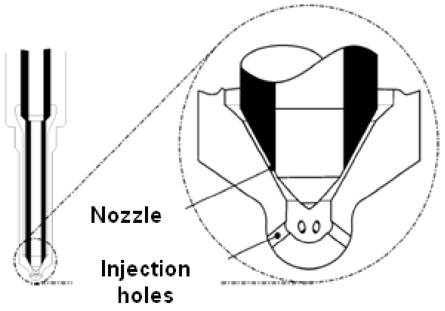


Figure 25: Nozzle with injection holes

The fuel used is a standard ISO4113 diesel. The media properties are based on measurements in the temperature range of +10 to +120°C and pressure range of 0 to 2000 bar:

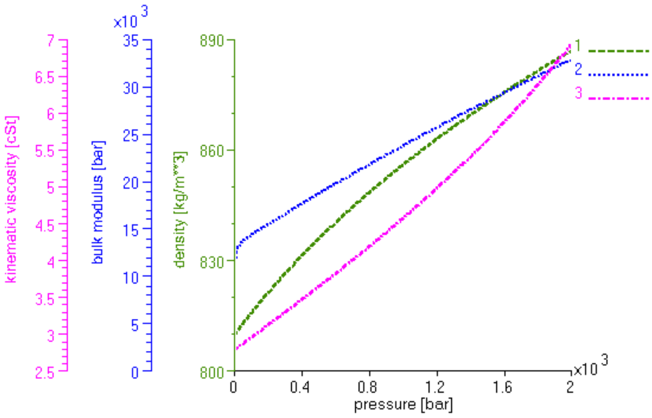


Figure 26 : ISO4113 properties – 0 to 2000 bar – +40°C, from Chaufour and All [6]

For computing the linear state-space representation of the injection system parts in the next section, we will only consider the P_{rail} = 1000 bar and T = +40 °C operating conditions.

ISO4113 media properties @ +40 ℃	Isentropic bulk modulus [bar]	Density [kg/m^3]	Kine-matic viscosity [cSt]
P_{rail} = 0 bar	9 700	810	2.8
P_{rail} =1000 bar	23 700	855	4.6
P_{rail} =2000 bar	32 800	886	6.9

Figure 27 : ISO4113 fuel properties

The L4 injection system for the common rail with 4 injectors sub-system without the pump can be simplified into the following system, the rail being considered as a distributed volume with closed-closed boundary conditions, and the injectors with connecting pipes being represented by their hydraulic inertances and hydraulic stiffnesses:

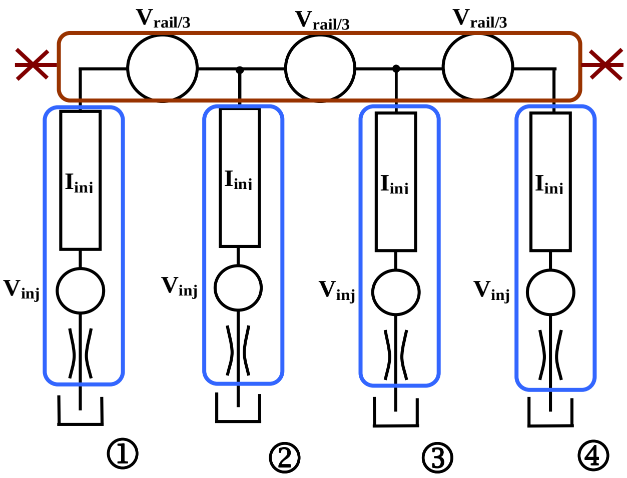


Figure 28: simplified injection system with 4 injectors

The model is based on a prototype LMS thermal-hydraulic library which is still under development. The components are limited to thermal-hydraulic capacity C, resistance R and inertance I, with usual equations. The specific point here is that the ISO4113 Diesel properties are called from the standard Imagine.Lab AMESim Bosch properties with external C-coded functions:

```
function mo_tfrhopti_
    input Real P;
    input Real T;
    input Real fluid_index;
    output Real rho;
    external "C";
end mo_tfrhopti_;
```

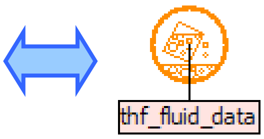
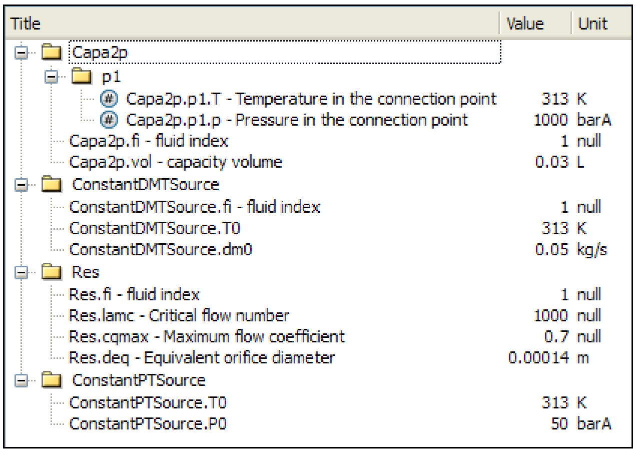


Figure 29: Modelica text for external function call of ISO4113 media properties

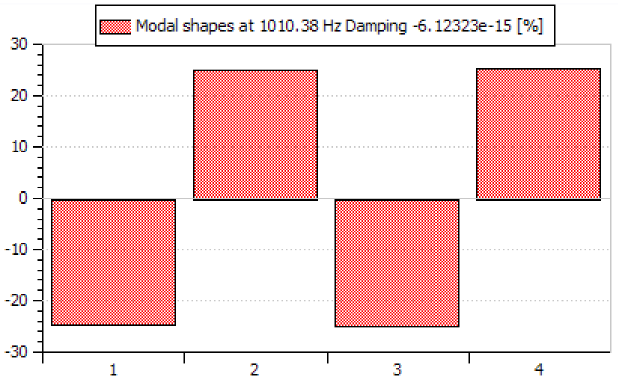
Currently, the library is not finalized then its design could be highly improved in the future.

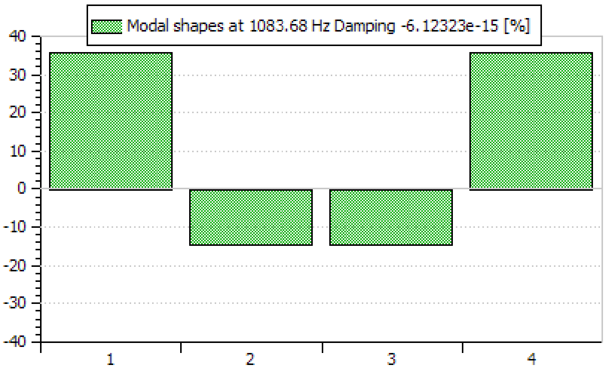


Title	Value	Unit
Capa2p		
p1		
Capa2p.p1.T - Temperature in the connection point	313	K
Capa2p.p1.p - Pressure in the connection point	1000	barA
Capa2p.fi - fluid index	1	null
Capa2p.vol - capacity volume	0.03	L
ConstantDMTSource		
ConstantDMTSource.fi - fluid index	1	null
ConstantDMTSource.T0	313	K
ConstantDMTSource.dm0	0.05	kg/s
Res		
Res.fi - fluid index	1	null
Res.lamc - Critical flow number	1000	null
Res.cqmax - Maximum flow coefficient	0.7	null
Res.deq - Equivalent orifice diameter	0.00014	m
ConstantPTSource		
ConstantPTSource.T0	313	K
ConstantPTSource.P0	50	barA

Figure 30: example of parameters window for a thermal-hydraulic system with the prototype library

This library is used for applying the Linear Analysis tools. For example, the modal shapes are computed for the first 4 natural modes. They highlight that the 4 injectors are actually coupled together, even through the large diameter rail, since the 4 inertances contributions are always combined together in every modal shape. Note also that their frequencies are very close.





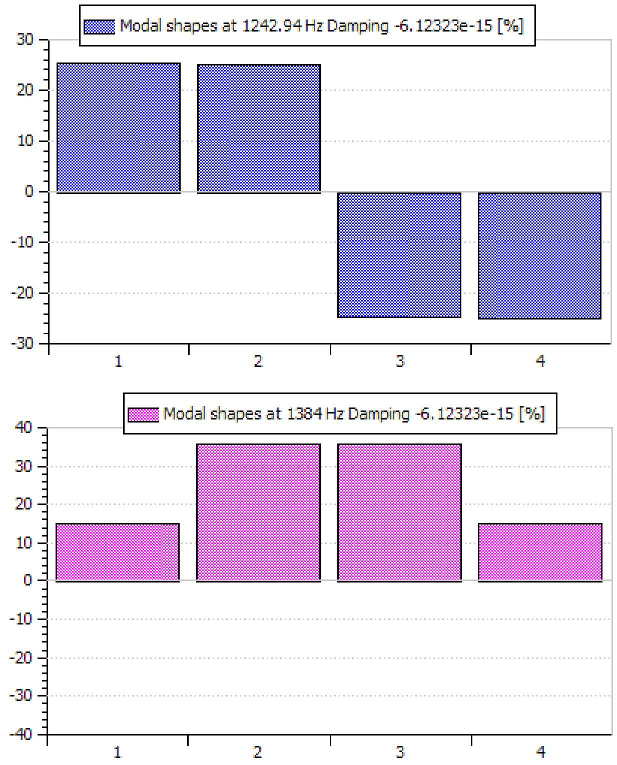


Figure 31: modal shapes of the first 4 modes – simplified injection system with 4 injectors

These values from f_1 = 1010 Hz to f_4 = 1384 Hz are finally close to the oscillations observed in experiments as presented in Figure 22 with a 1 ms period for one oscillation, which corresponds to approximately 1000 Hz.

Additional transfer functions could give more information. They could help determining if the associated gains [dB] of the nozzles pressures are important when an excitation occurs in one cylinder. It is indeed likely that the natural mode the closest to where the injection occurs would be the more excited one.

Finally, the modal shapes help understand how the system dynamics are organized. Some hydraulics/mechanics analogies (Viersma [8]) through the hydraulic stiffness Khyd [Pa/m^3] and the hydraulic inertance Ihyd [kg/m^4] would show that the rail with the 4 injectors and connecting pipes behave as a simple masses-springs system:

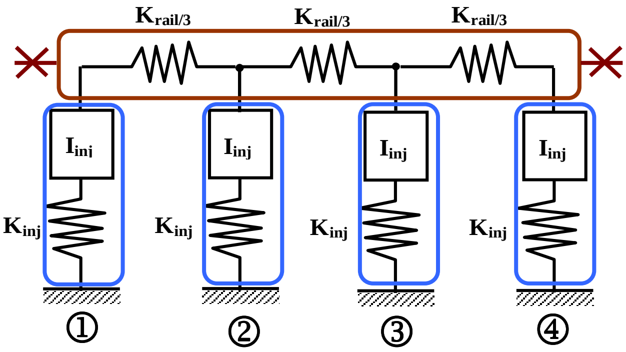


Figure 32: equivalent mass-spring model for common rail with 4 injectors

For the injector with connecting line included, we have:

$$Ihyd_{inj} = \frac{\rho \cdot Length}{Area} = \frac{855 \cdot 0.385}{\frac{\pi}{4}\left(2.4 \cdot 10^{-3}\right)^2} = 7.27 \cdot 10^7 \ kg/m^4$$

$$Khyd_{inj} = \frac{Bulk}{Volume} = \frac{23700 \cdot 10^5}{0.81 \cdot 10^{-6}} = 2.93 \cdot 10^{15} \ Pa/m^3$$

From this equivalent model, we would find directly the observed characteristic frequencies as:

$$f = \frac{1}{2\pi}\sqrt{\frac{Khyd_{inj}}{Ihyd_{inj}}} = \frac{1}{2\pi}\sqrt{\frac{2.93 \cdot 10^{15}}{7.27 \cdot 10^7}} = 1010 \ Hz$$

6 Conclusions

This paper demonstrated how Linear Analysis can be used to understand the dynamic behavior of a Modelica model. The view is independent from the time excitation so that the dynamic answer of such system can be understood for any type of excitations, with very few associated CPU-time. After a brief theoretical overview, the Linear Analysis approach has been applied to a series of different Modelica models to demonstrate that it is valid on different fields of physics, and that it can also be used for non-linear systems with a great efficiency. Next steps would be to highlight the use of such Linear Analysis tools in the context of models reduction (to reduce CPU-time to reach Real-Time performances during the MIL/SIL/HIL process) or for the design of the Control laws of closed loop systems (including Black-Nichols or Nyquist charts for control stability analysis). Alternatively, some works around formal linearization to get the Jacobian with Modelica libraries would bring even more added-value in the linearization process of nonlinear dynamic systems.

References

[1] “Multi-Formalism Modelling and Simulation” – H. Vangheluwe, Thesis, Universiteit Gent, Faculteit Wetenschappen, 2000-2001

[2] “The LinearSystems Library for Continuous and Discrete Control Systems” – M. Otter, Modelica Conference, 2006.

[3] “Mathematics of finite-dimensional control systems: theory and design” - D. L. Russell, Marcel Dekker, 1979.

[4] “System Dynamics – Modeling and Simulation of Mechatronics Systems” - D.C. Karnopp, D.L. Margolis, R.C. Rosenberg, 3rd Edition, John Wiley & Sons Inc., 2000

[5] “Object Oriented Modeling of a Gasoline Direct Injection System” - M. Corno, F. Casella, S. M. Savaresi, R. Scattolini, Modelica Conference, 2008.

[6] “Advanced Modeling of a Heavy-Truck Unit Injector System and its Application in the Engine Design process” - P. Chaufour, G. Millet, M. Hedna, S. Neyrat, E. Botelle – SAE2004-01-0020, 2004

[7] “Design and Implementation of Object-Oriented Model Libraries using Modelica” - H. Tummescheit, Lund, Sweden: PhD thesis, Department of Automatic control, Lund Institute of Technology, 2002.

[8] “Analysis, Synthesis and Design of Hydraulic Servosystems and Pipelines"- T.J. Viersma, Elsevier Scientific Publishing Company, 1980

TrueTime Network — A Network Simulation Library for Modelica

Philip Reuterswärd[a], Johan Åkesson[a,b], Anton Cervin[a], Karl-Erik Årzén[a]
[a]Department of Automatic Control, Lund University, Sweden
[b]Modelon AB, Sweden

Abstract

We present the TrueTime Network library for Modelica, developed within the ITEA2 project EUROSYSLIB. It allows for simulation of various network protocols and is intended for use within real-time networking. We describe some its features and discuss implementational issues. Since TrueTime Network is programmed in C, special attention is given to the how Modelica's external function interface is used. We also discuss briefly a future native Modelica implementation.

Keywords: Modelica; Network Simulation; TrueTime; Real-time Control

1 Introduction

Networked systems and networked control are increasingly common in many domains, e.g., in automotive systems. Sending signals over networks cause delays that, depending on the network protocol, can be more or less deterministic. Some examples of delays are network interface delays, transmission delays, propagation delays, and back-off times in case of collisions on a shared medium. In order to accurately simulate the consequences that these delays have on the overall system performance it is important to be able to model the network at an appropriate level. A too detailed network model including, e.g., the transmission of individual bits, will make the simulation too slow. Furthermore, this level of detail is in most cases unnecessary. A too coarse model, e.g., to model the network as a constant delay, will in many cases fail to capture the dynamics introduced by the network communication.

Within the ITEA2 project EUROSYSLIB, the Department of Automatic Control, Lund University is developing a Modelica network protocol library. The intended application area is real-time networking. In these networks, the upper layers of the ISO/OSI protocol stack are normally not used. Hence the library only models various wired or wireless data-link layer protocols with focus on the MAC-access related sources of delays. The library is based on the Matlab/Simulink toolbox TrueTime [1] developed in the same group.

The Modelica implementation of the TrueTime Network is based on the existing Simulink implementation of TrueTime, with some modifications. TrueTime Network makes it possible to simulate the sending of reals and arrays of reals over a network using different network protocols. It is implemented in C and used in Modelica through the external function interface.

The organization of the paper is as follows. In Section 2 we describe the TrueTime simulation package for Simulink and how it models networks and network protocols. Section 3 gives an introduction to the TrueTime Network library for Modelica from a user's perspective. In Section 4 we discuss the implementational aspects of the TrueTime Network library. Special attention is given to the usage of the external function interface. Section 5 discusses a future native Modelica implementation of the library.

2 TrueTime

TrueTime [1] is a Matlab/Simulink-based simulation tool that has been developed at Lund University since 1999. It provides models of multitasking real-time kernels and local-area wired and wireless networks that can be used in simulation models for networked embedded control systems. The TrueTime Network library is a Modelica port of TrueTime's network part. It supports six simple models of networks — CSMA/CD (Ethernet), CSMA/AMP (CAN), Round Robin (Token Bus), FDMA, TDMA (TTP) and Switched Ethernet.

DOI: 10.3384/ecp09430058

In addition, the wireless network protocols IEEE 802.11b/g (WLAN) and 802.15.4 (ZigBee) are also supported.

TrueTime models networks as a set of FIFO input queues, a shared communication medium, and a set of FIFO output queues. The queues model the send and receive buffers in the nodes connected to the network. A message that should be transmitted from one node to another is placed in one of the input queues. Messages are moved from the input queues, into the network, and into the output queues in an order that depends on the simulated network protocol. A message moves between a number of different queues on its way over the network in a fashion specified by the protocol. The transmission time of each message depends on the length of the message. Collisions and retransmissions are simulated in the relevant protocols. The wireless network models also take the path-loss of the radio signals into account, and as such uses coordinates to specify the locations of the nodes.

Propagation delays are not modeled, since they are typically very small in a local area network. Only packet-level simulation is supported, we assume that higher level protocols have divided long messages into packets.

3 Modelica Library

An overview of the library as shown in Dymola [3] is shown in Figure 1. The TrueTime Network library supports block based modeling with several different networks running in the same simulation. To each implemented protocol there is a corresponding block to allow for graphical modeling. The different network settings can be changed in the block masks. The inputs and outputs of the network blocks are signals that are used to trigger the sending and receiving of network packages.

Additionaly there are blocks for sending and receiving of network packages that are meant to be connected with the network blocks, see Figure 2. Separate blocks exist for the sending and receiving of scalars and arrays. There are also blocks that, given an interval, sample and send a signal periodically over the network. Finally there are blocks representing empty nodes, these should be connected to ground the network in case there are nodes that do not send or receive.

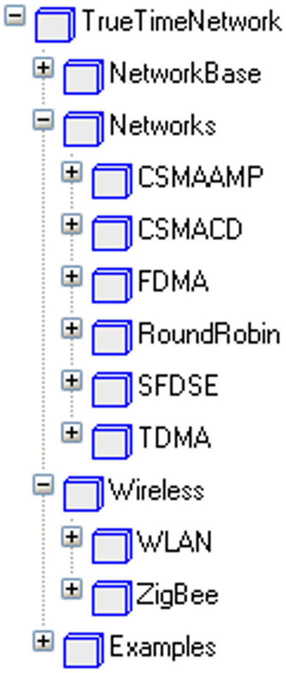


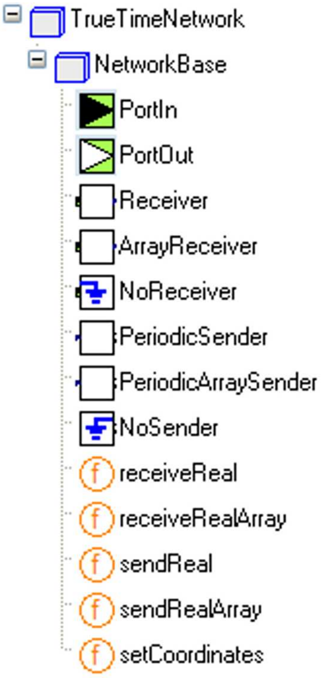
Figure 1: The TrueTime Network package

The network protocols have several settings, see e.g. Figure 3, some common to all and some specific to certain protocols. The network ID is an unique identifier for each network. The number of nodes in the networks must be known at the time of compilation and are specified by the user. The frame size and the speed of the network can also be tuned. The loss probability determines the probability that a message is lost in transit. Lost messages still consume bandwidth but never arrive at their destination. It is possible to set the value of the seed for the random number generator used to calculate if a package is lost or not. Setting of the seed makes it possible to conduct Monte-Carlo type simulations.

The wireless protocols have some additional settings. When simulating a wireless network the position of the nodes must be set. This is done either once at initialization, or continuously throughout the simulation in the case of a moving wireless network node. The transmission power and signal threshold parameters determines how the wireless network signals will be intercepted. There are also settings controlling the sending, resending and timing out of network packages.

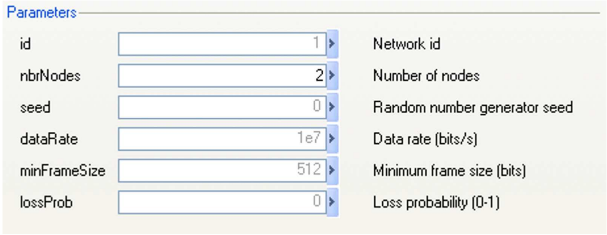
3.1 Example Usage

The library comes with some examples, showing the intended usage of the TrueTime Network library. The examples deals with control loops that



- TrueTimeNetwork
  - NetworkBase
    - PortIn
    - PortOut
    - Receiver
    - ArrayReceiver
    - NoReceiver
    - PeriodicSender
    - PeriodicArraySender
    - NoSender
    - receiveReal
    - receiveRealArray
    - sendReal
    - sendRealArray
    - setCoordinates

Figure 2: Blocks and external functions for sending and receiving network messages



Parameters

id	1	Network id
nbrNodes	2	Number of nodes
seed	0	Random number generator seed
dataRate	1e7	Data rate (bits/s)
minFrameSize	512	Minimum frame size (bits)
lossProb	0	Loss probability (0-1)

Figure 3: Parameter dialogue of the CSMA/CD network block

are closed over networks. Examples show both how to use a wired network, such as Ethernet, and a wireless network protocol, e.g., WLAN. The latter involves setting the positions of the network nodes during the simulation. It is possible to model both using blocks, see Figure 4, and standard Modelica, see Listing 1.

During the simulation it is possible to log the various networks signal and values sent over the network. The netwoks also log the network schedule, which show when packages were sent and when collisions happpened, see Figure 5.

4 Implementation

```
model NetworkExample
  CSMACD_Network
            network(id=1,nbrNodes=2);
  Receiver rcv1(id=1,address=1)
  Receiver rcv2(id=1,address=2)
  PeriodicSender snd1(id=1,address=1)
  PeriodicSender snd2(id=1,address=2)
  ...
equation
  connect(rcv1.portIn,
          network.portOut[1]);
  connect(rcv2.portIn,
          network.portOut[2]);
  connect(snd1.portOut,
          network.portIn[1]);
  connect(snd2.portOut,
          network.portIn[2]);
  ...
end NetworkExample;
```

Listing 1: The network simulation loop

The original TrueTime Simulink blocks are implemented as variable-step S-functions written in C++. Internally, each block contains a pointer to a network structure and a discrete-event simulator. Zero-crossing functions are used to force the solver to produce “major hits” at each internal (scheduled) or external (triggered) event. The events include sending and receiving of messages. Events are communicated between blocks using trigger signals that switch value between 0 and 1. At events the network is run and network packages are moved between the FIFO queues that the network comprises.

The C++ implementation of TrueTime was ported to C, so that it could be used with Modelica through the external function interface [2]. External objects are used to represent networks corresponding to different protocols. Since the external objects do not allow for member functions, auxiliary external functions are used to, e.g., run the network and to send and receive network packages. This hides the implementational details from the user.

Modelica currently does not support external states. This means that once we run the network there is no way to roll back to a previous state. Care must be taken when updating the network, so that we do not run the network in the “future”. This could happen, depending on the implementation, prior to event detection when the integrator tries to step. The Simulink simulator has richer interface to its integrator, which solves the problem in the Simulink environment. In Modelica we accomplish this by careful use of the `when`-construct.

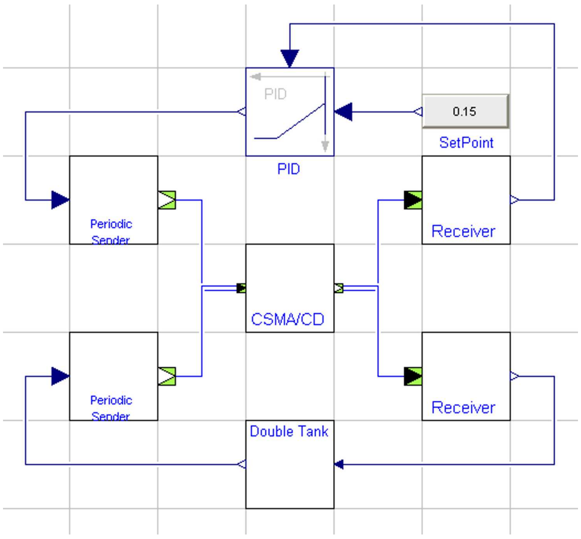
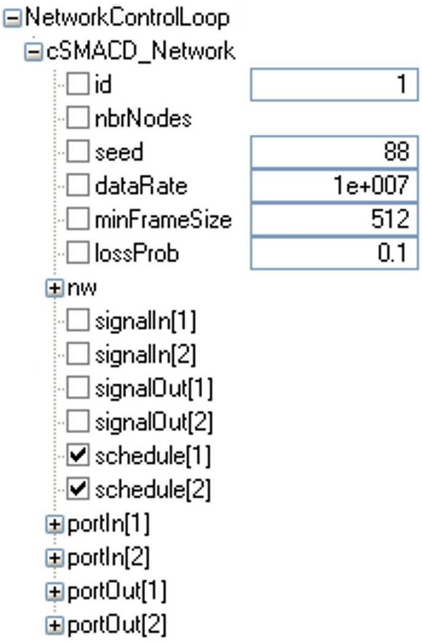


Figure 4: Simulating a PID control loop closed over a network

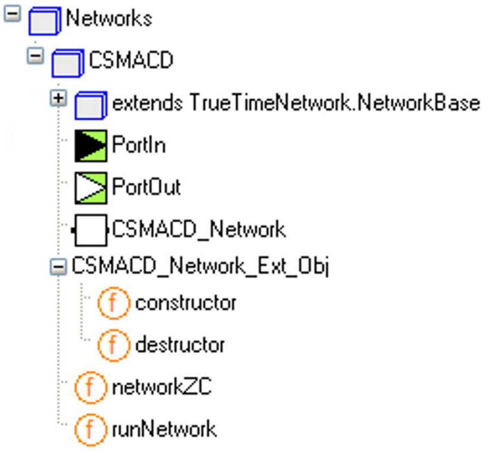
```
RTnetwork* nw;
int nw_id = 1;

nw = getNetworkPtr(nw_id);
```

Listing 2: Retrieving a network structure pointer



```
⊟NetworkControlLoop
  ⊟cSMACD_Network
    □id              1
    □nbrNodes
    □seed            88
    □dataRate        1e+007
    □minFrameSize    512
    □lossProb        0.1
  ⊞nw
    □signalIn[1]
    □signalIn[2]
    □signalOut[1]
    □signalOut[2]
    ☑schedule[1]
    ☑schedule[2]
  ⊞portIn[1]
  ⊞portIn[2]
  ⊞portOut[1]
  ⊞portOut[2]
```

Figure 5: Simulation variables



```
⊟ Networks
  ⊟ CSMACD
    ⊞ extends TrueTimeNetwork.NetworkBase
      PortIn
      PortOut
      CSMACD_Network
    ⊟ CSMACD_Network_Ext_Obj
        constructor
        destructor
      networkZC
      runNetwork
```

Figure 6: Network protocol implementation

4.1 External Network Objects

Figure 6 shows a network protocol model in Modelica. A network is implemented in dymola using an external object representing each network protocol. It points to the external C implementation of the network model. There is also a network wrapper-class that handles the access to the external object. This is done through the functions `networkZC` and `runNetwork`. Other external functions can also access the external network model by specifying the network ID. Externally, a pointer to the network structure can be obtained by doing a lookup on this number, see Listing 2.

To simulate network transmissions TrueTime Network relies on two functions, the zero-crossing function `networkZC` and `runNetwork`, see Listing 3. When a package is sent over the network, the network does not receive the package itself. Instead it reads a boolean signal and triggers on the flanks of it. When an incoming signal is received, signaling the arrival of a new network package, the network is run by calling `runNetwork`. By polling the network using the `networkZC`, we know when the network should be run the next time. If it returns zero, a `when`-clause triggers and the triggering package is either delivered to its destination or moved towards it through the FIFO queues that make up the network. Dropping a package simply means removing it from the network.

4.2 Sending and Receiving

Before triggering a signal on the send port of the network the sending node must create a network package and enqueue it in the external network data structure. When a message is sent, by calling the external function `sendReal`, see Listing 4, a message structure is created and is inserted into a FIFO queue. The network is accessed by doing a lookup using the network id number. All this takes

```
algorithm
  when change(signalIn) then
    (signalOut,schedule) :=
        runNetwork(nw,nbrNodes,time);
  end when;

  nextHit := networkZC(nw, time);
  when (nextHit <= 0.0) then
    (signalOut,schedule) :=
        runNetwork(nw,nbrNodes,time);
  end when;
```

Listing 3: The network simulation loop

```
function sendReal
  input Integer id "Network␣id";
  input Integer sender;
  input Integer receiver;
  input Real u "data";
  ...
  output Real y;
external "C" y = ...
  annotation(Include = ... );
end sendReal;
```

Listing 4: The sendReal external function

place in the external C code. At the same time, on the Modelica side, a boolean variable representing the input is toggled. When this happens the network is run, by calling the `runNetwork` function.

When making its way over the network, a message is transferred between a number of queues. The sending and receiving of messages is event based. How and when it is moved is determined by the protocol model of the network that is to be simulated. To determine when to run the network a zero-crossing function is used. A call to `runNetwork` placed within a `when` construct achieves this. When the network is run, it checks to see if any messages are to be transferred between the FIFO queues. The network also calculates the time of the next hit. This updates the value reported by the zero-crossing function.

When a message is ready to be received, a boolean variable is toggled. This triggers a call to the `receiveReal` function, which retrieves the message from the network. When sending and receiving arrays of data, the user specify at compile time the length of arrays, see Listing 5.

```
function receiveRealArray
  input Integer id "Network␣id";
  input Integer receiver;
  input Integer length;
  output Real[length] y;
external "C" ...
  annotation(Include = ... );
end receiveRealArray;
```

Listing 5: The receiveRealArray external function

5 A Native Implementation

In order to increase the transparency of the protocol implementations the network simulation engine may be implemented in native Modelica, rather than in C. Initial work following this approach was done.

The implementation was largely based on the design of TrueTime. The basic building block for this implementation is the `RingBuffer` class, which emulates a buffer of limited size containing network messages. The network messages in turn are represented by a record class `NWMessage` and subclasses thereof for each individual protocol. The implementation also contains connectors for connecting nodes to the network block, similar to TrueTime. The connectors then contain variables corresponding to the addresses of the sending and the receiving nodes, the actual data. The connectors also carry a boolean variable which is used to signal transmission. When this variable is toggled, the receiving side takes appropriate actions, for example reads the message and store it in an internal buffer. The project is still ongoing.

A particularly interesting extension of this work would be to use ModeGraph to model the state machines of sending and receiving nodes as well as the protocols. In particular since network protocols are often specified in terms of graphical state machine descriptions. Indeed, this approach would further improve the clarity and transparency of the network protocol implementations.

6 Summary

We have presented the TrueTime Network library, developed within the ITEA2 project EUROSYSLIB. The library is implemented using external objects and we have showed key aspects of the implementation related to the external function interface. We have also talked about a future native Modelica implementation, which is being worked on at the time of writing.

References

[1] Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker and Karl-Erik Årzén. *How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime* IEEE Control Systems Magazine 23, 16–30, 2003.

[2] Modelica Association. *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling — Language Specification Version 3.0*, 2007.

[3] Dynasim AB. Dymola - Dynamic Modeling Laboratory. `http://www.dynasim.se`

Design and Implementation of Animation Post-processor Based on ACIS and HOOPS in MWorks

Zhou Fanli[1], Zhang Hehua[2], Zhu Hengwei[2], Gong Xiong[1], Wang Boxing[1], Liu Jun[1], Chen Liping[1], Huang Zhengdong[1]
[1]Huazhong Univ. of Sci.&Tech., CAD Center, Wuhan, China
[2]Suzhou Toprank Software & Control Tech. Co. Ltd, Suzhou, China
{fanli.zhou, zhanghehuahust}@gmail.com, zhuhwei@126.com, {gongx, wangbx, liuj, chenlp}@hustcad.com, zdhuang@hust.edu.cn

Abstract

A complete Modelica-based simulation platform usually consists of modeling tool, compiler, analyzer, solver and post-processor. The 3D animation function is essential to the post-processor of a platform that supports MultiBody system simulation. Taking advantage of the complementarity and interoperability between graphical engines ACIS and HOOPS, MWorks, as a new generation of multi-domain modeling and simulation platform, implements the 3D animation of its post-processor based on these two graphical engines, and provides plentiful animation functions.

This paper firstly presents the overall design of the animation post-processor based on the analysis of visual features of the standard multibody library in Modelica; then describes its implementation, including mechanisms of geometry creation and display, data management and interactive interface; finally, verifies the effectiveness of the post-processor by some typical examples from the multibody library and application to aircraft landing gear simulation.

Keywords: Modelica; Post-processor; 3D animation; ACIS & HOOPS; MWorks

1 Introduction

A Modelica-based simulator usually consists of modeling tool, compiler, analyzer, solver and post-processor. The basic function of post-processor is to display simulation results in curves. If a platform supports multibody system, the 3D animation function is essential to its post-processor. The animation post-processor is used to deal with multibody animation, including geometry creation, graphic rendering, animation control and so on.

The popular graphic engines include PARASOLID, OpenGL, ACIS[1], HOOPS[2], Granite, etc. None of them has complete functions in animation. PARASOLID is good at modeling and visual interaction but has a defect in data management of complex models due to its unclear data structure; OpenGL has powerful graphical display and interaction functions but is short of professional geometric library; ACIS provides plentiful geometry modeling functions but is weak in visual operation and interaction; HOOPS has significant advantages in graphical display, interaction and data structure but is not good at modeling. Therefore, it's difficult to develop a powerful graphic system based on only one graphical engine.

Some simulation platforms provide animation function for multibody systems based on VRML, but this method is not powerful enough due to its defects in graphical quality, kernel interfaces and geometry library. Taking advantage of the complementarity and interoperability between ACIS and HOOPS, MWorks, as a new generation of multi-domain modeling and simulation platforms, implements the 3D animation of its post-processor based on these two graphical engines. MWorks provides plentiful animation functions, which have the advantages of convenient human-computer interaction, good geometric format compatibility, real-time geometric rendering, high fidelity animation effects, powerful model management and high expandability.

2 Design Overview

2.1 Visual Features of Standard MultiBody Library in Modelica

The standard MultiBody library in Modelica 2.2.2 or later consists of packages of World, Examples, Forces, Frames, Interfaces, Joints, Parts, Sensors, Types and Visualizers, as shown in Figure 2.1.

DOI: 10.3384/ecp09430076

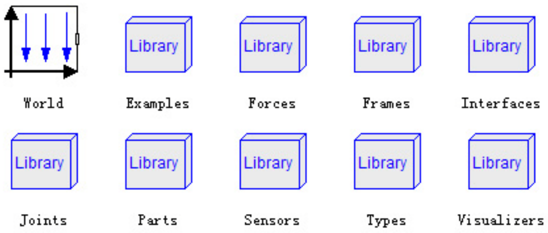


Figure 2.1 Modelica Standard MultiBody Library

Visualizers is the 3D graphic visualization package of the MultiBody library, which includes models of FixedShape, FixedShape2, FixedFrame, FixedArrow, SignalArrow, Advanced.Arrow, Advanced.DoubleArrow, Advanced.Shape, Internal.FixedLines and Internal.Lines. The Advanced.Shape model is the core of the Visualizers package, which gives the information about geometry construction in multibody animation.

The geometry is created according to 7 output variables in Visualizers.Advanced.Shape model, which are Form, rxvisobj[3], ryvisobj[3], rvisobj[3], size[3], Material and Extra. The variable Form represents the shape of multibody part, which may have two types: one is from the eight basic geometric elements in the standard library: box, sphere, cylinder, cone, pipe, beam, gearwheel and spring (see Figure 2.2); the other is from imported geometries defined by external geometric files, which have no unified format. The variables of rxvisobj[3], ryvisobj[3] and rvisobj[3] specify the position of model relative to the world coordinate system. The variable size[3] describes the length, width and height of model as shown in Figure 2.2, in which the dark blue arrow means the length direction and the light blue arrow means the width direction. The variable Material depicts material properties of model including color and specular coefficient. The variable Extra implies additional graphic properties, which have different meanings for different elements, as shown in Table 2.1.

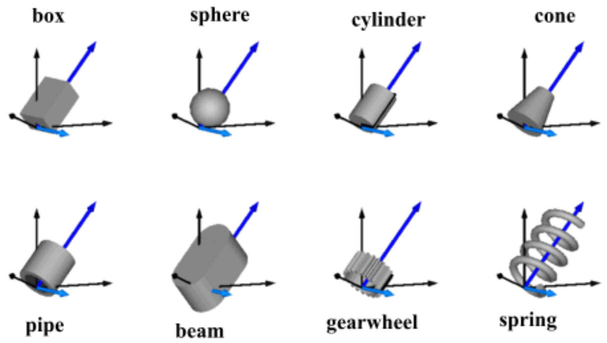


Figure 2.2 Eight Basic Geometric Elements

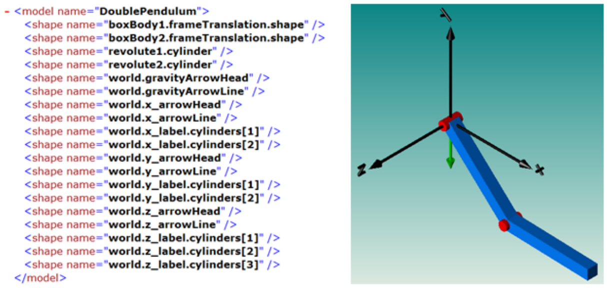
Table 2.1 Meaning of Variable Extra

Shape Type	Meaning of Variable Extra
cylinder	If Extra > 0, a black line is included in the cylinder to show its rotation.

cone	Extra = diameter-left-side / diameter-right-side, i.e;
	Extra = 1: cylinder;
	Extra = 0: “real” cone.

pipe	Extra = outer-diameter / inner-diameter, i.e; Extra = 1: cylinder that is completely hollow; Extra = 0: cylinder without a hole.
gearwheel	Extra is the number of teeth of the gear.
spring	Extra is the number of windings of the spring. Additionally, “height” is not the “height” but 2*coil–width.

The geometry of every multibody part is an assembly of different instances of the Visualizers.Advanced.Shape model. As an example, the instances of Shape in the example model Modelica.Mechanics.Examples.Elementary.DoublePendulum are shown in XML file in Figure 2.3.



```
- <model name="DoublePendulum">
    <shape name="boxBody1.frameTranslation.shape" />
    <shape name="boxBody2.frameTranslation.shape" />
    <shape name="revolute1.cylinder" />
    <shape name="revolute2.cylinder" />
    <shape name="world.gravityArrowHead" />
    <shape name="world.gravityArrowLine" />
    <shape name="world.x_arrowHead" />
    <shape name="world.x_arrowLine" />
    <shape name="world.x_label.cylinders[1]" />
    <shape name="world.x_label.cylinders[2]" />
    <shape name="world.y_arrowHead" />
    <shape name="world.y_arrowLine" />
    <shape name="world.y_label.cylinders[1]" />
    <shape name="world.y_label.cylinders[2]" />
    <shape name="world.z_arrowHead" />
    <shape name="world.z_arrowLine" />
    <shape name="world.z_label.cylinders[1]" />
    <shape name="world.z_label.cylinders[2]" />
    <shape name="world.z_label.cylinders[3]" />
  </model>
```

Figure 2.3 Geometries of DoublePendulum

2.2 Framework

MWorks[5][6] consists of five modules: Modeling environment, Compiler, Analyzer, Solver and Postprocessor. Modeling environment allows users to new a Modelica model by using drag-drop operation or text. Compiler compiles models by running lexical, syntax and semantic checks and generates flat equation systems of models. Analyzer analyzes flat equation systems from Compiler by carrying out structural consistency check, variable substitution, BLT decomposition and high index DAE reduction, and outputs index-1 DAE equation sequences. Solver solves the index-1 DAE equations in order and outputs the file of simulation results. Post-processor reads the result file and displays results in curves or in animation.

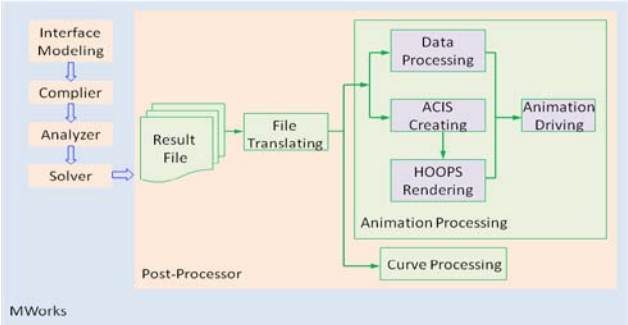


Figure 2.4 Process of MWorks

The process of animation post-processor in MWorks is as follows (see Figure 2.4): Firstly, post-processor reads and parses the result file to generate information for animation including geometric data (shape, position, material), animation data, curve data, etc.; Secondly, the post-processor uses ACIS to create geometries based on geometric data, and then uses HOOPS to render and display 3D geometric models; Thirdly, the post-processor generates transformation matrices of each animation frame based on animation data, which drive model to move; Finally, the post-processor responds to user's operations to begin animation, stop animation, rotate or translate model and so on.

3 Implementation

The key factors of the implementation of the animation post-processor include process of geometry creation and display, performance of data management and convenience of interactive interfaces.

3.1 Geometry Creation and Display

The mechanism of geometry creation and display is the core of animation post-processor, and its design directly affects the performance of the post-processor. The process of creating and displaying geometry in MWorks is shown below (see Figure 3.1):

(1) Create a top geometric model and initialize it;
(2) Check whether all parts have been created, if yes, go to step (6), if no, go to step (3);
(3) Create a part in the top model relative to the world coordinate system;
(4) Create a geometric entity relative to the part coordinate system by the following steps:

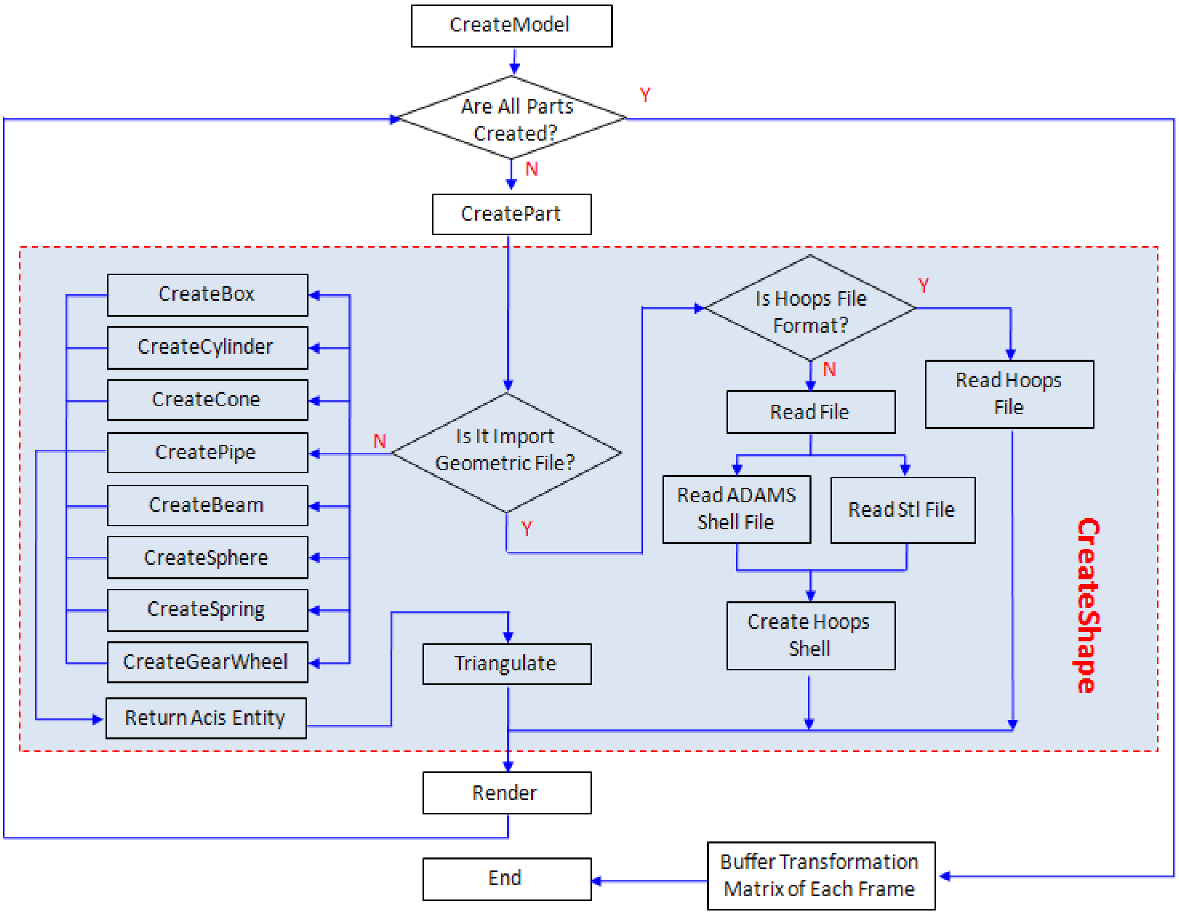


Figure 3.1 Process of Geometry Creation and Display

(i) Check whether the geometry is defined by an externally imported graphic file, if no, that is, the geometry is a standard graphic element, go to step (ii); if yes, go to step (iii);

(ii) Invoke the responding ACIS APIs to create geometric entity according to the element type of the geometry, then triangulate it and generate HOOPS Shell (a collection of polygons that forms a 3D object);

(iii) Check whether the imported file is in HOOPS format, if no, invoke self-defined APIs to parse the file and generate HOOPS Shell; if yes, invoke HOOPS APIs to parse the file and return the geometric object;

(5) Render the geometry, then go to step (2);

(6) Read the result file to generate transform matrices of each animation frame and save them to buffer;

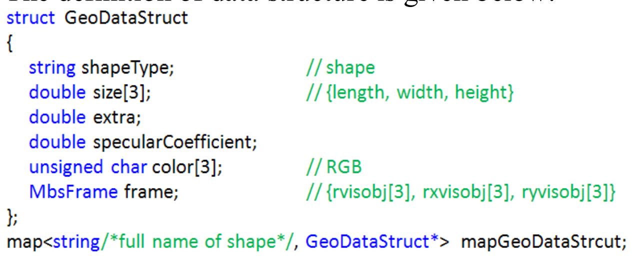
(7) Drive animation of the multibody model.

At present, the post-processor of MWorks can support the following formats: STL file (.stl), HOOPS file (.hsf and .hmf), ADAMS shell file (.shl), etc.

3.2 Data Management

3.2.1 Management of Geometric Data

After reading the result file, the post-processor obtains the data used for creating geometries. In order to improve the efficiency of accessing data, the geometric data of all instances of the Visualizers.Advanced.Shape model are saved in special data structure combining map container and struct pointer. The definition of data structure is given below:

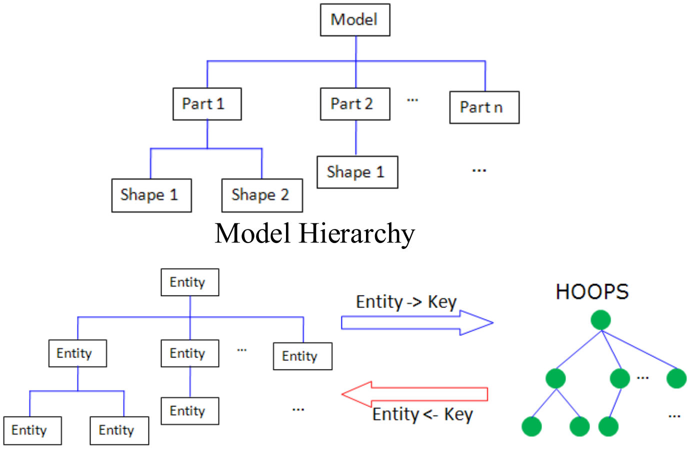


```
struct  GeoDataStruct
{
   string shapeType;                    // shape
   double size[3];                      // {length, width, height}
   double extra;
   double specularCoefficient;
   unsigned char color[3];              // RGB
   MbsFrame frame;                      // {rvisobj[3], rxvisobj[3], ryvisobj[3]}
};
map<string/*full name of shape*/, GeoDataStruct*>  mapGeoDataStrcut;
```

3.2.2 Management of Model Data

The post-processor of MWorks uses tree structure to represent model data. A Model, which represents a multibody model, contains a number of Parts. A Part consists of a number of Shapes, which implies an instance of Visualizers.Advanced.Shape model (see Figure 3.2). Meanwhile, HOOPS uses Segment to describe model data, and one segment maps one HOOPS key. So the key problem in management of model data is how to build the tree structure of model using the HOOPS key.

MWorks uses C++ inheritance mechanism to build the two-way mapping between Entity pointer and HOOPS key by creating Entity class (all of Model, Part and Shape are inherited from Entity). This method can effectively solve the key problem in management of model data. We can use the implementation of highlight picking up as an example: we firstly use mouse to select certain geometric object, and then invoke HOOPS API to obtain the HOOPS key of that object. The corresponding entity pointer of that object can be obtained by the two-way mapping. We finally invoke interfaces to modify the color and transparency of the entity, which indicates that the object is picked up.



Model Hierarchy

Two-Way Mapping between Entity and HOOPS Key

Figure 3.2 Structures of Model Data

3.2.3 Management of Animation Data

The 3D animation can be viewed as display of a sequence of picture frames. The position of each part of model has been changed once after each picture frame is displayed, which can be represented by a 4*4 matrix, namely transformation matrix. In order to enhance the efficiency of reading and writing animation data, we adopt the consecutive memory storage method (see Figure 3.3). This method stores the data of the same type in a continuous memory area, so that the data can be easily accessed through its first address and block length.

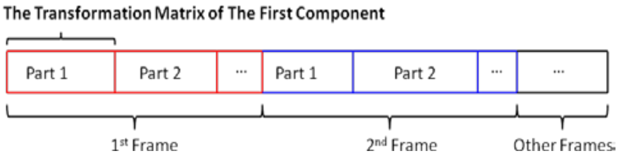


Figure 3.3 Physical Structure of Animation Data

3.3 Interactive Interfaces

The animation post-processor of MWorks uses MFC multiple document/view framework, which allows the user to open a number of relatively independent animation windows at the same time. The animation interface menu provides four kinds of functions (see Figure 3.4): (1) view splitting function to allow users to view animation in different split views from different perspectives; (2) model operation function to allow users to rotate, translate, zoom and highlight pick up model and to change the display mode, which can be wire-frame mode, hidden mode, perspective mode and shadow mode; (3) view operation function to allow users to change the observing view, which can be front view, rear view, left view, right view, upward view, downward view or axonometric view, etc.; (4) animation control function including the operations of starting animation, suspending animation, reversing animation, resetting animation, adjusting animation speed or recording animation video.

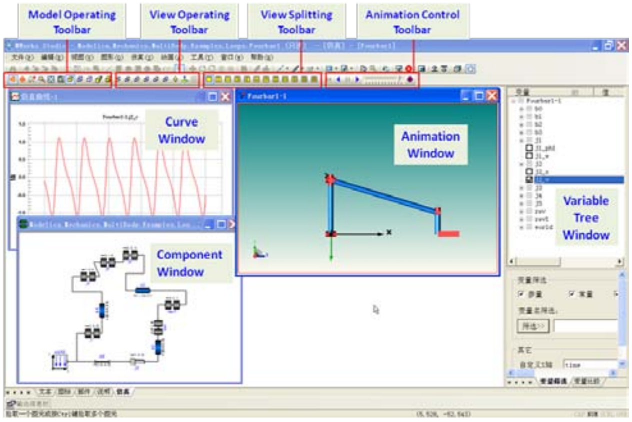


Figure 3.4 Animation Interfaces of MWorks

4 Examples

ACIS and HOOPS-based animation post-processor of MWorks has been successfully applied to simulation of MultiBody system based on Modelica.

4.1 Examples from Standard MultiBody Library

Take model Modelica.Mechanics.MultiBody.Examples.Loops.EngineV6 as an example. The results are shown in Figure 4.1 after the model is compiled, analyzed and solved. Figure 4.2 shows the results of another example of Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot. In post-processor window, the left is its axonometric view in shadow mode, the upper-right is its front view in hidden mode and the bottom-right is its default view.

4.2 Application in Aircraft Field

Cooperating with Commercial Aircraft Corporation of China, Ltd., MWorks accomplished the simulation of aircraft landing gear under various working conditions. (See Figure 4.3)

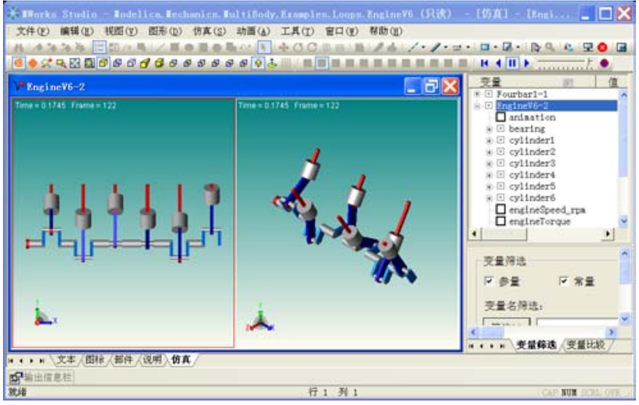


Figure 4.1 Animation of Example EngineV6

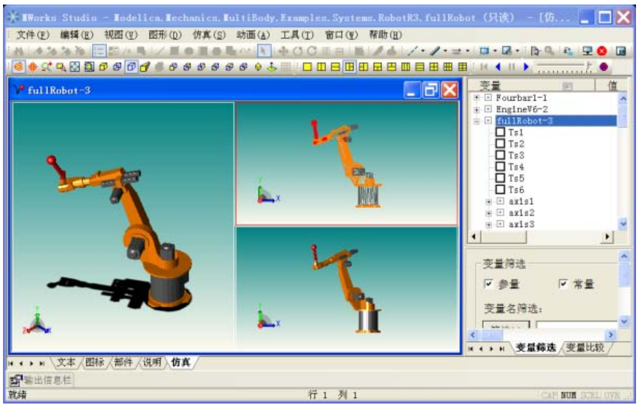


Figure 4.2 Animation of Example fullRobot

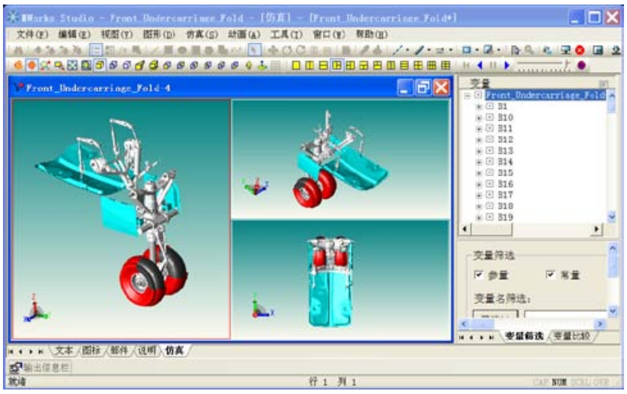


Figure 4.3 Animation of Aircraft Landing Gear

5 Conclusions

Based on ACIS and HOOPS, MWorks platform implements an animation post-processor for multibody systems. It has the advantages of convenient human-computer interaction, real-time geometric rendering, high fidelity animation effects, powerful model data management and good expandability, and has been successfully applied to some practical projects. The further work of animation post-processor of MWorks is to support flexible multibody animation, which has two tasks: (1) providing interface for common finite element software such as ANSYS and ABACUS; (2) supporting the animation of flexible body in flexible multibody library in Modelica.

Acknowledgments

The paper was supported by National Nature Science Foundation of China (No.60874064, No.60704019), National Science & Technology Major Project of China (No.2009ZX040001-015).

References

[1] ACIS online help. Spatial Technology Inc. http://www.spatial.com.

[2] HOOPS 3D Application Framework. HOOPS online help. Tech Soft American Inc. http://www.hoops3d.com.

[3] Zan Wang, Chao Xu, Xiang Xue. The Visualization Interaction Between ACIS and HOOPS. Group Technology & Production Modernization 2006, 1(23): 49 – 51.

[4] Hong-Wei Dong, Ru-Rong Zhou, Lai-Shui zhou. Developing 3D Application Software Based on ACIS. COMPUTER AIDED ENGINEERING 2002, 12 (4): 53 – 58.

[5] Ding Jianwan, Chen Liping, Zhou Fanli. A Component-based Debugging Approach for Detecting Structural Inconsistencies in Declarative Equation based Models. Journal of Computer Science & Technology, 2006, 21(3): 450-458

[6] FAN-LI Zhou, LI-PING Chen, YI-ZHONG Wu, JIAN-WAN Ding, JIAN-JUN Zhao, YUN-QING Zhang. MWorks: a Modern IDE for Modeling and Simulation of Multidomain Physical Systems Based on Modelica. Modelica 2006, September 4th – 5th: 725-732.

[7] Bo-Xing Wang, Bo Wang, Yun-Qing Zhang. Model Management in Complicated Mechanical System Simulation Platform. Journal of Computer-Aided Design & Computer Graphics, 2004, 16(4): 820 – 825.

Implementation of a Modelica Library for Smooth Spline Approximation

Jörg Ungethüm Dirk Hülsebusch
German Aerospace Center, Institute of Vehicle Concepts
Pfaffenwaldring 38-40, 70569 Stuttgart
joerg.ungethuem@dlr.de dirk.huelsebusch@dlr.de

Abstract

In system modeling, table data interpolation is frequently used for e.g. characteristic maps or as replacement of a complicated system. In many cases, the available data are noisy and of limited accuracy. It turns out, that data approximation is advantageous against data interpolation in these cases. A Modelica library for 1-D and 2-D spline approximation on basis of external functions is presented in this paper.

Keywords: interpolation, approximation, polynomial splines, data tables

1 Introduction

The Modelica standard library provides table objects for 1-D and 2-D data interpolation. However, there are several shortcomings of these implementations:

- Table data with of more than 2 dimensions is not supported (extension up to 4 dimensions is presented in [1])
- Only linear and 3rd order spline interpolation but not data approximation is supported
- Step functions cannot be modeled by standard tables as grids are required to be strictly increasing
- 2-D table data must be given on a rectangular grid. As measurement data is commonly available only as scattered data, the actual interpolation table must be generated by some kind of interpolation which introduces new inaccuracy.
- In case of inputs out of the table data range, data is extrapolated using the nearest 2 data points. In many real world problems this type of data extrapolation is not suitable.

2 Approximation vs. Interpolation

In system modeling, table data are often used for measured data or as a replacement for a complex external calculation (e.g. media properties). In many cases the table data are prone to random and discretization errors. In Figure 1 a simple example of 1-D measured data is given. The discretization of the measurement device is clearly seen in the sampled data (ΔU = 0.01V). Linear interpolation of these data leads to a partially stepwise constant curve, which obviously indicates zero gradients. Assuming a similar curve as a table based function definition this might introduce numerical problems. Unfortunately also cubic spline interpolation is not appropriate as the gradient of the spline is partially opposite to the gradient of the linear interpolation. It should be noted, that the spline interpolation introduces numerous local extrema which are not present in the original data. Again, this might lead to numerical trouble.

To overcome these difficulties, function approximation is occasionally used (e.g. in most media models). Unfortunately, finding suitable trial functions is

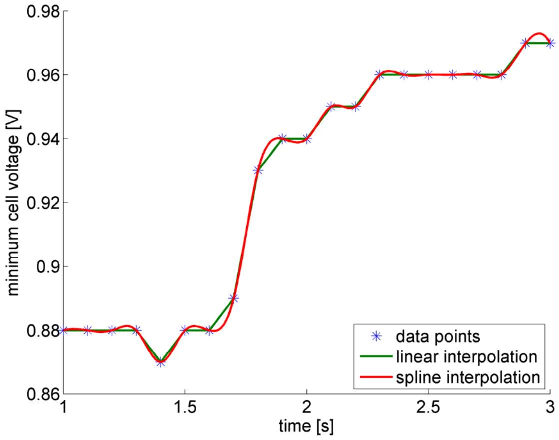


Figure 1 Linear and cubic spline interpolation of sample measured data points

not trivial at all. Only in very simple cases polynomials might be used.

Another approach is to use polynomial splines for piecewise approximation. That is, the knots of the generated splines are not fixed at the data points but selected by the algorithm. By means of the smoothing parameter s the smoothness of the approximation is controlled. Setting s to zero, an interpolating spline is constructed. Moving over to greater values of s, the closeness is relaxed for the benefit of smoothness. The smoother the approximation is the lesser knots are needed. Cubic spline approximation in 1D using different values for s is shown in Figure 2.

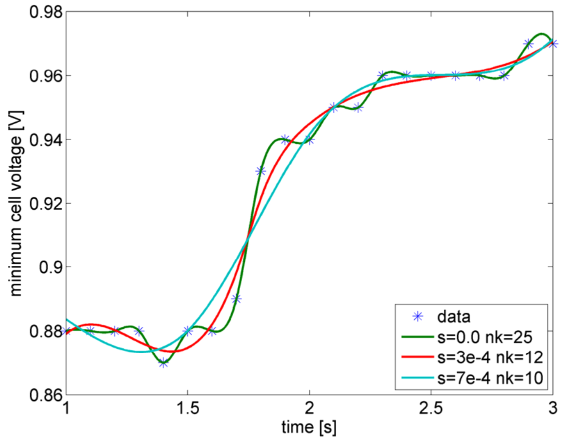


Figure 2 Cubic spline approximation of sample measured data points

3 Modelica implementation

The core routines for curve and surface fitting by spline approximation are public available in the library DIERCKX at Netlib [2]. Splines of order 1 to 5 are supported. DIERCKX routines are written in standard FORTRAN 77. To make them usable in the Modelica environment, only an interface had to be implemented. The DIERCKX library was translated into Ansi-C by f2c (also available from Netlib) to simplify mixed language programming.

The Modelica implementation uses the ExternalObject approach. Thus, splines are objects with constructor and destructor functions called automatically exactly once before the first respectively last use of the object. The internal representation of the splines as Ansi-C structures is completely hidden for the user. The spline generation involving dynamic memory allocation is done only during object initialization. Actual data interpolation is fast and does not involve memory allocation. As a nice feature, DIERCKX contains routines for derivative calculation. The first order derivatives are also available as Modelica functions. The polynomial order of the splines can be chosen between 1 and 5 whereas even numbers are strongly discouraged. Each data point might be weighted individually. The total number of spline objects is not limited nor is the size of data. For 1-D curve data fitting of periodic curves is supported. The data for surface fitting might be supplied on a rectangular grid in the same format that is used in the standard interpolation tables. However, also scattered input data are supported. In common, number and location of knots are chosen by the algorithm but this might be overridden by a user-specified knot selection.

4 Interfaces

The interface is designed with ease of use in mind. Thus, for any inputs reasonable defaults are supplied.

The basic usage of 1-D curve is:

```
/* spline initialization */
Import C=ApproxSpline.Curve1d;
C.Type curve=C.Type(data=data, s=0.01);
…
equation
/* spline data evaluation at x → y*/
y = C.eval(curve,x);
```

Any input arguments of the initialization routine except of the table data are optional. The inputs are:

data[:,2-3]	Data to be approximated as row wise triples: x,y,w (w is optional weight)
s = 0	Approximation smoothness
k = 3	Polynomial spline order (1..5)
periodic = false	Generate periodic spline if true
x_lim[2] = {min(data[:,1]), max(data[:,1])}	Lower and upper limits of the generated spline curve
t = fill(0,0)	if non-zero length, t is interpreted as user-specified knot selection

The order of the input data array does not matter as it is internally sorted. However, abscissa elements must be mutually unequal. The approximation smoothness has to be greater or equal zero. The generated spline approximation p(x) is found by minimizing the discontinuity of the k^{th} derivation whereas the condition

$$\sum_{i=1}^{n} w_i \left(y_i - p(x_i)\right)^2 \le s$$

is still fulfilled.

Per default, the limits of a non-periodic spline curve are set according to the minimum and maximum data value. However, the limits might be modified by parameter $x_{\lim}$. This might be used e.g. for extrapolation purpose. In any case, outside of the limits the spline evaluation will return the boundary coordinate.

$$p(x) = p(x_{\max}) \quad \big| \, x \ge x_{\max}$$
$$p(x) = p(x_{\min}) \quad \big| \, x \le x_{\min}$$

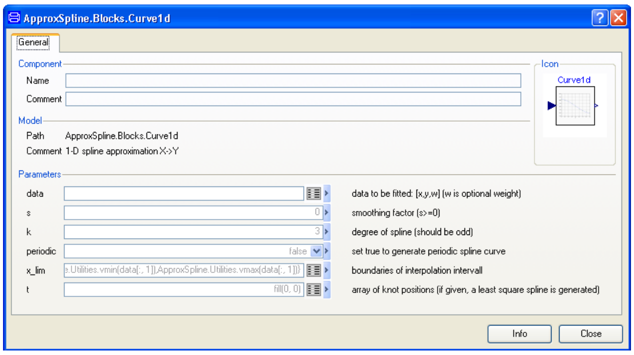
In case of periodic splines, the period is given by

$$\Delta = \max(x) - \min(x)$$

The y-coordinate of $x_{\max}$ as well as any given limits are ignored in this case.

If automatic knot selection is not sufficient, user might specify knots as parameter array.

For convenience, a block interface is provided. The block dialog is shown in Figure 3.


ApproxSpline.Blocks.Curve1d

General

Component

Name

Comment

Model

Path ApproxSpline.Blocks.Curve1d

Comment 1-D spline approximation X->Y

Parameters

data — data to be fitted: [x,y,w] (w is optional weight)

s 0 — smoothing factor (s>=0)

k 3 — degree of spline (should be odd)

periodic false — set true to generate periodic spline curve

x_lim e.Utilities.vmin(data[:, 1]),ApproxSpline.Utilities.vmax(data[:, 1])} — boundaries of interpolation intervall

t fill(0, 0) — array of knot positions (if given, a least square spline is generated)

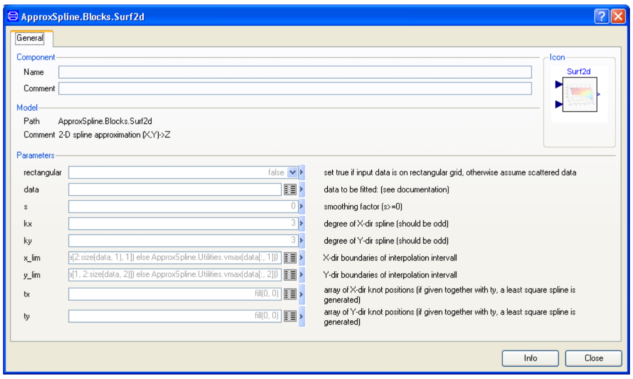
Figure 3 Parameter dialog of 1-D curve block

The interface of the 2-D surface is very similar to the 1-D case.

```
/* spline initialization */
Import S=ApproxSpline.Surf2d;

S.Type surf=S.Type(data=data, s=0.01)
…
equation
/* spline data evaluation [x,y] → y*/
y = S.eval(surf,x,y);
```

Any input arguments of the initialization routine except of the table data are optional. The inputs are:

rectangular=false	true if input data is on rectangular grid, otherwise assume scattered data
data[:,:]	Data to be approximated (either scattered or on rectangular grid)
s = 0	Approximation smoothness
kx = 3	Polynomial order in X-dir (1..5)
ky = 3	Polynomial order in Y-dir (1..5)
x_lim[2] = {min(x), max(x)}	Lower and upper limits in X-dir of the generated spline surface
y_lim[2] = {min(y), max(y)}	Lower and upper limits in Y-dir of the generated spline surface
tx = fill(0,0)	if non-zero length, tx is interpreted as user-specified X-dir knot selection
ty = fill(0,0)	if non-zero length, tx is interpreted as user-specified Y-dir knot selection


ApproxSpline.Blocks.Surf2d

General

Component

Name

Comment

Model

Path ApproxSpline.Blocks.Surf2d

Comment 2-D spline approximation (X,Y)->Z

Parameters

rectangular false — set true if input data is on rectangular grid, otherwise assume scattered data

data — data to be fitted: (see documentation)

s 0 — smoothing factor (s>=0)

kx 3 — degree of X-dir spline (should be odd)

ky 3 — degree of Y-dir spline (should be odd)

x_lim a[2:size(data, 1), 1]) else ApproxSpline.Utilities.vmax(data[:, 1])} — X-dir boundaries of interpolation intervall

y_lim a[1, 2:size(data, 2)]) else ApproxSpline.Utilities.vmax(data[:, 2])} — Y-dir boundaries of interpolation intervall

tx fill(0, 0) — array of X-dir knot positions (if given together with ty, a least square spline is generated)

ty fill(0, 0) — array of Y-dir knot positions (if given together with ty, a least square spline is generated)

Figure 4 Parameter dialog of 2-D surface block

The input data might be provided on a rectangular grid or as scattered data. In the first case, the input format is similar to the standard Modelica interpolation table. In case of scattered input data, a simple table with 3 or 4 columns has to be provided as shown in Figure 5 whereas the 4th column is the optional weight. For rectangular data it is not possible to provide individual data point weights.

$$\begin{array}{c|cccc} 0 & y_1 & y_2 & \cdots & y_{ny} \\ \hline x_1 & z_{1,1} & z_{1,2} & \cdots & z_{1,ny} \\ x_2 & z_{2,1} & z_{2,2} & \cdots & z_{2,ny} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{nx} & z_{nx,1} & z_{nx,2} & z_{1,1} & z_{nx,ny} \end{array} \qquad \begin{array}{c|c|c|c} x & y & z & w \\ \hline x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & w_n \end{array}$$

Figure 5 Input data schemes: rectangular grid (left) , scattered data (right)

5 Examples

As a typical example, a table based model of a turbo compressor is used. The model of the turbo compressor is built up from the equations:

The dimensionless isentropic compression work:

$$Y = \int_{p_1}^{p_2} v\,dp = RT_1 \frac{\kappa}{\kappa - 1}\left(\left(\frac{p_2}{p_1}\right)^{\frac{\kappa-1}{\kappa}} - 1\right)$$

By the use of dimensionless characteristic numbers the model becomes independent of variable inlet temperature and pressure. The most common characteristic numbers for turbo compressors are the head coefficient ψ and the flow coefficient φ

$$\psi = \frac{Y}{u^2/2} \qquad \varphi = \frac{\dot{V}}{\frac{\pi}{4}D^2 u}$$

whereas u is the wheel tip speed, D is the wheel diameter and $\dot{V}$ is the gas volume flow rate. The last characteristic number used is the tip speed Mach number, which is the ratio of tip speed and the sonic speed of gas.

$$Ma_u = \frac{u}{a}$$

The core of the compressor model are two 2-D lookup tables which maps head coefficient and tip speed Mach number to flow coefficient and inner efficiency.

$$\varphi = table_\varphi(\psi, Ma_u)$$
$$\eta_i = table_\eta(\psi, Ma_u)$$

In case of connecting both fluid ports with control volumes and the rotational flange with a rotating mass, the table inputs can be calculated directly from dynamic states. The mass flow rate is subsequently calculated from the flow coefficient.

$$\dot{V} = \varphi \max(u,0)\frac{\pi D^2}{4}$$

$$\dot{m} = \frac{\dot{V}}{\rho_1}$$

The shaft power might be computed by means of the isentropic and the mechanic efficiency from mass flow rate and isentropic compression work. However, to avoid a singularity at zero speed, it is advantageous to calculate the shaft torque without introducing the mechanic power.

$$\tau = \varphi \frac{\pi}{8}\frac{Y}{\eta_i}\rho_1 i_G D^3$$

The model has been implemented both with standard Modelica table interpolation as well as with approximating spline approximation. Table data were taken from a commercial automotive air compressor. There are 56 data points available. From these scattered data, the gridded data as needed for Modelica standard tables is generated by interpolation. As an example, the interpolated data using a 20x20 grid is plotted in Figure 6. The corresponding 3rd order approximating spline surface is shown in Figure 7.

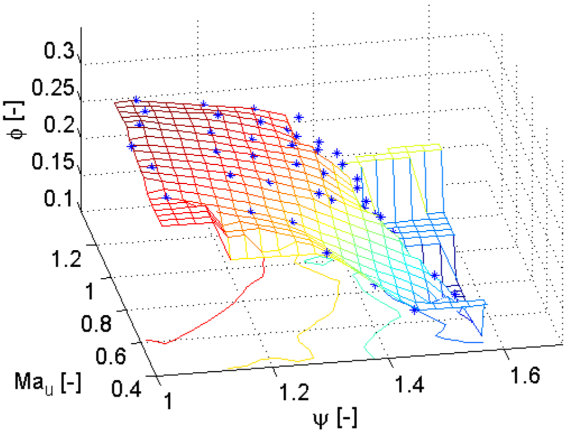


Figure 6 Flow coefficient as function of tip speed Mach number and head coefficient (mesh by interpolation, original data points as stars)

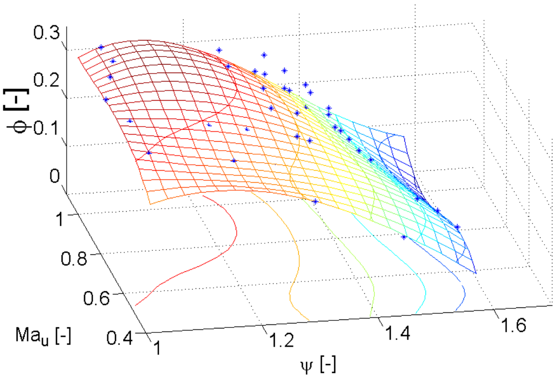


Figure 7 Flow coefficient as function of tip speed Mach number and head coefficient (mesh by spline approximation, original data points as stars)

The compressor model has been included into a simple test model which is shown in Figure 8. Both fluid ports of the compressor model are connected to control volumes. The left control volume is fed through a pipe from a boundary model. The right control volume blows off through an ideal nozzle. The compressor shaft is connected to a rotating mass which is driven by a speed-controlled torque. Starting with zero compressor speed, the torque accelerates the shaft up to the given rotating speed set value. Simultaneously the volume flow rate through the compressor and the pressure in the right control volume rise. The test model was run with both compressor implementations. The comparison of flow coefficient and inner efficiency which are calculated by table lookup respectively spline approximation are shown in Figure 9 and Figure 10. Obviously the spline approximation gives a smoother curve which in turn leads to slightly better simulation performance (Table 1)[1].

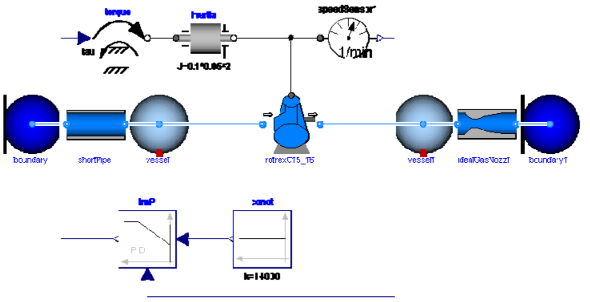


Figure 8 Compressor test model setup

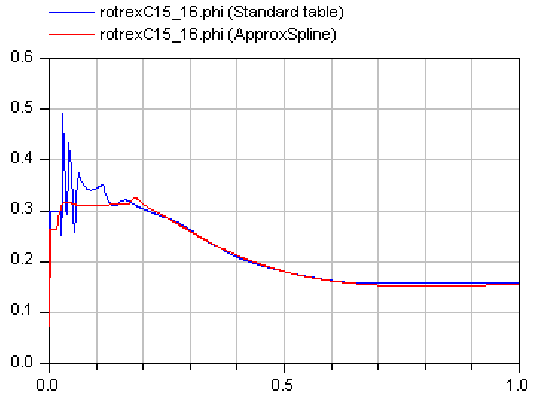


Figure 9 Comparison of flow coefficient

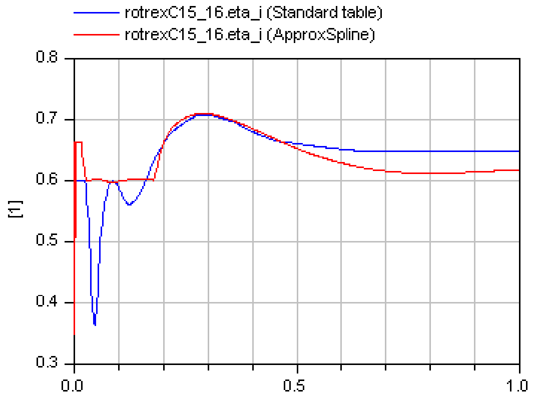


Figure 10 Comparison of inner efficiency

Table 1 Comparison of test model performance

	Standard	ApproxSpline	Diff
CPU-time [s]	0.094	0.079	-16%
F-evaluations	1399	885	-37%
H-evaluations	836	713	-15%
Jacobian-eval.	108	80	-26%

To make the equation system more difficult to solve, another pipe is introduced in the model between the compressor and the right control volume (Figure 11). In this modified model the pressure ratio of the compressor cannot be calculated from system states any more. As the pressure drop of the pipe depends on its mass flow rate, the pressure ratio of compressor in turn depends on the mass flow rate. As a result, a nonlinear equation system of dimension 4 after reduction which includes the table lookup procedure is generated. The simulation of the model using Modelica standard tables fails after approx. 0.1s simulation time due to a Newton solver failure. Simulation of the model using approximating splines works fine. To get the Model with the standard tables working, a modification of the table data is necessary. If an input signal of standard tables is outside of the defined interval, the corresponding value is determined by extrapolation through the last or first two points of the table. In several cases, this leads to nonsensical results, e.g. efficiency less than zero. By adding another 2 rows and columns to the table data, the extrapolation can be forced to return the last or first point of the table. Doing this makes the modified test model solvable even with Modelica standard tables. The comparison of the performance is shown in Table 2.

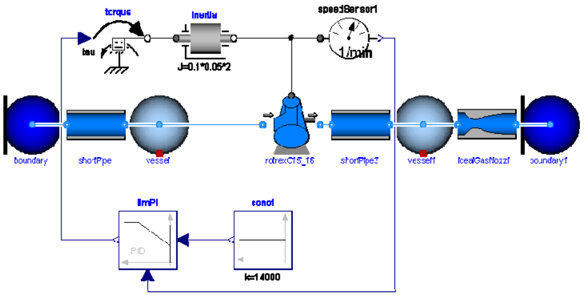


Figure 11 Modified compressor test model setup

Table 2 Comparison of modified test model performance

	Standard	ApproxSpline	Diff
CPU-time [s]	0.297	0.266	-10%
F-evaluations	1128	904	-20%
H-evaluations	786	714	-9%
Jacobian-eval.	87	82	-6%

[1] Any performance comparison was done on a double PC, dual processor, dual core AMD Opteron, 2.21 GHz, 2.75 GB RAM running Windows XP and Dymola 7.0

An interesting point is to study the sensitivity against random noise of table data. As mentioned before, table data are frequently obtained by measurements which are in most cases not smooth. To simulate this, a random number is added to each table data point:

$$\eta_i^* = \eta_i + \left(\max(\eta_i) - \min(\eta_i)\right)\mathrm{rnd}\left(\left[-0.5...0.5\right]\right)$$

$$\varphi^* = \varphi + \left(\max(\varphi) - \min(\varphi)\right)\mathrm{rnd}\left(\left[-0.5...0.5\right]\right)$$

The comparison of the models using these noisy table data is shown in Table 3. The shape of the spline surface only slightly changes in comparison to that generated from smooth data points (Figure 12). As a result, the simulation performance of the test model is nearly the same as if smooth data points were used (Table 3). In contrast the computational burden of the model using noisy data with standard Modelica table interpolation is much higher as if smooth data points were used. Thus it turns out, that spline approximation is advantageous especially if the table data is noisy.

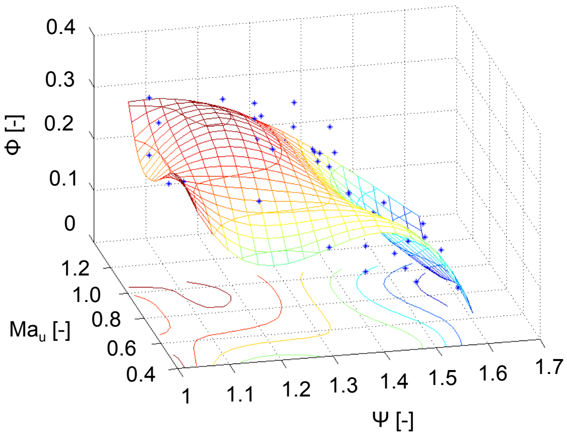


Figure 12 Flow coefficient as function of tip speed Mach number and head coefficient (mesh by spline approximation, noisy data points as stars)

Table 3 Comparison of test model performance with noisy table data

	Standard	ApproxSpline	Diff
CPU-time [s]	0.156	0.078	-50%
F-evaluations	2315	974	-58%
H-evaluations	1032	731	-29%
Jacobian-eval.	210	84	-60%

Another issue with measured table data is limited accuracy, e.g. because of the discretization error of an A/D-converter. To show this effect, the table data points are rounded to one digit (Figure 13). For the standard table model, the performance is clearly worse than with the original data. It is assumed, that the partially zero gradient of the data points is responsible for this behaviour. The approximating spline model is only marginal declined compared to the model that uses the original data (Table 4).

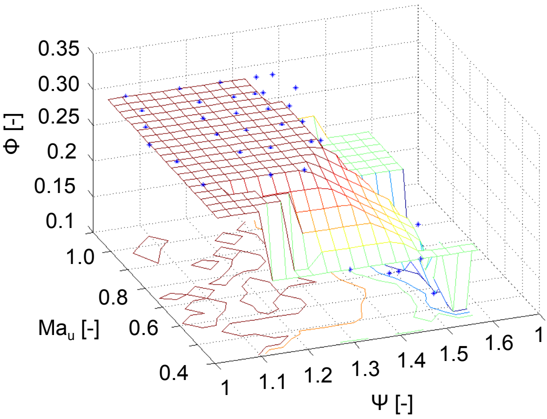


Figure 13 Flow coefficient as function of tip speed Mach number and head coefficient (mesh by interpolation, rounded data points as stars)

Table 4 Comparison of test model performance with rounded table data

	Standard	ApproxSpline	Diff
CPU-time [s]	0.109	0.078	-28%
F-evaluations	1520	924	-39%
H-evaluations	867	714	-18%
Jacobian-eval.	129	83	-36%

6 Conclusion

A new Modelica library for data approximation with polynomial splines in 1 and 2 dimensions is presented. Better performance compared to standard Modelica table interpolation was found in some cases, e.g. noisy or piecewise constant table data. The user interface is partially similar to the standard tables, so migration from standard interpolation tables to approximating splines should be of modest effort. For convenience, table data might be given as scattered data, so no external tool is necessary to generate regular gridded data points.

The library will be made freely available.

7 Acknowledgment

Many thanks to Prof. Paul Dierckx the author of the DIERCKX FORTAN code.

References

[1] T. Hirsch und M. Eck, “4-Dimensional Table Interpolation with Modelica,” *Proceedings of 6th International Modelica Conference 2008*, Bielefeld: 2008.

[2] P. Dierckx, *Curve and Surface Fitting with Splines*, Mcgraw Hill Book Co, 1995.

Point-to-Curve Constraints and other Contact Elements

Volker Beuter
Kämmerer AG
Wettergasse 18, 35037 Marburg
v.beuter@kaemmerer-group.com

Abstract

The MultiBody package of the Modelica Standard Library (MSL) contains a Prismatic Joint model with two Frame connectors where one frame can move with respect to the other along a direction *n*. This can be viewed as that the second frame can move along a straight line fixed in the first frame. In this work a constraint is developed where this line is replaced by some curve described by some suitable geometry: A Point-to-Curve (PtCv) constraint. It turns out that there are several options to define the orientation of the second frame with respect to the first one. Additional degrees of freedom are possible. These ideas can be applied to 2D: A Point-to-Surface (PtSf) constraint . PtCv and PtSf constraints seem to be suitable building blocks for higher order constraints: Curve-to-Curve (CvCv) resp. Surface-to-Surface (SfSf) constraints. As a by-product there are some Joint models not yet available in the MSL at all or not in that form, like an elementary Cylindrical joint.

Keywords: point-to-curve contact; osculating circle; point-to-surface contact

1 Introduction

Part of Kämmerer's involvement in the Eurosyslib project[4] is the development of a package with models for building convertible car roofs. One type of components of such a convertible roof are mechanisms where some kind of roller can move within something like a track. When we can abstract from effects like backlash and collisions with the bearings we can view this as a constraint with one translational degree of freedom. In this abstraction a point can move along a curve. The curve does not need to be fixed in space, it may be moving, but it is *rigid*. Conceptualized in Modelica MultiBody package terms this is a model with two Frame connectors. When we describe a roller - track component (where the track is fixed rigidly to some other part of the mechanism) the connecting point is not on the idealized line of the track. The moreover it is just a matter of the reference system what we consider the location of the connection. Translated to the Modelica model this means the curve is fixed to frame_a but it does not need to go through it. An idealized point can move along the curve. For convenience the other connector frame_b is located at this point on the curve.

The fact that there is just one translational degree of freedom along the curve does not imply that the point can move *freely* along the curve. There my be some friction, damping or even applied forces. But in a first stage we will not consider this.

Another question is if a Point-to-Curve constraint also ought to have *rotational* degrees of freedom. In the multi-body simulation program ADAMS[1] a point-to-curve contact always has all three rotational degrees of freedom (dofs), so it only constrains 2 translational dofs. Or think of the toddlers' toy where pierced pellets are beaded to a rigid wire. Here the pellets can rotate around the center axis of their hole, which is—again disregarding backlash—the tangential axis to the curve in the current contact point. An idealized model of this toy would be a point to curve constraint with two dofs: one translational dof along the curve and one rotational dof around the tangential axis in the contact point.

But it turns out that these additional rotational dofs can always be modeled—once tangential orientation along the curve can be represented—without putting them into the PtCv model itself: An ADAMS-like 4-dof PtCv can be built up from a PtCv without any rotational dof and a spherical joint connected to it. The mentioned toy can be modeled by a PtCv with tangential orientation with a revolute joint connected to it. Therefore here we will abstain from further complicating the models and will only consider PtCvs *with-*

DOI: 10.3384/ecp09430047

out any rotational dof: A PtCv constraint always has *just one dof.* (The idea of an additional rotational dof is only picked up for the special case of a straight curve in the section by-products where a cylindrical joint is described.)

The curve of an ideal PtCv constraint may be infinite or cyclic but it cannot be finite, i.e. cannot have end points: Such end points cannot be described by the notion of degrees of freedom. In contrast to this in technical realizations of PtCv elements we often have end points limiting the curve. But in most cases the stop is realized by means of introducing a repelling force and we can always describe a PtCv with stops as an ideal PtCv (with an unlimited curve) with additional force elements applying a repelling force preventing `frame_b` from leaving an admissible range of the curve too far. This will be described in some more detail in section PtCv with Stops.

As the Prismatic Joint from the MSL MultiBody package can be seen as the most simple case of a PtCv Joint, it is useful to have a look at it.

2 The Prismatic Joint

The Prismatic Joint has one translation dof in the direction specified by the direction parameter n: `frame_b` can move along a straight line trough `frame_a` in direction n, i.e the set $\left\{es \mid s \in R^3\right\}$ where $e = n/\|n\|_2$ is the normalized direction vector. As n is expressed in `frame_a` coordinates also the straight line is. (Here the roles of `frame_a` and `frame_b` are interchangeable, but we already view the straight line as fixed in `frame_a`.) Seen in this view the actual value of the position value s determines the "contact point" C in `frame_a` coordinates simply by $C(s) = es$. Here s is a one dimensional position variable (a distance, but may also become negative).

The only reasonable choice here is that both frames have the same orientation. The global positions of the frames is described by the equation $r_b = r_a + T^{-1}es$ where T is the orientation matrix of `frame_a` and T^{-1} is its inverse, which is simply the matrix transposed, because orientation matrices are symmetric. Disregarding the offset parameter `s_offset` we get `frame_a` = `frame_b` iff $s = 0$.

The sum of the forces acting on both frames is zero: $F_a + F_b = 0$. Regarding the torques at the frames we have $\tau_a + \tau_b + es \times F_b = 0$.

The tangential force, i.e. the force in direction n is $f = eF_a = -eF_b$. As there are no frictional, damping or applied forces in the basic Prismatic Joint model the tangential force is zero, i.e. $eF_a = 0$.

The question arising now is: What happens to these location and force balance equations when the straight line is distorted?

3 PtCv with parallel Orientation

Next we substitute the straight line by an arbitrary smooth curve, but keep the fact that both frames maintain parallel oriented permanently, i.e. the equation that both frame have the same rotation object. The curve is fixed in `frame_a` but does not need to run through it. Here we are not yet concerned with the concrete modeling of the curve. It is just a smooth mapping $C : R \rightarrow R^3$. What means "smooth" here will be elaborated later. It is not required that the curve is parameterized by it arc length. So we now use a variable s_0 for parameterizing the curve. $C(s_0)$ is the current contact point on the curve (in `frame_a` coordinates). The distance to $C(s_0)$ from the initial contact point along the curve, i.e. the arc length is denoted by the variable s. The same is with the velocities: v_0 is the curve parametrization velocity (i.e. the derivative of s_0) and v is the velocity along the curve. Note that s_0 and v_0 are not a physical length and velocity but just abstract `Real` variables. Only if the curve is parameterized by its arc length $s = s_0$ holds. (Another case where s_0 and v_0 are length and velocity is when the curve is parameterized with one of its components, say x, i.e. when $C(s_0) = \{s_0, C_y(s_0), C_z(s_0)\}$ holds, where C_y and C_z are the projections of the contact point function on y and z axis respectively. This is called a—lacking a better name—a "linear" curve in the PtCv package, because there is some main path in the curve, y and z can be viewed as deviations from this path.)

In the case of an arbitrary curve the equation $r_b = r_a + T^{-1}C(s)$ relates the global positions of the frames. Due to the parallel orientation of the frames their rotation object are the same. As the force at a frame is expressed in the frame coordinates we still have $F_a + F_b = 0$. The balance equation for the torques now becomes $\tau_a + \tau_b + C(s) \times F_b = 0$.

The property that the force along the axis of motion at the Prismatic Joint is zero here becomes that there is no force in the tangential direction along the curve in the contact point: $t_a F_a = 0$ where t_a is the (normalized) tangential vector in the contact point in `frame_a` coordinates.

A PtCv model with these equations is already suitable for building up an ADAMS-style PtCv by just connecting a Spherical Joint to `frame_b`.

4 Orientation of the second Frame

When something is moving along a curve it is quite natural that also the orientation of that object changes while traveling along the curve: A cornering car is normally oriented in the current direction of travel. When we want an object to follow a given curve we could use the old ADAMS users' trick of connecting the object to the curve by means of two PtCvs like described above in a short distance. But this has the disadvantage, that the object only follows the curve approximately. The closer the distance between both PtCvs, the better is the approximation but the more likely are numerical problems. The moveover when we do not need the remaining rotational dof along the axis connecting both PtCvs, we have to get rid of it by means of an additional joint. (In ADAMS this is done by a Perpendicular Joint which is not yet available in the MSL.) All these joints result in a model with many equations to describe such a simple mechanism.

So it seems desirable to have a tangential orientation of the second frame directly in the PtCv model. But here the problem arises that "tangential to the curve" only determines *one* direction vector of the orientation. Even if `frame_b` of the PtCv is to be connected to a revolute joint in order to introduce a rotational dof around the tangential axis (and therefore the second direction vector of the orientation does not matter) it has to be determined in order to have a unique solution of the equations. There are several opportunities: Take an arbitrary direction vector, e.g. the local z-axis of the `frame_a` coordinate system. But this does not work when the tangential vector t_a gets too near to the selected second direction vector. The other opportunity is to take the direction of the *normal* vector to the curve (pointing inward to a local curvature) But if the curve locally becomes a straight line the normal vector is not defined and special considerations have to be taken. The moreover using the normal vector to the curve demands a higher differentiability of the curve. Fore these reasons both options are available in the PtCv implementation and can be selected due to situation by parameter.

5 PtCv with tangential Orientation

A tangential orientation of `frame_b` keeps the position equation of the frames unchanged.

In order to archive a tangential orientation at least the tangential vector to the curve in the current contact point t_a (in `frame_a` coordinates) has to be determined. For a *moving* contact point this can in principle be done in Modelica by just applying the `der()` operator. Problems arise when the contact point is at rest, especially at simulation start. (So to speak you have to know where the road is going without walking.) Currently spatial derivations cannot be directly expressed in Modelica. Therefore not only an equation for the contact point depending on the curve parametrization variable s_0 has to be provided but also an equation for the tangential vector. The moreover, when the normal vector to the curve is taken as the second direction vector for defining the orientation of `frame_b`, also the second derivation of the contact point function has to be provided. (The normal vector can be determined from this second (spatial) derivation easily.)

When t_a and n_a are normalized vectors also the binormal vector $b_n = t_a \times n_a$ is and $T = \{t_a, n_a, b_a\}$ constitutes the transformation matrix of the relative rotation object from `frame_a` to `frame_b`.

The force and torque balance equations now have to account for the fact that both frames are no longer equally orientated. Forces and torques are transformed my means of the `resolve1` and `resolve2` functions from the `Modelica.Mechanics.Multibody.Frames` package. (Depending on whether `frame_a` or `frame_b` is closer to a root in the connection tree the balance equations are expressed resolved for both frames separately in order to improve numerical performance.)

An important observation is that the property that there are no forces in tangential direction *does no longer hold:* Even if there are no friction, damping or applied forces there is a force acting in tangential direction on a body attached at its center of mass to the curve when the curvature of the curve changes. When the body (with not only mass but also inertia) starts entering a corner the rotational energy rises. Due to the law of energy preservation the translational energy has to be decreased for the same amount. This means nothing but there is a breaking force acting along the curve. (When the curvature gets less again also the angular velocity and rotational energy go down again, so the translational energy rises and there is an accelerating force. So the process is reversible: after leaving the corner the travel velocity along the curve is the same as before entering the curve.)

When the curve is a circle there is a direct correspondence between rotational and translational energy.

For any two times differentiable curve there is a unique circle approximating the curve in the best way locally in a given point. This circle is called the *osculating circle*. Its radius is the inverse of the length of the normal vector. So the rotational velocity of a point connected to the curve is the same as the one of a point on this circle. Thus the equation relating translational and rotational energy can be applied to any two times differentiable curve and by differentiating this equation an equation for the tangential force can be applied.

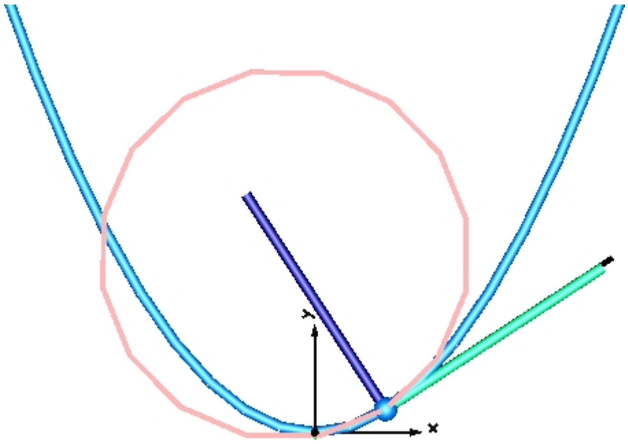


Figure 1: Planar curve (parabola) with osculating circle, tangential and normal vector in the contact point

The osculating circle can be visualized in the PtCv model. For the force equations we do not need its center coordinates and not even its radius but only its inverse, the *curvature* of the curve. This difference is important when the curve becomes a straight line locally, so that the radius of the osculating circle gets infinite and is not defined. The curvature simply gets zero and does not provide a problem.

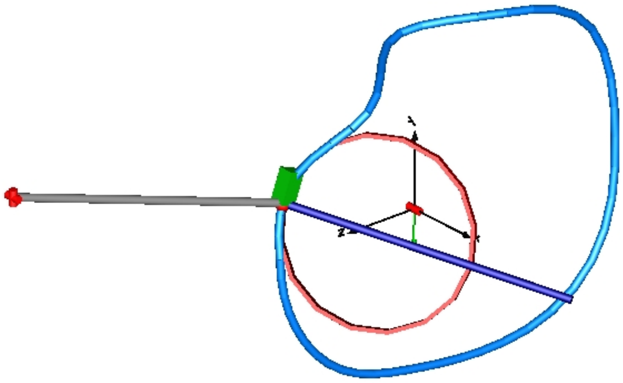


Figure 2: Osculating circle and normal vector at a 3D curve

Depending on the situation and the relation of tangential and rotational velocity this tangential force along the curve due to changing curvature can be neglected, e.g. when describing the cornering of an ICE train. On the other hand simulations of a body with a rather large inertia connected to a curve with parabola shape under the influence of gravity showed that the translational velocity is not highest in the lowest point of the parabola (as one might have expected) but at a symmetrical pair of points in a certain distance with a local minimum of the velocity in the lowest point of the parabola—where the curvature is highest.

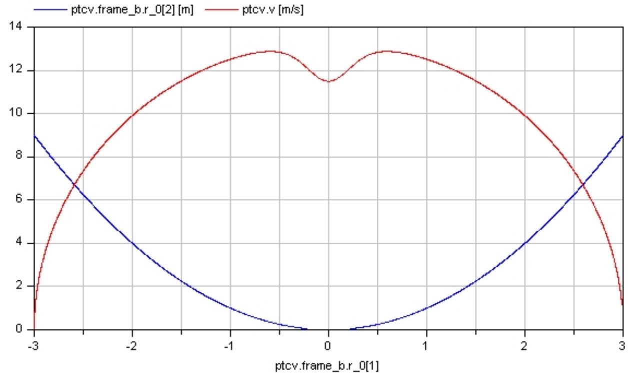


Figure 3: Velocity of a body with high inertia sliding freely along parabola

6 Geometry Definition

Up to now we only talked abstractly about the current contact point $C(s_0)$, the tangential vector t_a and the normal vector n_a (which are derivations of the contact point functions or are determined from derivations). In principle it is always possible to define the contact point function by its three cartesian components in the `frame_a` coordinate system. But it is rather inconvenient to define, say a helix curve with its center line in direction n by directly providing the three curve parametrization functions C_1, C_2, C_3 defining the curve in `frame_a` coordinates.

Therefore similar to the direction vector n in the Prismatic Joint two direction vector parameters n_x and n_y have been introduced defining a local x, y, z-system (with the z-direction orthogonal to both n_x and n_y) for a more convenient definition of the curve. E.g. a straight line in a direction n can be defined by setting $n_x = n$, taking for n_y any direction not parallel to n and the first coordinate function is the identity mapping, both other components are zero mappings.

The moreover there are model variants, where the curve is not defined in cartesian x, y, z coordinates but in cylinder coordinates: n_x here determines the axial direction of the cylinder. Here you have to provide the radial and axial component of the curve. To define a PtCv with the mentioned helix curve take $n_x = n$, the radial component is a constant and the axial component a linear function with suitable slope.

Now remains the question how the cartesian or cylindrical components of curve definitions are described.

7 Geometry Component Definition

There are several methods of defining the $x-$, $y-$ and $z-$ or radial and axial component of a curve definition. There is the opportunity to use *replaceable functions*. You write a function returning a single `Real` defining the desired function component. This method is good for defining geometric curves like the helix mentioned above, but is not applicable for defining curves with an arbitrary given shape. The moreover currently it is required not only to implement the functions describing the curve component but also their first and second derivations (because we need the spatial derivations of the curve to calculate the tangential and normal vector). Therefore this method is rather of theoretical interest to investigate the properties of an analytical function of concern.

As apparently there was no package ready to use we decided to implement our own cubic spline interpolation package. (It is not part of the PtCv package, because splines are of course applicable in many areas different from PtCv modeling.) Natural cubic splines are implemented, i.e the second derivation at the definition range borders are zero. A spline may have an arbitrary number of definition points which need not be equally spaced. Extrapolation is possible as constant or linear continuation or as periodic extrapolation with a repetition of the definition range infinitely many times. Although the complete calculation of the spline evaluation (and of the calculation of the second derivations at the definition points) is completely implemented in Modelica (like in the PtCv package, no external functions are used) Dymola is not able to calculate the derivations of the interpolation and extrapolation functions itself, so the derivations had to be provided explicitly. At a PtCv with parallel orientation using the normal vector for calculating the orientation object time derivations up to the 4^{th} order and spatial derivations of the evaluation function are needed. So a great deal of the development of the `Spline` package was implementing derivations.

Depending of the type of PtCv splines for the $x-$, $y-$, $z-$, radial or axial component of the curve definition can be provided. The radial spline is automatically extrapolated periodically, but it is up to the user to ensure that the lower and upper definition range border have the same radius to ensure the contact curve is continuous. All other component splines are linearly extrapolated. All definition splines have suitable defaults: The axial spline defaults to the zero spline, the radial spline defaults to the unit radius, so the default curve for a circular PtCv is a circle in the local $y-z$-plane (orthogonal to the provided n_x direction vector). The x-spline defaults to the identity mapping, $y-$ and z-spline to the zero mapping. By this means in many cases not all the definition splines have to be provided.

After we implemented our own `Spline` package we discovered that there is already a package for evaluation of Bezier Splines[3] developed at the DLR, Oberpfaffenhofen in 2002. (It is available under the Modelica license.) In order to use this `BSpline` package we had to write extrapolation features for it. (They were not added to the `BSpline` package, which was kept unchanged, but were placed into our `PtCv` package.) Now also PtCvs using BSplines for the curve definition are available in this package.

8 The PtCv Model Family

The sections above already mentioned that there are several PtCv models with different coordinate systems (cartesian or cylindrical) and different types the curve component functions are defined (replaceable functions, cubic splines, BSplines). All these models are extended from one basic model in several steps.

The partial model `PartialPtCv` contains everything common to all PtCv models. These are all parameters which are not directly related to the curve geometry definition, most of the parameters concerning animation, the equations relating the position and orientation of the two frames, the force balance equations and the equations for the force tangential to the curve. This model mostly uses the cartesian x, y, z-coordinate system mentioned above. The curve parametrization variables (e.g. s_0 are defined using replaceable types defaulting to `Real`, so they can be redeclared in situations, where they really mean positions and velocities, or angles and angular velocities. What is missing here is the equations for the current contact point $C(s_0)$, the current tangential t_a and normal vector n_a.

The next extension step is optional and rather intended for development and debugging purposes: The partial model `PartialPtCvExtended`, extended from

PartialPtCv contains variables (and visualizers) not really needed for the PtCv contact calculation but providing useful additional information, like visualizers for the osculating circle, the tangential and the normal vector, the traveled distance along the curve s and variables calculating the potential, translational and rotational energy for the special case that a mass with its center of mass is connected to a curve fixed in space.

The partial model PartialCircularPtCv is intended for all PtCv models using cylinder coordinates. (This does not mean that the curve itself is circular, like seen in the helix curve example. Therefore the name may be changed in future versions.) This model contains variables for transforming the cylinder coordinates into the cartesian x,y,z-system of the base model. There is a parameter revolutionLength determining the length of one revolution, i.e. if revolutionLength $= 360$ the radial and axial component are scaled on degrees, if revolutionLength $= 2\pi$ they have to be defined in radiant. The curve parametrization variables are redeclared to angles, angular velocities and angular accelerations.

Remember in the PtCv package the term "linear" means that the cartesian x-component of the curve is the identity mapping, i.e. there are only possible deviations into the y- and z-direction. For this case there is the partial model PartialLinearPtCv. In this case the curve parametrization variable s_0 is a distance, not along the curve but along the x-axis instead. Therefore the curve parametrization types are redeclared to SI.Position, SI.Velocity and SI.Acceleration. As also linear PtCvs use cartesian coordinates no transformation is required.

The partial model PartialGeneralPtCv only redeclares all curve parametrization types to Real, just to prevent further redeclaration. All these three partial models are currently extended from PartialPtCvExtended in order to have the additional debugging information at hand. In a final release they may be directly extended from PartialPtCv skipping the extra variables and visualizers.

The next step in this extension hierarchy are the completed (non partial) PtCv models, extending from one of the three models PartialCircularPtCv, PartialLinearPtCv or PartialGeneralPtCv. Here only the parameters for defining the curve components are declared (i.e. the replaceable functions for the geometry definition components currently together with their derivatives or the spline or BSpline parameters) and also the equations for determining the current contact point $C(s_0)$, the tangential vector t_a and the normal vector n_a, by evaluation of the functions or the spline resp. BSpline functions.

This separation into several model layers makes it easy to add new PtCv models with a custom geometry. There is even an instruction how to do so in the package documentation. On the other hand it enables to protect the base models by encryption in a version to be released in future.

9 PtCvs with Stops

So far we only dealt with PtCvs with an unlimited curve. For building a PtCv with a limited admissible range of the parametrization variable s_0, we take an existing full PtCv model and add the stops by extending it. (So we go one step further in the model extension hierarchy.) The implementation of a PtCv with stops has been performed exemplarily on a Linear PtCv where the curve is defined by two replaceable functions in y- and z-direction, but it can be implemented in the same way for any type of PtCv model.

It is important to note that adding stops to a PtCv does *not impose a new constraint* to it, but only applies an new additional force in tangential direction depending on the position and velocity of the contact point. If frame_b is forced to proceed by some prescribed motion it will do, regardless of the repelling forces. They may become huge, but as we have ideal elements nothing will break the mechanism like it will happen in a physical realization. A PtCv with stops still has one translational dof.

The stop position is defined by two new curve parametrization parameters $stop_1$ and $stop_2$. In this way the stop position is automatically located on the curve, namely at the positions $C(stop_1)$ and $C(stop_2)$. In case $stop_1 < s_0 < stop_2$ there is no additional force in tangential direction.

The stop is established by applying strong repelling forces to the point frame when it leaves the admissible parameter range. The repelling force consists of a non-linear spring force and linear damping where the damping coefficient is dependent on the actual penetration: If $s_0 < stop_1$ holds, there is contact to the lower stop and there is a force like the IMPACT force defined in ADAMS with a spring and a damping ingredient:

$$F(x) = \begin{cases} max(k(x_1 - x)^{exp} - cv, 0) & x < x_1 \\ 0 & else \end{cases}$$

where k is the spring stiffness, exp the stiffness exponent and $c = STEP(x, x_1 - d, c_{max}, x_1, 0)$ is the current damping coefficient depending on the position, which returns c_{max}, when $x < x_1 - d$, but zero, when $x > x_1$ and is smooth in between. This means that the damping force does not directly apply fully at the moment of contact, but is increased from contact to a penetration d where full damping c_{max} is archived. The `IMPACT` and `STEP` functions are implemented as separate functions to be used in other contexts than the PtCv stops.

The stops are visualized by cylinders with the origins at $C(stop_1)$ and $C(stop_2)$ pointing into the directions $-t_{stop_1}$ and t_{stop_2} (out of the admissible parametrization range) where t_{stop_1} and t_{stop_2} are the tangential vectors to the curve at $C(stop_1)$ and $C(stop_2)$. When the contact point leaves the range $stop_1 < s_0 < stop_2$ and there are contact forces, the relevant stop visualization cylinder changes its color.

Subject to further work on this topic is to establish a new partial model containing the stop implementation. So a particular PtCv model with stops ought to be little more than an extension of both the corresponding normal PtCv model (without stops) and the stop model.

9.1 Curve-to-Curve (CvCv) Constraints

A Curve-to-Curve (CvCv) constraint between two curves is defined by the property that at any time both curves have a common contact point and both curves are oriented tangentially to each other. This is a local condition. It does not require that the total shape of the curve would admit this constellation when physically built. (The curves may cross each other at regions away from the contact point.)

Some multi body dynamics programs (like ADAMS) provide Curve-to-Curve constraints only for *planar* curves and there is already a CvCv constraint implementation for planar curves in the `PlanarMultiBody` package [2]. But the concept of Curve-to-Curve constraint can also be transferred to smooth curves in 3D space.

In 2D CvCv constraint modeling when the contact point is found the position of both curves to each other is determined: A 2D CvCv constraint has just one (translational) dof. Her in 3D it is plausible to admit a rotational dof around the common tangential axis of both curves also.

Instead of modeling a CvCv constraint elementarily by stating its position and force balance equations we follow the approach of using two PtCv constraints connected to each other with their "point sides" to each other with a revolute joint in between. In case no rotational dof is admitted, there is a fixed rotation component instead where is can be set if both curves are to be oriented opposite to each other by using a rotation angle of 180 degrees or not.

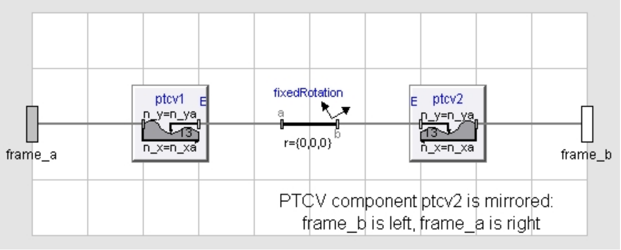


Figure 4: Diagram layer of a CvCv constraint

10 Point-to-Surface Constraints

It is quite natural to transfer the notion of a Point-to-Curve constraint to 2D: At a Point-to-Surface (PtSf) constraint a point can move along a smooth surface. This means a PtSf constraint has 2 dofs. For surfaces constituting analytical functions there are PtSf models with the surface described by replaceable functions. For practical applications there are PtSf versions where the surface is defined by 2D-Splines from the `AreaSpline` package. Currently only the option that the point frame is oriented parallel to the first frame is implemented, but in future orientation tangential to the surface will be an alternative.

Like at the PtCv models for all PtSf models there is one common partial base model and extensions with coordinate transformations for cylindrical and spherical coordinate systems (besides the core Cartesian coordinate system) from which the specific PtSf models are extended.

10.1 The AreaSpline Package

Here again it is straight forward to try to transfer cubic spline interpolation to 2D, i.e. to return the z-coordinate for a given (x, y) location. The spline is to be defined on a rectangular grid $(x_i, y_j)_{i=1,\dots m, j=1,\dots n}$. (This can be view as a landscape, where the height is tabulated at the points of this grid. Interpolation is the task to calculate the height $z = h(x, y)$ at any point (x, y) in between, under the assumption that the landscape is smooth.

10.1.1 Area Spline Interpolation

The idea here is to interpolate in the x- and y-direction rather independently. For any given x_0 location the projection $f(y) = h(x_0, y)$ can be considered a usual cubic spline (in the y-coordinate). In case x_0 is one of the grid values $(x_1,...x_m)$ we can determine the second derivations at these positions by solving the equations system like in the one-dimensional case. So we can even determine $h(x_0,y)$. (The same holds when the y-coordinate y_0 is one of $(y_1,...y_n)$: We can calculate $h(x,y_0)$.) To calculate the height at an arbitrary position (x,y) not matching any of the grid lines we can first determine the definition rectangle to which (x,y) belongs, i.e. indices i and j so that $x_i < x < x_{i+1}$ and $y_j < y < y_{j+1}$. Now we can calculate both $h(x_i,y)$ and $h(x_{i+1},y)$. The moreover we can determine $h(x_i,y)$ for all $i = 1,...m$ and consider these values as the definition points of a cubic spline in the x-coordinate. The problem is that for calculating this spline we'd had to solve the system of equations for this particular arbitrary y, i.e. at evaluation time, what would be time consuming at larger definition grids. But as the interpolation function of a spline is a 3rd order polynome between definition points, the 2nd derivation is a 1st order polynome which can be linearly interpolated easily.

The approach is now as follows: Instead of calculating the 2nd derivations of the spline defined by $(x_i,h(x_i,y_0))_{i=1,..m}$ by solving a system of equations, we take the coefficients of the splines $(x_i,h(x_i,y_j))_{i=1,...m}$ and $(x_i,h(x_i,y_{i+1}))_{i=1,...m}$, interpolate each pair linearly and take them as the coefficients of the spline through $(x_i,h(x_i,y))_{i=1,...m}$. By interpolating this spline at x we can finally calculate $h(x,y)$.

Here we started by working in y-direction, i.e. by first calculating $h(x_i,y)$ and $h(x_{i+1},y)$ but that is not crucial. It can be shown that we end up at the same result, when we calculate $h(x,y_j)$ and $h(x,y_{j+1})$ first and determine the coefficients of the spline trough $(y_j,h(x,y_j))_{j=1,...n}$ by linear interpolation of the coefficients in the columns x_i and x_{i+1}.

This approach has two advantages:

1. All spline coefficients can be calculated once for all when defining the spline (or when modifying it at an event). No solving of systems of linear equations is required at evaluation time.

2. The coefficients in x-and in y-direction can be calculated independently. The moreover the coefficients in each row and column can be determined independently. We simply can calculate the usual coefficients of 1D splines along all the definition grid lines. With an $m \times n$ definition grid we have just m systems of equations of size n and n systems of size m instead of one or two big systems of size mn or so.

Unfortunately this approach has one big disadvantage: Although it is true, that the 2nd derivation of the interpolation function between two definition points is a first order polynome which can be linearly interpolated without any loss of precision, we just get an approximation, when we interpolate between the 2nd derivations in y-directions at (x_i,y_j) and (x_i,y_{j+1}) in order to get the value at (x_i,y). Linear interpolations is exact here only in x-direction between (x_i,y_j) and (x_{i+1},y_j) but not in y-direction.

As a result of this inexactness we have the following effects. The interpolation function is:

1. continuous,

2. two times continuously differentiable in any point not matching one of the definitions grid lines,

3. two times partially continuously differentiable along the definition line grids, but in general

4. not partially differentiable when crossing the definition line grids, i.e. not (totally) differentiable at points on the grid lines.

So the resulting interpolation surface looks folded at the definition grid lines. The distances of the definition lines are the smaller the closer the definition grid lines get. Of course we are a lot better off than with interpolating the definition grid just linearly.

10.1.2 Area Spline Implementation

Like at the 1D splines there is a function `makeAreaSpline` to initialize a spline record by calculating the spline coefficients like described above. In the evaluation function `evalAreaSpline` extrapolation is possible constantly, linearly and periodically. It can be chosen between these three options independently in the x- and the y-direction.

Unfortunately we had to implement the 1st and 2nd derivation of this function manually. (Higher order derivatives were not yet needed because at the PtSf constraints up to now only parallel orientation of the point frame was implemented.) But on the other hand having these time derivatives it was easy to implement the partial derivatives into x- and y-directions that we also needed anyway.

10.1.3 Area Spline Visualization

There is a sub-package `Visualizers` for displaying area spline surfaces using the `Plot3D` package. An area spline can be displayed by entering its definition data into a function call. There are variants also displaying normal vectors to the surface or one of the partial derivatives instead of the spline surface itself. The definition grid lines are displayed also like in the example below.

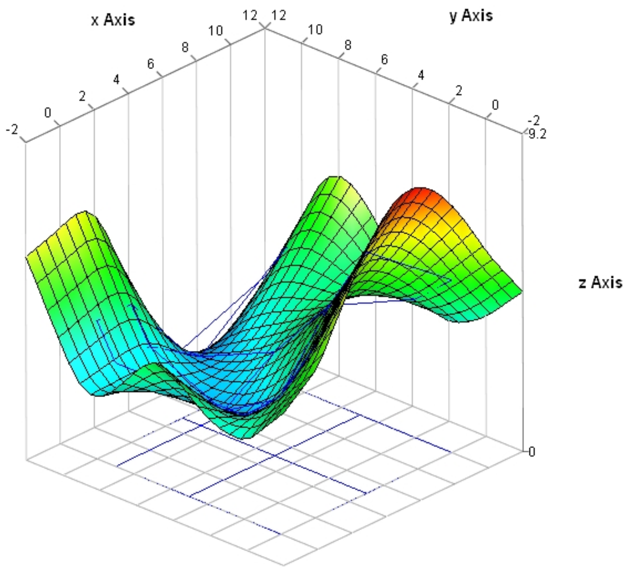


Figure 5: Example of Surface definition with `AreaSpline` Package

Unfortunately this package cannot be used for visualizing a surface in a mytt MultiBody model with the animation used there.

11 By-Products

In the development of the PtCv package an attempt was made to integrate a revolute dof to a PtCv with tangential orientation. This did not yet work properly. But it does work in the special case that the curve is simply a straight line. So there is a translational dof along an axis and a rotational dof around that axis. This is noting but a cylindrical joint. Therefore this was turned to a separate model where the arbitrary curve with all its parameters was reduced to a direction vector *n* and the hierarchy of partial models was turned into one model.

Of course there is a Cylindrical Joint model in the MultiBody package of the MSL, but this is composed of the connection of a Prismatic and a Revolute Joint. But compared to this standard implementation the Cylindrical Joint model in this package is described directly by equations. The number of equations is about 10% less than in the standard implementation and simple test models are considerably faster.

A PtSf constraint where the surface is a plane is a planar parallel joint, i. e. `frame_b` can move along a plane through `frame_a` defined by two direction vectors *n* and *m*. This is like the planar joint in the MultiBody library, but without the rotational degree of freedom. As such a joint is of general interest it has been implemented as a separate model. Despite the MultiBody planar joint here the translation in the plane is not modeled by two orthogonal prismatic joints but elementarily.

12 Conclusions

Although especially the PtCv models are up and running the packages described in this paper are to be seen as a work in progress. It seems valuable to incorporate some more ideas from the `PlanarMultiBody` package like providing the user with a collection of predefined curves like circles and elipsoids. Cubic splines will be then just one type of predefined curve. This applies even more to the `PtSf` package which will become much more usable if there would be a set of predefined parametrizised shapes.

The original plan to develop also contact *force* elements besides the constraints will not be addressed anymore within the Eurosyslib project due to lack of time but are subject to further work.

13 Acknowledgements

The `PtCv` (Point-to-Curve) and the `PtSf` (Point-to-Surface) packages and the used geometry packages `Spline` and `AreaSpline` have been developed as part of the ITEA2 **Eurosyslib** project (WP 8.6).

References

[1] http://www.mscsoftware.com/products/adams.cfm

[2] M. Höbinger, M. Otter: PlanarMultiBody - a Modelica Library for Planar Multi-Body Systems. Proceedings of the 5^{th} Modelica Conference, Bielefeld, Germany, 2008, pp. 549-556

[3] Schillhuber, Gerhard, `BSpline` package, Copyright Modelica Association and DLR, 2002.

[4] http://www.itea2.org/public/project_leaflets/EUROSYSLIB_profile_oct-07.pdf

Modeling and Simulation of a Solar Tower Power Plant with Open Volumetric Air Receiver

Nils Ahlbrink Boris Belhomme Robert Pitz-Paal
German Aerospace Center, Institute of Technical Thermodynamics
Linder Hoehe, 51147 Cologne
nils.ahlbrink@dlr.de, boris.belhomme@dlr.de, robert.pitz-paal@dlr.de

Abstract

The start-up of the PS10 power plant in Seville, Spain, in 2007 marked the entrance of solar tower power plants into the commercial state. Questions about the right operational strategy, particularly during unsteady operation states, come to the fore, and therewith the need to carry out transient simulations of entire tower power plants including the heliostat field. Meeting this necessity, the presented simulation approach opens the way to transient full plant simulations of solar tower power plants. A detailed heliostat field model was linked to a dynamic receiver model by coupling both simulation tools. A second coupling was established to a tool hosting a control panel of the heliostat field model. With this simulation approach, a start-up procedure and a tracking stop were simulated delivering different transient behaviors of local absorber temperatures and mass flows.

Keywords: Solar tower, modeling, tool coupling, plant simulation

1 Introduction

A detailed analysis of a full system out of heliostat field and receiver requires models, which represent both parts in an adequate, strong simulation environment. A fast raytracing tool, namely STRAL (Solar Tower Raytracing Laboratory) [1], is used to represent the heliostat field including optical characteristics as well as transient heliostat tracking.

For the receiver, the object oriented modeling language Modelica in combination with the simulation environment Dymola is applied to develop a transient, discretized receiver model. Modelica is an object-oriented modeling language designed for modeling of complex physical systems [2], [3]. Dymola interprets the modeling language Modelica and enables for convenient dynamic simulations of complex systems [4]. An object-oriented model library was developed for solar tower power plants with open volumetric air receiver technology, including the receiver, blowers, valves, pipes, the thermal storage system, and the power block. In combination with a heliostat field model, a complete solar tower power plant can be simulated [5]. The library is based on the open source Modelica library Modelica_Fluid [6].

The flux density distribution on the absorber surface of the receiver represents the physical interface between both models. To feed the receiver model with transient flux density data, both simulation environments need to be coupled. Therefore, STRAL was enhanced by an additional TCP/IP interface. An additional Dymola library was developed in parallel, enabling Dymola to communicate with STRAL via the same interface. In a second step, an additional tool coupling was established between STRAL and LabView for the purpose of a superior heliostat field control and human machine interface emulation.

The complete simulation system was tested running on three different computers. A solar tower power plant consisting of 153 heliostats and an open volumetric air receiver served as test case. The simulation scenario included a start-up procedure followed by a total heliostat field failure. During the failure time, the heliostats stopped tracking for 12 minutes and continued tracking afterwards.

The research results presented in this paper are part of a project, in which mainly control and operational strategies are developed for solar thermal power plants with open volumetric air receiver technology [5]. More information about the control strategy approach can be found in a related paper [7].

2 Transient system model

2.1 Heliostat field model

Like a couple of other well known flux density calculation tools for heliostat fields [8], the heliostat

DOI: 10.3384/ecp09430048

field model used in STRAL [1] comprises effects as shading and blocking of heliostats as well. Furthermore each single heliostat model comprises an individual highly resolved heliostat geometry data set, obtained from deflectometry measurements, and in addition an individual transient tracking model of the heliostat drive system, meeting the fact of limited movement speeds and specific drive constellations. This enables to simulate transient effects as for instance during start-up procedures, aim point changes or even worst-case scenarios as partial or total failure of the heliostat field.

The used simulation consisted of 153 heliostats, each with a separate model of the heliostat drive system embedded in an external Dynamic Link Library. The heliostat field is shown exemplary in Figure 1. The actual azimuth rotation angle and elevation angle represent the two state variables of the simple drive system model. As can be seen in Equation 1, the actual axis rotation angle φ_{act} is incremented and decremented respectively with a constant axis rotation speed ω_{axis} until the target axis rotation angle φ_{tar} is reached. The target axis rotation angle φ_{tar} depends on the position of the heliostat relative to the receiver, the current sun position and on the actual aim point assignment.

$$\varphi_{act,t} = \begin{cases} \varphi_{act,t-1} + \omega_{axis} \cdot \Delta t, & if \varphi_{act,t-1} < \varphi_{tar} \\ \varphi_{act,t-1} - \omega_{axis} \cdot \Delta t, & if \varphi_{act,t-1} > \varphi_{tar} \\ \varphi_{tar}, & if \varphi_{act,t-1} = \varphi_{tar} \end{cases} \quad (1)$$

It is assumed that each axis of the heliostat is moved with a constant axis rotation speed ω_{axis} and effects as for instance overshooting are not considered. The simple approach is justified in this case since heliostats usually operate with very low axis rotation speeds.

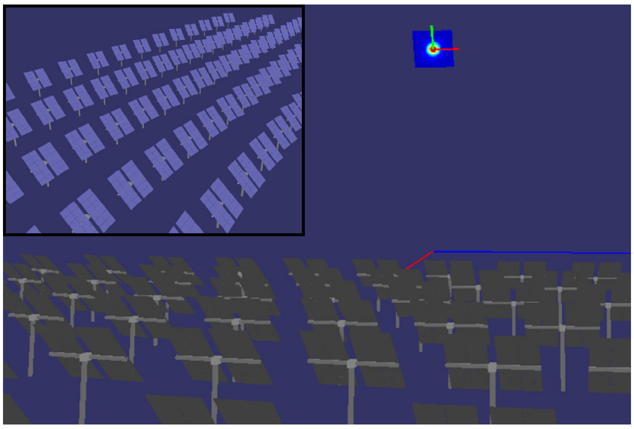


Figure 1 Heliostat field model in STRAL [1] showing the view towards the evaluation layer and upon the heliostat field (reduced window)

2.2 Receiver model

Depending on the purpose, the receiver can be modeled in different levels of detail. In this application, the dynamic behavior of the full receiver is the field of interest. Thus, the individual absorber components building the receiver are modeled in a more simplified way. Their individual temperature profile or the absorption process are not part of the investigations. The focus of the modeling is on the dynamic effects of the full receiver system including start-up, shut down and dynamic effects of an inhomogeneous flux density distribution in time and space. The main purpose of the model is to be used for analysis of transients in the receiver, for the development of control strategies, and for the prediction of receiver behavior.

Examplary, a receiver with open volumetric air receiver technology is modeled. This technology uses a grid of absorber modules to setup a receiver. Analogously, the receiver model will use a representative number of absorber modules to model the full receiver as shown in Figure 2. A number of real absorber modules can be represented by one absorber module in the model. A matrix of n x m modules is set up to represent the receiver. The highest possible discretization is the real number of absorber module. Higher discretizations are disadvantageous, as the focus of the receiver model lies on the dynamic effects of the complete receiver system and not on the absorption and heat transfer processes inside the module. It would cost additional computation time. Using absorber modules as the smallest unit, receivers of different sizes can be simulated. Effects of up scaling can be analyzed.

The setup of the receiver model in Dymola is shown in Figure 3. In alternating sequence, flow and volume elements are instantiated according to [2].

Model of the absorber module

The key model of the receiver is the absorber module model (Figure 2). Two air streams are modeled; the first stream represents the air inside the module, which is heated up in the absorber comb, and the second stream represents the return air at the outside of the module, which is lead back to ambient cooling the receiver structure. The return are is fed back to the receiver from the cold air side of the power plant.

The absorption process and the heat transfer in the absorber comb have to be modeled very accurate, as the performance of the absorber module is mainly determined by the absorption efficiency besides mass flow rate and air inlet temperature. However, the complexity of the model should be as simple as

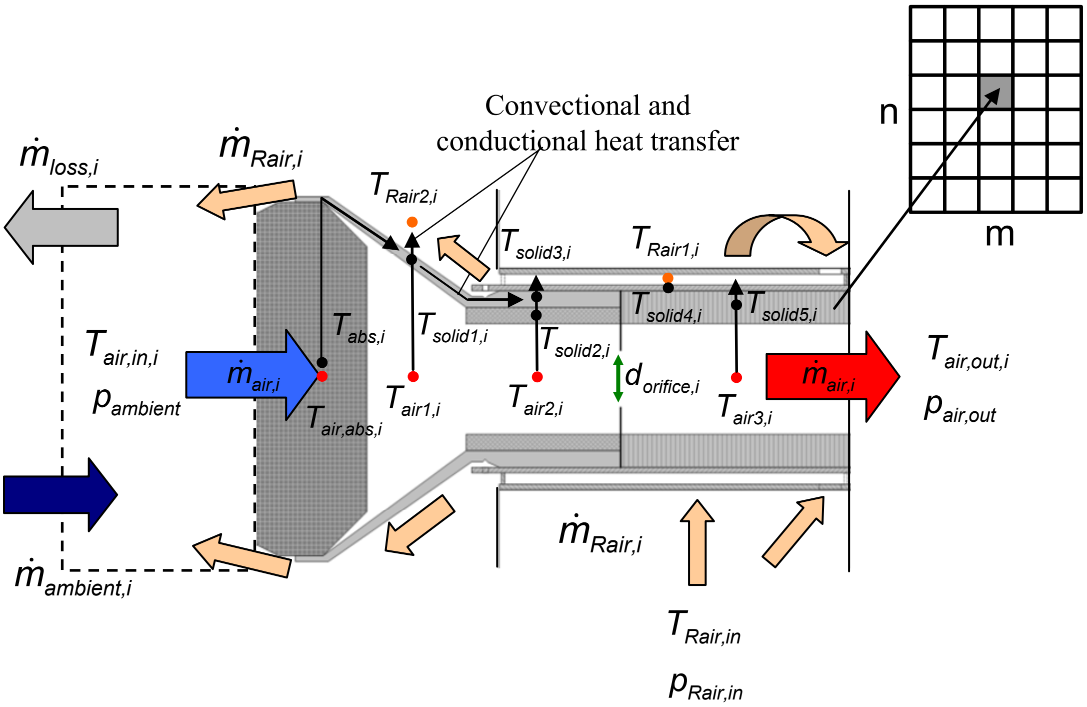


Figure 2: Absorber module model having 12 temperature nodes used for the receiver discretization of n x m absorber modules

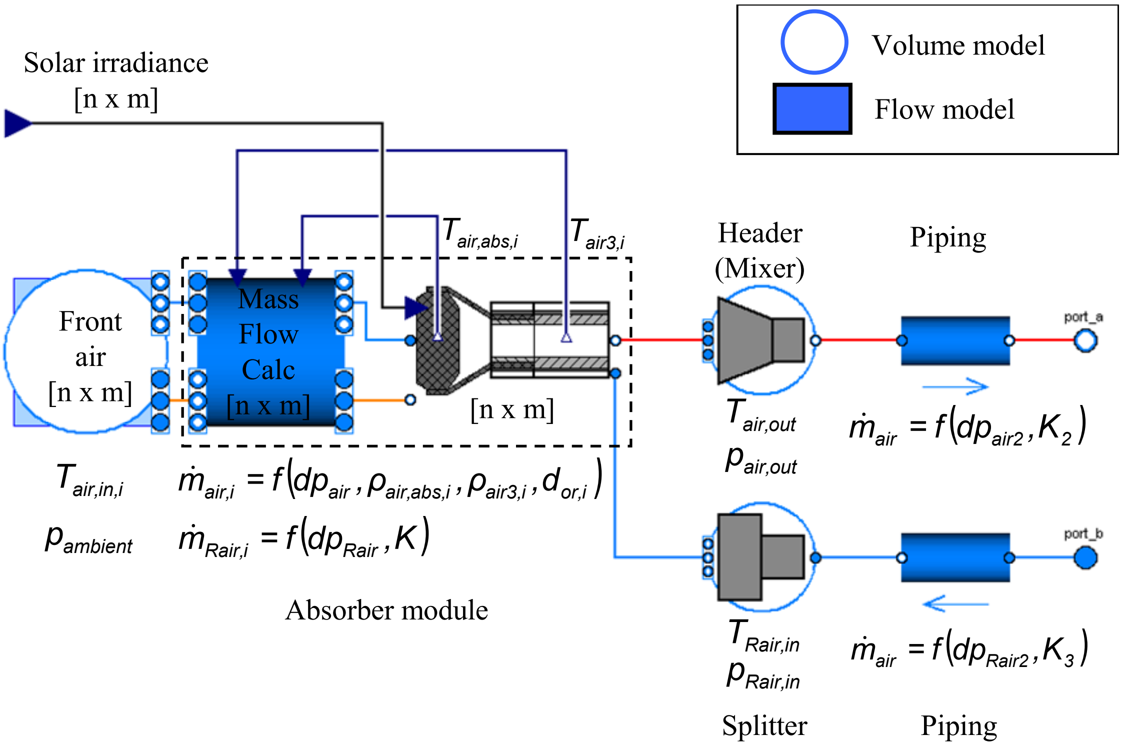


Figure 3: Schematic receiver model in Dymola including the described absorber module model

possible. Therefore, the absorption and heat transfer processes in the absorber comb are implemented by an efficiency interpolation matrix based on a detailed absorber model [9]. The inputs to the interpolation matrix are the flux density, the mass flow rate, and the inlet temperature. The result of the interpolation is the local efficiency from solar flux to heat absorbed in the absorber module.

Nusselt correlations are used for the heat transfer modeling between the hot air and the absorber cup, insulation, and the absorber pipe as well as for the heat transfer to the return air. 1-D heat conduction is assumed inside the solids in axial direction. In total, the absorber module model has 12 temperature nodes in the air streams and the solids. Depending on the up-coming model evaluation process, the number of nodes might be further reduced. Pressure at the nodes is assumed to be equal to the outlet pressure level $\boldsymbol{p_{air,out}}$ for the main stream and the inlet pressure level $\boldsymbol{p_{Rair,in}}$ for the return air. The pressure loss inside the modules is small, which results in a negligible influence on the fluid properties.

Mass flow rate distribution across receiver

Mass flow rate distribution across the receiver and the individual mass flow rates of the modules have a significant impact on the performance of the system. The mass flow rates of the individual absorber modules are dependent on each other since they are driven by a common pressure difference. The overall mass flow rate is split up into the absorber modules according to flow resistance of the modules. If the flow resistance of one module changes, for example by a changing flux density followed by an adapting air temperature, a new mass flow rate distribution over the receiver will establish. Thus, the absorber modules can not be considered individually.

In the absorber modules, like shown in Figure 2, a fixed orifice is used to adapt the mass flow rate according to a design flux density. Larger orifices are used in the center of the receiver as higher flux densities are expected. Smaller orifices are implemented in the less irradiated areas to achieve a desired temperature profile in the receiver at least for the design point flux density distribution. However, the orifices are not adaptable during operation, which results in deviant temperature profiles in off-design operating points.

The individual mass flow rates for the main air are modeled depending on the pressure difference $\boldsymbol{\Delta p_{air}}$, the air densities in the absorber comb $\boldsymbol{\rho_{air,abs,i}}$ and behind the orifice $\boldsymbol{\rho_{air3,i}}$, and the orifice diameter $\boldsymbol{d_{orifice,i}}$.

$$\dot{m}_{air,i} = f\left(\Delta p_{air}, \rho_{air,abs,i}, \rho_{air3,i}, d_{orifice,i}\right) = \frac{-a + \sqrt{(a^2 + 4b\Delta b_{air}}}{2b} \qquad (2)$$

The coefficients $\boldsymbol{a}$ and $\boldsymbol{b}$ are dependent on the densities, whereas factor $\boldsymbol{b}$ is as well a function of the orifice diameters $\boldsymbol{d_{orifice,i}}$.

For the return air, a quadratic reference pressure drop model is used.

$$\dot{m}_{Rair,i} = f\left(\Delta p_{Rair}, K\right) = \sqrt{\frac{\dot{m}_{Rair,Ref}^{\;2}}{\Delta p_{Rair,Ref}}\Delta p_{Rair}} \qquad (3)$$

For both air streams, it is assumed that the pressure difference is the same for all absorber modules.

$$\begin{aligned} \Delta p_{air} &= \Delta p_{air,i} = p_{ambient} - p_{air,out} \\ \Delta p_{Rair} &= \Delta p_{Rair,i} = p_{Rair,in} - p_{ambient} \end{aligned} \qquad (4)$$

In Modelica, the receiver model is implemented according to flow and volume elements strategy [2]. Flow elements determine the mass flow rate while they do not store mass. Volume elements, however, account for storage effects but not for flow calculations. In our case, the flow elements calculate the mass flow rate out of the pressure difference according to two adjacent volume elements. In the volume elements, the pressure is used as a state variable. The second state variable in the volume elements is either the temperature or the specific enthalpy. Flow elements can have an energy balance as well.

Boundary condition at inlet of absorber module

In front of the receiver, the front air is modeled with a volume element according to the discretization with n x m volumes. It uses ambient pressure level. The air inlet temperature $\boldsymbol{T_{air,in,i}}$ to the absorber modules is calculated. A part of the outflowing return air can be sucked in again. The amount is defined by an air return ratio. The other part is lost to ambient. Air from ambient mixes with the return air. The air inlet temperature to the absorber modules is a crucial variable for the performance of the absorber module. In the next step, a more detailed modeling of the effects of the air in front of the receiver will be implemented for example the rise of the hot air by natural convection.

The front air model is connected to the absorber module model, which uses the same discretization with n x m modules. It consists of two parts. As the absorber module model corresponds from its purpose to a flow model, the first block calculates the mass flow rates for both air streams and for each absorber module. The second block represents the described absorber module setup having 12 temperature nodes.

The separation is used to be able to optimize the mass flow rate distribution calculation regarding calculation time in one central component in the future. The thermal behavior of the modules can be exchanged separately. At the moment, the calculation of the mass flow rate results in a very complex equation system as all flows are dependent on each other and on the air temperature or rather the flux density.

Pressure is, as described earlier, not a state variable in the absorber modules. The pressure change is neglectable for the fluid property calculation. That way the correct pressure difference is transferred to the mass flow rate calculation block according to equation 4. Two temperatures, the air temperature in the absorber comb $T_{air,abs,i}$ and the air temperature behind the orifice $T_{air3,i}$, are fed back to the temperature dependent mass flow rate calculation.

Boundary condition at outlet of absorber modules

The lumped air volume behind the absorber module in the main stream represents the header. Heat losses are modeled by Nusselt correlations. Pressure and specific enthalpy are the state variables for the volume for which balance equations are solved. The mass flow rates of the individual absorber modules are mixed in the header. It is followed by a pipe model. On the return air side, an equivalent setup is formed by a volume representing the air in the splitter and the return pipe.

2.3 Tool coupling of Dymola and STRAL

Both simulation tools, STRAL incorporating the detailed heliostat field model and Dymola incorporating the complete receiver and air cycle model, are to be used in one overall simulation. For that purpose, a tool coupling had to be established. It works via a TCP/IP network connection. To be able to control the heliostat field during the simulation run, for instance to modify the aim point configuration of the heliostats, a second connection is established between STRAL and LabView as shown in Figure 4.

The tool coupling is based on a classical client server model approach, where Dymola acts as Client, LabView as Server and STRAL as Server and Client at the same time. Therefore, a specific heliostat field model class was implemented in Dymola, incorporating an external static library, which is linked to the STRAL TCP/IP interface. The class makes use of the external function interface in Modelica [10].

A simulation run is divided into an initialization stage and a main simulation stage. During the initialization stage, the Dymola client establishes a connection to the STRAL server and passes initial conditions, as for instance specific receiver configurations (size, expected flux density discretization). Then, STRAL establishes a second connection to the LabView server, acting as client. The succeeding simulation stage is a sequence of calculations steps with modifiable time steps defined by the Dymola model. A model, which triggers events at specific time instants, evokes a calculation step. The step size of the events can be set for specific time periods.

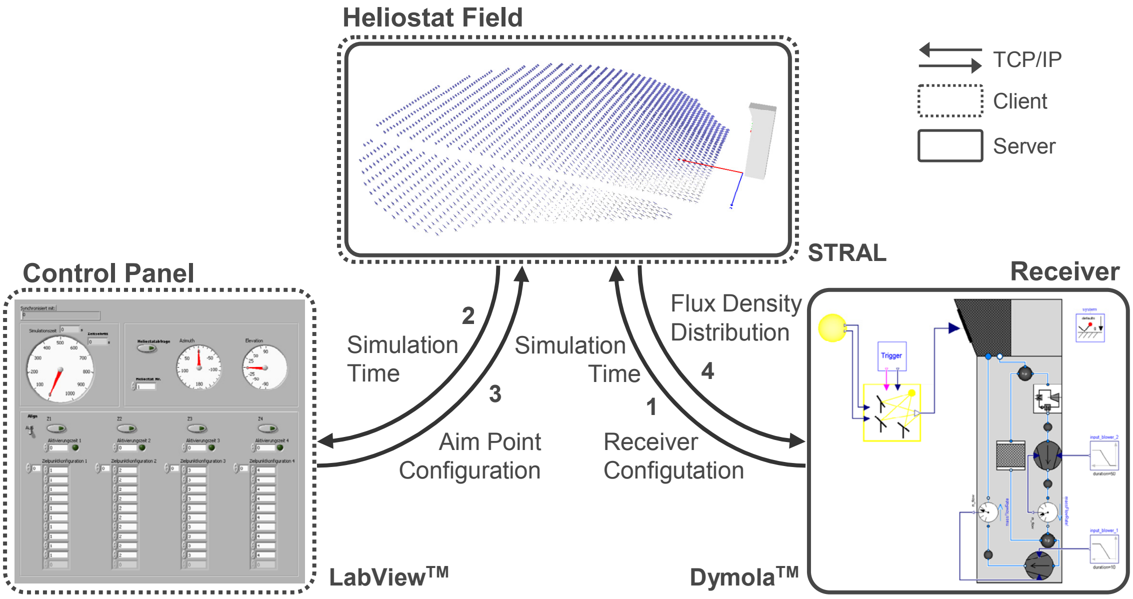


Figure 4: Tool Coupling Scheme of Dymola, STRAL and LabView

At the beginning of a single calculation step, Dymola sends a flux density calculation request to the STRAL server. This includes the actual simulation time and, optionally, the suns azimuth and elevation angle. If not defined by Dymola, the sun tracjectory can be read directly in STRAL. In the following step, the STRAL client sends a synchronization request to the LabView server providing the actual simulation time as well. The LabView Heliostat Field Control Panel Application exchanges status data of the heliostat field and, if needed, modifies the configuration of the field, as for instance the current aim point assignment. After this procedure, the heliostat alignment is updated by STRAL considering the specified drive system model of each heliostat. Then the requested flux density calculation is calculated based on STRAL's raytracing engine. The resulting flux density distribution is send back to the Dymola application. Independently from the main calculation steps, Dymola performs variable integration time steps inside of the Dymola model.

In addition to the described standard procedure, STRAL can calculate the next flux density distribution in advance based on the step size of the last call. At the next calculation request, it will check the desired time step, sun position and aim point configuration, and compares it to the assumed next calculation steps. That way, the next step can be calculated in advance before Dymola actually requests the next calculation step. No waiting time occurs for the Dymola model to wait for the STRAL answer, in case the time step and aim point configuration has not been changed, which is normally the case.

3 Generic test case

The start-up of a power plant, consisting of the described receiver model in Dymola and a heliostat field model in STRAL, was simulated. A control panel, modeled in LabView was used to interact with the heliostat field model to be able to shift the aim points.

The Dymola model includes the receiver, an adjustable heat sink to cool down the air to a design air temperature after the receiver, and a blower. The heat sink is implemented instead of a consumer like a thermal storage or a steam generator. The design return air temperature is set to 110°C. The mass flow rate of the receiver is adjusted by the blower power. It is increased at the beginning within one minute to the overall design mass flow rate of 2.2 kg/s. It is kept constant by a controller during the simulation. Mass flow rate disturbances are caused by changes of the air temperature in the absorber modules due to inhomogeneous irradiation. The mass flow rate distribution is dependent on the orifice diameters as well as the fluid properties and changes during the simulation.

The flux density is the input to the receiver, which is calculated by the heliostat field model. A discretization of 6x5 absorber modules is chosen. Each of the modules represents a matrix of 3x3 absorber modules. Thus, in total, 270 absorber modules of the real receiver are represented by 30 absorber modules in the model. For each of the modules, a specific orifice diameter was chosen depending on the location in the receiver. Larger diameters are used in the center to take care of expected higher flux densities and smaller diameters are implemented towards the edges of the receiver. The orifice diameters are important for the mass flow rate distribution, but cannot be adjusted during the simulation.

The Dymola model provides the receiver dimensions and discretization as well as the sun azimuth and elevation angle for the heliostat field model. The heliostat field model is set up in STRAL. It includes the 153 heliostats with a mirror area of 38.8 m^2 each. The heliostats have a constant rotation velocity of 0.01 mrad/s for the azimuth and elevation axis. At the beginning of the simulation, the heliostats point towards one common standby aim point west of the receiver surface. The aim point is shifted towards the center of the receiver, after the design mass flow rate is reached. The air temperatures inside the absorber modules will increase and approach a steady state level. At steady state level, a tracking stop of 12 minutes is simulated. The tracking stop represents possible mistakes or failure in the heliostat field control. In this scenario, the tracking is reactivated after 12 minutes, so that the heliostat focus is adjusted to the aim point at the center of the receiver again.

Figure 5 shows the solar irradiation, the material temperatures of five absorber modules in one receiver row and the average air outlet temperature, as well as the corresponding air mass flow rates. The locations of the absorber modules are displayed. They are positioned on the same horizontal axis in the fourth row of the receiver.

During the simulation, the material temperatures increase as the flux densities rise after shifting the aim point to the center. The shifting process is relatively fast compared to the temperature rise. Therefore, the solar irradiation increase appears similar to a step input. The temperatures increase and approach a steady state level. The start-up takes about 15 minutes for a temperature change of 800 °C. In this start-up procedure, all heliostats are shifted to one central aim point at the same time. As demonstrated, it causes a high temperature increase and therewith

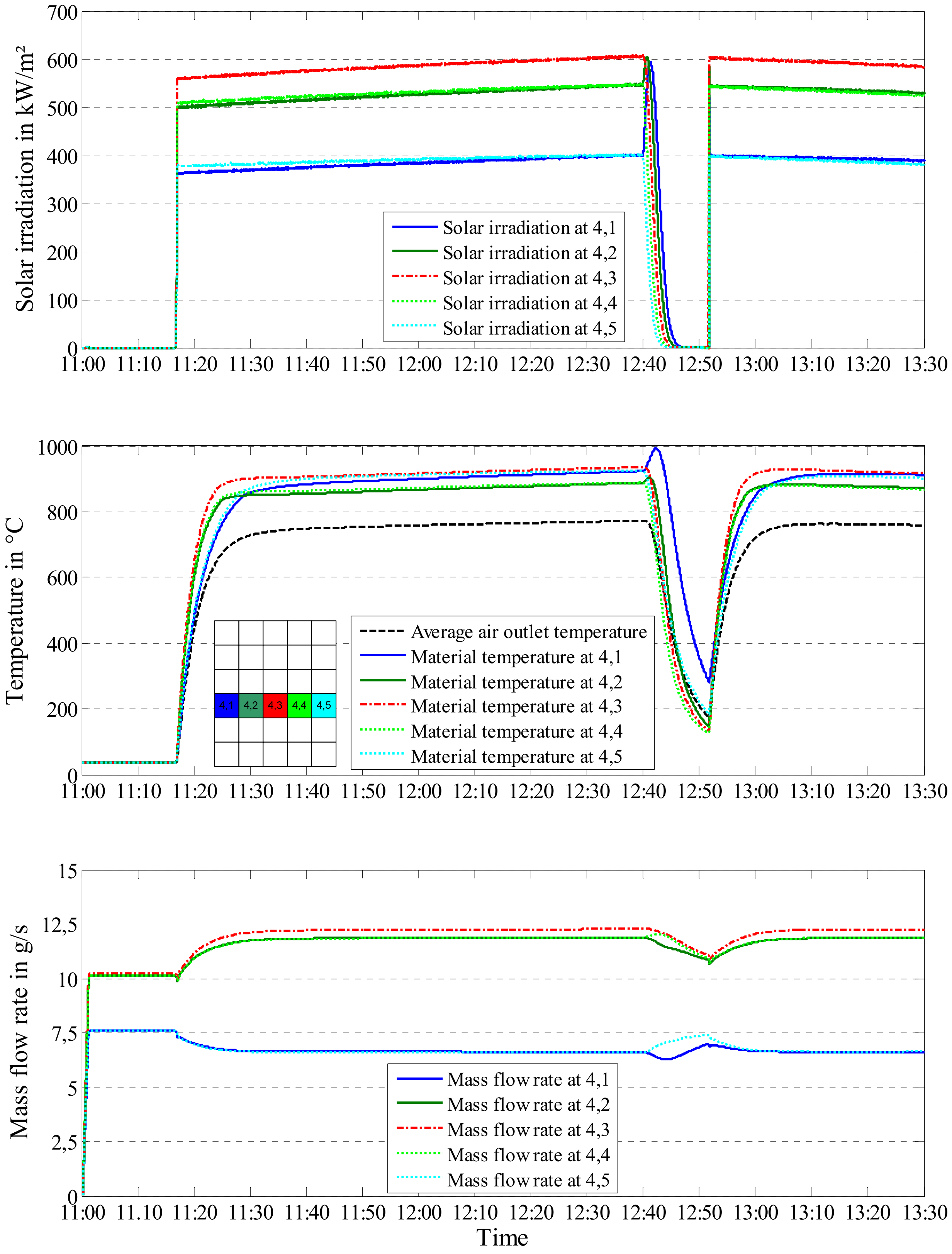


Figure 5: Solar irradiance, absorber material temperatures of five representative absorber modules of an open volumetric air receiver (6x5 total receiver discretization) and the average air outlet temperature, and the corresponding air mass flow rates of the five modules. The absorber modules have different orifice diameters and are located on the eastern side (4, 1) and (4, 2), in the center (4, 3) and on the western side (4, 4) and (4, 5) of the receiver in the fourth row. One module represents a matrix of 3 x 3 absorber modules.

high thermal stresses in the material. Other possible start-up procedures can be analyzed to define a more smooth and conservative start-up behavior.

From 12:40 ongoing, a tracking failure of 12 minutes is simulated. The heliostats remain in a fixed position. The focus of the irradiation moves towards the eastern edge of the receiver. The solar irradiation decreases immediately for the absorber module at the western edge and in the center of the receiver. In contrast, the irradiation on the module at the eastern edge increases with the focus moving towards the eastern edge. The temperatures follow the described behavior with a short time delay. An increase of around 70°C can be seen for the module at the eastern edge within some minutes. The temperature decreases as well, when the focus has shifted to the outside next to the receiver. After 12 minutes, the tracking is reactivated. The heliostats approach the aim point at the center of the receiver very fast. The temperatures increase to their previous level with time delay. The temperature rise is relatively small as the irradiation changes are manageable and a small receiver has been analyzed. In larger receiver, larger differences in mass flow rate and irradiation can occur. Unfortunately, the effects of irradiation changes are intensified.

The mass flow rate of each absorber module is dependent on the orifice diameter of the module and the air density. As the overall mass flow rate is kept constant, the individual mass flow rate of the modules changes with the temperature. The mass flow rates of the outer modules (4, 1) and (4, 5) decrease after the aim point shift, as they have a smaller orifice diameter as the central modules. The effect of rising temperatures is larger.

A similar phenomenon can be seen during the tracking failure. The modules on the eastern side with increasing irradiation show decreasing mass flow rates, the ones on the western side increasing mass flow rates until the focus of irradiation has left the receiver area. However, this time the orifice diameters are not well suited for this situation. Different temperature levels are reached. The material temperature of the module (4, 1) reaches a higher level leading to higher thermal losses. In order to be able to optimize the receiver behavior, adjustable orifices or air flaps could be implemented. That way, the mass flow rate distribution can be adjusted according to the irradiation conditions.

The simulation of a tracking stop was chosen to demonstrate one possible problem and its consequences during operation of a heliostat field. Problems like this can lead to temperature gradients in the receiver, which cannot be neglected. They can even cause great damage. In this specific case, the destructive temperature levels are not reached. It might still be necessary for other configurations to install an emergency defocusing of the heliostat field which still operates in case of an electrical power outage for the heliostat field and in case the blower fails as well.

4 Conclusions

For the first time, transient simulation of a solar tower power plant including heliostat field and a discretized solar receiver is described. A tool coupling was developed linking two simulation environments, namely STRAL representing the heliostat field and Dymola representing the receiver. Additionally, a second coupling was established to link a heliostat field control modeled in the LabView environment.

A detailed receiver model for the transient behavior of an open volumetric air receiver has been developed as one part of a Modelica model library. The structure of the absorber modules as the key components of the receiver is discussed as well as the implementation of the receiver in a Modelica model.

The simulation of a test case, comprising the start-up procedure and a tracking failure of the entire heliostat field, was accomplished. Occasionally large temperature gradients could be detected in some absorber modules of the receiver, which could lead to critical operation conditions. These exemplary results underline the necessity to carry out transient simulations of the entire solar tower power plant, especially during changing operational conditions, as for instance during start-up, failures or cloud passing.

References

[1] Belhomme, B., Pitz-Paal, R., Schwarzbözl, P., Ulmer, S. (2009): A new fast Ray Tracing Tool for High-Precision Simulation of Heliostat Fields, Journal of Solar Energy Engineering, 131 (3), 2009, in Press

[2] Tummescheid, H. (2002): Design and Implementation of Object-Oriented Model Libraries using Modelica. Thesis, Department of Automatic Control, Lund Institute of Technology, Lund, August 2002

[3] Modelica Association (2009): Modelica® - A Unified Object-Oriented Language for Physical Systems Modeling, Language Specifications, Version 3.1, May 27th, 2009

[4] Dynasim AB (2004): Dymola Version 7.1 http://www.dynasim.se

[5] Ahlbrink, N., Alexopoulos, S., Andersson, J., Belhomme, B., Boura, C., Gall, J., Hirsch, T. (2009): vICERP – The virtual Institute of Central Receiver Power Plants: Modeling and Simulation of an Open Volumetric Air Receiver Power Plant. Conference Proceedings, MATMOD Conference 2009, 263, Vienna, February 11-13, 2009

[6] Casella, F., Otter, M., Proelss, K., Richter, C., Tummescheid, H. (2006): The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks. Conference Proceedings, Modelica Conference 2006, 631-640, Vienna, September 4-5 2006

[7] Schmitz, M., Boura, C., Ahlbrink, N., Gall, J., Andersson, J. (2009): Optimized control of a hot –gas cycle for solar thermal power plants. Conference Proceedings, Modelica Conference 2009, Como, September 20-22, 2009, in Press

[8] Garcia, P., Ferriere, A. (2008): Codes for Solar Flux Calculation dedicated to Central Receiver System Application: A comparative Review, Journal of Solar Energy, 3 (2008) 189-197

[9] Hoffschmidt, B. (1997): Vergleichende Bewertung verschiedener Konzepte volumetrischer Strahlungsempfänger. Forschungsbericht / Deutsches Zentrum für Luft- und Raumfahrt e.V.; 1997, 35, Köln: DLR, 1997. 212 S.: ISBN: 1434-8454; Aachen, Technische Hochschule, Diss.; Reportnr.: DLR FB 97 35

[10] Fritzson, P. (2004): Principles of object-oriented modeling and simulation with Modelica 2.1. Wiley-IEEE Press, 2004

Modelling Steam Generators for Sodium Fast Reactor with Modelica

Franck David Annick Souyri
EDF R&D
6 quai Watier 78400 Chatou - France
franck.david@edf.fr annick.souyri@edf.fr

Guillaume Marchais
Altran Technologie
2 rue P Vaillant Couturier – 92300 Levallois Perret - France
guillaume.marchais@altran.com

Abstract

EDF is involved with CEA and AREVA in a common effort for the development of the future nuclear reactor generations. The studies, currently performed by the partners, concentrate on the design of Sodium Fast Reactor types that may include different kinds of innovative circuits and components as compared to the SPX (Super PheniX) plant design.

Based on previous knowledge on SG developed at EDF, with Sodium as hot fluid, and with the help of more recent methods of modeling using the Modelica libraries, a new model for the simulation of steam generator has been developed in order to help the designers of the heat exchangers to meet the requirements for a Sodium Fast Reactor plant design.

The paper will present the current status of the model and a comparison of the results with those of the actual SuperPhenix steam generator database.

Keywords: Power plants and energy conversion systems

1 Introduction

EDF is involved with CEA and AREVA in a common effort for the development of the future nuclear reactor generations. The studies concentrate on the future design of Sodium Fast Reactor types that may include different kinds of innovative circuits and components as compared to the previous SPX (Super PheniX) design.

Among the numerous subjects of interest driven by the project and technical issues, the present paper will focus on the energy conversion system between the reactor core, where liquid sodium is used as primary coolant, and the steam/water loop.

For safety purpose, i.e. to prevent the consequences of an interaction between water and “primary” sodium in case of tube leaks, an intermediate coupling fluid is required. The resulting design includes two heat exchangers (Figure 1) : the intermediate heat exchanger (IEX) and the steam generator (SG). The SG couples the intermediate fluid with water, which goes through a complete change of phase from liquid water to overheated steam. It is to be noticed that, according to the nominal power of the plant and also for safety reasons, the duty of the heat exchangers (IEX and SG) could be supported by a series of heat exchangers, with the same inlet and outlet conditions of temperature and pressure, rather than by big components.

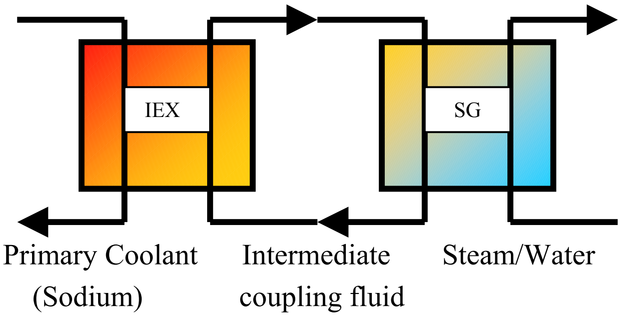


Figure 1 : Global Energy transfer scheme in a Sodium Fast Reactor

Since a final and completed design of the whole system is not decided yet, it was important for EDF/R&D to develop a model with the capability to integrate any kind of evolutions of geometry or operating conditions and to assess their effects on the energy conversion system. Also, this numerical

DOI: 10.3384/ecp09430051

model could offer relevant information for design optimization (especially concerning local temperature behavior on the SG two phase flow side).

The paper presents successively:

- A description of the physical model. This is a 1D model based on equations of conservation for mass, momentum and energy, compliant with transient analysis.

- a comparison of the results of the SG Modelica model with previous studies based on SPX measurements and calculations available at EDF.

- a preliminary assembly of modules to model the global energy conversion system including the connection of two heat exchangers (IEX and SG).

2 Motivation for the development

2.1 EDF's prior experience with Modelica

In order to improve the efficiency of its simulation tools while reducing their cost, EDF has chosen to use when possible, state-of-the-art readily available tools instead of proprietary codes developed by EDF.

The tools used should have open component libraries, be able to perform static and dynamic studies, compute steady states and solve inverse problems. They also should not induce an excessive dependency upon the tool providers. Modelica based tools offer such characteristics. That is why they are considered as good candidates for fulfilling EDF's needs, i.e. modelling and simulation at the system level for the sizing, design verification, validation and operation of its power plants.

In order to evaluate different tools, benchmark cases have been selected, covering the variety of studies made at EDF. Results obtained with our models written with Modelica language are quite satisfactory. Validation studies of the models with Dymola have shown good agreement with previous reference test cases ([1], [2]).

After these first positive results, and also because of the easy use and sharing of the Modelica models with Dymola, EDF decided to develop its own thermofluid Modelica library. The objective is to provide the physical and technological model components needed for steady state and dynamic simulation of power plants under normal and incidental operating conditions. This includes conventional nuclear and thermal power plants, but also future nuclear and thermal plants designs, and systems powered by renewable energy.

The library components must be able to describe single and two-phase flow, with heat transfer when needed, deal with zero and reverse flow, compressible and incompressible flow for water/steam but also other fluids used in specific heat exchangers for example, and smoke networks for thermal power plants.

A Modelica library of 0D and 1D thermal hydraulics components has been developed, based on mass, momentum and energy balance equations, completed with closure equations derived from empirical correlations valid for the operating domain under consideration. 1D models describe steam generator and vapour lines. Empirical correlations (heat transfer, pressure losses), adapted to the physical range of operation of the component, have also been translated into Modelica.

The library uses a finite volume approach, based on the staggered grid scheme for space discretization, and the upwind scheme for the handling of flow reversal [3]. Both schemes are well suited for convection, which is the predominant energy transport law within the network. Discretization is performed along the main flow direction only (1D modelling).

The basic model components are divided into two groups: nodes and edges. Nodes represent mixing volumes such as tanks, boilers, splitters and mergers, etc. They implement the mass and energy balance equations. Edges represent flow resistant elements such as valves, simple pressure loss pipes, etc. They implement the momentum balance equations. The network is built by connecting edges to nodes in order to obtain a complete set of mass, energy and momentum equations with their closure equations, and automatically fulfil the numerical scheme requirements. Complex library components such as heat exchangers, evaporator pipes or steam generator can be either developed as independent components or built by assembling edge and node elements.

It is also important to note that EDF has chosen not to use the Modelica inheritance mechanism, in order to keep the readability of the model: the complete set of equations can be found directly in the component model itself, instead of being scattered throughout the library when they are partially derived from super-classes.

2.2 Specifications for the model

The purpose of the final model will be to represent two heat exchangers connected with an intermediate fluid. The fluids to be considered are sodium liquid for the hot side at a temperature of about 500 °C and boiling water for the cold side at a pressure of about

200 bars. The intermediate fluid could be “classically” sodium liquid as in Phenix or SuperPhenix plants or another fluid to be decided in the future.

The main goal for the model is first to help researchers and pre-designers to assess the performance and the sizes of the different exchangers involved in the process. More over, the requirements for design of pumps will be derived from the calculation of head losses through the different parts of the components.

A second objective assigned to the model is to provide accurate local parameters such as velocity, temperature, pressure, void fraction,… in order to check if the parameters comply well with other concerns like fluid-structure interaction, corrosion and so on.

In a final step, the model will have to calculate off design situations or transients with respect to actual physical phenomenon and could be used in connection with the entire energy conversion system.

3 Model description

3.1 Two generic modules

To meet the above requirements, it was decided to develop separate modules for the fluids and for tube wall between the hot side and the cold side.

For our applications, the heat exchangers components can be considered as one dimensional with counter-flows even though the local flow can be locally perpendicular to the tube arrays.

Concerning the steam/water size, the fluid is supposed to enter at liquid sub-cooled conditions, then is heated and goes through a complete change of phase to become steam at super-heated conditions at the outlet. During the process, due to the phase change, the physical mechanisms that drive the heat transfer and pressure are very complex and differ strongly between the inlet and outlet of the tubes.

The complex mechanisms are represented by empirical correlations depending on geometry, direction of the fluid and local parameters (velocities, pressure, quality..). It appears clearly that it is necessary to have a local description of the average physical parameters all along the tubes to represent the actual physics.

As a result, hot side, tube wall and cold side modules are connected together, and represent a “cell” with a limited length (figure 2). The cells can then be connected together to form the whole length of the global heat exchanger.

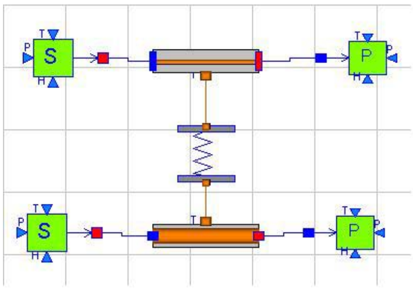


Figure 2 : generic cell with two modules for fluids and one module for the tube

The physical models employed should be able to calculate one phase or two-phase flows within the fluid module and heat transfers through the tube wall.

For a fluid module, the 1D model is based on a classic set of equations for the conservation of mass, momentum and energy (as given in equations 1,2 and 3).

$$\frac{\partial \rho_m}{\partial t} + \frac{\partial}{\partial l}\left(\rho_m . \mathrm{V}_m\right) = 0 \qquad (1)$$

$$\frac{\partial \rho_m V_m}{\partial t} + \frac{\partial}{\partial l}\left[\rho_m V_m^2 + \alpha(1-\alpha)\frac{\rho_l \rho_v}{\rho_m} V_{vl}^2\right] + \frac{\partial P}{\partial l} - F_m - \rho_m . \mathrm{g} = 0 \qquad (2)$$

$$\frac{\partial \rho_m H_m}{\partial t} + \frac{\partial}{\partial l}\left[\rho_m V_m H_m + \alpha(1-\alpha)\frac{\rho_l \rho_v}{\rho_m} L V_{vl}\right] - \frac{\partial P}{\partial t} - \frac{\partial P}{\partial l}\left[V_m + \alpha(1-\alpha)\frac{\rho_l - \rho_v}{\rho_m} V_{vl}\right] - \Phi_m = 0 \qquad (3)$$

where :

- ρ V, H and P are respectively the density, velocity, enthalpy and pressure of the fluid. The fluid is considered as an homogenous mixture (subscript m) in case of two-phase flow conditions.

- α is the void fraction (ratio of volume occupied by steam on total volume of fluid). Moreover, we have the relation : $\rho_m = \alpha \rho_v + (1-\alpha)\rho_l$ (4)

where v and l indicate respectively the vapor and liquid phase

- L is the difference of enthalpy between vapor and liquid at saturation condition for a given pressure

- V_{vl} is the difference of velocity between the steam and the liquid that results from different forces acting on each individual phase

- F_m is set for the sum of forces (friction, viscosity, obstacles..) acting on the mixture; g is the constant of gravity.

- Φ_m represents the heat transfers between the fluid and the tube walls. It can be expressed by the following equation :

$$\Phi_m = \frac{4}{d_{th}}\left[h_{tm}\left(T_t - T_m\right)\right] \tag{5}$$

where T_t and T_m are respectively the temperatures of the tube and the fluid; d_{th} is the thermal diameter and h_{tm} is the heat exchange coefficient between tube the tube and the fluid.

The 3 main unknowns solved by the solver from this 3 equations are, after a combination of equations, the pressure P, the dynamic enthalpy H_m and the mass flow rate $Q_m=\rho_m V_m$.

To characterize the two-phase mixture, it is assumed that the phases are always at thermodynamic equilibrium and that the relative velocity V_{vl} is known thanks to a relevant empirical correlation.

The terms F_m is given by an empirical correlations depending on geometry and local fluid parameters.

The tube module includes a unique equation of energy balance in the tube wall (see equation (6)). The equation links the time dependent variation of the tube wall temperature with the surface heat fluxes on both sides of the tube.

$$\rho_t . Cp_t . \frac{\partial T_t}{\partial t} = -\Phi_1 - \Phi_2 \tag{6}$$

where Cp_t is the thermal capacity of the tube and Φ_1 and Φ_2 the surface heat fluxes which can be written as equation (5).

It is necessary to take account of the thermal conductivity of the tube. This is made by means of the following relation (to be used for in-tube side):

$$h_{tm} = \frac{h_{tw}}{1 + \mathrm{A}.h_{tw}} \text{ with } \mathrm{A} = \frac{D_i}{2.\lambda_t}.Ln\left(\frac{D_i + e}{D_i}\right). \tag{7}$$

where D_i is the internal diameter of the tube and e its thickness; h_{tw} is the heat exchange coefficient on tube wall and λ_t the conductivity. A quite similar formula adapted to outer tube sides is also required.

Finally, the thermal properties of the fluids and materials are given either by the thermodynamic tables in the standard version of Modelica or by specific models.

This set of equations, with the complementary assumptions, was successfully used for SG modeling with Modelica in PWR applications [1]. Furthermore, the model is also widely employed at EDF in a 3D formulation to calculate two-phase flows in SG or reactor cores in PWR plants with a CFD component code [4].

3.2 Closure laws

All the correlations used in our model to calculate friction factors or heat exchange coefficients are taken from relatively old EDF's references [5]. Nevertheless the set of correlations used was adapted to represent correctly the physics in a SG heated by sodium liquid.

In this particular design, the exchanger consisted of helically coiled tubes and the correlations take into account these unusual features. The one phase sodium flows quite perpendicularly across the tube bundle. One phase or two-phase steam water flows inside tubes that are curved.

Since we didn't find any indication about relative velocities between the phases, the V_{vl} term is set to 0.

3.3 Input/output of the modules

The connections between the modules are as follow:

- Inlet connector of fluid modules consists of enthalpy and mass flow rate; outlet connector includes the pressure variable.
- The connection of tube wall module with fluid module exchanges the average tube temperature and the surface heat fluxes.

4 Application to a Steam Generator of Super Phenix

The objective of the work presented below was to validate the capability for the model to represent correctly the behavior of an helically coiled steam generator heated with sodium.

To be more precise, one must check the correct implementation of the modules, the reliability of correlations by comparing the calculations with test results or calculations made with an older code that is no longer available.

The database for the comparisons comes directly from the test campaign performed on the SG "D" of SPX plant at different power levels. Moreover, we used the results of an old code to compare temperature and pressure profiles for one phase sodium flow and two-phase steam water flow.

4.1 Geometrical features and operating conditions

A general view of the SG design is given in figure 3. The SPX Steam Generator was an helically coiled tubes heat exchanger. The sodium was flowing downwards on the shell side whereas the water was flowing upwards and was boiling inside the tubes of Incoloy 800.

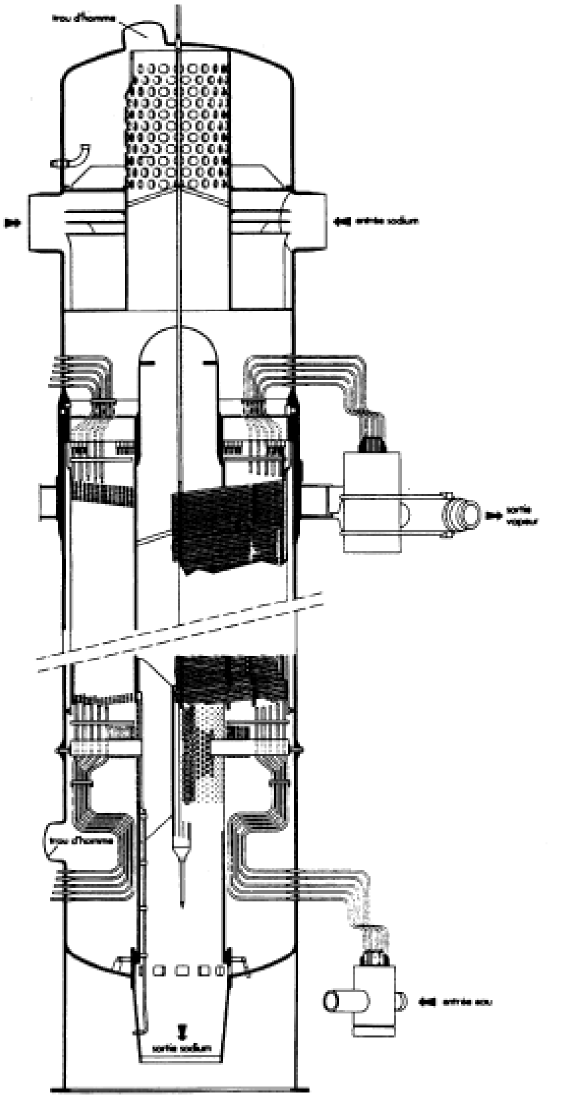


Figure 3 : Plan of SPX Steam Generator

Table 1 : Tube bundle data

Inner diameter	19,8 mm
Outer diameter	25,0 mm
Coil 1st diameter	1,17 m
Coil pitch	45 mm
Coil number	17
Tube number	357
Helical angle	7,45 °

The bundle was contained in an annulus of inner diameter 1,125 m and external diameter 2,655 m.

Main characteristics and layout of the bundles are presented Table1. The tube bundle is completed with straight and curved parts to be connected to water supply and steam outlet pipes.

The heated part covers a height of about 14,9 m and corresponds to an average tube length of 90,9 m.

Operating conditions at nominal power during the tests are presented Table 2.

Table 2 : Operating conditions (measured) near 100% of nominal power

Power exchanged	724 MW
Sodium inlet temperature	518 °C
Sodium outlet temperature	343 °C
Sodium mass flow rate	3249 kg/s
Water inlet temperature	233,8 °C
Water outlet temperature	489,8 °C
Water mass flow rate	331 kg/s
Steam outlet pressure	185,4 bar

4.2 Calculation results and comparison with tests

Beyond the instrumentation dedicated to global performances and inlet and oultet flow parameters, local instrumentation (60 thermocouples) was set at different levels in the sodium side and at the outlet of steam water side. The measurements were performed by Novatome during 1986.

The two categories of measurements provide global and local information and are used in the following chapter for comparison with calculation results. Nevertheless, local steam temperature measurements were not documented in EDF's reports and can not be used.

However, global calculations and profiles are compared with EDF's older code that was validated with the present data and with a 45 MW experiment [6] having similar geometrical features.

For our simulations, a number of 20 "cells" was used to represent the whole length of the tubes. That number is a balance between the complexity of the model and its capability to "catch" the changes of phase and local physical behavior.

Moreover, a specific arrangement was used to take into account a by pass of sodium at the bottom of the bundle (see detail Figure 4).

For the two fluids, input data are temperature and mass flow rate at the inlet and pressure at the outlet.

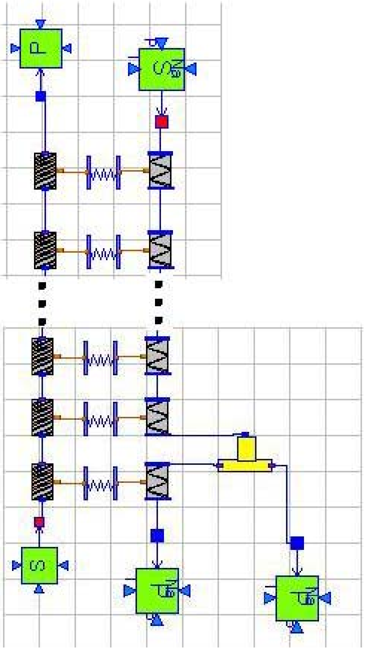


Figure 4 : sodium bypass arrangement

Calculation results at nominal power :

The first comparison was made for a thermal power exchanged (724 MW) very close to the designed nominal power (750 MW). It was necessary to slightly adjust the calculated thermal power that was 2% under the expected one. This was achieved by tuning the conductivity of tube material.

Figures 5 and 6 present respectively the profiles of temperatures and steam water pressure calculated along the tubes with EDF's previous code. The dots that appear in Figure 5 come from temperature thermocouples located at different levels in the shell side (i.e. where the sodium liquid flows). The comparison was satisfactory, so we can rely on calculations with the previous code.

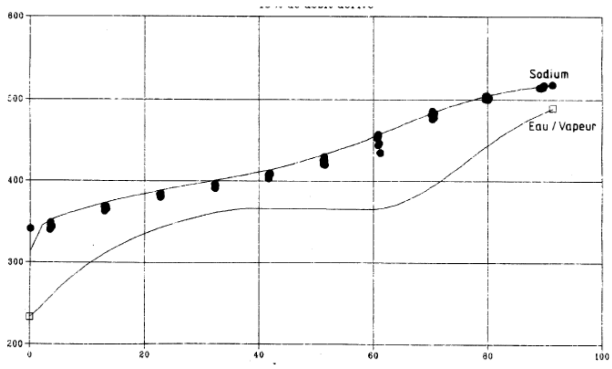


Figure 5 : temperature profiles (upper curve is sodium, lower curve is steam water) – calculated with previous code

The steam/water temperature profile shows the different parts where the fluid is liquid, two-phase or vapor. The minimum of temperature pinch appears at the upper part of the tubes but near the beginning of boiling section the temperature pinch is also narrow.

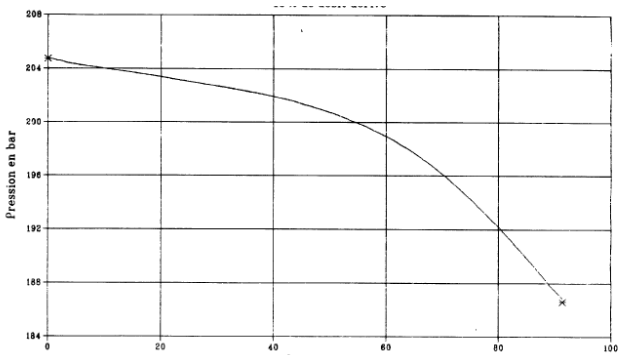


Figure 6 : Pressure profile on steam/water side, calculated with previous code

Unsurprisingly, the pressure profile on Figure 6 shows different slopes depending on whether one area is liquid (left hand side), two-phase (center) or vapor (right hand side).

Figures 7 and 8 present the calculation results of temperatures and steam/water pressure profiles with our Modelica current modeling.

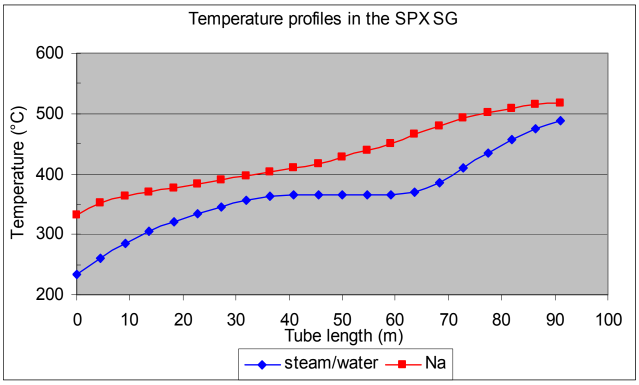


Figure 7 : temperature profiles along the tubes with Modela model

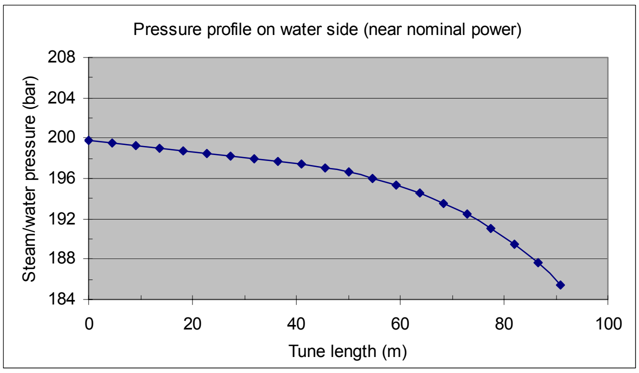


Figure 8 : pressure profile along the tubes with Modela model

As it can be seen on figures 5 and 7, the temperature profiles compares perfectly. For the two simulations, beginning and end of boiling sections are located at the same tube heights.

Although the shapes of pressure profiles (figure 6 and 8) are quite similar, there is a significant difference between the values of pressure drops of about 2,5 bar. The difference is concentrated in the upper part of the tubes at high two-phase flow quality and beyond. That may be due to the relative velocity set to 0 in our model or friction factors in two-phase flow.

This difference will have to be investigated in a future work.

Results at intermediate power exchanged (50% and 25% of nominal power):

Two other calculations were made with the same physical assumptions and closure relation ships for two thermal powers of 388 and 151 MW.

The calculations compare very well with the previous ones : differences of power exchanged remain below 1,5%.

5 Toward the modeling of a complete EI/SG system

The calculation results presented in chapter 4 are very satisfying. It is now possible to move with confidence to a more complete modeling of the coupled system with two heat exchangers (EI and SG) as discussed in chapter 1.

Main objective for this work is first to bring out a first geometrical design that complies with plant requirements. More over, it is suggested to study other fluids than sodium as coupling fluids.

Figure 9 presents a possible and very preliminary architecture that couples an Intermediate Heat Exchanger with a Steam Generator. The intermediate fluid can be chosen by entering the right thermo-physical properties in the modules.

To perform this work, all useful parameters for the geometrical design are lumped in a specific “Geometry” module. Each generic module will find current geometrical parameters values in the “Geometry” module. Moreover, the “Geometry” module contains all the specific and coherent calculations between the geometry features and it is used to help the designer to find out geometrical parameters if another one is specified.

Once all the geometric details have been set for the model, the calculation allows to check the technical constraints (total power exchanged, pressure drops, local parameters such as temperatures or velocities..).

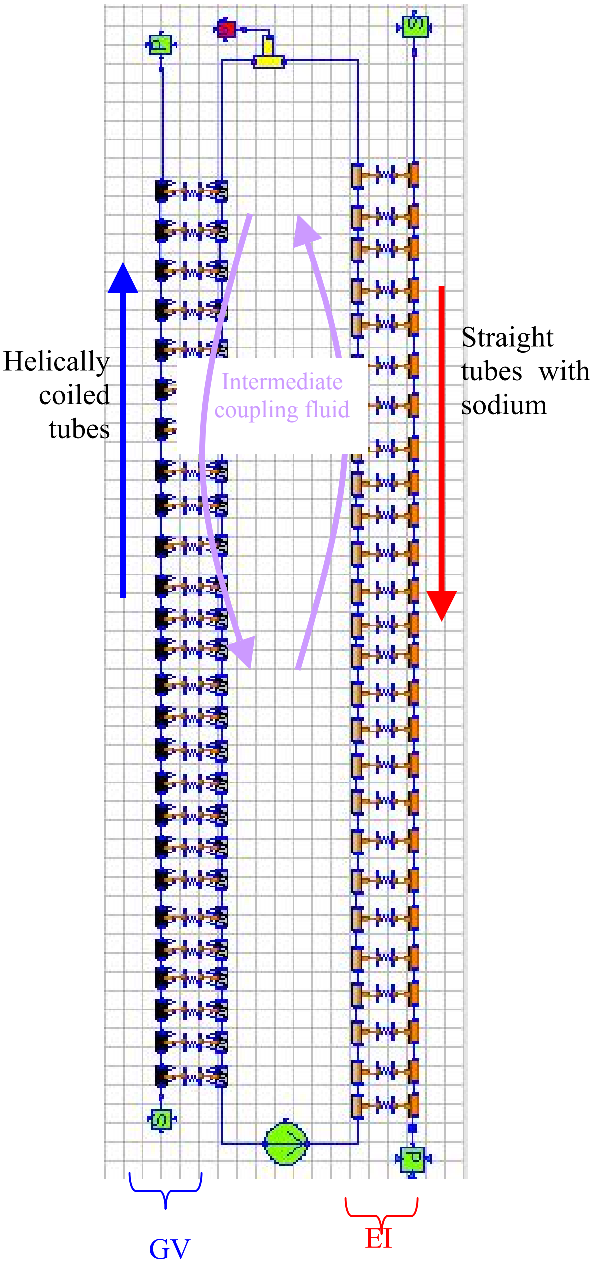


Figure 9 : architecture of coupled heat exchangers

6 Conclusions

In the framework of the studies for development of future Sodium Fast Reactor, the Modelica library is used, at EDF/R&D, to model the systems of heat exchangers coupling the primary loop with sodium to the steam/water flows by means of an intermediate coupling fluid.

The work presented here is focused on the development of a physical model to represent a Steam Generator made of helically coiled tubes and heated with liquid sodium.

The set of equations, the closure laws and their correlations form a comprehensive model that shows its capability to calculate correctly the global heat exchanges and the temperatures and pressure profiles along the tubes, as compared to SuperPhenix Steam Generator database and previous modeling.

On this basis, the model will be used in near future to participate in the design of the complete and coupled system including an Intermediate Heat Exchanger and a Steam Generator.

References

[1] Avenas C. et al, “Quasi-2D Steam Generator Modelling with Modelica”, ISC’2004, Malaga, Spain.

[2] Souyri A. et al., “Pressurized Water Reactor Modelling with Modelica”, 6th international Modelica Conference, March 3-4, 2008, Bielefeld, Germany

[3] Patankar S.V., “Numerical Heat Transfer and Fluid Flow”, Hemisphere Publishing Corporation, 1980.

[4] David F., “Three dimensional thermal-hydraulic simulation in steam generators with THYC Exchanger code – Application to the UTSG model 73/19”, NURETH9, October 3-8, San Francisco, Californie

[5] Fontaine et al., “Synthèse d’essais de maquettes d’un générateur de vapeur chauffé au sodium” , Nuclear energy maturity, Paris, 1975,716-737

[6] Lecoeuvre JM et al., “Numerical simulation of operation of the model for the Super-Phenix steam generator in non-stationary regime” , EDF, Bulletin de la Dierction des Etudes et Recherches, Serie A Nucleaire, Hydraulique, Thermique N°3, 1981, pp 27-33

Simulation of the dynamic behaviour of steam turbines with Modelica

Juergen Birnbaum[a], Markus Joecker[b], Kilian Link[a], Robert Pitz-Paal[c], Franziska Toni[a], Gerta Zimmer[d]
[a]Siemens AG, Energy Sector, Erlangen, Germany
[b]Siemens AG, Energy Sector, Finspång, Sweden
[c]German Aerospace Center, Institute of Technical Thermodynamics, Cologne, Germany
[d] Siemens AG, Energy Sector, Muelheim, Germany
juergen.jb.birnbaum@siemens.com, markus.joecker@siemens.com, kilian.link@siemens.com, robert.pitz-paal@dlr.de, franziska.toni@cbi.uni-erlangen.de, gerta.zimmer@siemens.com

Abstract

Steam turbine technology is one of the leading technologies used in electricity production since more than one hundred years. In recent time requirements for steam turbines have been changing slowly. Steam turbines are not longer used in power plants with high operation times and a high full load share only, but are also implemented in combined cycle power plants or solar thermal power plants. This type of plants requires good dynamic behavior of the steam turbine due to fast and frequent start ups and daily cycling.

To optimize the performance of this kind of power plants and their components it is necessary to simulate and analyze their dynamic behavior. Therefore, a general model approach for steam turbines within Modelica has been developed. This model approach is based on a general model, which can be adjusted to the necessary model depth as described in this paper.

Steam turbines in a solar thermal power plant with direct steam generation must fulfill special requirements regarding their dynamic behavior. Hence, this model is applied as an example to explain the behavior of an industrial steam turbine used in such plants. Furthermore, this paper shows first results of simulations with turbine models. To validate the model, the results are compared with results from the Siemens internal steady state calculation tool. Since results stay within the estimated accuracy, the model approach can be used for further calculations.

The dynamic behavior of the turbine is analyzed by using typical solar irradiance disturbances. This analysis shows that no critical operation points occur within the turbine.

Keywords: solar thermal power plant, dynamic turbine behavior, turbine modeling

1 Introduction

Steam turbines are typically used in different types of power plants (e.g. fossil fired steam power plants, nuclear power plants, combined cycle power plants, solar thermal or biomass power plants) with different requirements regarding their dynamic behavior.

Generally, a fossil fired steam power plant is operating more than 7000h a year with the rated power output. Therefore, the dynamic behavior of the steam turbine is not essential for this kind of plants. Combined cycle power plants normally operate in middle load, hence the dynamic behavior of the steam turbine is of main importance for the start-up procedure. This aspect gained even more significance since, due to the new grid requirements arising from renewable electricity generation, combined cycle power plants are also used for peak load supply.

The value of the dynamic behavior of steam turbines in a solar thermal power plant has been even rising, since daily cycling, fast start-up behavior and good transient behavior have become essential requirements. The dynamic behavior of the solar field and of the power block, especially the steam turbine and their interaction to bring solar thermal power plants with direct steam generation into the market, are questions still to be answered. As an appropriate solution, the layout of such a plant has been analyzed and optimized in a first step as described in [1] and [2]. Further steps will be a dynamic analysis of the solar field, of the turbine itself and their interactions. Parts of this analysis are already done and described in [3]. This paper addresses the modeling of such a steam turbine and provides a first analysis of its dynamic behavior.

DOI: 10.3384/ecp09430055

2 Plant configuration

The basic layout of the analyzed solar thermal power plant with direct steam generation is shown in figure 1. It consists of a solar field, the power block based on a Rankine-cycle and an optional thermal storage system.

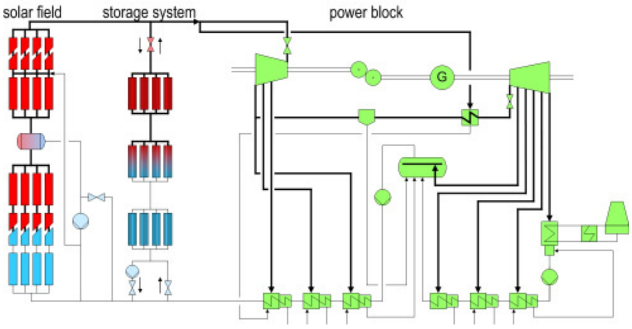


figure 1: 50MW$_{el}$ parabolic trough power plant with direct steam generation

During the ITES project different power plant configurations have been designed [2]. One of these configurations, a plant layout for 50MW$_{el}$ with main steam parameters of 500°C and 110bar, has been chosen for the modeling of the steam turbine and the analysis of its dynamic behavior.

The solar field is divided into four subfields, each with an evaporator and a superheater section, and is operated in recirculation mode. [1] provides a detailed description of the solar field layout. The power block contains a steam turbine, which is divided into a Hp- and a Lp- turbine, a feed water preheater section with six preheaters, the feed water pumps, the feed water tank and a wet cooling tower. The steam between the turbines is reheated again through a steam-steam reheater with condensation. This study does not focus on the optional thermal storage system which will, therefore, not be explained within this paper.

3 Modeling

A general model has been developed to show the dynamic behavior of a "solar" steam turbine. This general model can be adjusted to different turbine types and the necessary model depth. In this example, the "solar" steam turbine for the above described plant layout is modeled within Modelica. The validation of the model approach is done through a comparison of the simulation results of this model with results from stationary calculations by a Siemens inhouse tool.

3.1 Modeling approach

Steam turbines consist of different components, which are casing and shaft with the blading. Depending on the question to answer, the steam turbine has to be modeled with a certain level of detail. Therefore, the model has been designed in a way that allows to simulate a single blading group as well as the turbine as a whole.

The basic model, shown in figure 2, has been built-on a turbine section, an inlet and outlet volume and the flange. The masses for the casing and the shaft are considered in corresponding wall models.

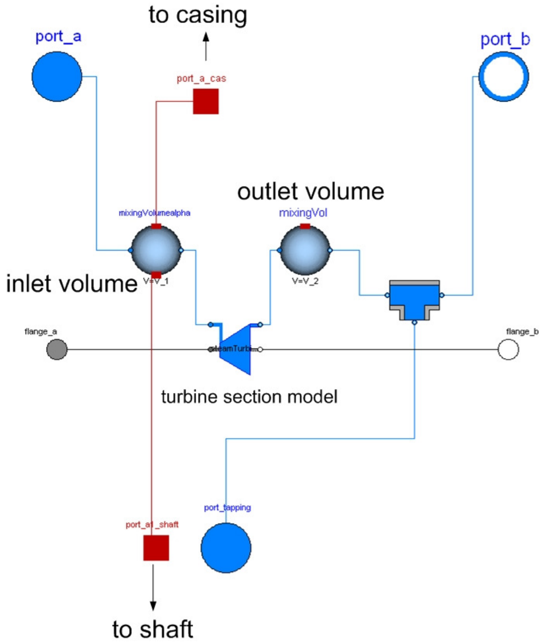


figure 2: basic steam turbine model

The mass flow through the turbine section is calculated by using Stodola's law [4]. Heat transfer between the steam and the casing respectively the shaft is calculated at the inlet volume. The level of detail for the basic model is predefined depending on the initial values for pressure and enthalpy and the geometric inputs (e.g. one basic model to model the Hp-turbine).

3.2 Modeling of a "solar" steam turbine

Figure 1 shows a schematic drawing of a parabolic trough power plant. However, to analyze the dynamic behavior of the steam turbine, it is not necessary to simulate the whole power plant as long as reasonable data for disturbances at the boundaries is available.

Since enough data for disturbances around the "solar" turbine had been available, reasonable boundaries have been chosen (figure 3). The solar field input and the mass flow at the extractions have been chosen as boundary conditions for the Hp-turbine. Similar boundary conditions have been selected for the Lp-turbine, where, additionally, the condenser backpressure has been selected. The mass flow at the condensate side of the reheater is suitable as reheater boundary condition.

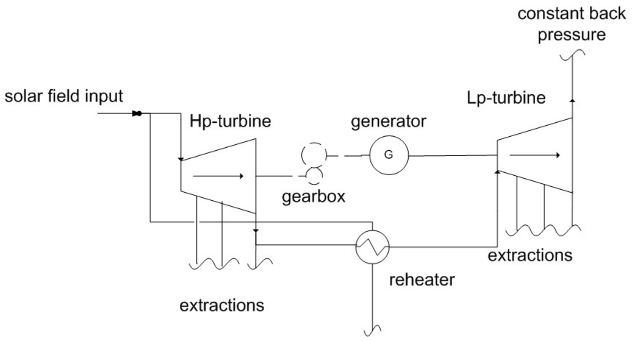


figure 3: reasonable boundaries for the turbine model

Due to the chosen boundary conditions a reasonable design for the turbine model has to be applied which is dividing each turbine into different sections. Each of these sections is described by the general model. Three sections have been selected for the Hp-turbine and five sections for the Lp-turbine. Each section is defined from turbine inlet respectively turbine extraction to the following turbine extraction. Figure 4 shows the overall turbine model and the division of the Hp-turbine into the three suitable sections.

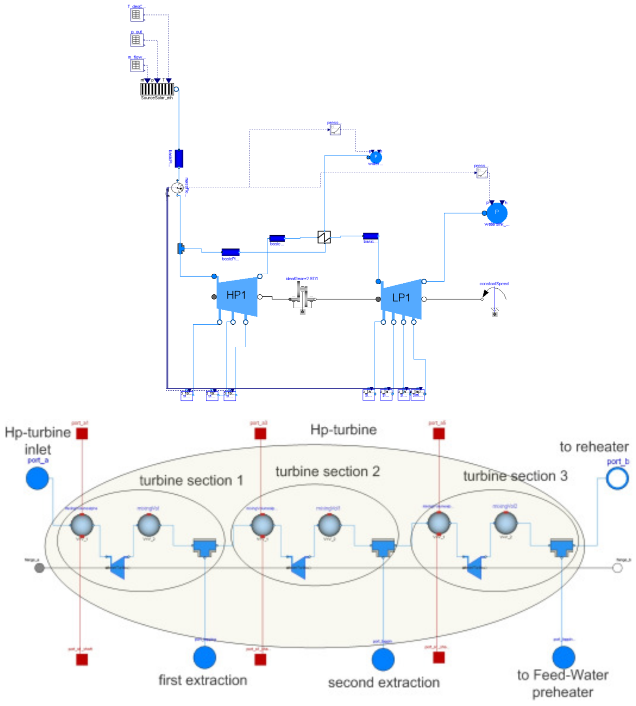


figure 4: Overall turbine model and division of the Hp-turbine into suitable turbine sections

3.3 Validation of the model

In order to validate this model, results of the stationary calculations by the internal Siemens tool have been recalculated with Dymola/Modelica. Typically, load cases from 100% load down to 40% load in steps of 20% are calculated for this analysis. The results for these load cases of both tools were compared at the inlet of the different turbine sections and the outlet of the Lp-turbine.

The relative failure in the calculated mass flow through each turbine section is within +1.3% and –0.1%; for the enthalpy it is within +0.3% and -0.2% for all calculated load cases. Regarding pressure calculation, the relative failure stays within the same range except for the Lp-turbine inlet, leading to a failure in the pressure calculation in every modeled section of the Lp-turbine. Within those, the relative failure varies over all load cases between +6.5% and -0.1%.

This failure occurs from different calculation methods used for the calculation of the pressure loss over the reheater within the tools. In the Dymola/Modelica model the pressure loss over the reheater is calculated according to a given geometry. In contrast, within the Siemens internal calculation tool the pressure loss in part load is calculated with an approximation considering the design pressure loss, the design mass flow and the actual mass flow.

Exemplary the results for the 80% load case is shown in table 1.

table 1: Comparison of the calculation results for 80% load (turbine with reheat)

rel. Failure	**p in %**	**h in %**	**m_{flow} in %**
Hp-inlet	0.192	0.035	-0.004
1. extraction	0.474	0.055	0.023
2. extraction	0.495	0.061	0.049
Lp-inlet	2.225	0.094	0.434
3. extraction	5.706	0.292	0.434
4. extraction	5.953	0.288	0.459
5. extraction	6.081	0.273	0.494
6. extraction	0.486	-0.048	0.559
Lp-outlet	-0.156	-0.032	0.677

A comparison of the pressure failure calculation with the failure calculations of the enthalpy and the mass flow shows a relatively high difference between the calculated pressure in Modelica and the Siemens inhouse tool. Due to this significant difference, a turbine model without reheater has been analyzed in the same way as described above.

The relative failure of the mass flow and the enthalpy calculation were in the same range as the relative failure calculated for these properties for the turbine design that has been analyzed first. The relative failure of the pressure calculation over the Lp-turbine decreased to values between +0.1% and -1%. Hence, the different calculation methods used for pressure loss over the reheater could be determined as the cause for the relatively high failure in pressure calculation (table 2).

table 2: Comparison of the calculation results for 80% load (turbine without reheat)

rel. Failure	p in %	h in %	m_{flow} in %
Hp-inlet	0.034	0.000	0.000
1. extraction	-0.065	-0.006	-0.030
2. extraction	-0.124	0.035	-0.039
Lp-inlet	-0.017	0.027	-0.021
3. extraction	-0.092	0.023	-0.025
4. extraction	0.014	0.022	-0.023
5. extraction	-0.067	0.022	-0.017
6. extraction	-0.137	0.022	0.001
Lp-outlet	-0.084	0.072	0.001

Taking into account the different calculation methods for the pressure loss of the reheater, the comparison of results between the two calculation tools shows a very good accuracy of the results. The turbine model within Dymola/Modelica can, therefore, be used for further calculations. However, it should be kept in mind, that the pressure loss calculation of any component used within the turbine model must first be analyzed separately.

4 Simulation of the dynamic behavior of a “solar” steam turbine

The simulation of the dynamic behavior of the turbine is done for typical solar disturbances. These disturbances are resulting in main steam parameter disturbances. Within the ITES-Project, the German Aerospace Center (DLR) is simulating the dynamic behavior of the solar field for such power plants [3]. The solar field outlet data (mass flow, temperature, enthalpy), calculated by the DLR, are used as input data for a simulation of the dynamic behavior of the steam turbine. The typical disturbance used for this analysis is a step disturbance in the solar radiation from 550W/m² down to 275W/m² over 600s (figure 5).

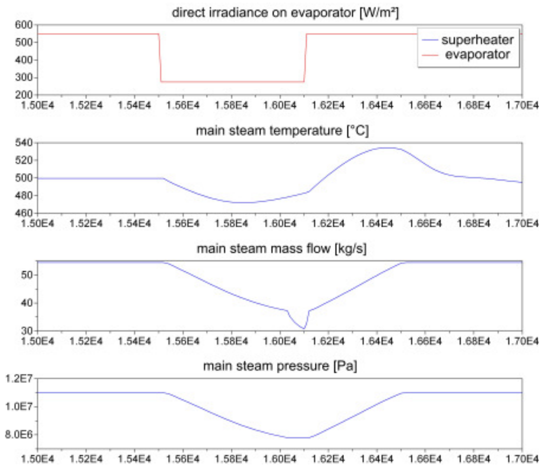


figure 5: typical irradiance disturbance and the corresponding solar field behavior

To analyze the simulated dynamic behavior of the turbine the standard specification for steam turbines IEC 60045-1 is used [5].

4.1 Steam temperature limits

Looking at the steam temperature behavior of the modeled Hp- and Lp-turbine sections one can clearly see that the temperature disturbance is softened over the turbine especially over the reheater (figure 6). This behavior occurrs due to the thermal inertia of the turbine and the reheater. Therefore, the steam temperature limits are analyzed only for the Hp-turbine inlet, where the highest temperature stress occurs.

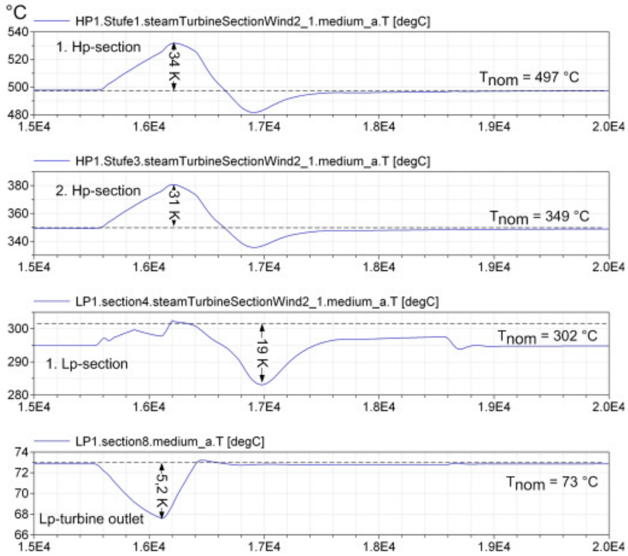


figure 6: steam temperature behavior simulated for different modeled turbine sections

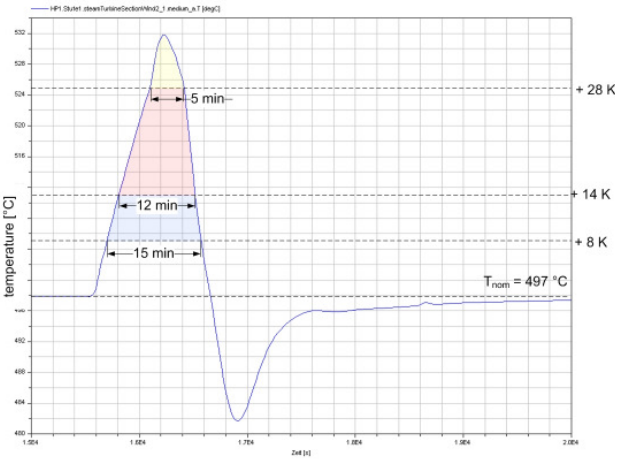


figure 7: excess steam temperature at the first modeled Hp-section

A detailed analysis at the Hp-inlet shows that temperature limits of the turbine are significantly overshot (figure 7). The temperature limits +8K and +14K over rated temperature must be analyzed on a yearly basis, as done in [3], to establish, whether the specified limits have been exceeded. The temperature limit of +28K is an absolute temperature limit. Overshooting this temperature limit will normally lead to a turbine trip influencing the steady electricity production, or reducing the expected lifetime of the turbine. As this limit has been exceeded for 5 minutes, this would already lead to a trip or significant reduction in turbine lifetime.

4.2 Temperature differences between steam and casing respectively shaft

Another limit of the turbine, which has to be fulfilled in operation, is the temperature difference between steam and casing respectively shaft. In case the temperature difference is exceeding a certain degree, the thermal stresses within the turbine will get too high which will lead to a reduced lifetime of the turbine.

The temperature difference between the steam and the shaft respectively casing is analyzed for the Hp-inlet (figure 8) and the Lp-inlet (figure 9). The difference between the steam temperature and the shaft temperature of the first Hp-section as well as the first Lp-section is below 0.4K. Like the shaft temperature, the temperature of the casing at the first Hp-section is nearly exactly following the steam temperature. The temperature of the first Lp-section casing for the analyzed disturbance is ~5.4K lower than the steam temperature. The bigger mass of the Lp-casing compared to the Hp-casing and the shafts of the Hp- and Lp-turbines mainly cause this temperature difference. However, all analyzed temperature differences are well below their specified limits.

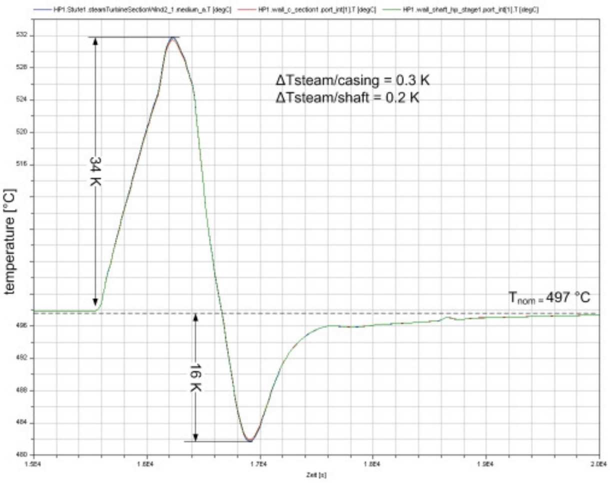


figure 8: temperature difference between steam and casing respectively shaft at first Hp-section

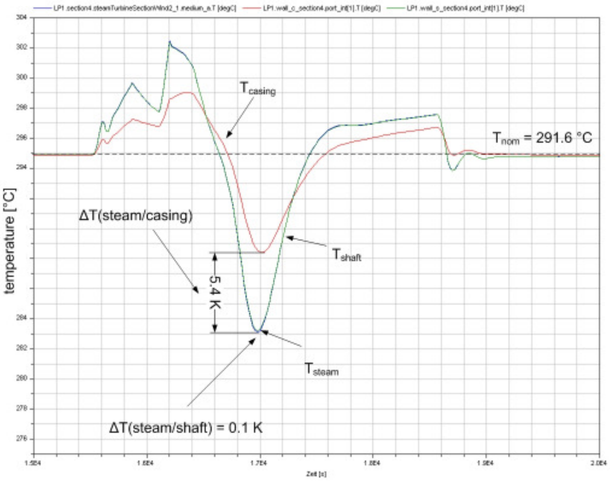


figure 9: temperature difference between steam and casing respectively shaft at first Lp-section

4.3 Temperature distribution within the casing

Another characteristic for the dynamic behavior of the turbine is the temperature distribution in the casing. If the temperature gradient from the inner to the outer wall of the casing is too big, the thermal stress within the casing will exceed its limits, which will again lead to a reduced lifetime of the turbine.

Since the highest temperature stress has been simulated for the first turbine section, the temperature distribution within the casing is analyzed for this section. Therefore, the casing is divided into six layers each with the same mass from the inner to the outer wall.

Figure 10 shows the maximal temperature difference of 34K between the inner and the outer wall. The maximal tolerable temperature gradient, previously determined during first approximations, is ~50K. The determined temperature gradient stays, therefore, well within these limits.

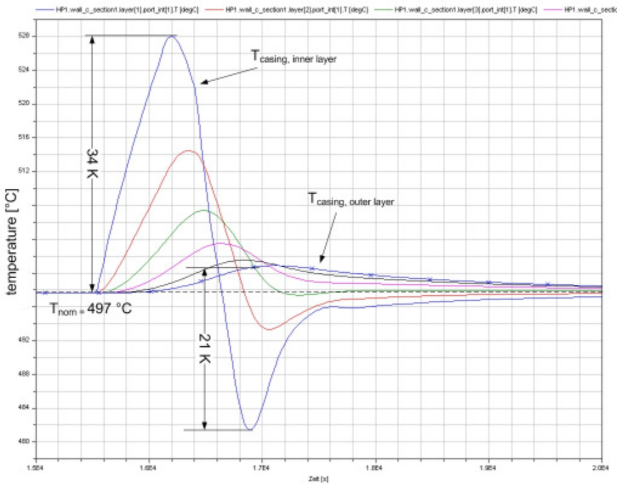


figure 10: temperature gradient within the casing of the first Hp-section (6 wall layers)

4.4 Pressure limits

The standard specification of the pressure disturbances for steam turbines defines only maximal pressure limits [5]. The main steam pressure for the analyzed disturbance coming from the solar field is never exceeding the rated pressure (figure 7). Since in the solar field no overpressure has been simulated it is not necessary to analyze the turbine regarding its pressure behavior.

5 Conclusions

This paper presents a model approach for the simulation of the dynamic behavior of steam turbines. For the purposes of an industrial steam turbine within a solar thermal power plant with direct steam generation the model approach has been compared with stationary calculations for different load points. This comparison revealed a very high accuracy of Dymola/Modelica simulations results, hence, this model can be used for further calculations.

The analysis of the dynamic behavior of the turbine within a solar thermal power plant shows that most of the typical limits for steam turbines have not been exceeded. The limit, which in fact has been exceeded, is the absolute temperature limit at the Hp-turbine inlet. One possibility to control the main steam temperature within its allowed limits is to optimize the solar field control strategy. Another possibility has been evaluated within the ITES-project, where a short time storage had been integrated into the main steam path. The results of this analysis are published in [3].

Acknowledgements

The authors would like to thank the German Ministry for the Environment, Nature Conservation and Nuclear Safety for the financial support given to the ITES project (contract No 16UM0064).

References

[1] Birnbaum J., Eck M., et al. A Direct Steam Generation Solar Power Plant with Integrated Thermal Storage. Las Vegas, USA: 14th Bienial SolarPACES Symposium, 2008.

[2] Birnbaum J., Hirsch T., et al. A Concept for Future Parabolic Trough Based Solar Thermal Power Plants. Berlin, Germany: 15th International Conference on the Properties of Water and Steam, 2008.

[3] Birnbaum J., Feldhoff J., et al. Steam Temperature Stability in a Direct Steam Generation Solar Power Plant. Berlin, Germany: 15th Bienial SolarPACES Symposium, 2009.

[4] Traupel W. Thermische Turbomaschinen II., 3. edition. Berlin, Heidelberg, New York: Springer Verlag, 1982.

[5] International Electronic Commission (IEC) Steam turbines – Part 1: Specifications, IEC 60045-1. Geneva, 1991

System-Level Modeling of an ICE-powered Vehicle with Thermoelectric Waste-Heat-Utilization

Thomas Braig Jörg Ungethüm
German Aerospace Center (DLR), Institute of Vehicle Concepts
Pfaffenwaldring 38-40, 70569 Stuttgart
thomas.braig@dlr.de joerg.ungethuem@dlr.de

Abstract

The reduction of automotive fuel consumption is a major challenge of automotive OEMs and suppliers. It is expected, that during the next decade the majority of vehicles will still be driven by internal combustion engines (ICE). As a rule of thumb about 2/3 of the fuel energy fed to the engine is converted into heat and is used only occasionally and only partly for heating the interior. One promising technology to convert exhaust heat into usable electric energy is the thermoelectric generator (TEG) [1]. As this component has significant retroactive effects on the automotive system, a system-level model of the car was developed to calculate the net benefit of the TEG for steady-state operating-points as well as for dynamic driving cycles.

This paper presents a model and simulation results of an ICE-powered vehicle with thermoelectric waste-heat-utilization. The model uses component models of the AlternativeVehicles-, VehicleInterfaces-, PowerTrain- and YaFluid-Library.

Keywords: waste heat utilization, thermal management, thermoelectric generator

1 Introduction

A first approach of a TEG-vehicle-model has been presented by Eschenbach et al. on the 5th International Modelica Conference 2006 [3]. Meanwhile several prototypes of thermoelectric generators have been developed at the Institute of Vehicle Concepts which is part of the German Aerospace Center (DLR) (Figure 5).

Simultaneously to this hardware development the AlternativeVehicles-Library has been enlarged and improved. For high level system simulation of thermo-fluid components a new library called YaFluid-Library has been developed.

2 Model

At the current stage of development, the TEG prototype is not integrated into the cooling system of the car. Cooling of the TEG is done by a separate cooling cycle with an extra fender mounted cooler. For the prototype phase of development this variant has several advantages:

- reproducible measurements due to minimal thermal interaction between the TEG and other components (especially ICE)
- comparatively simple integration of the TEG into a production car
- simple and flexible control of the coolant flow due to separate coolant pump

In a future stage of development the TEG will be integrated in the common cooling system. This integration variant has following advantages:

- the already existing coolant pump and cooler can be used so that additional weight and aerodynamic drag is avoided
- exhaust heat can be used to shorten the warm-up phase

The aim of the vehicle model is to study the interactions between the TEG and other subsystems and to maximize the energetic net benefit. The TEG is part of the exhaust system but is also connected with the cooling medium as well as with the electric system (Figure 1). Mainly the cooling system and the ICE are affected by the TEG. As shown in Figure 1, ICE and TEG are connected in parallel concerning the cooling agent. The coolant flow through the TEG has to be propelled by the coolant pump and the additional heat flow has to be transferred by the front cooler to the ambient. The TEG can also affect the ICE by an increased exhaust gas back pressure The energetic net benefit of the TEG P_{nb} is the difference between generated electric power P_{TEG} and the sum of the losses due to the TEG P_{losses}.

DOI: 10.3384/ecp09430044

$$P_{nb} = P_{TEG} - P_{losses} \qquad (1)$$

$$P_{losses} = P_{pump} + P_{fan} + P_{backpressure} \qquad (2)$$

P_{TEG} is calculated according equation (5). In case of driving cycles it is more convenient to evaluate work instead of the power.

The backpressure influences the gas-exchange process losses as well as the exhaust residual percentage. Obviously the exhaust pumping work increases with increasing backpressure due to an additional heat exchanger. However, this depends heavily on design and control strategy of the engine. No adequate modeling approach to quantify this retroactive effect has been found in literature, thus this effect is still not considered in the simulation. As the TEG is integrated in the existing cooling system and the front cooler size has not changed, the TEG doesn't cause a higher aerodynamic drag. As the mass of the TEG is low, the influence on the driving resistance can be neglected. During the warm up phase the engine oil temperature is low and therefore the engine friction is high. As the TEG shortens the warm up time it also reduces indirectly the engine loss. This positive effect is still not considered in the model.

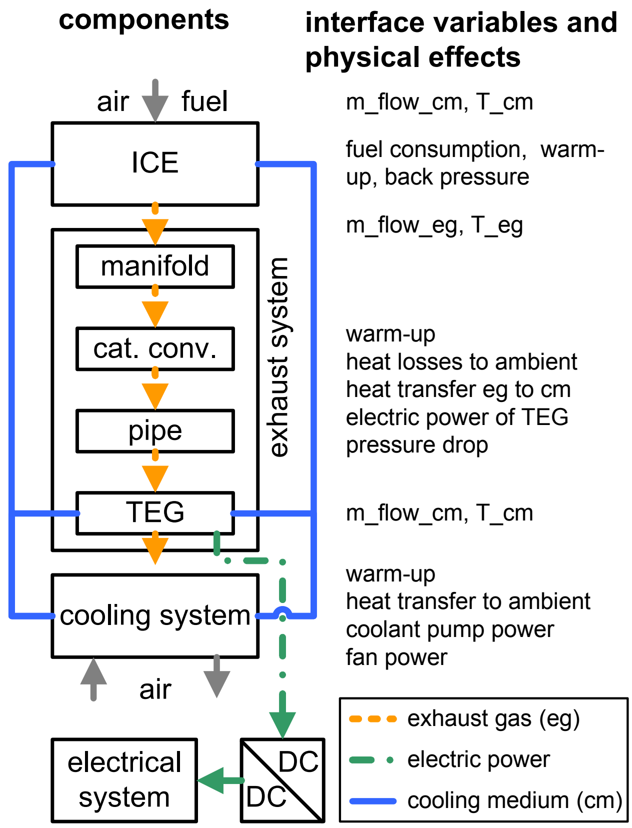
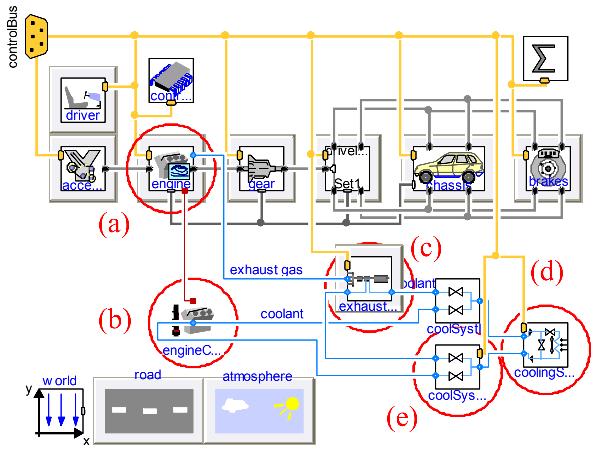


Figure 1 TEG interacting with other components

In the following sub chapters the vehicle, engine, exhaust system and cooling system models will be explained.

2.1 Vehicle model

The vehicle system model uses components of the AlternativeVehicles-, VehicleInterfaces-, PowerTrain- and YaFluid-Library (Figure 2). The base model is a conventional mid sized vehicle with manual gearshift. The engine model has a fluidPort for the exhaust gas flow and a heatPort for the heat flow into the engineCooling model. EngineCooling, exhaustSystem and coolingSystem are connected by the coolant flow. In one of the spitters (e) the coolant mass flow control system is implemented.



(a) engine, (b) engineCooling, (c) exhaustSystem with TEG, (d) coolingSystem, (e) splitter

Figure 2 Diagram of the vehicle model

2.2 Engine model

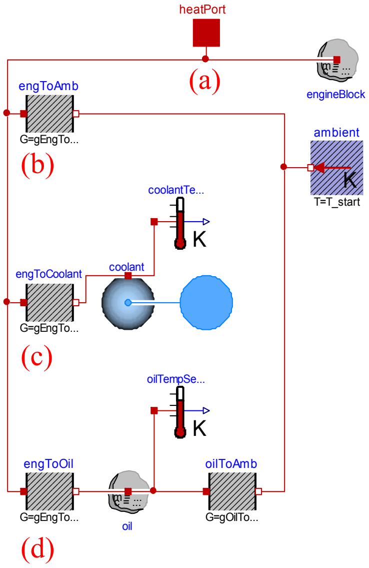
The engine model generates the required mechanical power and calculates using a characteristic map also the loss power as well as the fuel consumption.

This loss power ($\dot{Q}_{engLoss}$) is divided into one part that heats the exhaust mass flow ($\dot{Q}_{engExh}$) and another part that warms up the engine block. In the engineCooling model the heat flow $\dot{Q}_{engCooling}$ which is rejected from the engine block, is calculated. The rejected heat flow consists of the parts ambient heat flow, coolant heat flow and oil heat flow (Figure 3). Each heat transfer coefficient is assumed constant in this model.

$$\dot{Q}_{engCooling} = \dot{Q}_{engToAmb} + \dot{Q}_{engToCoolant} + \dot{Q}_{engToOil} \qquad (3)$$

The thermal masses and heat transfer parameters can be determined with engine test benches measurements.

For the calculation of the generated electric power of the TEG, exhaust mass flow and temperature are important input values. The exhaust mass flow is the sum of the fuel flow and the air mass flow. In normal mode the air mass rate can be calculated with the fuel consumption and the lambda value. As at the current state of development only spark-ignition engines are concerned, stoichiometric combustion can be assumed. In the inertia fuel shutoff mode the air mass flow is calculated by the cylinder capacity and the engine speed. The exhaust gas temperature is calculated from its specific enthalpy which in turn is calculated using the mass flow and exhaust loss power ($\dot{Q}_{engExh}$).



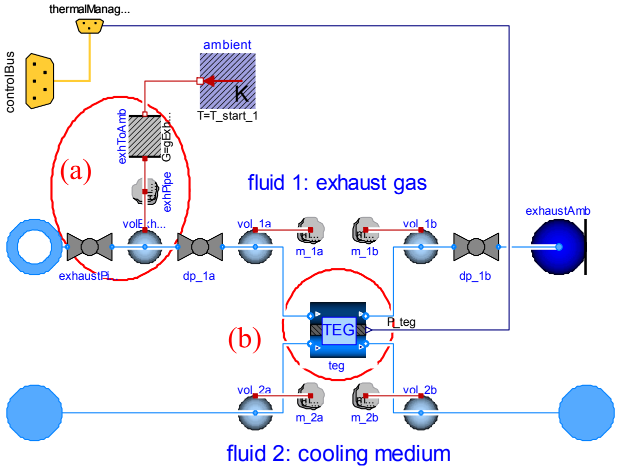
(a) $\dot{Q}_{engCooling}$, (b) $\dot{Q}_{engToAmb}$, (c) $\dot{Q}_{engToCoolant}$, (d) $\dot{Q}_{engToOil}$

Figure 3 Diagram oft the EngineCooling model (modified PowerTrain WarmUpModel)

To avoid negative retroactive effects of the TEG due to immoderate increased backpressure, at peak loads the exhaust mass flow is partially bypassed. As the TEG and the engine are coupled by the coolant circuit, the TEG shortens the warm-up phase and reduces the engine friction. In the case that the cooling system gets to its limit, the additional heat flow from the TEG into the coolant might not be passed to ambient and could lead to a further temperature rise of the coolant and engine. In this cases the TEG should also be bypassed.

2.3 Exhaust system model with TEG

To ensure that the light-off temperature of the catalytic converter is reached within a reasonable time after cold start, the TEG is mounted in the exhaust system behind the catalytic converter. Any component of conventional exhaust gas systems (manifold, catalytic converter, pipes) does transfer significant amounts of heat to the ambient. These heat losses reduce the exhaust temperature and should be minimized by an adequate isolation if exhaust heat is utilized. In the model the three components are combined to a single valve, a control volume, a thermal mass and the heat transfer to the ambient.



(a) combined manifold, catalytic converter and pipe, (b) TEG

Figure 4 Diagram of the ExhaustSystem model with TEG

2.4 Thermoelectric Generator (TEG)

The TEG is a gas-liquid heat exchanger that transfers a heat flow from the hot exhaust gas to the cold coolant (Figure 5). Thermoelectric material (TE) in the wall between the hot and cold sides use the temperature difference to produce an electric potential due to the Seebeck effect. The efficiency of thermoelectric material can be defined as the quotient of the generated electric power and the heat flow at the hot side [2].

$$\eta_{TE} = \frac{P_{TE}}{\dot{Q}_{TE}} \rightarrow P_{TE} = \dot{Q}_{TE} \cdot \eta_{TE} \qquad (4)$$

Not the total wall surface is covered by the thermoelectric material, therefore the factor c_{TE} has to be introduced to calculate the generated power of the TEG.

$$P_{TEG} = \dot{Q}_{TEG} \cdot \eta_{TE} \cdot c_{TE} \tag{5}$$

$$\eta_{TE} = \underbrace{\frac{T_h - T_c}{T_h}}_{\eta_C} \cdot \frac{1}{\frac{4}{ZT} + 2 - \frac{1}{2} \cdot \frac{T_h - T_c}{T_h}} \tag{6}$$

The heat flow $\dot{Q}_{TEG}$ is calculated using the P-NTU-Method [4] and the efficiency of the thermoelectric material η_{TE} is interpolated from a lookup table depending on the hot and cold side temperatures. The electric current also affects the efficiency η_{TE}. However, as a current converter is necessary anyway for the connection to the vehicle electrical system, it is assumed the electric current is forced to an optimum by an appropriate control module. The material data are not available as an equation but as a lookup table. To increase the numeric efficiency the cubic spline approximation provided in the ApproxSpline-Library is used.

As indicated in the equation (6), a large temperature difference between the hot side T_h and cold side T_c of the thermoelectric material is essential for a reasonable good performance of the TEG. Also the transferred heat flow rises with the temperature difference. The module cold side temperature T_c is mainly influenced by the coolant temperature that in turn depends on the engine operation point. The module hot side temperature depends on the exhaust temperature. In general the exhaust temperature rises with the engine load and consequently fast driving cycles with high engine loads are more advantageous for the generated power P_{TEG}.

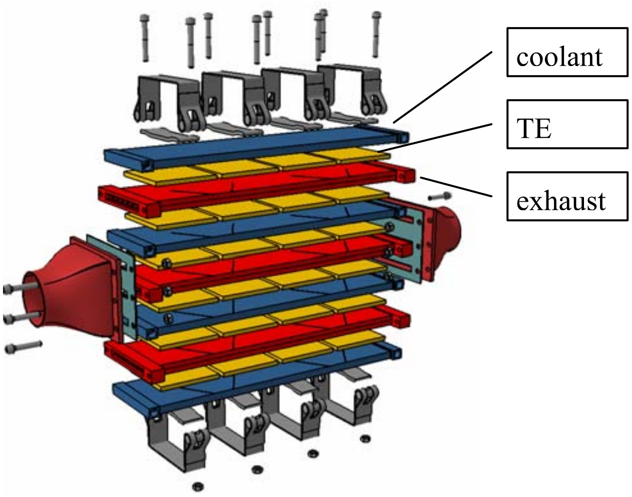


Figure 5 Exploded view of the TEG [2]

For design purposes a detailed discretized model of the TEG is used. For system level simulation it is more convenient to use a concentrated model of the TEG. This model is based on a P-NTU-Method heat exchanger model which is included in the YaFluid-Library. Several geometries and fin types are available, e.g. louvered fin, wavy fin or offset strip fins. Depending on the geometry several characteristic variables are calculated, e.g. heat transfer areas, Reynolds number, Nusselt number and Fanning friction factor.

Measuring values were available from the Institute of Materials Research which is also part of the German Aerospace Center (DLR). After calibrating the model parameters measurement values and simulation values fit well (Figure 6, Figure 7).

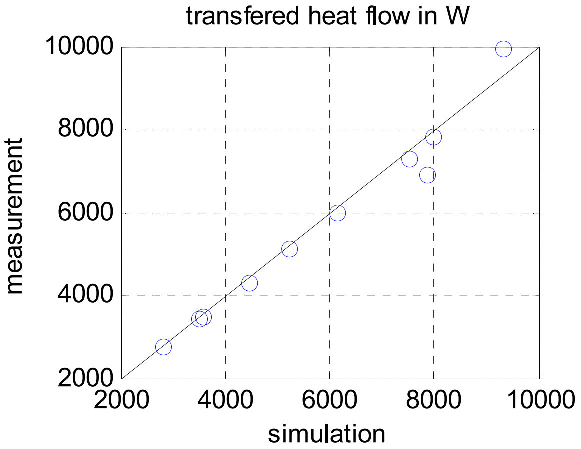


Figure 6 Comparison of simulated and measured transferred heat flow

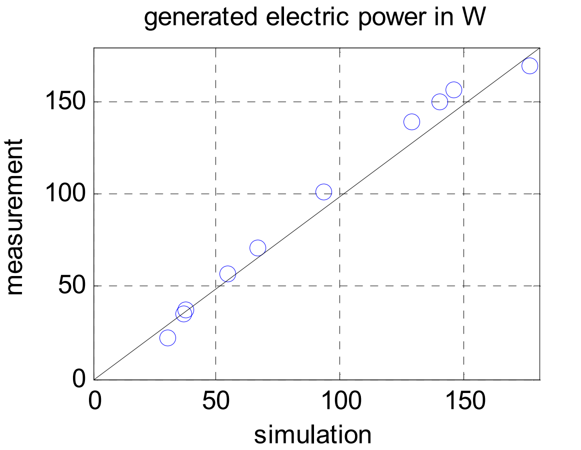
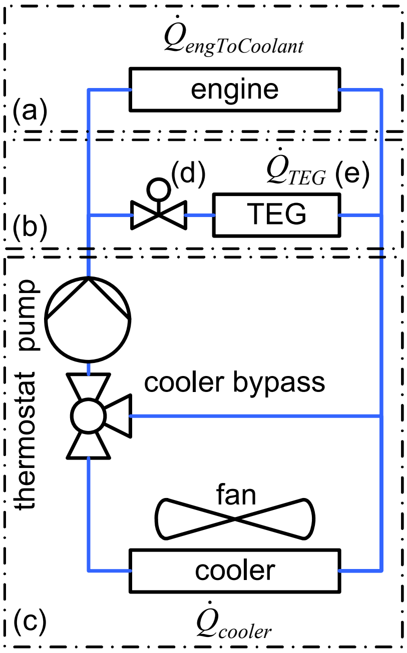


Figure 7 Comparison of simulated and measured generated electric power

2.5 Cooling system model

The amount of heat that the cooling cycle has to transfer to the ambient is given as $\dot{Q}_{engToCoolant}$. In most cases, the airflow is sufficient to reject the heat from the coolant. Only at low vehicle speed additional energy effort is necessary to force air through the front cooler by a fan. The TEG causes an additional heat flow $\dot{Q}_{TEG}$ into the cooling cycle, which is calculated in the TEG model as mentioned above.



(a) engineCooling, (b) exhaustSystem, (c) coolingSystem, (d) controlled valve, (e) TEG coolant outlet

Figure 8 Simplified cooling circuit

After cold start the operating temperature of the engine should be reached as fast as possible. Therefore the bypass is fully opened and almost no coolant runs through the cooler. Between the coolant temperatures T_{lower} and T_{upper} the bypass is gradually closed. If the coolant temperature exeeds the upper temperature limit, the bypass is fully closed. Up to a certain velocity (v_{FanOn}) the heat transfer to ambient can be improved by switching on the fan.

In the cooling system model the heat flow $\dot{Q}_{cooler}$ from coolant to ambient is calculated (Figure 10). Therefore the simple approach

$$\dot{Q}_{cooler} = G_{cooler} \cdot \Delta T_{cooler} = G_{cooler}(T_{coolant} - T_{amb}) \quad (7)$$

is used, where $T_{coolant}$ is the coolant temperature, T_{amb} is the air temperature of the ambient and G_{cooler} is the thermal conductance (product of the heat transfer coefficient and heat transfer area). G_{cooler} depends primarily on the geometry and furthermore on mass flows of air and coolant.

At engine peak load G_{cooler} has to be sufficient to reject the engine waste heat $\dot{Q}_{engToCoolant}$ to the ambient. The driving resistance of a given vehicle varies with slope and payload. Thus, the lowest vehicle speed at which maximum engine load is achievable is defined by the cooler design. Figure 9 shows a simple cooler design procedure. A simulation of the engine waste heat flow rate in dependence of vehicle speed at zero gradient which covers the complete power range of the engine is done. To ascertain an adequate thermal conduction of the cooler, the following assumptions were made:

- coolant mass flow and temperature are constant
- $\dot{Q}_{engToAmb}$ and $\dot{Q}_{engToOil}$ are neglected, thus $\dot{Q}_{engToCoolant} = \dot{Q}_{engCooling}$
- the air velocity through the cooler is equal to the car velocity

Thus, G_{cooler} depends only on the cooler air flow which is directly linked to the car velocity. At maximum vehicle speed the cooler has to transfer the maximum engine waste heat (Figure 9 point (a)).

$$\dot{Q}_{cooler}(v_{\max}) = \dot{Q}_{engToCoolant}(v_{\max}) \quad (8)$$

As a worst-case approximation, it is assumed the thermal conductance depends linear on the vehicle velocity (line (d) in Figure 9). Cooler maps show, that this underestimates the actual thermal conductance of real world coolers (the characteristic curve (c) promises a higher heat transfer than the linear approach (d) delivers).

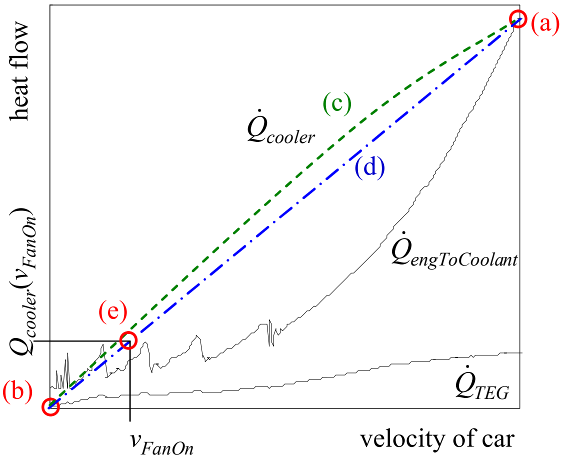


Figure 9 Identification of Q_{cooler} respective G_{cooler}

Thus, the cooler conductance is calculated as follows.

$$G_{cooler,v}(v=0)=0 \tag{9}$$

$$G_{cooler,v}(v=v_{\max})=\dot{Q}_{engToCoolant}(v_{\max})/\Delta T_{cooler} \tag{10}$$

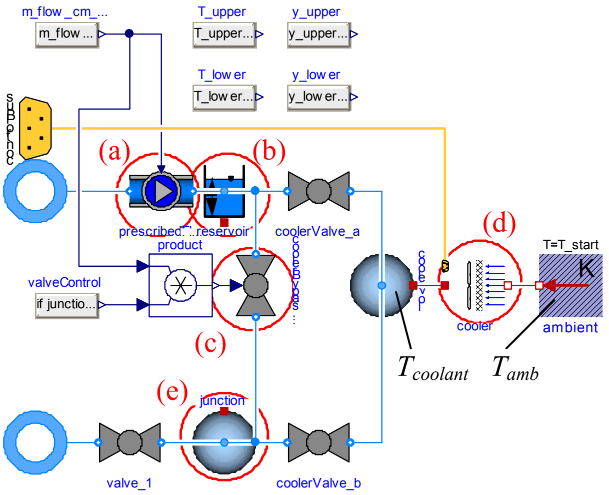
$$G_{cooler,v}(v)=v\cdot G_{cooler,v}(v_{\max})/v_{\max} \tag{11}$$

As a consequence of the linear approach, the heat rejection of the cooler is insufficient at low vehicle speed. This leads to a continuously rising coolant temperature up to a certain temperature T_{fanOn} where the fan is switched on and forces the airflow through the cooler with the velocity v_{fanOn}.

$$G_{cooler,fan}=G_{cooler,v}(v=v_{FanOn}) \tag{12}$$

$$G_{cooler}(v)=\begin{cases} T_{coolant}>T_{FanOn}:G_{cooler,fan} \\ \quad else:G_{cooler,v}(v)\end{cases} \tag{13}$$

The heat rejection is implemented in the cooler model (Figure 10).



(a) pump, (b) reservoir, (c) bypass valve, (d) cooler, (e) junction

Figure 10 Diagram of the CoolingSystem model

As indicated at Figure 8 the coolant flow through the bypass depends on the coolant temperature (Figure 10 (e)) and the temperature limits T_{lower} and T_{upper}.

The coolant flow through the TEG has to be forced by the coolant pump and consequently this pumping power is considered in the energetic net benefit (eq. (1)). It is the product of volume flow and pressure loss divided by pump efficiency. The pressure loss is calculated using a fitting curve on the base of measurement values. Only the pressure loss of the TEG is considered, not the pressure loss of the control valve. The latter is determined by the pressure loss of the engine, as the total pressure differences of both parallel passes are equal. To minimize the pumping power the coolant mass flow is controlled depending on the difference of coolant outlet temperature of the TEG (Figure 8 (d), (e)) and the coolant outlet temperature at design point (set point).

3 Simulation

For the simulation model a mid sized vehicle with an 85 kW powered gasoline engine was used (Table 1). Whether the energetic net benefit of the TEG is positive or not depends crucially on the concrete driving cycle and the TEG efficiency. The TEG used here is designed for waste heat recovery at medium to high engine load. Thus, a suitably driving cycle must be used to achieve reasonable results. In general the Artemis Driving Cycles show a much more realistic daily life driving behaviors than the often used New European Driving Cycle (NEDC) [6]. Considering the TEG design, the Artemis motorway driving cycle is an adequate driving cycle for this study (Figure 11).

Table 1 Parameters of the vehicle model

Parameter	Value
Vehicle mass (2/3 load)	1575 kg
Engine power	85 kW at 6000 1/min
Engine displacement	1.6 l
Engine max. torque	155 Nm at 4000 1/min
TEG 1	bismuth telluride (Bi_2Te_3)
TEG 2	lead telluride (PbTe)

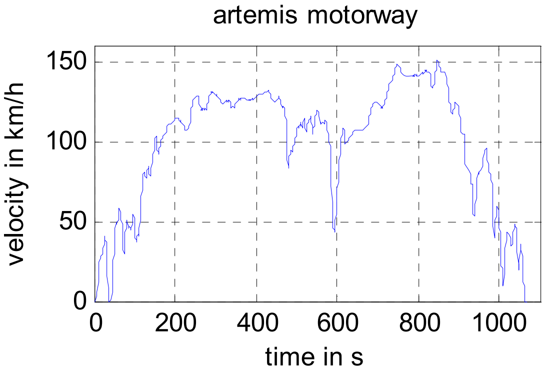


Figure 11 Velocity in the artemis motorway driving cycle

Two variants of a TEG are simulated, first a state of the art TEG with the thermoelectric material bismuth telluride (Bi_2Te_3) (TEG 1) and second an improved TEG with lead telluride (PbTe) and an enhanced heat transfer characteristic (TEG 2).

For both TEG variants the power according eq. (1) has been calculated and plotted. The generated electric power of the TEG has been calculated in dependency on fluid mass flow rates and temperatures as well as the geometry of the TEG. Due to the high vehicle velocity in the Artemis motorway cycle and consequently high heat transfer to ambient the front cooler fan is never switched on. In this simulation the power loss of eq. (2) is equal to the coolant pumping power.

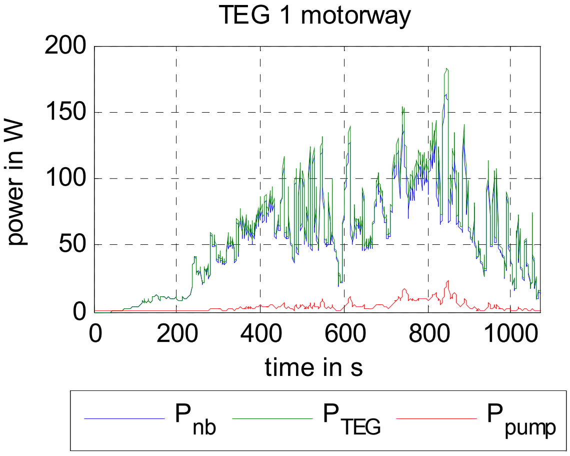


Figure 12 TEG 1 motorway: powers

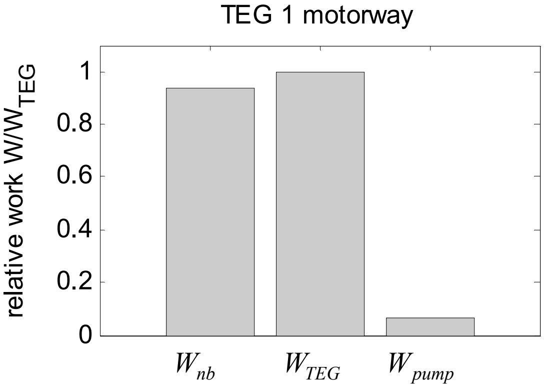


Figure 13 TEG 1 motorway: works

At simulation start the temperature difference between the hot and cold side of the TEG is zero, thus the heat transfer and the generated electric power are also zero. As the thermal mass of the coolant and exhaust system warm up, the temperature difference across the thermoelectric module and the generated electric power rises.

Due to the effective control of the coolant mass flow rate the pumping power is kept quite small in comparison to the generated electric power (Figure 12 and Figure 13). During the warm up time the coolant temperature is still low. Consequently the coolant mass flow as well as the pumping power are minimal.

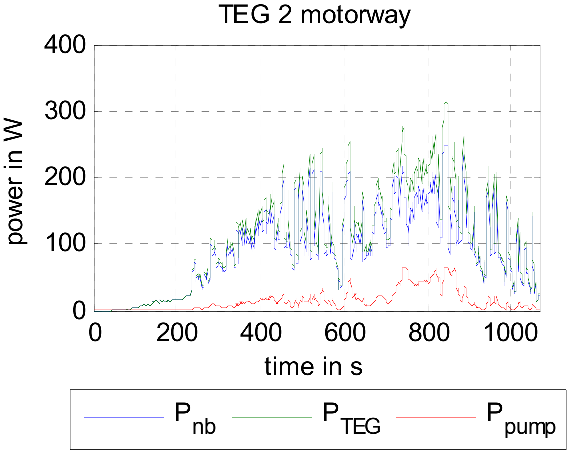


Figure 14 TEG 2 motorway: powers

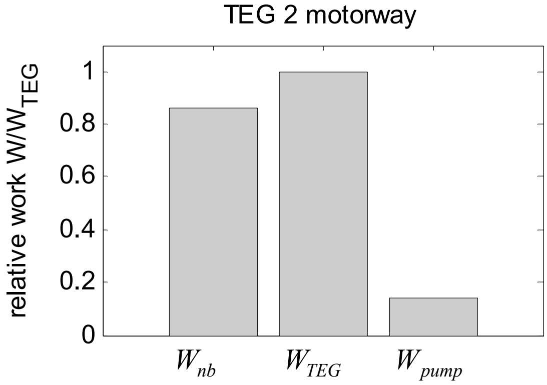


Figure 15 TEG 2 motorway: works

Thanks to the higher operation temperature of the thermoelectric material lead telluride (PbTe) and the enhanced heat transfer characteristic the TEG 2 generates up to 310 W electric power. In comparison to the TEG 1 the net benefit work is about 70 % higher. Due to the higher transferred heat flow the warm up time is even more reduced. The higher heat flow leads also to a higher coolant mass flow through the TEG and consequently the pumping work rises. As the pumping power rises quadratically, also the pumping work increases disproportionately to the generated electric energy.

4 Conclusions

A mid sized car equipped with a thermoelectric generator (TEG) for exhaust heat recovery has been modeled based on the Modelica libraries AlternativeVehicles, VehicleInterfaces, PowerTrain, ApproxSpline and YaFluid. The mutual interferences of the components have been studied and on the basis of two TEG variants the potential augmentation of the generated net benefit power as well as the net benefit work have been analyzed. However using commercially available thermoelectric modules (Bi_2Te_3) the TEG reaches values up to 180 W, 310 W are obtainable using lead telluride (PbTe). Improved thermoelectric materials, control of coolant and exhaust flow as well as possible reductions of the exhaust and coolant pressure losses have a significant potential to improve the energetic net benefit of TEG equipped cars in future.

For the calculation of the net benefit of waste-heat-utilization-technologies it is important not just to focus on single components but to examine the entire system. The AlternativeVehicles-library is suitable to model various vehicle concepts and enables understanding and optimizing the whole vehicle system.

References

[1] Friedrich, Treffinger, Müller: Management von Sekundärenergie und Energiewandlung von Verlustwärmeströmen. ATZ/MTZ-Konferenz – Energie, München, 2007

[2] Treffinger, Häfele, Weiler, Eder, Richter, Mazar: Energierückgewinnung durch Wandlung von Abwärme in Nutzenergie (Recovery of energy through conversion of waste heat), Innovative Fahrzeugantriebe, Dresden, 2008

[3] Eschenbach, Ungethüm, Treffinger: Vehicle model for transient simulation of a waste-heat-utilisation-unit containing extended PowerTrain and Fluid library components, 5th International Modelica Conference, Vienna, 2006

[4] Shah, Sekulic: Fundamentals of Heat Exchanger Design, John Wiley and Sons, Hoboken, 2003

[5] Ungethüm, Hülsebusch: Implementation of a Modelica Library for Smooth Spline Approximation, 7th International Modelica Conference, Como, 2009

[6] M. André: Real-world Driving Cycles for Measuring Cars Pollutant Emissions. Part A: The Artemis European Driving Cycle. Institut National de Recherche surles Transports et leur Securité (INRETS), Bron, 2004

Some aspects of the modeling of tube-and-shell heat-exchangers

Anton Sodja[1]   Borut Zupančič[1]   Janko Šink[2]

Fakulteta za elektrotehniko, Univerza v Ljubljani[1]
Tržaška 25, 1000 Ljubljana

Izoteh d.o.o.[2]
Brnčičeva 15b, 1000 Ljubljana

Abstract

Modern control design of large industrial plants relies strongly on modelling, while experimentation on the real system is extremely limited. The usable model should be intuitive and easily reusable to model potential plant adaptations or similar plants.

In this paper some aspects of the modelling a recuperator plant consisting of four heat exchangers are presented. With the object-oriented modelling tool Dymola, using Modelica as a modelling language, the building of clear, intuitive and reusable model of basic, but most important plant's blocks, heat exchangers is enabled. However, the proposed model of heat exchanger proved to be intrinsically stiff and therefor at the moment problematic to be used as a submodel of more complex models.

Keywords: recuperator, heat exchanger; tube-and-shell, cross-flow, object-oriented modelling

1 Introduction

It is well known that practical constraints exists in control design of production lines, where one of the most problematic is the limited experimentation possibilities. The production line should be namely brought in operation as fast as possible but as rarely as possible halted for maintenance during its life-time. So there are not many opportunities to learn about the system through experimentation and to improve the control design. Consequently tere is enough time merly for tuning the controllers. Control design must be somehow supported by modelling.

In this paper a model of recuperator is presented, shown schematically in Fig. 1. The plant comprises of four heat exchangers in series which are used to recover waste heat from exhaust gases. The latter are created by combustion of a mixture of the waste gases coming from the main plant and a natural gas. The pipe they flow through is shown in the middle of the Fig. 1. On the secondary side of the heat exchangers incoming gases have various purposes and destinations in the main plant.

The flow of gases through recuperator can be manipulated by flap valves and ventilators (not shown in Fig. 1).

The recuperator plant is included in different industrial lines and can be modified according to the needs and specifics of the plant. The model of the process must thus be very flexible and user friendly for a control designer. As an appropriate modelling environment Dymola with object-oriented modelling language Modelica [6] was chosen. Object-oriented and acausal modelling approach simplifies modelling process. On contrary to traditional modelling, where model is represented by a set of functional blocks with causal connections (inputs and outputs), object-oriented acausal models are composed as sets of related, interacting objects (submodels) and transformation of the model in a proper form suitable for computation is left to translator. Object-oriented models thus preserve topology of the system being modelled and are as such more intuitive and easily reusable since submodels do not have explicitly defined computational order.

Modelica was found as appropriate also because free Modelica libraries from the domain of thermo-fluids [1, 3] are available what significantly mitigates the model development procedure.

2 Mathematical model of the heat exchanger

Heat exchanger is a device in which energy in the form of heat is transferred what is usually realized by the confinement of both fluids in some geometry in which they are separated by a conductive material. The properties of heat exchanger are strongly dependent on geometry and material as well as on properties of both fluids. It is known that such devices had usually non-linear behaviour [5].

DOI: 10.3384/ecp09430101

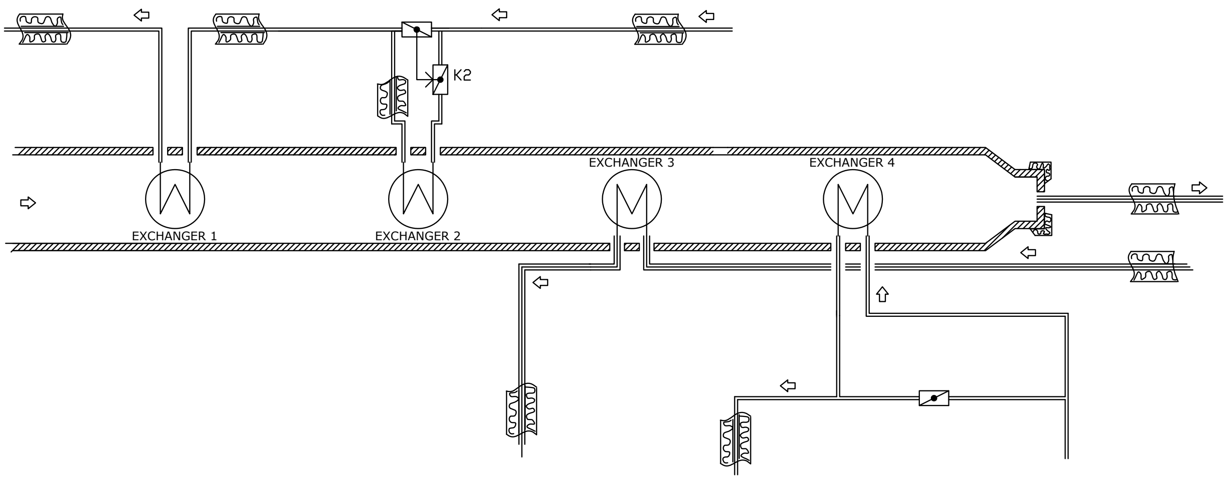


Figure 1: A scheme of the recuperation process: hot exhaust gases flow pass four heat exchangers and heat up gases used in other part of the plant.

2.1 Model of the heat flow rate between fluids

Our aim is to build a reusable model which will enable better understanding of the process. So the model should be described by analytical equations. The observed recuperator process comprises of shell-and-tube cross-flow heat exchangers depicted in Fig. 2. Particular heat exchanger consists of a bundle of tubes with circular cross section which is inserted into a shell so that a flow through the tubes is perpendicular to the flow through shell as illustrated in Fig. 3. Since heat transfer takes place across the tubes surface, the arrangement of flows (geometry of the heat exchanger) has important impact on the efficiency of the device.

However, analytical solution for shell-and-tube heat exchanger exists only when the flows of both fluids are parallel. They can be co-current or counter-current. The energy balance for tube-side fluid and shell-side fluid for a tubular counter-current heat exchanger is given in Eq. (1) and Eq. (2) respectively. Equations

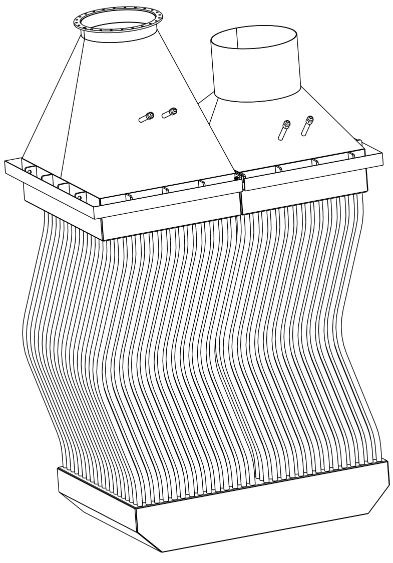


Figure 2: Scheme of the cross-flow heat exchanger.

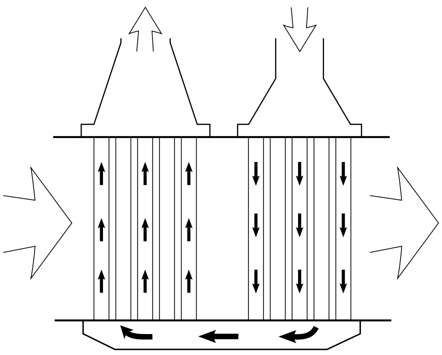


Figure 3: Flow directions in cross flow heat exchanger.

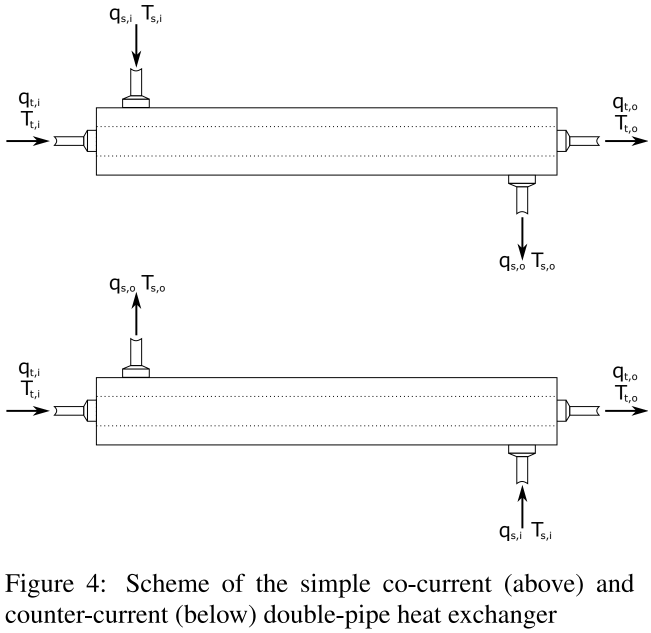


Figure 4: Scheme of the simple co-current (above) and counter-current (below) double-pipe heat exchanger

for co-current heat exchanger are different (due to flow direction) only in the sign of the first term on the right side of Eq. (2).

$$\begin{aligned}\frac{\partial}{\partial t}(\rho_t \cdot A_{c,t} \cdot \hat{C}_{p,t} \cdot T_t) =& -\frac{\partial}{\partial z}(\rho_t \cdot q_t \cdot \hat{C}_{p,t} \cdot T_t)\\ & -\frac{U \cdot A}{L} \cdot (T_s - T_t)\end{aligned} \tag{1}$$

$$\begin{aligned}\frac{\partial}{\partial t}(\rho_s \cdot A_{c,s} \cdot \hat{C}_{p,s} \cdot T_s) =& \frac{\partial}{\partial z}(\rho_s \cdot q_s \cdot \hat{C}_{p,s} \cdot T_s)\\ & -\frac{U \cdot A}{L} \cdot (T_s - T_t)\end{aligned} \tag{2}$$

In Eq. (1) and (2) indices t and s denote quantities of tube-side and shell-side fluid respectively; T designates mean temperature of cross section which depends on time and position along the pipe's length (z-axis): $T = T(t,z)$, A_c is area of the pipe's cross section, ρ and $\hat{C}_p$ are density and specific heat capacity of the fluid respectively (also time and position dependent), q is the mass flow, U is the overall transfer coefficient, A is area through which heat is exchanged and L is length of the pipes.

From Eq. (1) and (2), analytical expressions for the temperatures of the fluids at the pipes' outlets and heat flow rate between the fluids can be derived [4]. However, for a cross-flow heat exchanger it is known only that its performance is worse from counter-current and better from co-current heat exchanger [5]. However, no analytical expressions exists. So the following supposition was taken into account: while the shell's length is relatively small in comparison to its cross section and the flow through it is highly turbulent (Reynolds number is of the order 10^5), temperature differences across the shell are negligible and shell can be modelled sufficiently accurate as a lumped model. The energy balance equations are then:

$$\begin{aligned}\frac{\partial}{\partial t}(\rho_t \cdot A_{c,t} \cdot \hat{C}_{p,t} \cdot T_t) =& -\frac{\partial}{\partial z}(\rho_t \cdot q_t \cdot \hat{C}_{p,t} \cdot T_t)\\ & -\frac{U \cdot A}{L} \cdot (T_s - T_t)\end{aligned} \tag{3}$$

$$\begin{aligned}\frac{d}{dt}(\rho_s \cdot V_s \cdot \hat{C}_{p,s} \cdot T_s) =& \rho_s \cdot q_s \cdot \hat{C}_{p,s} \cdot (T_{s,i} - T_{s,o})\\ & -U \cdot A \cdot (T_s - \bar{T}_t)\end{aligned} \tag{4}$$

In Eq. (4) V_s denotes the volume of the shell, $T_{s,i}$ and $T_{s,o}$ temperatures of the shell-side fluid at inlet and outlet respectively and $\bar{T}_t$ is a mean temperature of fluid in tube bundle at a given time.

As Modelica does not support solving partial differential equations implicitly, Eq. (3) was discretized by a finite volume method.

2.2 Model of the wall's heat capacity

In the proposed heat exchangers fluids in tube- and shell-side are hot gases that passes through the exchanger at relatively high speed (about 15 m/s in the shell), and their mass (and thus heat capacity) is much smaller then the mass of the tube-bundle's wall which weights 3.4 tons. The influence of the wall heat capacity is thus considerable and must be included in the model.

On the other hand, the thermal conductivity of the wall is a few orders greater then conductivity of the gas and was thus neglected.

In order to consider dynamics due to wall's heat capacity, additional equation has to be added to the energy-balance equations Eq. (3) and Eq. (4):

$$\begin{aligned}\frac{\partial}{\partial t}(\rho_t \cdot A_{c,t} \cdot \hat{C}_{p,t} \cdot T_t) =& -\frac{\partial}{\partial z}(\rho_t \cdot q_t \cdot \hat{C}_{p,t} \cdot T_t)\\ & -\frac{\alpha_t \cdot A}{L} \cdot (T_w - T_t)\end{aligned} \tag{5}$$

$$\begin{aligned}\rho_w \cdot A \cdot d \cdot \hat{C}_{p,w} \cdot \frac{dT_w}{dt} =& -\alpha_t \cdot A(T_w - \bar{T}_t)\\ & -\alpha_s \cdot A \cdot (T_w - T_s)\end{aligned} \tag{6}$$

$$\begin{aligned}\frac{d}{dt}(\rho_s \cdot V_s \cdot \hat{C}_{p,s} \cdot T_s) =& \rho_s \cdot q_s \cdot \hat{C}_{p,s} \cdot (T_{s,i} - T_{s,o})\\ & -\alpha_s \cdot A \cdot (T_s - T_w)\end{aligned} \tag{7}$$

In the new equations, index w designates quantities of the wall, α_t and α_s are the convective heat transfer factors of the tube- and shell-side gas respectively and d is thickness of the wall.

2.3 Model of the fluid dynamics

The purpose of the model is temperature dynamics description of the recuperator. The thermal dynamics of its component, heat exchanger, is described by Eqs. (5), (6) and (7). However, some parameters, for example, ρ_t, $C_{p,t}$, are not constant (or nearly constant) in operating-temperature range, so additional relations must be introduced. It means flow-equations derived from the laws of mass and momentum balances (to calculate mass flow and pressure respectively) and algebraic equations (Eqs. (8)) which determine properties of the media. They describe relation among pressure p, temperature T, internal energy u, enthalpy h and density ρ.

$$\begin{aligned}p &= p(\rho,T)\\ u &= u(\rho,T)\\ h &= u + \frac{p}{\rho}\end{aligned} \tag{8}$$

Eqs. (8) introduce a non-negligible nonlinearites in the model and also increase differential algebraic equations index of the system [7].

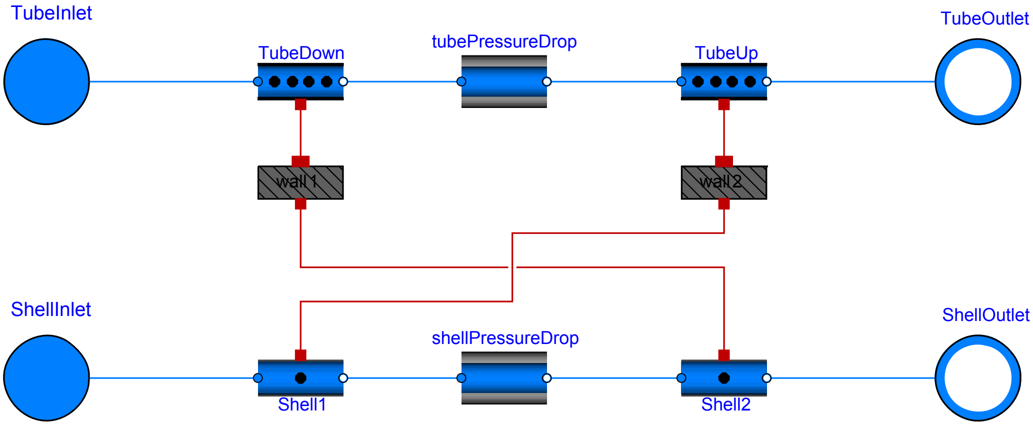


Figure 5: Component scheme of the heat exchanger submodel in Dymola

2.4 Model of the pressure drop

Determination of the pressure drop plays a major role in the design of heat exchanger along with heat-transfer coefficient [5]. However, for the control design of the observed recuperator is not very important. Partly it is the consequence of the lack of measurements of the pressure-drop dynamics. Only nominal pressure drop at nominal volume flow rate was namely available. So, a pressure drop Δp of the whole pipe (shell and tube) of the heat exchanger is modelled:

$$\Delta p = kv^2 \tag{9}$$

Coefficient k in Eq. (9) was calculated by inserting nominal pressure drop and nominal velocity (derived from nominal volume flow) into the equation.

2.5 Convective heat-transfer coefficient

The convective heat-transfer coefficient of the gas is problematic as it is influenced by geometry of the heat exchanger and chemical properties of the gases. Usually it is provided as an empirical expression including Reynolds and Prandtl numbers. To keep the model simple, the expressions in Eq. (10) and (11) (for tube- and shell-side respectively) were found to be sufficient:

$$\frac{\alpha_t \cdot d_{h,t}}{\lambda_t} = 0.040 \cdot (Re_t \cdot Pr_t)^{0.75} \tag{10}$$

$$\frac{\alpha_s \cdot d_{h,s}}{\lambda_s} = 0.113 \cdot (Re_s \cdot Pr_s)^{0.75} \tag{11}$$

In Eq. (10) and (11) indices t and s indicate tube- and shell-side respectively, d_h is hydraulic diameter, λ thermal conductance, Re Reynolds number and Pr Prandtl number.

3 Implementation in Modelica

The model of the heat exchanger basically consists of two thermally coupled pipes. So it should be built up by two pipe submodels and intermediate heat-transfer submodel.

3.1 Heat exchanger

As already mentioned in the introduction, many freely available Modelica libraries for modeling thermodynamics systems exist. In our case basic components from the *Modelica_Fluid* library are used and adopted. Here a tube bundle is described by distributed parameters. So a component of a pipe with distributed parameters discretized by finite volume method is taken from the library. For the model of shell a component of a pipe with lumped parameters from the library is used. Thermo-fluid governing equations and properties of the media (Eq. (8)), are realized by a nested component of the *Modelica.Media* library), enabling the avoidance of flow-dynamics equations formulation by hand. Components of the tube bundle and the shell are connected over a wall model, what is a custom made component simulating Eq. (6). The resulting scheme is shown in Fig. 5.

In the Fig. 5 it can be seen that each pipe is consists of three components – pipe is split in two parts and a pressure drop component is placed inbetween. The model of tube bundle is composed from components *TubeDown*, *tubePressureDrop* and *TubeUp* components, while shell comprises *Shell1*, *shellPressureDrop* and *Shell2*. At the outermost edges four connectors are place, namely inlet and outlet for the tube and shell. They represent interface of the heat exchanger component.

3.2 Convective heat transfer

The mentioned components of the pipes from the *Modelica_Fluid* library represent a replaceable nested component of convective heat transfer. However, this component can be replaced by the one including also convective-heat-transfer coefficient from Eq. (10) and

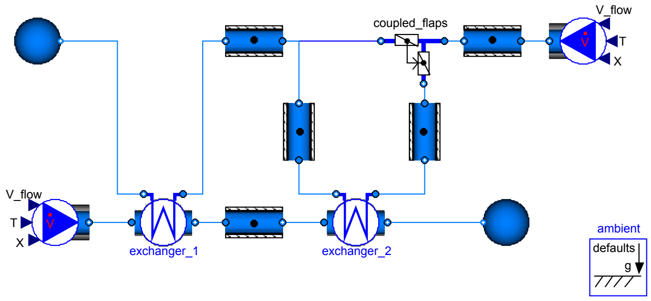


Figure 6: Model of the recuperator's exchangers 1 and 2 with some periphery

Eq. (11). The code listing of our component for tube's convective heat transfer is the following:

```
model PipeHT_TubeConv
  extends PartialPipeHeatTransfer;
  outer input Medium.BaseProperties[n] medium;
  SI.CoefficientOfHeatTransfer alpha0[n];
  SI.ThermalConductivity lambda[n];
  SI.PrandtlNumber Pr[n];
  outer input SI.ReynoldsNumber Re[n];
equation
  for i in 1:n loop
    lambda[i] = Medium.thermalConductivity(medium[i]);
    Pr[i] = Medium.prandtlNumber(medium[i]);
    alpha[i] = lambda/d_h*0.040*(Re[i]*Pr[i])^0.75;
    thermalPort[i].Q_flow=noEvent(if alpha[i] < 5 then 5
      else alpha[i]*A_h/n*(thermalPort[i].T-T[i]);
  end for;
  thermalPort.Q_Flow=Q_flow
end PipeHT_TubeConv
```

The *PipeHt_TubeConv* is very simple component, because all the needed thermodynamic variables (thermal conductivity and Prandtl number) of the medium in Eq. (10) are computed by *Modelica.Media* library and some other variables (Reynolds number and medium base properties) are computed in embedding pipe model.

3.3 Recuperator

The model of recuperator is decomposed into subsystems, i.e., heat exchangers, pipes and flaps. It means that it is built by connecting components simulating the subsystems.

However, certain difficulties were encountered at the connections of pipes and heat exchangers. An implicit system of nonlinear equations for pressure/flow correlation is namely needed in the connection point [2]. This is the consequence of the discretization of the partial differential equations in the pipe's model. A staggered grid approach is used, leading to half momentum balances between the pipe's boundary and first and last segment (finite volume). The system of nonlinear equations in connection points is especially problematic at the initialization phase due to unsatisfactory knowledge of the pressure drop across the system it is hard to define corresponding initial values. The consequence is that time necessary for simulation severely increases and in some cases a solution can not be found.

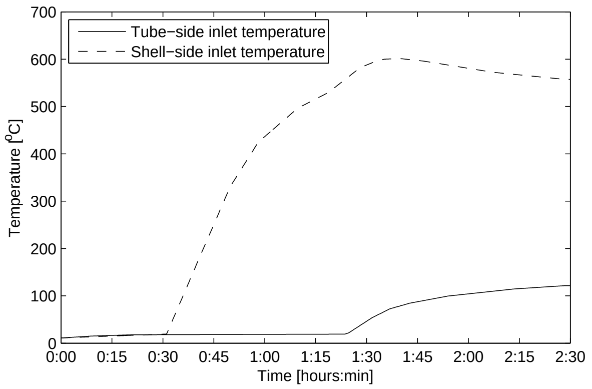


Figure 7: Temperatures of the tube- and shell-side gases at the inlet.

Additional difficulties arise from the different cross-areas of the connecting pipes (especially on the tube side – the outlet gases are less dense then the inlet ones and take larger volume, so the outlet pipe have larger diameter than inlet one). This fact causes numeric problems in the kinetic terms of the flow-equations and special care must be taken to handle them appropriately [2]. Due to numerical problems in the junctions of the pipes the building of models from prepared heat-exchanger and pipe (sub)models is still very tedious task.

In Fig. 6 a model of the connected heat exchangers 1 and 2 is shown (illustrated also on the left side of Fig. 1). It represents part of the recuperator. Tubes of both exchangers are connected and the temperature at the outlet of the exchanger 1 is controlled by two coupled flaps which define the portion of the flow passing through heat exchanger 2. The simulation run of the model takes more than 10 hours, while the simulation of the single heat exchanger finishes in a few minutes.

4 Validation

Validation of the heat exchanger model was performed on a real measurements data (validation of the whole recuperator is impossible due to the lack of data). The available measurements were temperatures of the exhaust gases at the entrance into the shell of the heat exchanger and at the exit from the shell, input and output temperatures of the tube gases as well as volume flow through the tube. Unfortunately, volume flow through the shell was not measured. So it was supposed to be constant during the simulation and equal to the nomi-

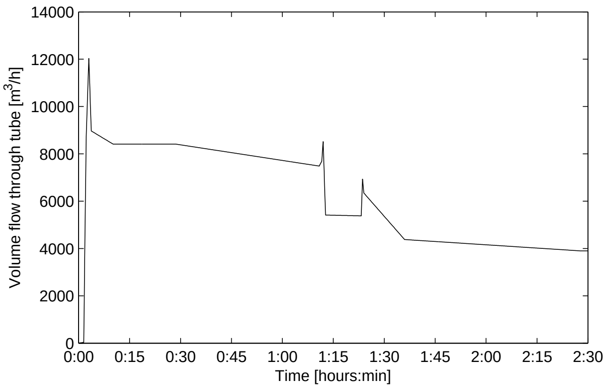


Figure 8: Volume flow through the tube.

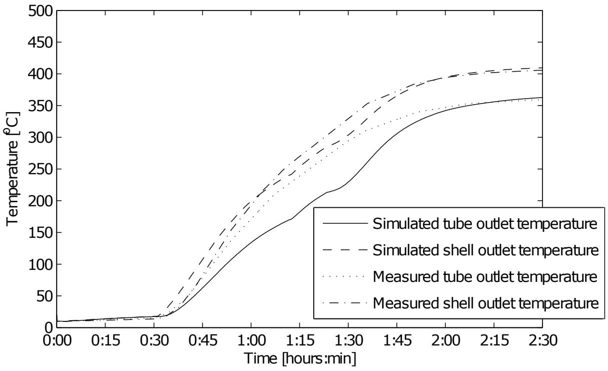


Figure 9: Comparison of the measured and temperatures obtained by simulation of the gases at outlet of tube and shell.

nal one.
The temperature measurements of the entering gases into the heat exchanger are shown in Fig. 7 and measurements of volume flow through tube in Fig. 8. Simulation results, i.e. temperatures of tube and shell gases at outlet, compared with measured ones are shown in Fig. 9.

As it can be seen in Fig. 9, response of the model fits the measured data relatively well. The biggest discrepancy is during the rise time what can be assigned to the missing shell flow measurements. At the end of the simulation, when the shell flow should reach its nominal value, also the difference between measured and calculated response is smaller.

Important property of the model is that it is relatively unaffected by a sharp changes in tube flow as it can be seen in Fig. 8 and Fig. 9. The steep changes of the tube flow cause only little disturbances in the temperatures at the outlet. By experimenting on the model it was found out that it is a consequence of the nonlinear convective heat transfer.

Nevertheless, validation of the model is not yet satisfactory. It should be validated on more measurements and the data should also include measurements of the flow through the shell.

5 Conclusion

As it was shown in the paper, object-oriented acausal modelling approach with support of freely available libraries offers a very rapid development of the complex and highly nonlinear models. In the paper a model of the heat exchanger is shown.

However, in the case of more complex case of connecting heat-exchanger models, many difficulties appear. Solving them makes the model unnecessarily complicated. The numerical problems of the recuperator model thus originate in the components of the *Modelica_Fluid* library and the design of fluid connector due to limitation of Modelica [2]. This has been improved in the latest language-standard change and the *Modelica_Fluid* library was reimplemented.

In our future work we plan to port our model to the newest version of the *Modelica_Fluid* library which will help us to approach our goal of creating a clear and reusable model for control engineers. We will also intend to acquire better measurements from the new plants which will enable more proper validation of the single heat exchanger model as well as the whole recuperator plant.

References

[1] Elmqvist H, Tummescheit H and Otter M. Object-Oriented Modeling of Thermo-Fluid Systems. In: Proceedings of the 3rd International Modelica Conference 2003, Linköpig, Sweden.

[2] Modelica Association, Modelica_Fluid Users Guide, *http://www.modelica.org/libraries/Modelica_Fluid*, 2009.

[3] Casella F, Leva A. Modelica open library for power plant simulation: design and experimental validation. In: Proceedings of the 3rd International Modelica Conference 2003, Linköpig, Sweden.

[4] T. W. F .Russell, A. S. Robinson and N. J. Wagner, *Mass and Heat Transfer*. Cambridge University Press, 2008.

[5] T. Kupran, *Heat Exchanger design handbook*. Taylor & Francis, 2000.

[6] Modelica Association, Modelica - A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, version 3.0, 2007.

[7] K. Sørensen, N. Houbak, T. Condra, *Solving differential–algebraic equation systems by means of index reduction methodology*. Simulation Modelling Practice and Theory, Simulation Modelling Practice and Theory. Volume 14, Issue 3. April 2006, pages 224-236.

Reversed-flow models

Tomas Skoglund[a]

[a]Tetra Pak Processing Systems
Ruben Rausings gata, SE-221 86 Lund, Sweden
tomas.skoglund@tetrapak.com

Abstract

Component models for bidirectional flow in process lines were developed to account for propagation of fluid properties and balance equations for mass, heat and momentum. The models performed well in simulations with flow in arbitrary direction. The models increased the stability in simulations of cases with short, unintentional reversed flow.

Keywords: reversed flow; bidirectional flow; process line; property propagation; balance equations; fluid composition; mass fraction; liquid food

1 Introduction

Fluid flow accounting for mass, thermal and momentum balances have been modelled in Modelica [6] and was reported in e.g. [9] (Eborn, 2001), [10] (Tummescheit, 2002) and [11] (Casella and Leva, 2003). The latter paper also addressed flow reversal.

A special case is liquid food process lines, e.g. lines for production of UHT milk in dairies. Dynamic simulation of this has been practiced during some years by means of a Modelica-based dynamic model library [1] (Skoglund, 2003), [2] (Skoglund and Dejmek, 2006), [5] (Skoglund, 2007) and [12] (Gäfvert et al, 2008). Beside the fundamental laws of conservation, e.g. mass and heat, the model library addressed particular characteristics of liquid food process lines. For example dynamic propagation of fluid properties was considered, [3] (Skoglund and Dejmek, 2006), due to the need of simulating start-up and shut-down with fluid changes, which are occurring frequently in the addressed applications.

Also typical for liquid food process lines is that flow directions in many cases are reversed. As fluid propagation has to be considered simultaneously, it has to be addressed in the models for correct mass and thermal balance equation, which otherwise may lead to non-physical behaviour and eventually cause a crash of the simulation. This article describes how the model library was adapted for reversed flow.

2 The "FoodProcessing" library

Since the start of the development of the "FoddProcessing" library [1] much more work was spent to address characteristics of liquid food process lines. Thus a model for axial-dispersed plug flow (ADPF) was derived [3] and some models were extended for reaction kinetics [8]. Figure 1 shows the library "FoodProcessing" in the Modelica tool Dymola [7], which was also described in earlier publications, [1], [2] and [12].

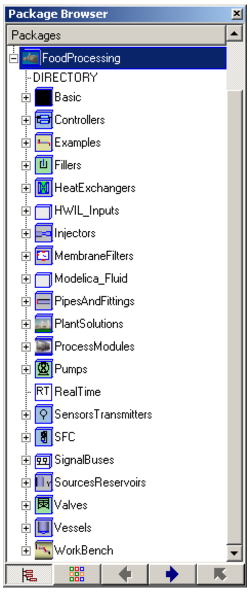


Figure 1. The "FoodProcessing" library, which has been described earlier in [1], [2] and [12].

DOI: 10.3384/ecp09430017

The library has since been used to configure many process lines to test various performances, e.g. product losses. Thus the arise, development and propagation of mixing zones was simulated for product filling and emptying in a commercial UHT line for milk sterilization [4] (Skoglund & Dejmek, 2007). The library was also used for trouble shooting and testing new design ideas as regards both process design and control algorithms. Furthermore simulation for functional test of complete process lines with the actual control system was enabled through further library development. Hence the library was adapted for real-time Hardware-in-the-loop simulation (HILS) [12]. Software for signal communication with the control system was developed too.

3 Reversed-flow processes

In some processes the direction of the fluid flow will vary as the plant state shifts. Figure 2 shows a typical case where different flow directions are normal at different plant states, e.g. filling and emptying of product (food medium) to/from a tank.

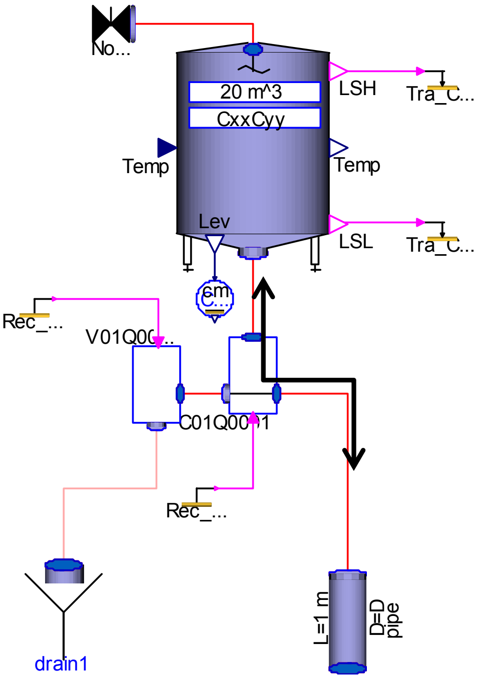


Figure 2. A typical case with two flow directions. The black, bold arrows show the product (food medium) flow at filling and emptying of the vessel.

In volumes and junctions mass and heat balances require that the properties from the correct connections are used in the balance equations. In Figure 2 this is valid for the tank and three-port valve at the bottom of the tank. Furthermore, in fluid channels (e.g. the pipe in Figure 2) it is important that fluid properties from the correct connection are used for the dynamic momentum balances and friction forces.

3.1 Transients – unintentional reversed flow

In many situations the flow direction should normally never change. However due to transients during valve transitions, the flow direction might be reversed for a short period of time, This might be enough to cause simulation problems if reversed flow is not handled adequately in the models. Figure 3 shows an example of that

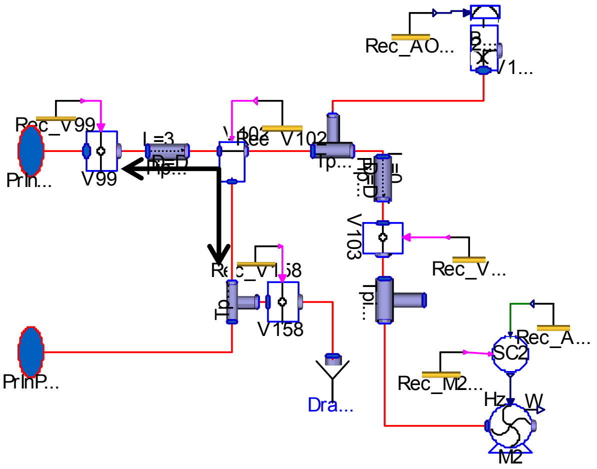


Figure 3. Example of case where the flow direction may be reversed just during state transition in a plant. The black, bold arrows indicate two flow directions, which both deviate from the normal.

3.2 The problem

The problem arises when flow direction and fluid properties not match. The problem occurs in balance equations for mass and heat. To illustrate the problem we can study a simple mass balance in a volume with one port for normal flow in and one port for normal flow out of a component. We can assume an incompressible fluid with constant density flowing through the constant volume, *V*, with a positive constant flow rate, *Q*. This yields the following mass balance for a certain fluid component with the mass fraction *X* in the volume, thus also flowing out from the volume.

$$V\frac{\partial X}{\partial t} = (X_{in} - X)Q \qquad \textbf{Eq. 1}$$

X_{in} is the mass fraction into the component through the normal input port. With all variables and constants positive, Eq. 1 describes a stable system where

X approaches X_{in}. If the flow rate changes direction, Eq. 1 changes character to Eq. 2, which describes an incorrect and unstable system.

$$V\frac{\partial X}{\partial t} = (X - X_{in})|Q| \qquad \textbf{Eq. 2}$$

The correct conversion of Eq.1 would be

$$V\frac{\partial X}{\partial t} = (X - X_{out})Q = (X_{out} - X)|Q| \qquad \textbf{Eq. 3}$$

X_{out} is the mass fraction into the component through the normal output port. This conditional rearrangement of the equations must be handled when the flow changes sign.

4 Reversed-flow models

The fundamental idea with the reversed-flow models presented in this paper is to use a `record` containing fluid properties declared as `input` and `output` in the connectors. Hence there is a need for two types of connectors "In" and "Out" to match the `input` with an `output` and the `output` with an `input`. This signal-based approach was suggested by Casella and Leva [11]. Thus the connectors must be connected in pairs "In" to "Out". Therefore all components are modelled for a "nominal" flow direction. The consequence is that some component models have to be made in more than one version to match different nominal flow directions. One example of that is the three-port valve in Figure 2, a valve which in some applications normally splits the flow and in other applications normally merges two flows. Another example is the T-pipes in Figure 3, where two of them normally split the flow whereas one of them normally merges two flows.

4.1 Modelica code for connectors

A Modelica code for the nominal flow direction "into" a component is given below.

```
connector ProductIn
  flow SIunits.VolumeFlowRate Q;
  SIunits.Pressure p;
  input ProductData PrData
    "Fluid data for forward flow";
  output ProductData PrDataRev
    "Fluid data for reversed flow";
end ProductIn;
```

In this example the thermal properties are not declared as a pair of flow and cross variables. The fluid properties are declared in the `record ProductData`. In [11] the `input` and `output` handled the specific enthalpy. In this case they include not only thermal properties, but also density and mass fraction properties for the fluid composition as this is important to consider in many liquid food cases (see e.g. [3])

Corresponding to the above connector is a connector for the nominal flow direction "out of" a component. The Modelica code is given below.

```
connector ProductOut
  flow SIunits.VolumeFlowRate Q;
  SIunits.Pressure p;
  output ProductData PrData
    "Fluid data for forward flow";
  input ProductData PrDataRev
    "Fluid data for reversed flow";
end ProductOut;
```

4.2 Conditional equations

As the flow direction alters, the fluid properties used in the equations, e.g. mass fraction balance, must swap the used data from the connectors. The following Modelica code shows the principle used in a constant volume in which the fluid (with the density `rho`) is mixed ideally (instantly).

```
 // Connectors
ProductIn PrIn "Nominal flow in";
ProductOut PrOut "Nominal flow out";
 // Fluid data
ProductData PrData "Actual in volume";
ProductData PrInData "Actual at nominal
flow in";
ProductData PrOutData "Actual at nomi-
nal flow out";
equation
 // Propagation of fluid properties
PrInData = if PrIn.Q >= 0 then
      PrIn.PrData else PrData; // "In"
PrOutData = if PrOut.Q <= 0 then
PrData else PrOut.PrDataRev; // "Out"
PrInData.PrDataRev = PrData; // Rev. in
 // Mass balance
der(m) = PrInData.rho*PrIn.Q +
         PrOutData.rho*PrOut.Q;
 // Mass fraction balance
mX = m*PrData.X;
mFlowX = PrInData.rho*PrIn.Q*PrInData.X
   + PrOutData.rho*PrOut.Q*PrOutData.X;
der(mX) = mFlowX;
```

5 Sensors for reversed flow

System simulations mostly need sensor models, Thus, in a system where the fluid may flow in either of the two directions, sensors for fluid properties must account for that, i.e. the property (e.g. concentration of a fluid component) must account for the flow direction. A sensor adaptor was modelled to pick up fluid properties from the flow path. A part of the code for that adaptor is given below.

```
connector ToSensor "Connector for sen-
sors"
 output ProductData PrData;
.
.
end ToSensor;

model SensorAdaptor
  ToSensor toSensor;
  ProductIn PrIn;
  ProductOut PrOut;
  equation
    toSensor.PrData = if PrIn.Q>=0 then
       PrIn.PrData else PrIn.PrDataRev;
     .
     .
  connect(PrIn, PrOut).
end SensorAdaptor;
```

The sensor models have a corresponding connector to the `ToSensor` to pick up the fluid properties.

6 Process examples and results

As already mentioned, reversed flow models are required both to get correct simulation results in systems where reversed flow takes place normally, and to get stable simulations in situations with short, unintentional reversed flow.

6.1 Normal situations with reversed flow

A simple example illustrating the solution of reversed flow models is shown in Figure 4. The system comprises two pressure tanks, a pipe, a centrifugal pump and two sensors for the concentration of a fluid component. (In this case the sensor adaptors are incorporated in the pipe.) Tank 1 has an initial content of 10 m^3 with a concentration of 10 % carbohydrates, while Tank 2 has an initial content of 2 m^3 with 0 % concentration. The pump M1 runs intermittently 250 seconds with a stop time equal long. During the pumping the fluid volumes in the vessels will change. The levels and gas pressures will change accordingly. The consequence is that after a while the flow will go in the reversed direction during the periods when the pump is stopped.

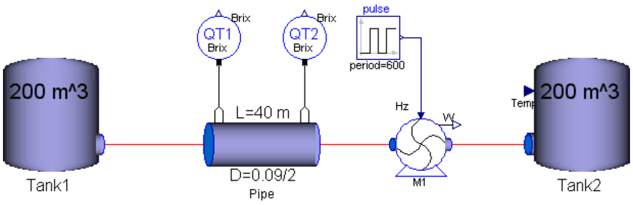


Figure 4. Example of simple system with forward and reversed flow

The pipe model is based on mass balance as axially dispersed plug flow (Eq. 4), and momentum balance (Eq. 5) in an incompressible fluid.

$$\frac{\partial C}{\partial t} + v\frac{\partial C}{\partial x} - D\frac{\partial^2 C}{\partial x^2} = 0 \qquad \textbf{Eq. 4}$$

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial p}{\partial x} + p_f + \rho g\frac{\partial z}{\partial x} = 0 \qquad \textbf{Eq. 5}$$

Explanations of symbols are given in section 8, Notation. Figure 5 shows the simulation result. Note; When the fluid changes flow direction there is a time lag – due to the transportation through the pipe – until the “new” fluid composition reaches the sensor at the current pipe exit. The sensors are modelled as ideal sensors with no dynamics.

Another more realistic example, though still simplified, is shown in Figure 6. It shows a milk supply, a valve cluster and two tanks. Figure 7 shows the simulation result of tank filling with milk, milk transport from tank to tank, and water flush of a tank.

6.2 Reversed flow during transitions – stability

The example in Figure 8 is used to illustrate the problem with instability in situations with short periods of reversed-flow without accounting for it. The example was run both without and with models accounting for reversed flow.

In the valve V1 the normal flow direction is from the sources of water and juice concentrate to the out port on the right hand side. The valve is of type change-over and connects either of the two ports from the sources of water or juice concentrate to the out port. During the valve transition (after 10 s), all three ports were connected to each other for a short moment of time and, due to the pressures and flow dynamics, the flow was reversed for about 1 s in the valve port for water. The simulation results are shown in Figure 9. In the case of models not accounting for the reversed flow, the simulation crashed after approximately 11.5 s (1.5 s after the valve activation). The reason is the incorrect balance equations, due to the reversed flow of water, which cause the non-physical increase of temperature and sugar concentration, which can be seen in the lower graphs of Figure 9. In the case with models accounting for the reversed flow, the temperature and sugar concentration were correct and the simulation did not crash.

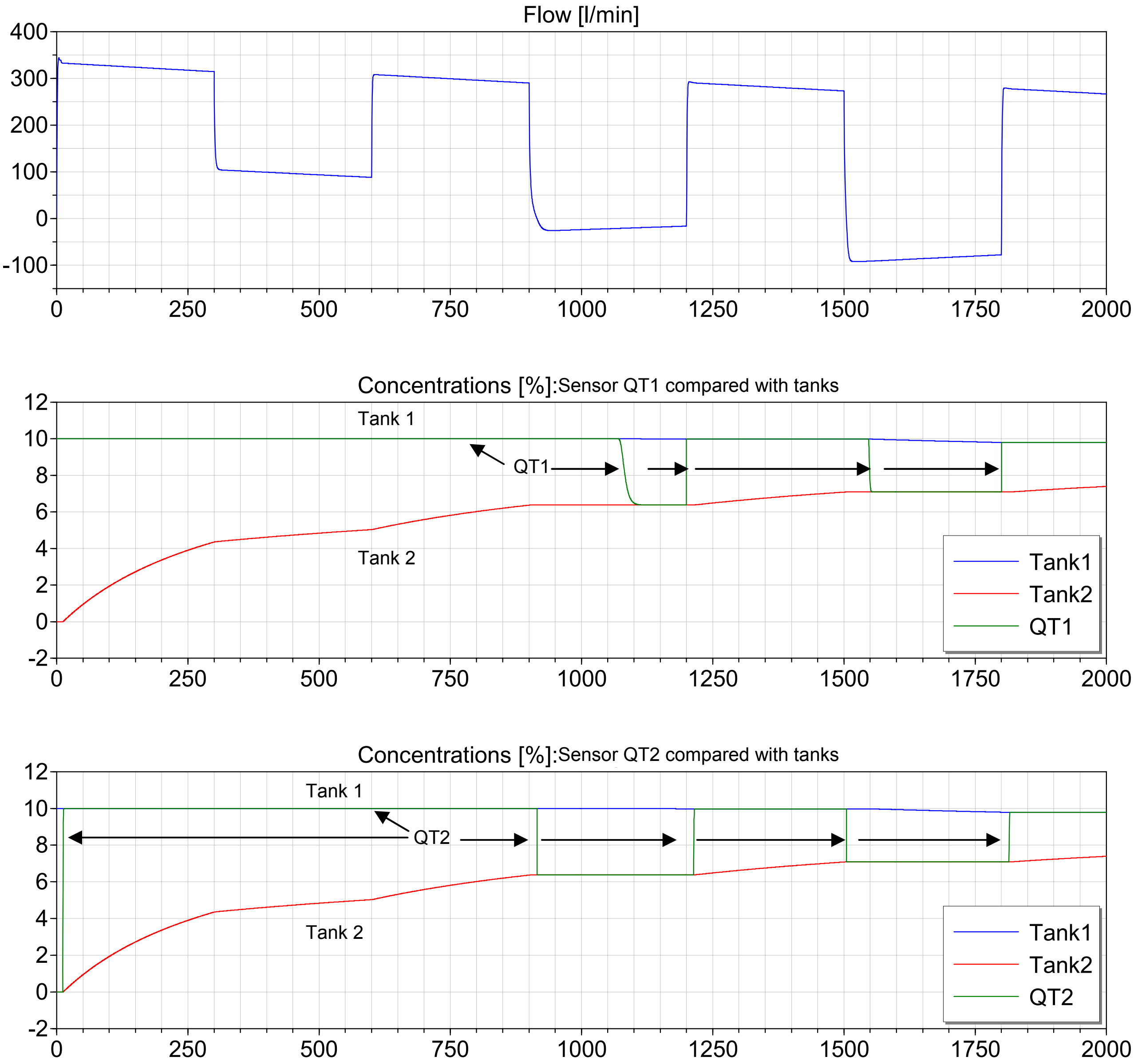


Figure 5. Simulation result of system shown in Figure 4. The time scale is in seconds. The concentrations in the tanks are shown in both middle and bottom figures to facilitate comparison. The locations of the sensors QT1 and QT2 are shown in Figure 4. Note; When the fluid changes flow direction there is a time lag – due to the transportation through the pipe – until the "new" fluid composition reaches the sensor at the current pipe exit. (For clarity in black/white print, some curves are marked directly in the graphs as a complement to the legends.)

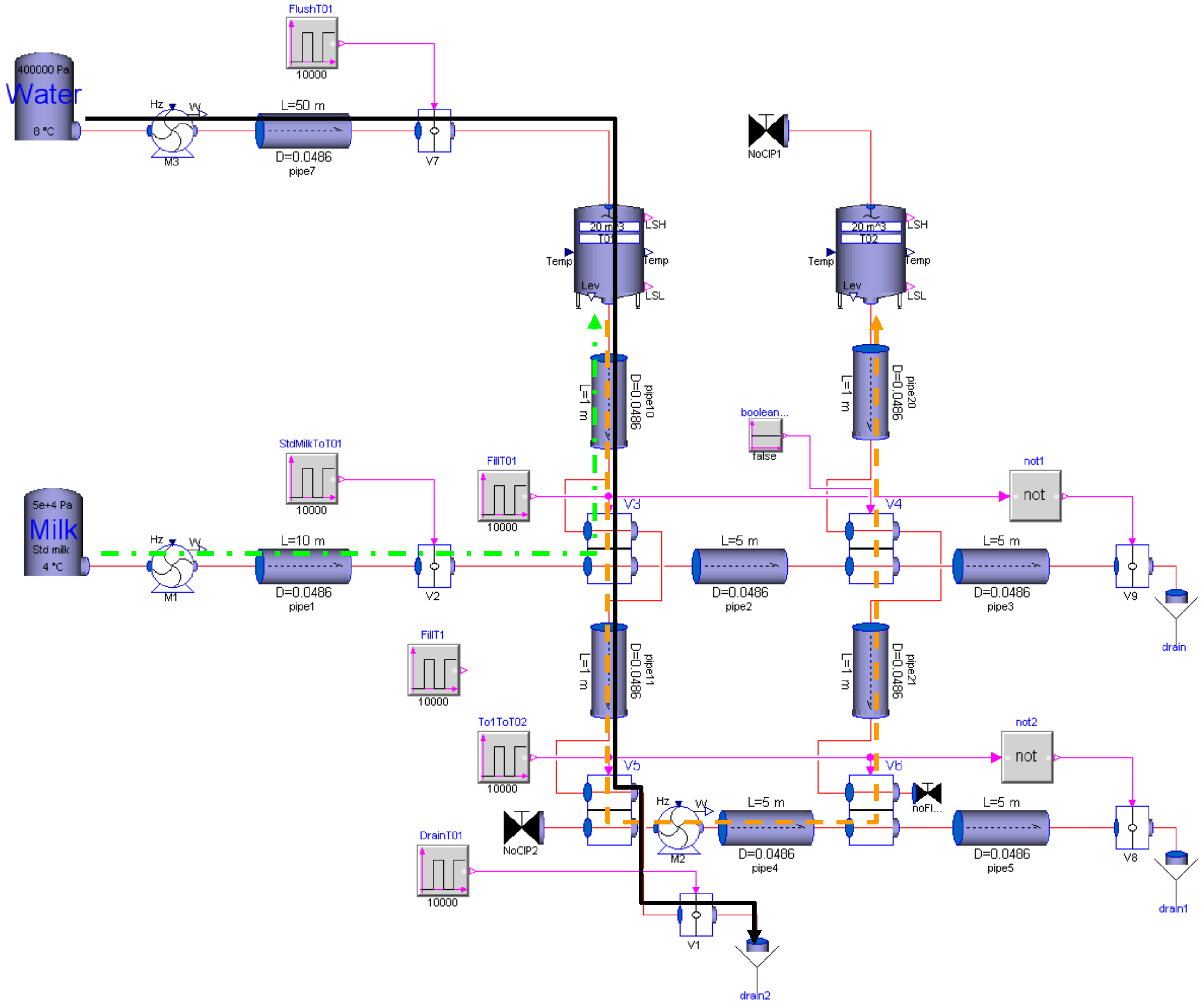


Figure 6. A typical, though simplified, valve cluster (the four valves: V3, V4, V5, V8) with two tanks. A sequence of three steps is run through. 1: (green, bold, dash dotted arrow) Milk is pumped to tank T01. 2: (orange, bpld, dashed arrow) The milk is pumped from tank T01 to tank T02. 3: (black, solid, bold arrow) Tank T01 is flushed with water to drain. During these three phases fluids are running in two directions through the tank T01, the valve V3 and the pipe pipe10. Simulation results are shown in Figure 7.

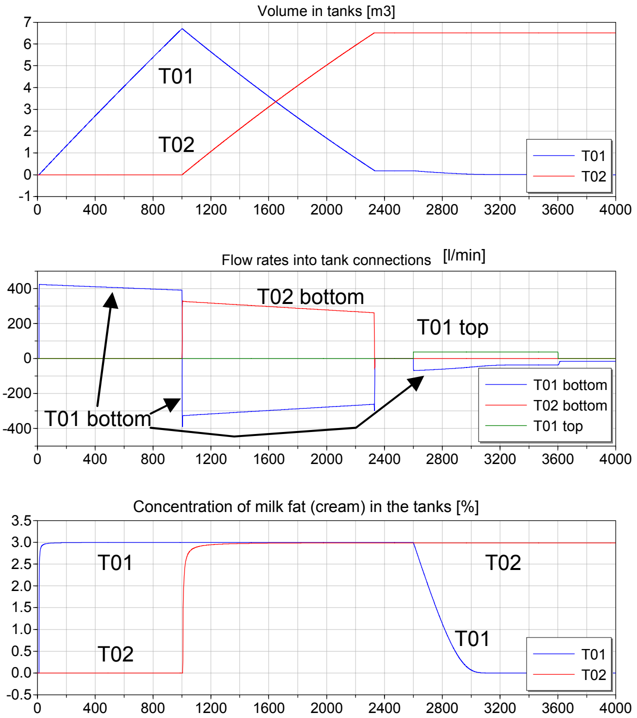


Figure 7. Simulation result of system shown in Figure 6. The time scale is in seconds. (For clarity in black/white print, the curves are marked directly in the graphs as a complement to the legends.)

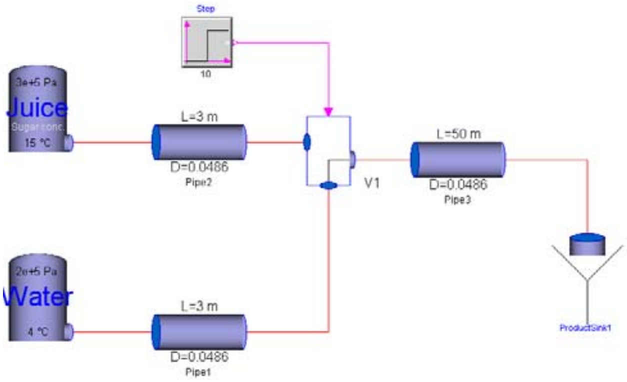


Figure 8. Simple model to illustrate problem with instability due to short, unintentional reversed flow. In the valve V1 the normal flow direction is from the sources of water and juice concentrate. The valve is a change-over type and connects either of the two ports from the sources of water or juice concentrate to the output. During the valve transition (at 10 s), all three ports were connected to each other for a short moment of time, which in this case caused a short moment of reversed flow of water

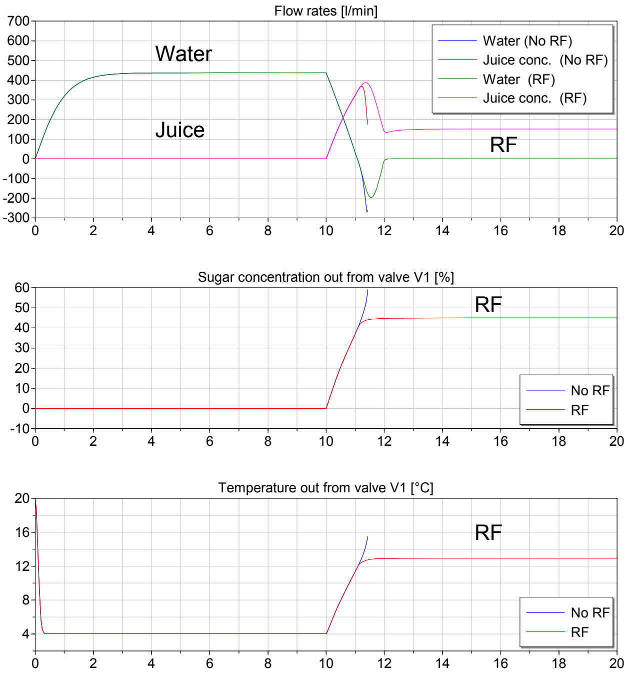


Figure 9. Simulation result of system shown in Figure 8. RF means "reversed flow model". As can be seen, the simulation without reversed-flow models was unstable and crashed after approximately 11.5 s, which was 1.5 s after the valve V1 was activated. (For clarity in black/white print, some curves are marked directly in the graphs as a complement to the legends.)

7 Conclusions

Fluid flow is often intentional bidirectional. In other cases, in spite of the intended unidirectional flow, *unintentional* reversed flow may occur during short periods of time. Therefore, models accounting for bidirectional fluid flow are required in many applications. Models accounting for this must address all properties propagating along the 1D flow coordinate, and the fact has to be considered, that the properties will differ in a connection point of a component depending on the flow direction in each moment of time.

The modelling problem is twofold; the connector problem and the balance equation problem. In this paper the solution was based on the input/output connector principle with conditional use of connector data in the heat balance, [11]. A solution, which also includes mass fraction balances, was demonstrated in this paper. Furthermore, models to adapt sensor models for arbitrary flow direction was proposed and demonstrated.

Correct and stable simulations were achieved for flow in arbitrary direction during simulations of fluid flow systems particularly relevant for liquid food applications.

8 Notation

C	Volumetric concentration, kg/m^3
D	Dispersion coefficient in axial direction, m^2/s
g	Constant of gravity acceleration, approx. 9.81 m/s^2
p	Pressure, Pa
p_f	Pressure drop due to flow friction, Pa
Q	Volumetric flow rate, m^3/s
t	Time, s
V	Volume, m^3
v	Mean velocity over a channel cross-sectional area, m/s
X	Mass fraction of a fluid component, 1
x	Axial spatial coordinate (along the fluid channel), m
z	Vertical spatial coordinate, m

Greek letters

ρ	Density, kg/m^3

References

[1] Skoglund, T., Simulation of Liquid Food Processes in Modelica. Proceedings of the 3rd International Modelica Conference 2003, 51-58. Linköping, Sweden, November 3-4, 2003, Organized by the Modelica Association and Linköping University, Sweden. Available at www.modelica.org.

[2] Skoglund, T. and Dejmek P., A model library for dynamic simulation of liquid food process lines. Proceedings of FOODSIM 2006, 5-12, Naples, Italy, June 15-17, 2006, Organized by EUROSIS.

[3] Skoglund, T. and Dejmek, P. A dynamic object-oriented model for efficient simulation of fluid dispersion in turbulent flow with varying fluid properties. Chemical Engineering Science, 2007, 62, pp. 2168-2178.

[4] Skoglund, T. and Dejmek, P., Fuzzy traceability – A process simulation derived extension of the traceability concept in continuous food processing. Trans IChem Part C, Food and Bio products Processing, 2007, 185, pp. 1-6.

[5] Skoglund, T., Dynamic Modelling and Simulation of Liquid Food Process Lines. Doctoral thesis, Department of Food Technology, Engineering and Nutrition, Faculty of Engineering, LTH, Lund University, Lund, Sweden, 2007. (ISBN 978-91-976695-1-1)

[6] Modelica Association, http://www.modelica.org

[7] Dynasim AB, http://www.dynasim.se

[8] Skoglund, T. and Dejmek, P., A dynamic object-oriented model for efficient simulation of microbial reduction in dispersed turbulent flow. Journal of Food Engineering, 2008, 86, pp. 358–369.

[9] Eborn, J. On Model Libraries for Thermo-hydraulic Applications. Doctoral thesis, Department of Automatic Control, Faculty of Engineering, LTH, Lund University, Lund, Sweden, 2001.

[10] Tummescheit, H., Design and Implementation of Object-Oriented Model Libraries using Modelica, Doctoral thesis, Faculty of Engineering, LTH, Lund University, Lund, Sweden, 2002.

[11] Casella, F. and Leva, A., Modelica open library for power plant simulation: design and experimental validation. Proceedings of the 3rd International Modelica Conference 2003. Linköping, Sweden, November 3-4, 2003, Organized by the Modelica Association and Linköping University, Sweden. Available at www.modelica.org.

[12] Gäfvert, M., Skoglund, T., Tummescheit H., Windahl, J., Wikander, H., Reuterswärd, P., Real-Time HWIL Simulation of Liquid Food Process Lines. Proceedings of the 6th International Modelica Conference 2008, 709-715. Bielefeld, Germany, March 3-4, 2008, Organized by the Modelica Association and University of Applied Sciences, Bielefeld, Germany. Available at www.modelica.org.

Control system design for the starch mashing phase in the production of beer

Alberto Leva Filippo Donida
Politecnico di Milano, Dipartimento di Elettronica e Informazione
Via Ponzio, 34/5 - 20133 Milano, Italy

Abstract

The starch mashing phase, the first one in the brewing process, has a fundamental influence on the quality of the final product. In particular, a good temperature control can significantly reduce the product variability, and also improve the process efficiency by (slightly) reducing the mashing phase duration. In this work, control-oriented models of the mashing process, including biochemical reactions' representation and energy balance equations, are used to synthesise and test some temperature control schemes. The mix of equation- and algorithm-based modelling allowed by Modelica allows to size the control equipment to the (nearly) final detail, including for example the comparison of different types of heating actuators.

Keywords: brewing; process control; process/control co-simulation

1 Introduction

Many models were proposed in the literature for the starch mashing phase in the brewing process, mostly with the aim of understanding the underlying biochemistry, and therefore correctly sizing the necessary equipment. This manuscript continues the previous work [7], where a well-established literature model (that given its purpose however takes the mash temperature as an exogenous variable) was complemented with suitable energy equations, so as to obtain a new model suitable for simulation studies aimed at the synthesis of the necessary temperature control. Based on said model, two studies are illustrated, First, the temperature profile typically used (that at present comes essentially from heuristic considerations) is re-designed so as to obtain the required final product composition with a (slightly) shorter mashing phase, to the advantage of process throughput and energy efficiency. Then, the control system is simulated both as a continuous-time system and including a (quasi-)*replica* model of the control code: in particular, the possibility of employing an on-off heating actuator instead of a modulating one is investigated, so that the devised control solution can be applied in the presence of realistic equipment.

2 The mashing model

The mashing model used here is based on the work by [2]. Through the enzymatic activity and the consequent degradation of the starch, the carbohydrates production is evaluated; the phenomenon is ruled by the amount of fermentable sugars (glucose, maltose, and maltotriose) that constitute the nourishment base for the yeast during the fermentation phase, and also by non-fermentable sugars (dextrins), that participate in the process, as shown by the equations later on. Through the knowledge of the initial compound of the barley malt (i.e., initial amount of sugar and dextrins, starch concentration, and amylase potential) and of the initial mash temperature, it is possible to determine the enzymatic activity and the carbohydrates concentration dynamics—see works such as [6, 3] for deeper discussions on the matter.

The key part of the model is in the prediction of the starch hydrolysis, since that phenomenon determines the quantity of fermentable carbohydrates in the wort, and therefore the alcoholic degree of the finally obtained beer. As for that particular reaction, the objective of mashing is to reach the maximal fermentable carbohydrates productivity, subject to a convenient (and frequently product-specific) specification on the final dextrins concentration (i.e., non-fermentable carbohydrates) so as to ensure the organoleptic

DOI: 10.3384/ecp09430083

qualities of the beer. The starch is firstly gelatinised, and then transformed into fermentable sugars (sucrose, maltose, and maltotriose) and non-fermentable sugars (dextrins) through the enzymatic activity. Note that also part of the dextrins will be transformed into fermentable sugars.

Obviously, gelatinisation is not an instantaneous phenomenon, since it depends on the concentration of the involved reactants. It therefore depends on temperature, and if temperature is taken as exogenous, its only dynamics comes from mass (concentration) balances; in that framework, the starch gelatinisation kinetic is represented as a first-order reaction. From some experiments it is noticed in the literature that at a given temperature T_g (typically around 60°C) a discontinuity is present in the phenomenon; to deal with that problem, the starch gelatinisation rate r_g, expressed in $g/kg\ s$, is traditionally modelled by two alternative equations, namely

$$r_g = k_{g1} \exp\left(-\frac{E_{g1}}{RT}\right)[S_s] \quad for\ T < T_g \tag{1}$$

and

$$r_g = k_{g2} \exp\left(-\frac{E_{g2}}{RT}\right)[S_s] \quad for\ T > T_g, \tag{2}$$

where $[Ss]$ is the ungelatinised starch concentration ($g/kg\ of\ mash$); E_{g1}, E_{g2} the activation energy (J/mol); k_{g1}, and k_{g2} the pre-exponential factor (s^{-1}); T the temperature (K) of the mash; T_g the threshold temperature (K) and R the gas constant ($8.31 J/mol\ K$). The relationship between the global enzymatic activity and the temperature may be represented as the composition of two terms: the temperature effect on the specific activity of each one enzyme site, and the coupled time-temperature effect on the denaturation of active sites. This leads to

$$r_{de\alpha} = k_{d\alpha} \exp\left(-\frac{E_{de\alpha}}{RT}\right)[E_\alpha] \tag{3}$$

and

$$r_{de\beta} = k_{d\beta} \exp\left(-\frac{E_{de\beta}}{RT}\right)[E_\beta], \tag{4}$$

where r_{de} is the reaction rate ($U/kg\ s$) of denaturation, $[E_\alpha]$ and $[E_\beta]$ the active site concentrations ($U/kg\ of\ mash$), $E_{de\alpha}$ and $E_{de\beta}$ the activation energies (J/mol) for the denaturation and $k_{d\beta}$ and $k_{d\beta}$ the pre-exponential factors (s^{-1}). The global enzyme activation rate (r_{ac} expressed in $U/kg\ s$) can be represented by

$$r_{ac\alpha} = k_{d\alpha} \exp\left(\frac{-E_{de\alpha}}{RT}\right)[E_\alpha]\, a_\alpha \tag{5}$$

and

$$r_{ac\beta} = k_{d\beta} \exp\left(\frac{-E_{de\beta}}{RT}\right)[E_\beta]\, a_\beta, \tag{6}$$

where a_α and a_β are the enzymatic site specific activities, that can be approximated by a suitable polynomial in the temperature range of interest (we omit details for brevity). The chemical reactions for the hydrolysis of starch and dextrins into glucose, maltose, and maltotriose are

$$\begin{gathered}
(C_6H_{10}O_5)_n + n(H_2O) \rightarrow n(C_6H_{12}O_6) \\
(C_6H_{10}O_5)_n + \frac{1}{2}n(H_2O) \rightarrow \frac{1}{2}n(C_{12}H_{22}O_{11}) \\
(C_6H_{10}O_5)_n + \frac{1}{3}n(H_2O) \rightarrow \frac{1}{3}n(C_{18}H_{32}O_{16}) \\
(C_6H_{10}O_5)_n \rightarrow x(C_6H_{10}O_5)_{n/x} \\
(C_6H_{10}O_5)_{n/x} + \frac{n}{x}n(H_2O) \rightarrow \frac{n}{x}n(C_6H_{12}O_6) \\
(C_6H_{10}O_5)_{n/x} + \frac{n}{2x}n(H_2O) \rightarrow \frac{n}{2x}n(C_{12}H_{22}O_{11}) \\
C_6H_{10}O_5)_{n/x} + \frac{n}{3x}n(H_2O) \rightarrow \frac{n}{3x}n(C_{18}H_{32}O_{16})
\end{gathered} \tag{7}$$

According to the reaction scheme (7) and the specific action of *alpha*− and *beta*−amylases, the kinetics for the gelatinised starch hydrolysis (expressed in $g/kg\ s$) into glucose, maltose, maltotriose and dextrins are, respectively, represented by

$$r_{gl} = k_{gl} a_\alpha [S_g] \tag{8}$$

$$r_{mal} = k_{\alpha,mal} a_\alpha [S_g] + k_{\beta,mal} a_\beta [S_g] \tag{9}$$

$$r_{mlt} = k_{mlt} a_\alpha [S_g] \tag{10}$$

$$r_{dex} = k_{dex} a_\alpha [S_g] \tag{11}$$

and similarly, the kinetics for the dextrins hydrolysis is given by

$$\acute{r}_{gl} = \acute{k}_{gl} a_\alpha [D] \tag{12}$$

$$\acute{r}_{mal} = \acute{k}_{\alpha,mal} a_\alpha [D] + \acute{k}_{\beta,mal} a_\beta [D] \tag{13}$$

$$\acute{r}_{mlt} = \acute{k}_{mlt} a_\alpha [D] \tag{14}$$

where k_{gl}, k_{mlt}, $k_{\alpha,mal}$, $k_{\alpha,mal}$, k_{dex}, $\acute{k}_{gl}$, $\acute{k}_{mlt}$, $\acute{k}_{\alpha,mal}$, $\acute{k}_{\beta,mal}$ are the kinetic factors ($kg/U\ s$), $[D]$ and $[S_g]$ the dextrins and the gelatinised starch concentrations ($g/kg\ of\ mash$), and a_α and a_β

are the real activity of *alpha* and *beta* amylase ($U/kg\,of\,mash$). Finally, the ordinary differential equations

$$\frac{d\,[S_s]}{dt} = -r_g \tag{15}$$

$$\frac{d\,[S_g]}{dt} = r_g - r_{gl} - r_{mal} - r_{mlt} - r_{dex} \tag{16}$$

$$\frac{d\,[D]}{dt} = r_{dex} - \acute{r}_{gl} - \acute{r}_{mal} - \acute{r}_{mlt} \tag{17}$$

$$\frac{d\,[gl]}{dt} = -r_{gl} + \acute{r}_{gl} \tag{18}$$

$$\frac{d\,[mal]}{dt} = -r_{mal} + \acute{r}_{mal} \tag{19}$$

$$\frac{d\,[mlt]}{dt} = -r_{mlt} + \acute{r}_{mlt} \tag{20}$$

express the carbohydrate concentration evolutions.

Starting from the above work, the objective here is to create models suitable for system studies related to control. It is therefore not appropriate to take the mash temperature as an exogenous variable (as is instead correct, according to the literature, for bio-chemical only studies). On the other hand, for our purposes the temperature dynamics has to be described based on the power generations and exchanges involved in the process, namely the heat released to the tank wall from the external in order to warm up the mass, the heat lost towards the external environment, and the heat coming from the saccharification reactions. Such an extension causes a notable increase of the model complexity, and to maintain that complexity to an acceptable level for system studies, we have to introduce a few assumptions. In detail,and quite reasonably for a system study, we assume that the wort is perfectly mixed, with homogeneous temperature and concentrations, specific heat and density of the components are constant in the considered range of temperature, and the heat contribution of the mechanical mixing actions are negligible.

The typical commercial tanks used have very different sizes and geometries. However, for this study, the tank is assumed to be a vertical cylinder (therefore yielding results immediately suitable also for a home-brewing context, incidentally); the exchange area (A) is the sum of the area of the wall around the cylinder plus the area of the bottom, and the net power entering in the wort contained in the vessel (in W) is

$$Q = Q_{rea} + Q_{exc} \tag{21}$$

where Q_{rea} is the power generated by the mashing process (obviously with its sign, as mashing is actually an endothermic reaction), and Q_{exc} the power exchanged with the wall and through the wort surface in contact with the air. The energy equation is therefore

$$Q = C_{mass}\frac{d(T_m)}{dt} \tag{22}$$

where T_m is the temperature (in K) of the mash and C_{mass} is the heat capacity (in J/K) of the mash

$$C_{mass} = M_{water}Cp_{water} + M_{grain}Cp_{grain} \tag{23}$$

with M_{water} and M_{grain} respectively the mass (in kg) of the water and the grains, and Cp_{water} and Cp_{grain} the specific heat (in $J/kg\,K$) of the water and the grain. The net power exchanged by the mash is a sum of a two terms, i.e.,

$$Q_{exc} = Q_{mw} + Q_{ma} \tag{24}$$

where Q_{mw} is the heat exchanged with the reactor wall and Q_{ma} is the heat exchanged by the mash in contact with the air above the mash,

$$Q_{ma} = U_{ma}A_{sup}(T_{ext} - T_m) \tag{25}$$

where U_{ma} is the overall heat transfer coefficient (in $W/m^2\,K$) between the mash and the air above, referred to a bulk air temperature as usual in similar cases; A_{sup} is the exchange area (in m^2) on the top of the mash ($A_{sup} = \pi r^2$) and T_{ext} is the external temperature (in K) of the environment where the tank is located, and

$$Q_{mw} = U_{mw}A_{wall}(T_{wall} - T_m) \tag{26}$$

where U_{mw} is the overall heat transfer coefficient between the mash and the reactor wall; A_{wall} is the exchange area, and T_{wall} is the temperature of the reactor wall. As for the reactor wall temperature, apparently another energy equation is needed, i.e.,

$$C_{wall}\frac{d(T_{wall})}{dt} = Q_{we} - Q_{mw} \tag{27}$$

where Q_{we} is the heat exchanged by the wall in contact with the external air and C_{wall} is the heat capacity of the reactor wall

$$C_{wall} = M_{reactor}Cp_{metal} \tag{28}$$

with $M_{reactor}$ the mass (in kg) of the reactor and Cp_{metal} the specific heat (in $J/kg\,K$) of the metal which compose the tank. The overall mashing reaction is endothermic, thus needs heat to take place. This heat (Q_{rea}) is taken away from the mash is the sum of three terms: the power generated by the glucose production Q_{glu}, by the maltose production Q_{mal} and by the maltotriose production Q_{mlt}.

$$Q_{rea} = Q_{glu} + Q_{mal} + Q_{mlt} \tag{29}$$

For brevity, we report here only the equations for the power originating from the glucose production; those related to maltose and maltotriose are analogous. The power generated by a reaction is typically in the form

$$Q_{reaction} = -ReactionRate * ReactionEnthalpy \tag{30}$$

where the enthalpy of reaction H is the sum of the products' enthalpies minus the reactants' enthalpies.

$$H = H_{prod} - H_{reac}. \tag{31}$$

This enthalpy is however a *standard enthalpy*, calculated in a standard condition (atmospheric pressure and 25°C). This is not our case since the mash temperature is variable. Thanks to the Kirchhoff's equation, we can calculate the reaction enthalpy considering the condition of our process. Kirchhoff's equation (for the glucose production) is represented by

$$H_{reac} = H0_{reac} + C_{glu}\Delta T \tag{32}$$

where $\Delta T = T_m - T_0$ with T_0 the standard temperature (25°C) and T_m the mash temperature; C_{glu} is the specific heat capacity of reaction; $H0_{reac}$ is the enthalpy of the reaction in the standard condition which is (looking 31) equal to

$$H0_{glu} = Hf_{glu} - Hf_{water} - Hf_{starch} \tag{33}$$

As for C_{glu}, it is the difference between the sum of the specific heat capacities of the products and of the reactants ($Cp_{prod} - Cp_{reac}$), thus

$$C_{glu} = Cp_{glu} - Cp_{water} - Cp_{starch} \tag{34}$$

and of course, the same calculations can be done also for the maltose and maltotriose production. The main part of the Modelica implementation of the so derived model is shown below.

```
//--- energy balances -----------------------------
  Cmash*der(Tm) = Qmash_air + Qmash_tank + Qreact;
  Cmash = Mwater*CpWater + Mgrains*CpMalt;
  Qmash_air = Umash_air*SupArea*(Te - Tm);
  Qmash_tank = -Qbp-Qbb;

  Clat*der(Tlat) = Qbp + Qpe + Qbap;
  Clat = CpMetal*MvessLat;
  Qbp = LatArea *Umash_tank*(Tm - Tlat);
  Qpe = LatArea *Utank_outside*(Te - Tlat);
  Qbap = (pi*radO^2-pi*radI^2)*lambda_m/Lrifbp
         *(Tbase-Tlat);

  Cbase*der(Tbase) = Qbb + Qbe + Q_gasburner - Qbap;
  Cbase= CpMetal*MvessBase;
  Qbb=SupArea*Umash_tank*(Tm - Tbase);
  Qbe=SupArea*Utank_outside*(Te - Tbase);

  radO=radI+LatThick;
  volume = radI^2*pi*level;
  volume = Mwater/densWater + Mmalt/densMalto;
  ContactArea = pi*radI^2 + level*radI*2*pi;
  SupArea = radI^2*pi;
  LatArea =level*radI*2*pi;
  MvessLat=2*radI*pi*H*LatThick *densMetal;
  MvessBase=radI^2*pi*BaseThick *densMetal;


//--- reaction powers -----------------------------
  Qreact = Qglu + Qmal + Qmlt;
  Qglu = -((Rgl + R1gl)*Mgrains)*Hglu;
  Hglu = H0glu/mmGlu + Cglu*(Tm - T0);
  H0glu = HfGlu - HfWater - HfStarch;
  Cglu = CpGlu - CpStarch - CpWater;
  Qmal = -((Rmal + R1mal)*Mgrains)*Hmal;
  Hmal = H0mal/mmMal + Cmal*(Tm - T0);
  H0mal = HfMal/2 - HfWater/2 - HfStarch;
  Cmal = CpMal/2 - CpStarch - CpWater/2;
  Qmlt = -((Rmlt + R1mlt)*Mgrains)*Hmlt;
  Hmlt = H0mlt/mmMlt + Cmlt*(Tm - T0);
  H0mlt = HfMlt/3 - HfWater/3 - HfStarch;
  Cmlt = CpMlt/3 - CpStarch - CpWater/3;

//--- mass balances -------------------------------
  M = Mwater + Mmalt;
  Mmalt = Mcarbo + Mother;
  Mcarbo = Mglu + Mmal + Mmlt + MstarchGel
         + MstarchNG + Mdex;
  Mother = ((1 - AmIni - GluIni - MalIni - MltIni
          - DexIni)*Mgrains);
  Mglu = Mgrains*glu;
  Mmal = Mgrains*mal;
  Mmlt = Mgrains*mlt;
  MstarchGel = Mgrains*Sg;
  MstarchNG = Mgrains*Ss;
  Mdex = Mgrains*D;
  TotProd = glu + mal + mlt + D - GluIni - MalIni
          - MltIni - DexIni;

//--- sugars' creation rates ---------------------
  Rg = Kg*Ss;
  Kg1 = kg1*exp(-Eg1/(R*Tm));
  Kg2 = kg2*exp(-Eg2/(R*Tm));
  s = arctan((Tm - Tg)*kk)/pi + 0.5;
  Kg = Kg1*(1 - s) + Kg2*s;
  Rgl = kgl*RealActAlfa*Sg;
  Rmal = kamal*RealActAlfa*Sg + kbmal*RealActBeta*Sg;
  Rmlt = kmlt*RealActAlfa*Sg;
  Rdex = kdex*RealActAlfa*Sg;
  R1gl = k1gl*RealActAlfa*D;
  R1mal = k1amal*RealActAlfa*D + k1bmal*RealActBeta*D;
  R1mlt = k1mlt*RealActAlfa*D;
  der(glu) = Rgl + R1gl;
  der(D) = Rdex - R1gl - R1mal - R1mlt;
  der(mal) = Rmal + R1mal;
  der(mlt) = Rmlt + R1mlt;
  der(Sg) = Rg - Rgl - Rmal - Rmlt - Rdex;
  der(Ss) = -Rg;
  der(Ealfa) = -RdeAlfa;
  der(Ebeta) = -RdeBeta;

//--- enzimatic activities ---------------------------
  RdeAlfa = KdeAlfa*exp(-EdeAlfa/(R*Tm))*Ealfa;
  RdeBeta = KdeBeta*exp(-EdeBeta/(R*Tm))*Ebeta;
  RealActAlfa = aalfa*Ealfa;
  RealActBeta = abeta*Ebeta;
  aalfa = if aalfa < 0 then 0
     else if Tm < 315 then 1
     else 1.3333333e-07*(Tm -300)^6 - 2.2e-05
          *(Tm - 300)^5+ 0.00144333333333*(Tm - 300)^4
          - 0.0490499999999 *(Tm - 300)^3
          + 0.92383333333122*(Tm - 300)^2
          - 8.89999999997759*(Tm - 300)
          + 34.299999999904;
  abeta = if abeta < 0 then 0
      else if Tm < 304 then 1
      else if Tm >= 304 and Tm < 336
           then 0.049*Tm - 13.9
      else if Tm >= 336 and Tm < 343
           then -0.374*Tm + 128.23
      else 0;
```

In accordance with the notation introduced and used above, the reported code should be practically self-explanatory.

3 Mashing temperature control

Temperature control plays a decisive role for the decomposition of malts, and is therefore crucial for beer qualities such as stability and taste [5], while bing also of interest for the overall brewing process, see e.g. [4]. Sticking however to the mashing process, the saccharification temperature control should track the required set point curve (pre-specified based on the ingredients and the desired product characteristics, so that it can be thought of as a recipe *datum*) quickly, accurately and without overshoots during the temperature rise; the same control should also exhibit good load disturbance rejection properties, so as to rapidly lead the temperature back to the setpoint should the system exhibit any error caused by external disturbances.

A mashing kettle normally has a large volume, and therefore the temperature dynamics is a lag-dominant process. Given that, generally, many breweries control the temperature by a standard PID algorithm, or similar ones, the main differences residing in the actuation mechanism, see e.g. [1]; we therefore adhere to that approach while structuring our schemes.

The first scheme considered is a single-loop temperature control. To enhance the significance of the study, the scheme is applied to the mashing phase, with the dimensions of the vessel compatible with a home brewing case, where simple controls are more likely to be used than in industrial brewing. In the typical home brewing mashing vessel (with a volume of say 25 litres), the available heating actuator is a gas burner, placed below the vessel base, and fed through a modulating valve (see later on for considerations on the possible use of on/off actuators, however, form more realistic a setting); the available thermal power from such an actuator is about 5 kW.

At first, we can introduce some slight modifications to the mashing vessel model in order to account for the fact that heat is only released to the bottom of the vessel. The walls exchange heat with the external air, but also with the heated vessel bottom. For the vessel base we thus have:

$$\begin{aligned} C_{base} &= M_{base} Cp_{Metal} \qquad (35) \\ C_{base}\frac{d(T_{base})}{dt} &= Q_{fb} + Q_{gasb} + Q_{be} - Q_{bap} \qquad (36) \end{aligned}$$

with M_{base} the mass (in kg) of the reactor's base, T_{base} its temperature and Cp_{metal} the specific heat (in $J/kg\,K$) of the metal which composes the tank; Q_{fb} is the heat exchanged between the mash and the base; Q_{gasb} the power contribution from the gas burner, Q_{be} the heat exchanged with the external environment and Q_{bap} the heat exchanged with the tank wall. For the vessel wall, instead, we have

$$\begin{aligned} C_{wall} &= M_{wall} Cp_{Metal} \qquad (37) \\ C_{wall}\frac{d(T_{wall})}{dt} &= Q_{fw} + Q_{we} + Q_{bap} \qquad (38) \end{aligned}$$

where M_{wall} is the mass of the reactor's wall and T_{wall} its temperature; Q_{we} the heat exchanged with the external environment. To notice how rightly the power Q_{bap} pass by vessel's base through the wall; Q_{bap} is the heat exchanged between the base of the vessel and the wall, written as

$$Q_{bap} = (\pi r_O^2 - \pi r_I^2)\frac{\lambda_m}{L_{rif}}(T_{base} - T_{wall}) \qquad (39)$$

where r_O and r_I are respectively the internal and external radius of the vessel, then the first term represents the annulus formed by the difference between the internal and external circumferences;

λ_m the thermal conductivity of the metal (typically aluminium or steel) and L_{rif} the reference length for the heat exchange. Q_{fb} and Q_{fw} are given by

$$Q_{fb} = U_{fb} A_{base} (T_m - T_{base}) \tag{40}$$

$$Q_{fw} = U_{fw} A_{wall} (T_m - T_{wall}) \tag{41}$$

where U_{fb} and U_{fb} are respectively the overall heat transfer coefficient between the mash and the reactor base and between the mash and the reactor wall; A_{base} the area of the base given by $A_{base} = \pi r_I^2$ and A_{wall} the area of the wall given by $A_{wall} = 2 r_I \pi H$, with H the wort level. Finally we have to calculate the heat exchanged by the tank with the outside. This is the sum of two terms:

$$Q_{be} = U_{be} A_{base} (Te - T_{base}) \tag{42}$$

$$Q_{we} = U_{we} A_{wall} (Te - T_{wall}) \tag{43}$$

where U_{be} and U_{we} are the overall heat transfer coefficient between the base and the outside and between the wall and the outside; Te the outside temperature. Then, the heat lost by the mash is

$$Q_{loss} = Q_{ma} - Q_{fb} - Q_{fw} \tag{44}$$

where Q_{ma} is the heat exchanged by the mash in contact with the air above.

$$Q_{ma} = U_{ma} A_{sup} (T_{ext} - T_m) \tag{45}$$

where U_{ma} is the overall heat transfer coefficient (in $W/m^2\,K$) between the mash and the air above; A_{sup} the exchange area on top of the mash. We assume for simplicity here that the heat released by the burner itself to the vessel is related to the heater command by the first order transfer function

$$G(s) = \frac{\mu}{1 + Ts} \tag{46}$$

Notice how simple it is, with the proposed approach, to devise quite accurate a model, easy to parametrise with dimensional data, and suitable for control studies. Control examples are reported later on, using as controller blocks an analogue PI with antiwindup, a digital PID in the ISA form with antiwindup and tracking, and a digital ISA PI(D) with antiwindup, tracking, and on-off (time division) output, the Modelica code for the first and third of said controller blocks (the second is a mere restriction of the third) is shown below.

```
model PI
  parameter Real CSmax=1;
  parameter Real CSmin=0;
  parameter Real k=0.7;
  parameter Real Ti=280;
  parameter Real b=0;
  Real SPf;
  Real xRff;
  Real fbOut;

public
  Modelica.Blocks.Interfaces.RealInput PV;
equation
  CS = max(CSmin, min(CSmax, k*(SPf-PV) + fbOut));
  Ti*der(fbOut)+fbOut= CS;
  Ti*der(xRff)+xRff = (1-b)*SP;
  SPf = b*SP+xRff;
public
  Modelica.Blocks.Interfaces.RealInput SP;
  Modelica.Blocks.Interfaces.RealOutput CS;
initial equation
  SPf = SP;
end PI;
model digital2dofPID_TDO
  parameter Real K = 1 "Gain";
  parameter Real Ti = 10 "Integral time [s]";
  parameter Real Td = 0 "Derivative time [s]";
  parameter Real N = 5 "Derivative filter ratio [#]";
  parameter Real b = 1 "SP weight in P action [#]";
  parameter Real c = 0 "SP weight in D action [#]";
  parameter Real CSmax = 100 "Maximum CS";
  parameter Real CSmin = 0 "Minimum CS";
  parameter Real TDsteps = 100 "TDO resolution";
  parameter Real Ts = 0.1 "Sampling time [s]";
  Real counter;
//protected
  Real sp;
  Real spo;
  Real dsp;
  Real pv;
  Real pvo;
  Real dpv;
  Real dp;
  Real di;
  Real d;
  Real dold;
  Real dd;
  Real cs;
  Real cso;
  Real dcs;
  Real StepsUp;
public
  input Modelica.Blocks.Interfaces.RealInput SP;
  input Modelica.Blocks.Interfaces.RealInput PV;
  output Modelica.Blocks.Interfaces.RealOutput CS;
  Modelica.Blocks.Interfaces.RealInput TR;
  Modelica.Blocks.Interfaces.BooleanInput TS;
algorithm
  when sample(0,Ts/TDsteps) then
       counter := counter+Ts/TDsteps;
         if counter>=Ts then
            // Compute control signal
            counter :=0;
            sp := SP;
            pv := PV;
            dsp := sp-spo;
            dpv := pv-pvo;
            if not TS then
               dp := K*(b*dsp-dpv);
               di := K*Ts/Ti*(sp-pv);
               d  := (Td*pre(d)+K*N*Td*(c*dsp-dpv))
                   /(if Td>0 then Td+N*Ts else 1);
               dd := d-dold;
               dcs := dp+di+dd;
               cs := cso + dcs;
             else
               cs := pre(TR);
             end if;
             if cs>CSmax then
                cs:=CSmax;
             end if;
             if cs<CSmin then
                cs:=CSmin;
             end if;
             spo := sp;
             pvo := pv;
             cso := cs;
             dold := d;
             StepsUp := floor
                     ((cs-CSmin)/(CSmax-CSmin)*Ts);
         end if;
         CS := if (counter<StepsUp) then CSmax
               else CSmin;
  end when;
end digital2dofPID_TDO;
```

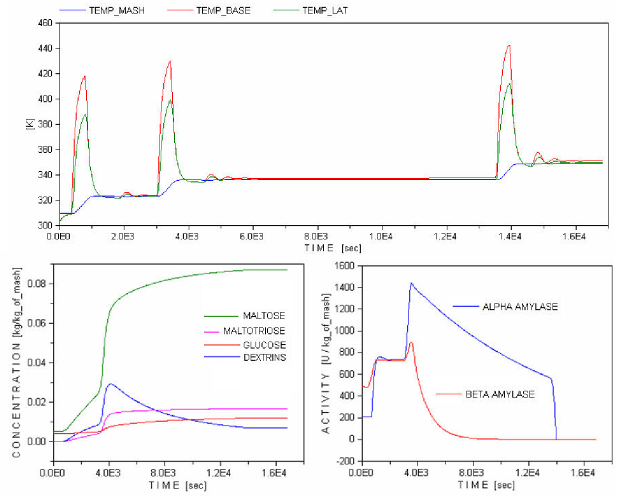


Figure 1: Evolution of the main variables in the gas-burner heated mash. In the first box temperature of the vessel's base, vessel's wall and mash. In the second box the carbohydrate evolutions and in the third one the enzymatic activities.

4 Simulation examples

4.1 Simulation model assessment

The first example aims at validating the model presented above, that includes both biochemical and energy-related phenomena, with respect to biochemical-only ones, and considering essentially carbohydrates' concentrations and enzymatic activities. To do so, the temperature is controlled via a PI regulator acting on the heating power, and "reasonably" tuned by hand based on some simulated open-loop responses, and the evolution of the variables of interest is compared to that obtained by impressing the temperature (with no energy equation, like in [2]) instead of giving only its set point, and having the temperature determined by energy phenomena.

Figure 1 reports an example of such tests. As can be seen, with a well functioning temperature control, the relevant process variables follow the expected behaviour closely enough, and both carbohydrates' concentrations and enzymatic activities are practically identical to those obtained with the models in [2]. The model presented here can thus be taken as reliable for system studies aimed at control design.

4.2 Analogue control

To further witness the obtained results, we now present a mashing simulation where temperature is controlled by a cascade structure, having the heating fluid flow rate as the inner variable, instead of a single loop. In figure 2 an industrial tank is considered, that contains $1000 kg$ of water and $300 kg$ of grains: through our control structure, we try to have the mash follow the same temperature profile as in the previous simulation (with quite a different plant setup, notice). Observe the good temperature evolution of the mash in response to the reference profile, in particular the correct rise of about 1 K/min. A fast and constant rise of the temperature toward the set point is considered important in mashing, which means that the presented control is performing definitely well. As a particular advantage of the cascade structure, we can also show what happens if, upstream of the valve, a significant temperature decrease occurs (before 8000 sec, from $417K$ to $390K$, in $10sec$). We can notice how the valve quickly reacts, thanks to the internal loop of the cascade structure; this compensation allows the heating jacket temperature to stay almost constant, and consequently the same to be true for the mash temperature inside the tank. After 10000 sec heating fluid temperature comes back to the previous value. As for the heat exchanges, the thermal energy generation and transfer that influence the mash temperature can be observed. As shown in equations 21 and 24 the mash is affected by lost heat, by the heat produced in the saccharification reaction and by the heat exchange through the tank wall. In figure 3 these thermal powers are shown for the first $100min$ of mashing.

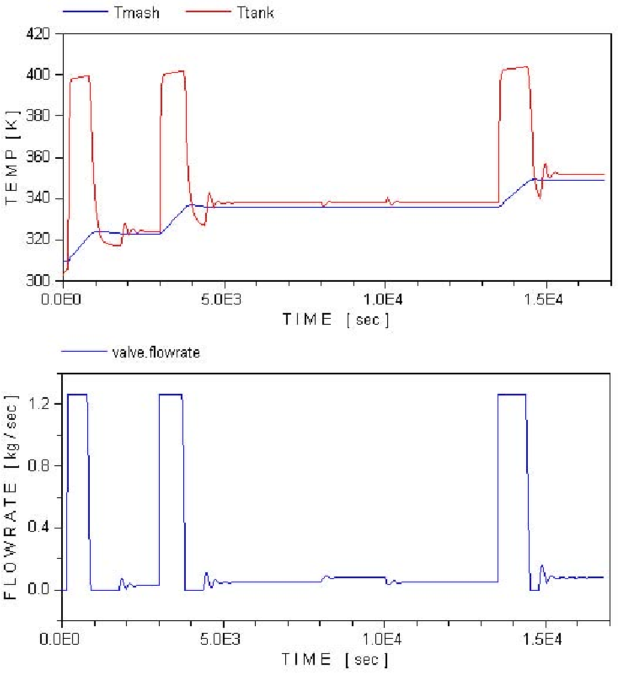


Figure 2: Mashing cascade control.

Having such variables available can apparently be of help for equipment sizing, and overall process and control commissioning.

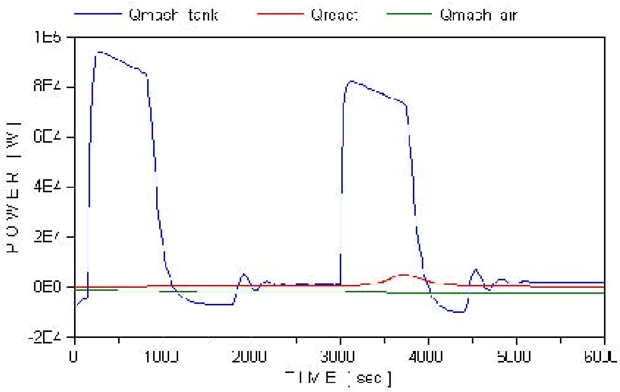


Figure 3: Thermal powers through the mash.

4.3 Analogue versus digital control and on-off actuation

In this section we briefly present two simulated versions of a single-loop mashing temperature control scheme, one (entirely in the continuous time) aimed at control law commissioning and/or regulator parameter tuning, the other (hybrid) at control equipment specification and assessment. The diagram of the Modelica model used for the reported test is shien in figure 4.

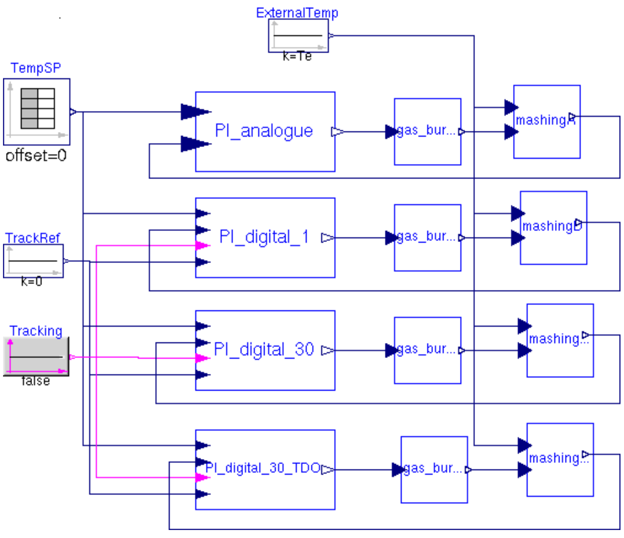


Figure 4: Diagram of the Modelica model used for the simulations on analogue versus digital control, and on-off actuation.

The controller is a PI with normalised gain of 0.6, and an integral time of 340s (the way that tuning was achieved is irrelevant for the scope of this manuscript). Figure 5 shows the results obtained with that PI as an analogue controller and as a digital one, with a sampling tome of 1s (suitable for a modulating actuator) and of 30s (suitable for an on-off one, managed by a time division output as is typically done in such cases). It can be seen that the analogue controller behaves very well, while the other two (hybrid) simulated systems show that the situation can be successfully managed also with the time division controller and the on-off actuator. In addition, some differences in the initial phase are evidenced, that are due to the different way the antiwindup is realised in the analogue and the digital PI. Hence, the presented models allow for an effective simulation, actually useful for both the control law synthesis, and the sizing of the corresponding equipment.

Finally, figure 6 compares, through a hybrid simulation, the PI of figure 5 with sampling time of 30 seconds and its “time division output” version, where the control signal is interpreted as the duty cycle of the activation for an on-off actuator, of course with a base period of 30 seconds. Notice how the conclusions previously drawn are confirmed by this definitely realistic simulation.

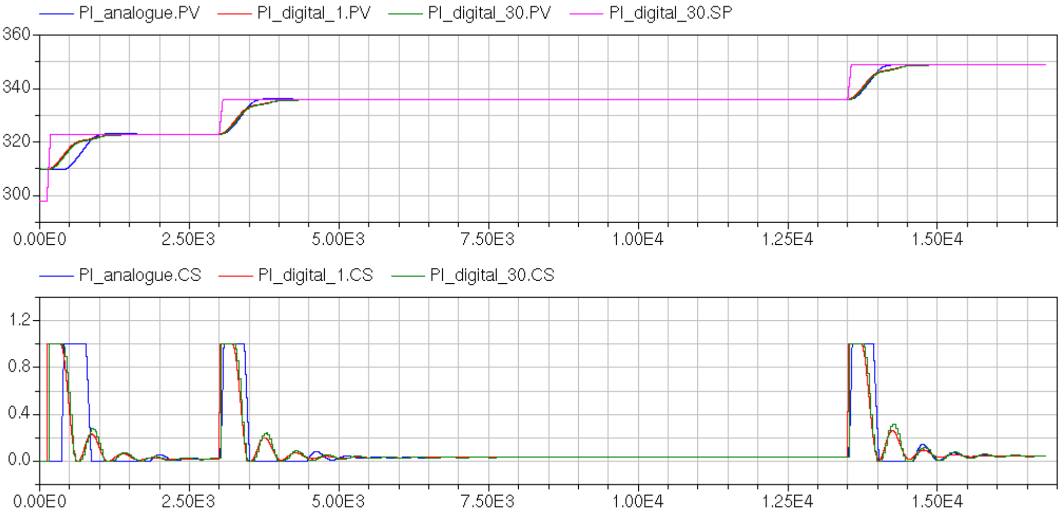


Figure 5: Analogue and digital mashing temperature control: set point and controlled variable (upper plot), control signal (lower plot).

5 Conclusions and future work

Literature models of the starch mashing phase in the brewing process were complemented with suitable energy balance equations, so as to make them suitable for the design and the simulation-based assessment of the corresponding temperature controls. Models were implemented in the Modelica language, and verified against literature data. Temperature control schemes were then presented, that were set up with one of those (first principle) models, in order to illustrate the usefulness and practical applicability of the idea. It is worth noticing that the models are suitable for control studies were the overall system is simulated as a continuous-time one, and also as a hybrid one including both time- and event-based controllers, so as to allow both for the tuning of a control law, and the sizing and verification of the corresponding algorithm and equipment.

Future developments will include more extensive use of model-based control techniques, more extensive validations of both the models and the control schemes developed with them, and also the integration of model forecasts and process measurements, to further improve the control performance. The obtained results are also being ported into the OpenModelica environment [8] to foster their diffusion.

References

[1] E. Alvarez, J.M. Correa, J.M. Navaza, and C. Rivero. Injection of steam into the mashing process as alternative method for the temperature control and low-cost of production. *Journal of Food Engineering*, 43(4):193–196, 2000.

[2] C. Brandam, X.M. Meyer, J. Proth, P. Strehaiano, and H. Pingaud. An original kinetic model for the enzymatic hydrolysis of starch during mashing. *Biochemical Engineering Journal*, 13(1):43–52, 2003.

[3] B. de Andrés-Toroa, J.M. Girón-Sierra, J.A. López-Orozco, C. Fernández-Conte, J.M. Peinado, and F. Garcia-Ochoa. A kinetic model for beer production under industrial operational conditions. *Mathematics and Computers in Simulation*, 48(1):65–74, 1998.

[4] D.A. Gee and W.F. Ramirez. Optimal temperature control for batch beer fermentation. *Biotechnology and Bioengineering*, 31(3):224–234, 1988.

[5] S. Jiliang, Y. Wei, and G. Dexin. Study of compound optimal control for beer saccharification temperature. In *Proc. 26th Chinese Control Conference*, pages 356–359, Zhangjiajie, PR China, 2007.

[6] T. Koljonen, J.J. Hämäläinena, K. Sjöholmb, and K. Pietiläb. A model for the prediction of fermentable sugar concentrations dur-

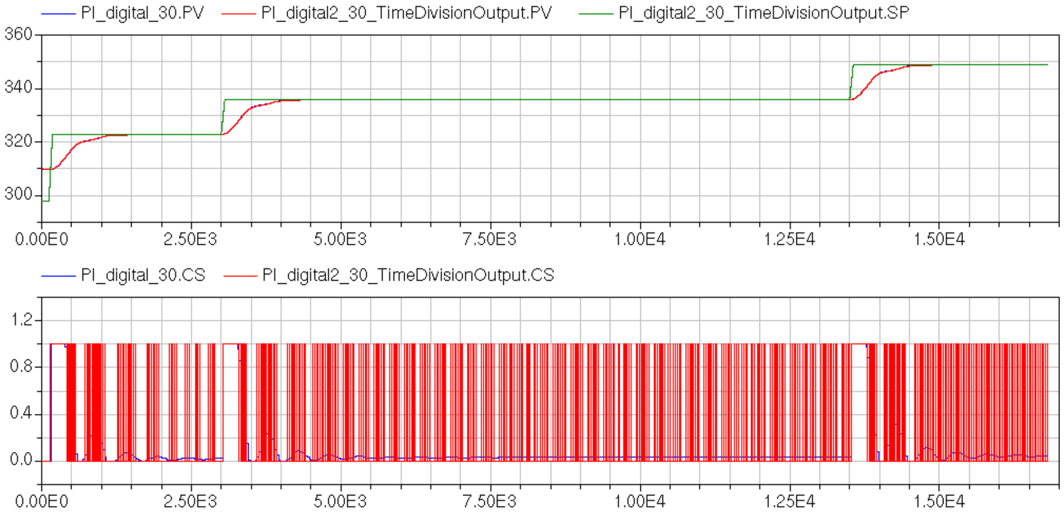


Figure 6: Comparison between the discrete-time and time division output PIs: set point and controlled variable (upper plot), control signal (lower plot).

ing mashing. *Journal of Food Engineering*, 26(3):329–350, 1995.

[7] A. Leva, F. Donida, and M. Bordoni. Object-oriented modelling and simulation of starch mashing. In *Proc. 3rd IFAC Conference on Analysis and Design of Hybrid Systems*, Zaragoza, Spain, 2009.

[8] The Open Source Modelica Consortium. *OpenModelica home page*, 2009. http://www.openmodelica.org.

Optimal Robot Control Using Modelica and Optimica

Martin Hast[a] Johan Åkesson[a,b] Anders Robertsson[a]
[a)] Department of Automatic Control, LTH,
Lund University, Sweden
[b)] Modelon AB, Sweden

Abstract

In this paper, Modelica along with Optimica is used to formulate and solve a minimum time optimization problem. The problem concerns the traversal of a given path with a robot in as short time as possible under input constraints. Several problem reformulations, increasing the chance of finding optimal solutions, are discussed. This paper also discusses the use of these optimal solutions for control of industrial robots. A control structure, in which the optimal trajectories are essential, is used on an ABB IRB140B to ensure robustness for model errors and disturbances.

Keywords: Modelica, Optimica, Optimization, Robot Control

1 Introduction

In several robot applications such as gluing, painting and arc welding, not only the end points but also the path as such and the speed of traversal are strongly connected to quality and efficiency. Optimal input trajectories, minimizing the traversal time along a path, can be found by solving an optimization problem. The results can be used by a controller to slow down the motion along the path rather than deviate from it, if the robot is subject to disturbances or model errors. A controller with ability to slow down the motion along the path while still trying to minimize the traversal time is a Path Velocity Controller, PVC [7]. The PVC depends on the existence of nominal acceleration and velocity profiles. The profiles are obtained by formulating and solving a minimum time optimization problem. Modelica along with Optimica [2] can be used to formulate optimization problems in a natural and compact way. Modelica is used to formulate the dynamical system and the initial values for the optimization problem, while the Optimica language extension is used to impose limits on the variables and to formulate the cost function and the constraints. Hence, Modelica and Optimica provide a convenient method for formulating and solving optimal control problems which is necessary when using PVC.

2 Background

2.1 JModelica.org

JModelica.org is a novel Modelica-based open source project targeted at dynamic optimization [4]. JModelica.org features compilers supporting code generation of Modelica models to C, a C API for evaluating model equations and their derivatives and optimization algorithms. The compilers and the model C API has also been interfaced with Python [9] in order to enable scripting and custom application development. In order to support formulation of dynamic optimization of Modelica models, JModelica.org supports the Optimica extension [3]. Optimica offers constructs for encoding of cost functions, constraints, the optimization interval with fixed or free end points as well as specification of the transcription scheme.

The JModelica.org platform contains an implementation of a simultaneous optimization method based on orthogonal collocation on finite elements [6]. Using this method, state and input profiles are parametrized by Lagrange polynomials, of order three and four respectively, based on Radau points. This method corresponds to a fully implicit Runge-Kutta method, and accordingly it possesses well known and strong stability properties. By parametrizing the variable profiles by polynomials, the dynamic optimization problem is translated into a non-linear program (NLP), solved by a numerical NLP solver. The NLP is, however, very large. In order to efficiently find a solution to the NLP, derivative information as well as the sparsity patterns of the constraint Jacobians need to be provided to the solver. The simultaneous optimization algorithm has been interfaced with the large-scale NLP solver IPOPT [13], which has been developed particularly

DOI: 10.3384/ecp09430089

to solve NLPs, arising in simultaneous dynamic optimization.

The choice of a simultaneous optimization algorithm fits well with the properties of the dynamic optimization problems treated in this paper. In particular, simultaneous methods handle unstable systems well, and also, state and input inequality constraints are easily incorporated.

Formulating the Minimum Time Optimization Problem

We consider the optimization problem of traversing a given path in as short time as possible under given input constraints. The minimum time optimization problem and reformulations thereof follow the presentation in [7]. Subsequently, we assume that a model of order p with states q_i and n inputs, τ_i, is available.

A model of a rigid robot can be written

$$\tau = H(q)\ddot{q} + v(q,\dot{q}) + d(q)\dot{q} + g(q), \qquad (1)$$

whereas a model of a flexible robot is given by

$$\begin{aligned} &H(q)\ddot{q} + v(q,\dot{q} + d(q)\dot{q} + g(q) + K(q-\theta) = 0 \\ &J\ddot{\theta} = \tau + K(q-\theta). \end{aligned} \qquad (2)$$

See [12] for details concerning robot modelling. Common for these models, and for the models used in work described in this paper, are that they all can be written on the form

$$\tau = h(q,\dot{q},\ldots,q^{(p)}) \qquad (3)$$

with limited inputs described by

$$\tau_i^{min} \le \tau_i \le \tau_i^{max}, \qquad 1 \le i \le n \qquad (4)$$

We further assume that a traversable path,

$$q(t) = f(t) \qquad (5)$$

is available, i.e., the path trajectories are defined in such a way that all states can reach all points on the prescribed path.

The objective is to traverse the path as fast as possible, i.e, minimizing the traversal time t_f. By setting the start time, $t_0 = 0$, the time-minimum optimization problem is formulated as

$$\min_{\tau} t_f = \min_{\tau} \int_0^{t_f} 1\mathrm{d}t \qquad (6)$$

under the constraints imposed by (4) and (5), along with boundary conditions for

$$\begin{aligned} &q(t_0), \dot{q}(t_0), \ldots, q^{(p)}(t_0) \\ &q(t_f), \dot{q}(t_f), \ldots, q^{(p)}(t_f) \end{aligned} \qquad (7)$$

The problem formulation consists of pn states and is generally hard to solve. Consequently a reduction of the number of states is desirable. The reductions are conducted as presented in [7] and are here given for completeness.

Reducing the Number of States

The number of dynamical states in the optimization problem is reduced to p as described in [7]. By introducing the *path parameter*

$$s(t_0) = s_0 \le s(t) \le s(t_f) = s_f \qquad (8)$$

and parametrizing the path f as a function of the nominal, scalar path parameter i.e., $f(s)$, the number of states can be reduced. Setting $q = f(s)$ and using the chain rule, $\frac{df}{dt} = \frac{df}{ds}\frac{ds}{dt}$, the model given by (3) is rewritten as

$$\tau = h_s(s,\dot{s},\ldots,s^{(p)}) \qquad (9)$$

In addition to (4), (9) serves as constraints for the reduced optimization problem. The dynamics of the optimization problem are now expressed as a chain of p integrators

$$\begin{aligned} &\frac{ds}{dt} = \dot{s} \\ &\frac{d\dot{s}}{dt} = \ddot{s} \\ &\vdots \\ &\frac{ds^{(p-1)}}{dt} = s^{(p)} \end{aligned} \qquad (10)$$

Following the state reduction the cost function is expressed as

$$\min_{s^{(p)}} \int_0^{t_f} 1\mathrm{d}t \qquad (11)$$

and the boundary conditions are imposed on s and its time derivatives

$$\begin{aligned} &s(t_0), \ldots, s^{(p-1)}(t_0) \\ &s(t_f), \ldots, s^{(p-1)}(t_f) \end{aligned} \qquad (12)$$

Reformulating the Optimization Problem

The number of states in the optimization problem can be further reduced to $p-1$ as the problem is reformulated over a fixed interval. The problem is converted to optimization over a fixed interval by deriving a dynamic system in s [7]. New states $x_1, \ldots, x_{p-1}$ are introduced. They are defined as

$$\begin{aligned} x_1 &= \frac{\dot{s}^p}{p} \\ x_i &= \frac{dx_{i-1}}{ds}, \qquad i = 2, \ldots, p-1 \end{aligned} \tag{13}$$

The free variable, u, in the optimization problem is defined as $u = s^{(p)}$. By defining a function, g, as

$$g(x_1) = (px_1)^{1/p} \tag{14}$$

the p-th order derivative of s is expressed as [7]

$$s^{(p)} = g'(x_1)g(x_1)^{(p-1)}\frac{dx_{p-1}}{ds} + F_p(x_1, \ldots, x_{p-1}) \tag{15}$$

where

$$\begin{aligned} F_p(x_1, \ldots, x_{p-1}) =& g''(x_1)x_2 g(x_1)g(x_1)^{p-2}x_{p-1} \\ &+ g'(x_1)(p-2)g(x_1)^{p-3}g'(x_1)x_2 g(x_1)x_{p-1} \\ &+ \sum_{i=1}^{p-2} \frac{\partial F_{p-1}(x_1, \ldots, x_{p-2})}{\partial x_i} x_{i+1} g(x_1) \end{aligned} \tag{16}$$

The constraints (9) are written as functions of s, x and u as

$$\tau = h_x(s, x_1, \ldots, x_{p-1}, u) \tag{17}$$

still subject to (4). We now have boundary conditions for

$$\begin{aligned} & x_1(s_0), \ldots, x_{p-1}(s_0) \\ & x_1(s_f), \ldots, x_{p-1}(s_f) \end{aligned} \tag{18}$$

The cost function for the minimum time optimization problem is reformulated

$$\int_0^{t_f} \mathrm{d}t = \int_{s_0}^{s_f} \frac{1}{\dot{s}} \mathrm{d}s = \int_{s_0}^{s_f} (px_1)^{-1/p} \mathrm{d}s \tag{19}$$

Equations (17), (18) and (19) along with the dynamic system

$$\begin{aligned} \frac{dx_1}{ds} &= x_2 \\ \frac{dx_2}{ds} &= x_3 \\ &\vdots \\ \frac{dx_{p-2}}{ds} &= x_{p-1} \\ \frac{dx_{p-1}}{ds} &= u - F_p(x_1, \ldots, x_{p-1}) \end{aligned} \tag{20}$$

state the full optimization problem, reformulated to a fixed interval.

Figure 1: An ABB IRB140B. Picture from [1].

3 Modeling

The robot considered in this paper is an ABB IRB-140B, see Figure 1. The IRB140B is a six joint serial robot which allows an arbitrary positioning and orientation within the robot's work envelope. Due to the mechanical structure of robots, depicted in Figure 2, robot models in general are quite complex and nonlinear, cf., Model (1) and (2). However, the robot model used here consists of six independent linear models. Each of the robot's six joints are modeled by a linear second order model. This is possible since the input-output relation for which the model has been identified contains a linearizing controller. The identified models, and the models used for optimization, describe the relation between a joint's velocity reference, τ, and its angular position, q. The model structure is given by (21).

$$T_i\ddot{q}_i + \dot{q}_i = K_i\tau_i \tag{21}$$

The parameter values for each joint are displayed in Table 1.

4 Path

A path has been recorded using so called *lead-through*, a force-control mode which allows the operator to freely move/lead the robot in the workspace, with the IRB140B [1]. The joint angles have been parametrized

Joint	K_i	$T_i \times 10^{-2}$
1	1.031	1.907
2	1.077	2.043
3	1.061	1.913
4	1.051	1.716
5	1.062	1.791
6	1.062	1.745

Table 1: Parameters for the controlled joint model of Eq. (21).

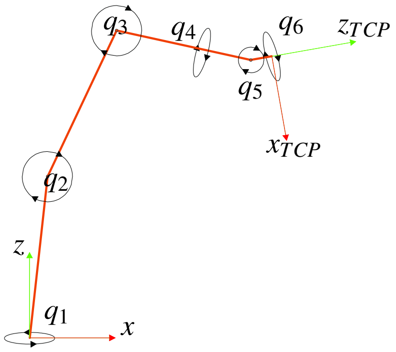


Figure 2: Mechanical structure of the ABB IRB140B showing the joint angles q_i. Picture from [8].

by the path parameter, s, defined between $s_0 = 0$ and $s_f = 1$. The path is described by splines implemented in Modelica as if-clauses and the derivatives of $f(s)$ have been calculated by derivation of the splines.

Figure 3 shows the TCP position. TCP is an abbreviation for Tool Center Point which is a user-defined point on the robot's end effector.

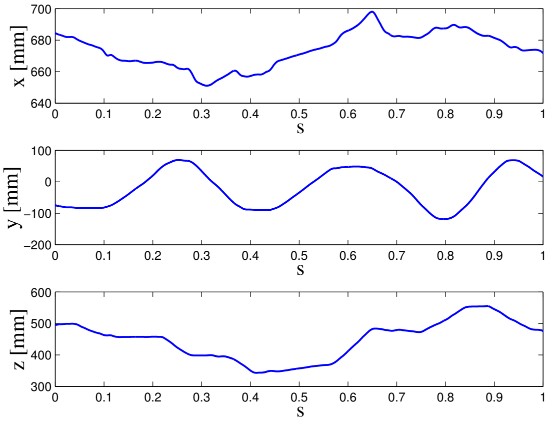


Figure 3: TCP as function of the path parameter.

5 Optimization

The goal of the optimization is to find the minimum time acceleration profile, $v_2(s)$, and the velocity profile, $v_1(s)$, to be used in the PVC, see Section 6. The acceleration profile is defined as the acceleration along the path expressed as a function of the nominal path parameter i.e., $v_2(s) = \ddot{s}(s)$. The velocity profile is defined analogous by i.e., $v_1(s) = \dot{s}(s)$. This can be done if the time derivative of s, $\dot{s}$, is assumed greater or equal to zero [7]. Formulation of the optimization problem is done using Modelica and Optimica. Modelica is used to code the constraints (17) and the dynamics (20), while Optimica is used to define the input limits, (4) the boundary conditions, (18), and the cost function, (19). The Modelica and Optimica codes are presented in Appendix A, Listings 1 and 2. Note that the built-in Modelica variable `time` is equivalent to the path parameter s due to the reformulation. Hence, the `der()`-operator is equivalent to derivation with respect to the path parameter s.

Using the reduction techniques from Section 2 the reformulated optimization problem considered is now given by

$$\begin{aligned}
&\min_{u(s)} \int_{s_0}^{s_f} \frac{1}{\sqrt{2x_1}} \mathrm{d}s \\
&\text{s.t.} \\
&\quad x_1 = \frac{\dot{s}^2}{2}, \quad \frac{dx_1}{ds} = u \\
&\quad K\tau = T(f''(s)\dot{s}^2 + f'(s)u) \\
&\quad \tau^{min} \leq \tau \leq \tau^{max} \\
&\quad x_1(s_0) = 0, \quad x_1(s_f) = 0 \\
&\quad \dot{s} \geq 0
\end{aligned} \tag{22}$$

This work was done using a predecessor of the JModelica.org platform, see [11] and [4]. The predecessor version uses AMPL as intermediate representation format and supports an early version of Optimica. The Modelica and Optimica code is compiled by the Optimica compiler which translates the optimization problem into AMPL [5]. The external solver, IPOPT [10], is then called to solve the problem.

In order for IPOPT to find an optimal solution the occurrence of a good initial guess is crucial. Because of the free end time, finding an optimal solution for a general minimum time problem requires an initial guess close to the optimal solution. The reformulation of the optimization problem to a fixed interval is therefore preferable.

Finding an optimal solution for the optimization problem (22) turns out to be feasible. But in order to find a solution with a smooth acceleration profile the optimization problem has to be solved in two steps. First, the optimization problem as it is stated in (22) is solved. This result can now be used as an initial guess when solving a second optimization problem. In the second optimization problem a two per cent slack on the final time is introduced and the cost function penalizes the square integral of the input signals. This renders smoother acceleration and velocity trajectories which are suitable for implementation on the robot actuators while the robot still traverses the path in close to minimum time.

6 Control System

The PVC algorithm [7] modifies the acceleration of a new path parameter, σ, in such a way that the path, $f(\sigma)$, is not deviated from while ensuring that the input limitations (4) are not violated. The algorithm uses the nominal acceleration profile, $v_2(\sigma)$, as a reference to update the path acceleration, $\ddot{\sigma}$. The path acceleration is limited to make sure that the input constraints (4) are not violated. Moreover, the algorithm includes internal feedback $\frac{\alpha}{2}(v_1(\sigma)^2 - \dot{\sigma}^2)$ that makes the path velocity, $\dot{\sigma}$, approach the nominal velocity, $v_1(\sigma)$.

A controller written on the form

$$\tau = \beta_1(\sigma)\ddot{\sigma} + \beta_2(\sigma, \dot{\sigma}, q, \dot{q}) \tag{23}$$

is assumed to be available. Combining (23) with the limits (4), it is possible to calculate the minimum and the maximum acceleration, $\ddot{\sigma}^i_{min}$ and $\ddot{\sigma}^i_{max}$, for each joint i.

$$\tau_i^{min} \leq \tau_i = \beta_{1_i}\ddot{\sigma} + \beta_{2_i} \leq \tau_i^{max}, \qquad 1 \leq i \leq 6 \tag{24}$$

$$\begin{aligned} \ddot{\sigma}_i^{max}(\beta_{1_i}\beta_{2_i}) &= \begin{cases} \frac{\tau_i^{max} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} > 0 \\ \frac{\tau_i^{min} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} < 0 \\ \infty, & \beta_{1_i} = 0 \end{cases} \\ \ddot{\sigma}_i^{min}(\beta_{1_i}\beta_{2_i}) &= \begin{cases} \frac{\tau_i^{min} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} > 0 \\ \frac{\tau_i^{max} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} < 0 \\ -\infty, & \beta_{1_i} = 0 \end{cases} \end{aligned} \tag{25}$$

By choosing the limits on $\ddot{\sigma}$ according to (26), the acceleration along the path is chosen in order not to violate the limits on the input

$$\begin{aligned} \ddot{\sigma}_{max}(\beta_1, \beta_2) &= \min_i \ddot{\sigma}_i^{max}(\beta_1, \beta_2) \\ \ddot{\sigma}_{min}(\beta_1, \beta_2) &= \max_i \ddot{\sigma}_i^{max}(\beta_1, \beta_2) \end{aligned} \tag{26}$$

The PVC algorithm is given by (27). The full control structure is presented in the block diagram in Figure 4.

$$\begin{aligned} \frac{d\sigma}{dt} &= \dot{\sigma} \\ \frac{d\dot{\sigma}}{dt} &= \ddot{\sigma} \\ u_r &= v_2(\sigma) + \frac{\alpha}{2}(v_1(\sigma)^2 - \dot{\sigma}^2) \\ \ddot{\sigma} &= sat(u_r, \ddot{\sigma}_{min}(\beta_1, \beta_2), \sigma_{max}(\beta_1, \beta_2)) \end{aligned} \tag{27}$$

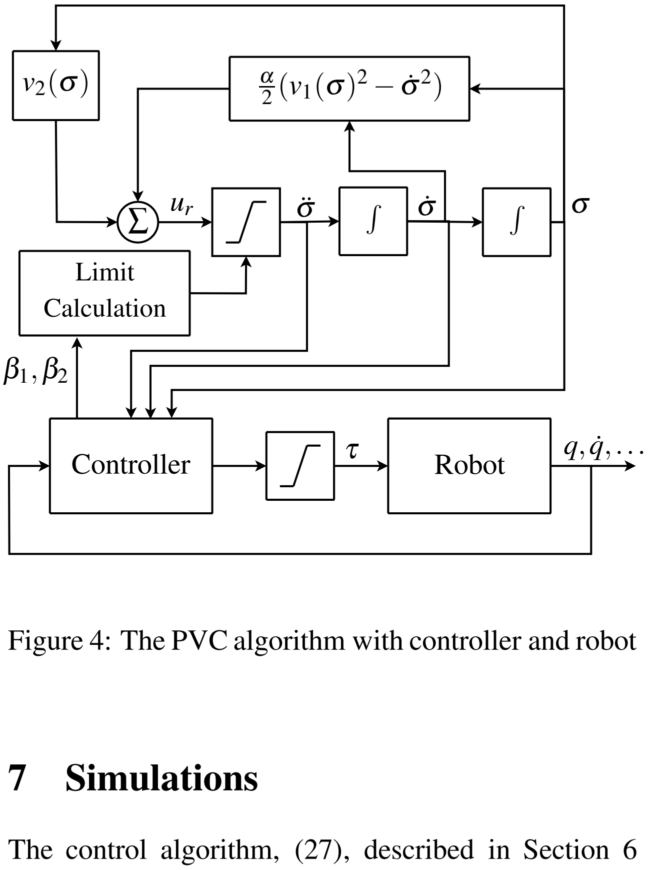


Figure 4: The PVC algorithm with controller and robot

7 Simulations

The control algorithm, (27), described in Section 6 has been implemented in Simulink. In order to evaluate the performance of the PVC a model error was introduced. The model error introduced was a 20% decrease in the gain for joint 1, i.e., $\tilde{K}_1 = 0.8K_1$. Two simulations were done using the perturbed model. In the first simulation, the regular controller 23 was used but the PVC was disabled, see Figure 5, whereas in the second simulation the PVC was used together with the controller 23. The internal feedback gain in the second simulation was chosen to $\alpha = 500$.

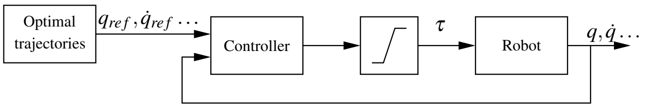


Figure 5: Block diagram showing the setup used in the first simulation, without the PVC.

The minimum time for traversing the path with the unperturbed model is 10.87 seconds and with the perturbation the minimum time is 12.09 seconds.

In the first simulation the path was traversed with the velocity profile obtained from the optimization. Since there was no PVC the deviated path was traversed in 10.87 seconds. In the second simulation, with the PVC, the path was instead traversed in 12.41 seconds which is 0.32 seconds longer than the optimal time for the perturbed model. The velocity along the path, $\dot{\sigma}(\sigma)$, for both simulations is displayed in Figure 6. Here one can clearly see that the PVC lowers the velocity along the path. When using the PVC, the path is traversed without violating the input boundaries, τ^{min} and τ^{max}, as can be seen in Figure 8. When not using the PVC, see Figure 9, it is obvious that the input signal calculated by the controller is well above the input limitations of the process. Since the input signals are saturated at their limits this gives rise to the deviation from the path visible in Figure 7. Figure 7 shows that perturbation mostly effects the error in y direction which is due the robot's mechanical structure and and the traversed path.

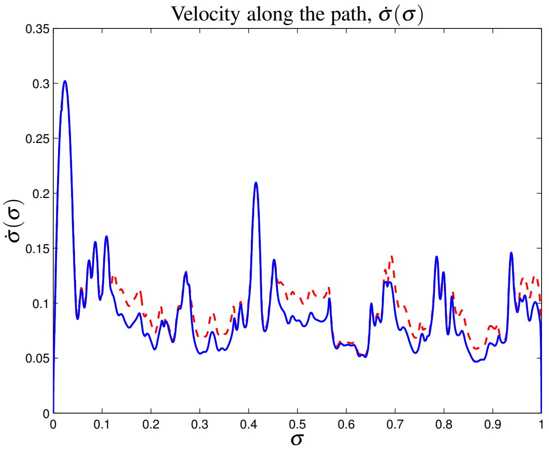


Figure 6: Continuous line is with the PVC enabled and dashed line is with the PVC disabled

8 Summary and Conclusions

In this paper we formulated and solved a minimum time optimization problem for an industrial robot. Reformulations were done in order to obtain acceleration and velocity profiles without having to find an initial guess close to the optimal solution. The robot joints were modeled with simple second order linear transfer functions. This was possible due to the presence of linearizing controllers working within the identified models. Lead-through was used to record the path which was represented by the joints angular positions. The dynamical model with its constraints and boundary conditions, the path and the cost function were done in Modelica and Optimica. Modelica along with Optimica provided an efficient and convenient way to formulate the dynamic optimization problem. The Optimica formulation is both in structure and syntax close to the mathematical description of the optimization problem which is beneficial. The closeness to the mathematical description facilitates the formulation of optimization problems which makes the work less time consuming as well as less error-prone than coding in for instance AMPL.

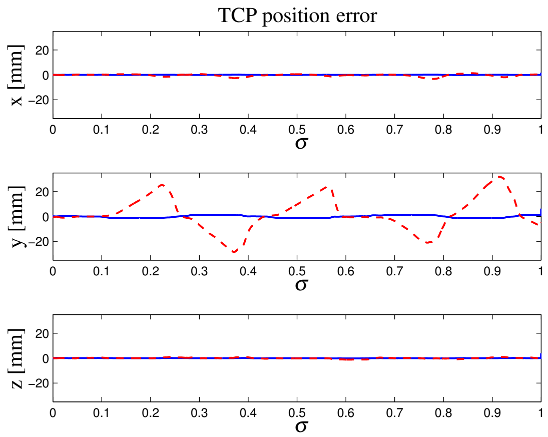


Figure 7: Continuous line is with the PVC enabled and dashed line is with the PVC disabled.

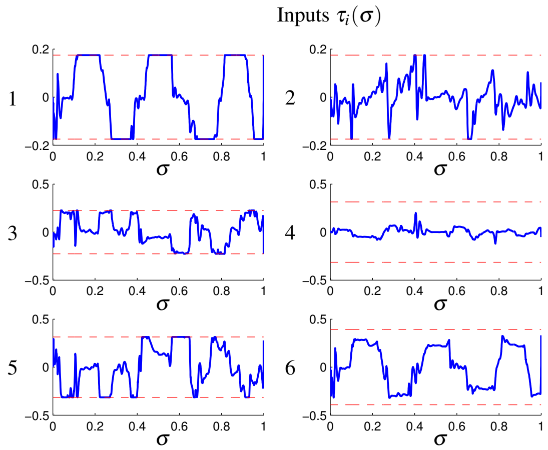


Figure 8: Input signals for the six joints with the PVC enabled

The optimization results were used as nominal acceleration and velocity trajectories in a PVC. The PVC has been tested, both in simulations and on an ABB IRB140B robot, showing that the path deviation is small, the input limitations are not violated and that the path traversal speed is close to the optimal. This paper shows that Optimica is well suited for the task

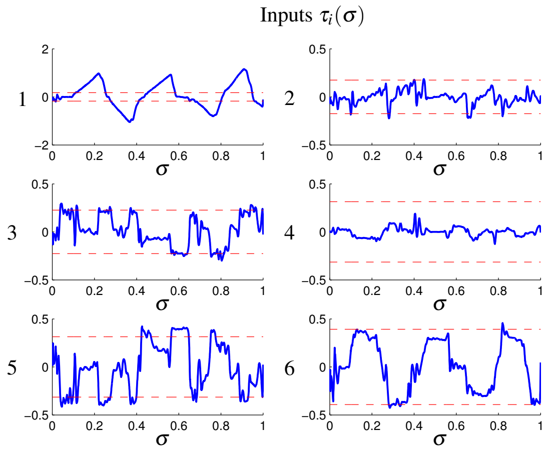


Figure 9: Input signals for the six joints with the PVC disabled

of finding the optimal acceleration and velocity profiles needed in order to use PVC.

References

[1] ABB. ABB Home Page, 2009. `http://www.abb.com/`.

[2] Johan Åkesson. *Tools and Languages for Optimization of Large-Scale Systems*. PhD thesis, Department of Automatic Control, Lund University, Sweden, November 2007. ISRN LUTFD2/TFRT--1081--SE.

[3] Johan Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *In 6th International Modelica Conference 2008*. Modelica Association, March 2008.

[4] Johan Åkesson, Tove Bergdahl, Magnus Gäfvert, and Hubertus Tummescheit. Modeling and Optimization with Modelica and Optimica Using the JModelica.org Open Source Platform. In *Proceedings of the 7th International Modelica Conference 2009*. Modelica Association, September 2009.

[5] AMPL - A Modeling Language for Mathematical Programming. AMPL Home Page, 2009. `http://www.ampl.com/`.

[6] L.T. Biegler, A.M. Cervantes, and A Wächter. Advances in simultaneous strategies for dynamic optimization. *Chemical Engineering Science*, 57:575–593, 2002.

[7] Ola Dahl. *Path Constrained Robot Control*. PhD thesis, Department of Automatic Control, Lund University, Sweden, April 1992. ISRN LUTFD2/TFRT--1038--SE.

[8] Fredrik Eriksson and Marcus Welander. Haptic interface for a contact force controlled robot. Master's Thesis ISRN LUTFD2/TFRT--5837--SE, Department of Automatic Control, Lund University, Sweden, May 2009.

[9] Python Software Foundation. Python Programming Language – Official Website, 2009. `http://www.python.org/`.

[10] IPOPT - Interior Point OPTimizer. IPOPT Home Page, 2009. `https://projects.coin-or.org/Ipopt`.

[11] Modelon AB. JModelica Home Page, 2009. `http://www.jmodelica.org`.

[12] Mark W. Spong, Seth Hutchinson, and M.Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, Inc, 2006.

[13] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line- search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–58, 2006.

A Modelica and Optimica Code

```
model Robot
  // Robot inputs, tau
  Real tau[6];
  // Robot model gains, K
  parameter Real K[6] = {1.031, 1.077, 1.061,
    1.051, 1.054, 1.062};
  // Robot model time constants, T
  parameter Real T[6] = {0.019086, 0.020433,
    0.019129, 0.017158, 0.017909, 0.017447};
  // First and second derivative of s
  // with respect to time
  Real sd(start=0);
  Real sdd;
  // Auxiliary variable
  Real x1;
  // Optimization variable
  Modelica.Blocks.Interfaces.RealInput u
  // Path, f, stored as splines
  Splines f;
  // First and second derivative of the path f
  // with respect to s
  Real df[6];
  Real ddf[6];
equation
  K*tau = T*(ddf*sd^2 + df*u) + df*sd;
  df = der(f.f);
  ddf = der(df);
  x1 = sd^2/2;
  der(x1) = u;
  // Independant variable
  f.s = time;
end Robot;
```

Listing 1: The Modelica code for the optimization problem in (22).

```
optimization OptTraj (objective=cost,
                      startTime=0,
                      finalTime=1)
  // Cost function
  Real cost;
  // Instance of robot model
  Robot robot(u(free=true));
equation
  der(cost) = 1/sqrt(2*x1+1e-10);
constraint
  // Lower bounds on robot inputs
  robot.tau >= {-0.175, -0.175, -0.227,
               -0.314, -0.314, -0.395};
  // Upper bounds on robot inputs
  robot.tau <= {0.175, 0.175, 0.227,
               0.314, 0.314, 0.395};
  // Terminal constraint on x1
  robot.x1(finalTime) = 0;
  // Inequality constraint
  robot.sd >= 0;
end optTraj;
```

Listing 2: The Optimica code for the optimization problem in (22).

Using Modelica models in real time dynamic optimization – gradient computation

Lars Imsland* Pål Kittilsen Tor Steinar Schei
Cybernetica AS
7038 Trondheim, Norway
{lars.imsland,pal.kittilsen,tor.s.schei}@cybernetica.no

Abstract

This paper reports on implementation of gradient computation for real-time dynamic optimization, where the dynamic models can be Modelica models. Analytical methods for gradient computation based on sensitivity integration is compared to finite difference-based methods. A case study reveals that analytical methods outperforms finite difference-methods as the number of inputs and/or input blocks increases.

Keywords: Nonlinear Model Predictive Control, Sequential Quadratic Programming, Gradient computation, Offshore Oil and Gas Production.

1 Introduction

Nonlinear model predictive control (NMPC) is an advanced control technology that enables the use of mechanistic multi-disciplinary process models in achieving process control objectives (economical, safety, environmental). NMPC algorithms formulate an 'open-loop' constrained dynamic optimization problem, which is re-solved and re-implemented at regular intervals to combine the advantage of the optimal control solution with the feedback achieved through updated information (measurements and estimated states and parameters). The number of applications of NMPC is increasing, especially within certain process industries [11, 4], but there is certainly potential for further growth.

Models developed primarily for dynamic process simulation and design are often not appropriate for NMPC, for example for reasons related to numerical robustness and computational speed. Nevertheless, issues such as modularity, reuse, model libraries, documentation, etc., make it advantageous to use advanced modeling languages such as Modelica also for development of prediction models for NMPC. This is discussed in [7].

NMPC optimization algorithms are often *Sequentially Quadratic Programming* (SQP) algorithms. That is, the NMPC nonlinear dynamic optimization problem is discretized and solved by applying quadratic programming sequentially to quadratic/linear approximations of the optimization problem, and upon numerical implementation the solution generally converges to a local optimum. SQP algorithms for NMPC are often classified based on how discretization is done, but nevertheless have in common that they need (at least) gradient information of the discretized dynamic optimization problem.

This paper is concerned with gradient computation for a class of NMPC optimization algorithms often referred to as sequential (or single-shooting) NMPC optimization, using Modelica models developed in Dymola as prediction models. Of particular concern is the exploitation of symbolic/analytical Jacobians of the Modelica model. Furthermore, we discuss briefly the implementation of gradient computation in commercial NMPC software such as CYBERNETICA CENIT [4]. Finally, we use a semi-realistic NMPC problem from offshore oil and gas production as a case to illustrate issues such as accuracy and efficiency in gradient computation.

[*] Also affiliated with the Norwegian University of Science and Technology, Department of Engineering Cybernetics.

DOI: 10.3384/ecp09430067

2 Nonlinear Model Predictive Control

2.1 Models

A model of the physical plant we want to control by NMPC is implemented in Modelica. The underlying mathematical representation could be either as (hybrid) ODE or DAEs, but here we assume, mainly for simplicity, that it is formulated as a (piecewise) continuous ODE:

$$\dot{x} = f(x,u,p), \;\; y = h(x,u,p), \;\; z = g(x,u,p), \quad (1)$$

where x are states, u are manipulated inputs, p are parameters (which might be candidates for online estimation), y are measured outputs and z are controlled (not necessarily measured) outputs.

Model-based control typically apply some estimation scheme, for instance a sigma-point extended Kalman-filter approach or moving horizon estimation [12]. The task of the estimation is to

- consolidate measurements to obtain a best estimate of the process state,
- estimate unknown and changing parameters (adaptation), and
- achieve zero steady state error in the desired controlled variables (integral control).

We will not be concerned about estimation in this paper, and assume therefore (unrealistically) that we know all parameters and measure the state.

Note that typically, we will in addition to manipulated inputs also have other (measured) inputs that in essence make the system time-variant. However, we will employ a discrete-time NMPC formulation and are therefore only interested in integration over one sample interval where these inputs typically are assumed constant.

For the same reason, we are only interested in the solution of (1) in the sense that it is used to calculate states and outputs at the next sampling instant. That is, we are interested in the discrete-time system

$$x_{k+1} = x_k + \int_{t_k}^{t_{k+1}} f(x(\tau), u_k, p_k)\mathrm{d}\tau, \qquad (2a)$$

$$y_k = h(x_k, u_{k-1}, p_{k-1}), \qquad (2b)$$

$$z_k = g(x_k, u_{k-1}, p_{k-1}). \qquad (2c)$$

The integration involved is in general solved by ODE solver routines.

For NMPC, we will use the above system for prediction. In prediction for NMPC, we are not concerned with the measured outputs, and disregard these for now. With abuse of notation, we will write the time-varying discrete-time NMPC predictor system as

$$x_{k+1} = f_k(x_k, u_k), \quad z_k = g_k(x_k, u_{k-1}). \quad (3)$$

2.2 NMPC optimization problem

We formulate here a simplified discrete-time NMPC optimization problem using the model (3). We assume the desired operating point $(x,u) = (0,0)$ is an equilibrium ($f_k(0,0) = 0$, $g_k(0,0) = 0$), and we minimize at each sample (using present measured/estimated state x_0 as initial state for predictions) the objective function

$$J(x_0, u_0, u_1, \ldots, u_{N-1}) = \frac{1}{2}\sum_{i=0}^{N-1} z_{i+1}^{\mathsf{T}} Q z_{i+1} + u_i^{\mathsf{T}} R u_i$$

over future manipulated inputs u_i, where z_i are computed (predicted) from (3), and Q and R are weighting matrices. Importantly, the future behavior is optimized subject to constraints:

$$z_{\min} \leq z_k = g_k(x_k, u_{k-1}) \leq z_{\max}, \quad k = 1, \ldots, N$$
$$u_{\min} \leq u_k \leq u_{\max}, \quad k = 0, \ldots N-1.$$

The first input u_0 is then implemented to the plant.

It is important to note that the problem formulation used here is simplistic, For the sake of brevity and with little loss of generality, it does not contain features usually contained in NMPC software packages, such as:

- Features related to non-regulation problems (for instance control of batch processes).
- Input blocking (for efficiency).
- Incidence points (for efficiency and feasibility).
- Control horizon longer than input horizon.
- End-point terminal weight/region (in regulation, for efficiency/stability).
- Input moves instead of inputs as optimization variables.

Further details about such issues can be found in MPC textbooks, for instance [9].

2.3 Sequential NMPC optimization

In most cases, the (discretized) dynamic optimization problem is solved using numerical algorithms based on sequential quadratic programming (SQP). A SQP method is an iterative method which at each iteration makes a quadratic approximation to the objective function and a linear approximation to the constraints, and solves a QP to find the search direction. Then a line-search is performed along this search direction to find the next iterate, and the process is repeated until convergence (or time has run out). General purpose SQP solvers may be applied to NMPC optimization, but it is in general advantageous to use tailor-made SQP algorithms for NMPC applications.

The main approaches found in the literature are usually categorized by how the dynamic optimization problem is discretized/parametrized. The most common method is perhaps the *sequential* approach [3], which at each iteration simulates the model using the current value of the optimization variables $(u_0, u_1, \dots, u_{N-1})$ to obtain the gradient of the objective function (and possibly the Hessian), thus effectively removing the model equality constraints and the states $x_1, x_2, \dots, x_N$ as optimization variables. Other methods are the *simultaneous* approach [1], and the *multiple shooting* approach [2]. In this paper, a sequential approach is taken.

3 Gradient computation in sequential NMPC optimization

3.1 The sensitivity (step/impulse response) matrix

As explained above, sequential SQP approaches to NMPC optimization sequentially simulates and optimizes. The simulation part should calculate the objective function- and constraint gradients with respect to the optimization variables u_i. This is typically done via the *step response matrix*, or as in the simplified exposition here, the impulse response matrix. To avoid confusion, we will mostly refer to this in the following as the (NMPC) sensitivity matrix.

Rewrite the objective function by stacking future inputs and outputs as

$$J(x_0, \mathbf{u}) = \frac{1}{2}\left(\mathbf{z}^\mathsf{T}\mathbf{Q}\mathbf{z} + \mathbf{u}^\mathsf{T}\mathbf{R}\mathbf{u}\right),$$

where $\mathbf{u} = (u_0, u_1, \dots, u_{N-1})$, $\mathbf{z} = (z_1, \dots, z_N)$, $\mathbf{Q} = \text{blkdiag}\{Q\}$ and $\mathbf{R} = \text{blkdiag}\{R\}$. The gradient of the objective function is

$$\frac{\partial J}{\partial \mathbf{u}} = \mathbf{z}^\mathsf{T}\mathbf{Q}\frac{\partial \mathbf{z}}{\partial \mathbf{u}} + \mathbf{u}^\mathsf{T}\mathbf{R}.$$

Similarly, the linearization of the output constraints are also given by the matrix $\Phi = \frac{\partial \mathbf{z}}{\partial \mathbf{u}}$, which we call the sensitivity matrix (which in this case is the truncated impulse response matrix).

That is, once we have calculated the sensitivity matrix Φ, we can easily evaluate the objective function- and constraints gradients in sequential NMPC optimization. From this, one can argue that gradient computation in sequential NMPC is mostly about efficient computation of the sensitivity matrix.

The rest of this section treats calculation of the sensitivity matrix by finite differences, and by forward ODE sensitivity analysis. We remark that one could also use adjoint sensitivity methods for calculating the desired NMPC gradients [8]. However, for NMPC problems with a significant number of constraints, this is likely to be less efficient than forward methods.

3.2 NMPC sensitivity matrix by finite differences

Finding the sensitivity matrix by finite differences is achieved by in turn perturbing each element of all input vectors u_i and simulate to find the response in the z_js. The perturbation is typically either one-sided (forward finite differences) or two-sided (central finite differences), the latter taking about twice the time but being somewhat more accurate [10].

3.3 NMPC sensitivity matrix by sensitivity integration

Assuming we have a time-varying linearization of (3) along the trajectories:

$$x_{k+1} = A_k x_k + B_k u_k, \tag{4a}$$

$$z_k = C_k x_k + D_k u_{k-1}, \tag{4b}$$

we can calculate the sensitivity matrix which in this simple case is as shown in eq. (5) on the bottom of the following page. (The sensitivity matrix shown there is the impulse response matrix, the step response matrix is the cumulative sum of the columns of the impulse response matrix, from right to left.) To find the linearization (4), it is usually most practical to calculate sensitivities of the solution of (1) with respect to initial values $x(0) = x_k$ and inputs u_k (assumed constant over each sample interval). Stacking these sensitivities in matrices S and W, they are given by the following matrix ODEs [5]:

$$S := \frac{\partial x}{\partial u_k}: \quad \dot{S} = \frac{\partial f}{\partial x} S + \frac{\partial f}{\partial u}, \qquad S(0) = 0, \quad \text{(6a)}$$

$$W := \frac{\partial x}{\partial x_k}: \quad \dot{W} = \frac{\partial f}{\partial x} W, \qquad W(0) = I. \quad \text{(6b)}$$

We will only be interested in the sensitivities at the end of each sample interval, which are the system matrices in (4a):

$$B_k := \frac{\partial x_{k+1}}{\partial u_k} = S_{k+1}, \; A_k := \frac{\partial x_{k+1}}{\partial x_k} = W_{k+1}. \quad (7)$$

Defining also $C_k := \frac{\partial z_k}{\partial x_k}$, $D_k := \frac{\partial z_k}{\partial u_{k-1}}$, we get the linearized (LTV) system above. The required Jacobian matrices

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial u}, \quad \frac{\partial g}{\partial x}, \quad \frac{\partial g}{\partial u}$$

can be found from finite differences, symbolically (for instance from a Dymola model), or by automatic differentiation methods. The two latter should be preferred.

The sensitivity ODEs (6) are solved together with (2a) by ODE solvers. Although the size of S and W might be large, the fact that the systems (6) have a block-diagonal Jacobian with the individual blocks being the Jacobian of (2a) can and should be exploited in ODE solvers, leading to efficient computation of the ODE sensitivities [6]. (For systems with a very large number of states and relatively few inputs/outputs, it might be more efficient to directly integrate the elements in the sensitivity matrix, and thus avoiding calculation of state sensitivities.)

It is important to note that the above is described for zero order hold and no input blocking. For efficient implementation, it is essential to exploit input blocking in the sensitivity computations.

4 Implementation

A software package for model-based estimation and control (NMPC) will typically include an *offline* part for model fitting (parameter optimization) to data, data-based testing of estimation and simulation-based testing of NMPC (including estimation), and an *online* part for a complete NMPC real-time solution (including estimation). The workflow in taking a parametrized model (implemented in Modelica/Dymola, or 'by hand' in lower level languages such as C) to online application is attempted illustrated in Figure 1. A Modelica tool (such as Dymola) will need a method for exporting the models so they can be used efficiently in the offline tool and the online system. Dymola has the option of C-code export, which is platform independent and gives models that are efficiently evaluated and easily integrated with ODE/DAE-solvers and optimizers, typically implemented in C.

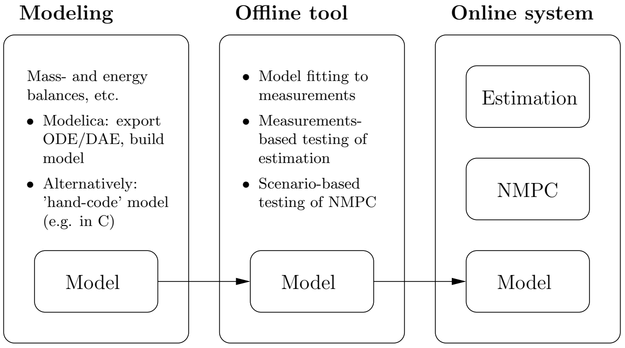

Modeling
Offline tool
Online system
Mass- and energy balances, etc.
• Modelica: export ODE/DAE, build model
• Alternatively: 'hand-code' model (e.g. in C)
• Model fitting to measurements
• Measurements-based testing of estimation
• Scenario-based testing of NMPC
Estimation
NMPC
Model
Model
Model

Figure 1: Overview over workflow in model usage. Data storage, data acquisition/exchange, and GUI not shown.

NMPC software that should use analytical methods for gradient computation, need the discrete-time model to provide A_k, B_k, C_k and D_k matrices (4) (typically found via sensitivity integration using the exported model, as discussed earlier). Figure 2 indicates the data flow in a discretized model component that is based on a

$$\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix} = \begin{pmatrix} C_1 B_0 + D_1 & 0 & 0 & 0 & \cdots \\ C_2 A_1 B_0 & C_2 B_1 + D_2 & 0 & 0 & \cdots \\ C_3 A_2 A_1 B_0 & C_3 A_2 B_1 & C_3 B_2 + D_3 & 0 & \cdots \\ C_4 A_3 A_2 A_1 B_0 & C_4 A_3 A_2 B_1 & C_4 A_3 B_2 & C_4 B_3 + D_4 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix} \qquad (5)$$

continuous-time ODE Modelica/Dymola model, using an ODE solver which implements sensitivity integration.

Several specialized solvers for calculation of ODE sensitivities exist. For instance, CVODES [6] implements variable-order, variable-step multistep ODE solvers for stiff and non-stiff systems, with sensitivity analysis capabilities. For efficiency of sensitivity integration, it is a significant advantage if we have symbolic ODE Jacobians available, for instance from a Dymola model.

As a side-remark, for models that do not provide symbolic Jacobians, we have the option of using automatic differentiation packages (CppAD, ADOLC, or others). However, this typically requires C++ compilation.

5 Case

5.1 Case description

In the North Sea (and on other continental shelves), petroleum is produced by drilling wells into the ocean bed. From the wells, typically a stream of oil, gas and water arrives at a surface production facility (platform or ship) which main task is to separate the products. Oil and gas are exported, either through pipelines or by ship. Water is cleaned and deposited to sea or pumped back to the reservoir.

A schematic picture (in the form of a Dymola screendump) of such an offshore oil and gas processing plant is given in Figure 3. Oil, gas and water enter the plant from several sources. In reality the sources are reservoirs connected to the production facility through wells and pipelines. The separators are large tanks which split the phases oil, water and gas. The produced oil is leaving in the lower right corner of the figure, while the gas enters a compression train (not included in the model) from the first and second separator (two leftmost tanks). Water is taken off from each separator and sent to a water treatment process.

Generally, this type of process is a fairly complex system in terms of numbers of components. However, many of the components are of the same type (mainly separators, compressors, valves, controllers, in addition to minor components such as sources, sinks, splitters, sensors, etc.), which simplifies overall modeling and make it efficient to reuse model components. Furthermore, construction of this process model benefited significantly from using models and concepts introduced by the new Modelica_Fluid library. Models for valves, sources, sinks, and sensors were used directly, whereas other models and functions in the new library inspired the development of our own models. For real-time efficiency reasons, we take care to ensure that we end up with an ODE-type model. The new *stream* class is an improvement to ease the construction of models satisfying this criteria.

In addition to the unit modes, medium models are necessary in order to calculate physical properties like density and heat capacity, in addition to phase transitions between oil and gas. The model should have real time capabilities, favoring simple/explicit relations. For phase equilibrium calculations, correlations of k-values (as function of temperature, pressure and molecular weight) were used together with a simplified representation of the many chemical species found in the real process. Gas density was described by a second-order virial equation, where the model coefficients were fitted to an SRK-equation for the relevant gas composition evaluated for the temperature and pressure range of current interest.

The model we use as NMPC prediction model (cf. Figure 3) has 27 states. We consider two NMPC problem formulations, one being 2×2 (2 MVs and 2 CVs), the other 4×4.

5.2 Issues in preparing a Dymola model for use with a NMPC system

Before a Dymola simulation model can be used in a NMPC system, it must be prepared. It is worthwhile to mention some of the issues involved:

- In addition to making sure that the model does not contain nonlinear systems of equations that must be solved to evaluate it (i.e, the model is an ODE as explained above), for Dymola to export symbolic Jacobians we must of course ensure that the model is differentiable. Especially if Modelica functions are used extensively, this might in some cases involve some effort.
- Dymola provides ODE-style Jacobians (A, B, C and D matrices). Unfortunately, it seems not possible to specify a subset of inputs that we want to evaluate Jacobians with respect to. This is especially critical for the B-matrix. For instance, we have in the case study a total of 9 possible MVs/DVs, all of

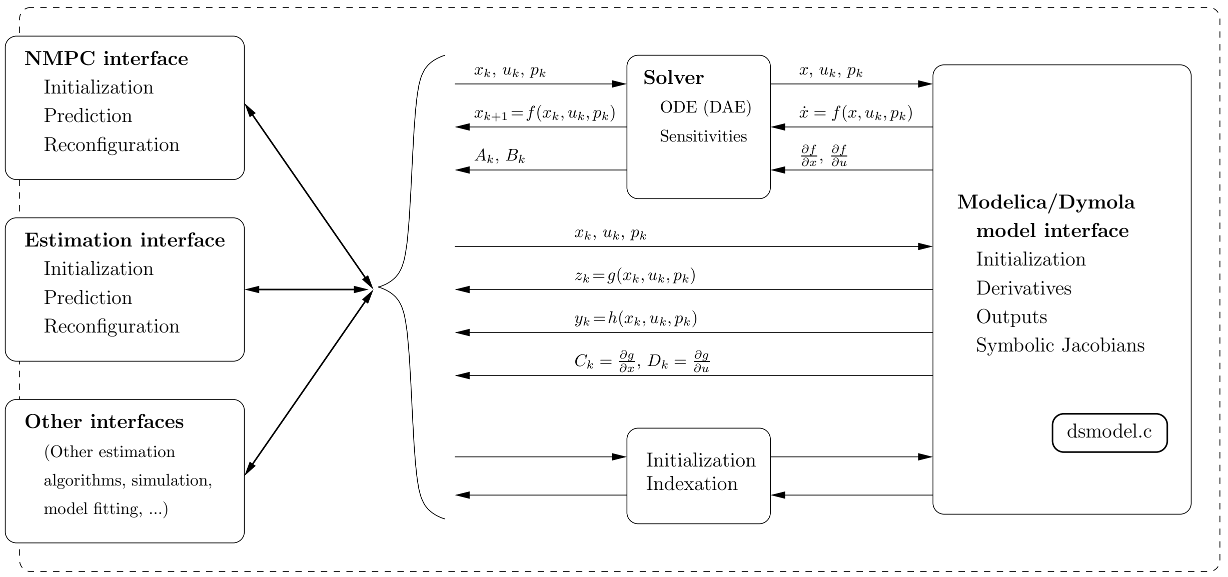


Figure 2: Overview over model component data flow.

which are modeled as Dymola inputs. We have chosen to control 2 or 4 of these. This means that we evaluate a B-matrix of dimension 27×9, instead of 27×2 or 27×4. This incurs considerable unnecessary complexity. If we in addition have a considerable number of parameters to be estimated also modeled as Dymola inputs, this makes the situation even worse.

- It seems the most natural way to implement communication between a NMPC system and the model, is to use 'top level' Modelica inputs and outputs. This is usually rather straightforward to implement for NMPC inputs and outputs (MVs, DVs and CVs) and measurements, but not very flexible: The Dymola C-code model interface could have been more sophisticated when it comes to identification and indexing of inputs, outputs and states. Furthermore, the use of top-level inputs and outputs can become rather awkward when it comes to model parameters that should be estimated.

6 Gradient computations applied to case

This section aims to illustrate the advantages and differences between finite differences and sensitivity-based sensitivity computations. Strictly speaking, the results only apply to this specific case, but we believe there is some generality in the trends reported. Issues that will be discussed, are computational complexity (timing), accuracy, and implementational aspects. The results are of course influenced by many factors not investigated (i.e., kept constant) here, as for example number of states, stiffness, exact definition of input blocks, etc.

We use a Matlab interface to the NMPC system, and choose to compare computational complexity by measuring execution time in Matlab. Although this has some drawbacks, it should give a fairly accurate picture of the relative performance. To increase the reliability, for each recording of execution time we run 5 consecutive identical NMPC scenarios (with significant excitation), and record the smallest execution time. This execution time includes the Kalman filter and NMPC optimization, but as the gradient computation is the most computationally expensive part, and the other parts are independent of choice of method for gradient computation, the difference in execution times should give a fairly good estimate of the difference in complexity of gradient computation.

6.1 Correctness and accuracy

It is clear that using finite difference (from now on FD) and analytic sensitivity methods based on sensitivity integration (AS) should give the same gradients “in the limit” (of perturbation-size and integration tolerances). Nevertheless, it is of interest to test this, also to get a feel for how large errors (or differences) relaxed integration tolerances

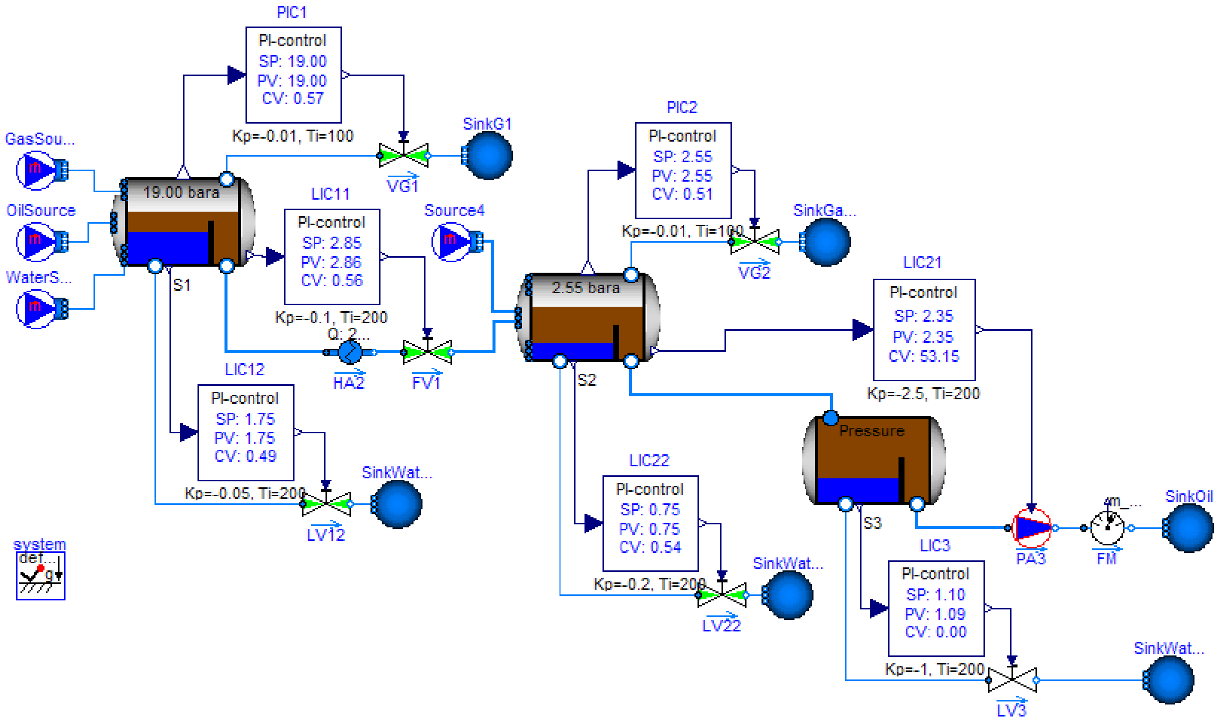


Figure 3: Overview of an offshore oil and gas processing plant, as implemented in Dymola.

and realistic perturbations will lead to.

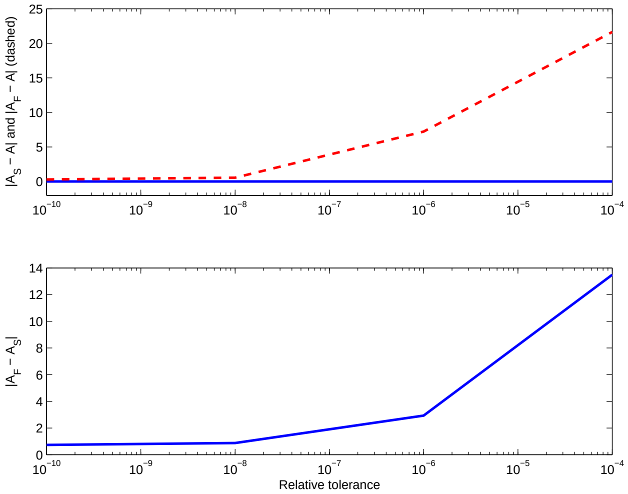


Figure 4: Top: Difference between 'real' (as found by AS methods with tolerances of 1e-12) step response matrix A and step response matrices for higher tolerances. Bottom: Step response matrices using AS and AF (almost) converge as tolerances decreases.

From Figure 4, we see that for small integration error tolerances, the gradients found are fairly correct in both methods, but the error in the FD sensitivity matrix increases much faster as the error tolerances are increased. An important note regarding the implementation of the AS-method is that we have chosen to have error control on both states *and* sensitivities, not merely states. In our experience, this can be essential to ensure accurately enough sensitivities when using AS.

We do not discuss here the choice of perturbation size in FD methods, as this does not differ from the general discussion in e.g. [10]. Suffice it to say that the general trends in Figure 4 are fairly independent of choice of perturbation size.

It is notable that even though both methods only give correct gradients in the limit, FD methods gives a direct approximation to the gradient of the objective function that is actually being optimized (including integration errors). This can in theory be an advantage in the line-search step of SQP algorithms.

6.2 Computational complexity

In this section we will compare the computational cost of different gradient computations as the number of NMPC degrees of freedom increases. We increase the degrees of freedom both in number of input blocks as well as number of inputs. The cases we will compare, are gradient computation using finite differences (FD) with or without using symbolic Jacobians in the ODE solver, and analytical gradient computation using sensitivity integration (AS), also with and without using symbolic Jacobians.

We use the same ODE tolerances in all cases, and for sensitivity integration the sensitivities are included in the error control. The relative time usage for different number of inputs and input blocks are shown in Figure 5. In Figure 6, the experiment is repeated for $N_u = 2$, but without using symbolic Jacobians in the ODE solver.

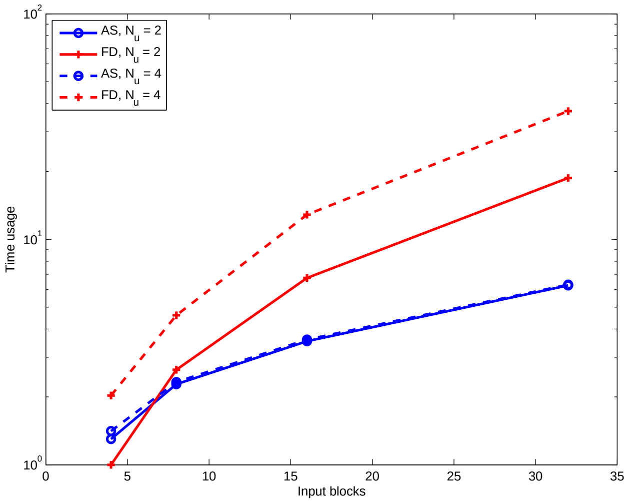


Figure 5: Relative time usage for different number of inputs and input blocks.

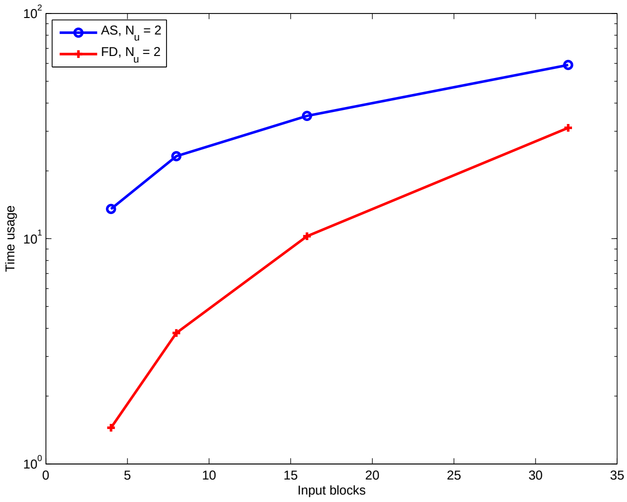


Figure 6: Relative time usage for different number of inputs and input blocks, without exporting symbolic Jacobians from Dymola.

The following observations are made:

- FD grows approximately quadratically in input blocks (as additional input blocks incurs both additional perturbations and ODE solver resetting), while AS grows approximately linearly in input blocks (incurs only additional ODE solver resetting). Note the logarithmic scale in Figure 5. To the extent that this is general, this means that AS will always outperform FD when many input blocks are used.
- Increasing number of input blocks (leads to more frequent ODE solver resetting) is more expensive than increasing number of inputs (leads to larger “sensitivity state”) when using AS. We attribute this both to the efficiency of CVODES in exploiting the structure in the sensitivity equations, but also to the next issue:
- Increasing number of inputs does not significantly increase complexity of AS. This may be surprising, but can be explained in this case by the fact that all ODE Jacobians are calculated irrespectively of how many of the inputs are actually active, as discussed in Section 5.2. If Dymola allowed calculation of only those Jacobians that are needed, this could considerably speed up execution time.
- AS suffers significantly more than FD from not having symbolical Jacobians available.

Finally, we mention that in our experience, implementing Modelica-functions in C can significantly speed up FD (not done in this case), see also [7]. If this can be combined with export of symbolic Jacobians, it can also speed up AS, but to a much less extent. In other words, implementing Modelica-functions in C is less important when using AS.

7 Concluding remarks

To construct the sensitivity matrix using analytical methods becomes significantly faster than finite difference-based methods as the number of inputs and/or input blocks increases. Therefore, this technology is expected to be important in NMPC systems for 'larger' (in a NMPC context) models. Furthermore, we expect that many of these 'larger' models will be implemented in high-level languages (Modelica) rather than in lower-level languages (C), due to issues like reuse and modularity, but also due to availability of symbolic Jacobians.

Moreover, we conclude that to use analytical methods, we should have ODE Jacobians available. Dymola provides these for us, but it seems we do not have the opportunity to evaluate 'partial' Jacobians, which would be a significant advantage.

We hope we have provided an incentive for developers of other Modelica-tools to generate symbolic Jacobians. Using automatic differentiation may also be possible in cases with C-code export-type functionality, but this could have some other drawbacks.

Acknowledgments

The authors thank StatoilHydro for providing information and input to the case study. Trond Tollefsen is acknowledged for his contributions to the development of the model library CyberneticaLib and particularly for the implementation of the case study model.

References

[1] L. T. Biegler, A. M. Cervantes, and A. Wächter. Advances in simultaneous strategies for dynamic process optimization. *Chem. Eng. Sci.*, 57:575–593, 2002.

[2] H. G. Bock, M. Diehl, D. B. Leineweber, and J. P. Schlöder. A direct multiple shooting method for real-time optimization of nonlinear DAE processes. In F. Allgöwer and A. Zheng, editors, *Nonlinear Predictive Control*, volume 26 of *Progress in Systems Theory*, pages 246–267. Birkhäuser, Basel, 2000.

[3] N. M. C. de Oliveira and L. T. Biegler. An extension of newton-type algorithms for nonlinear process control. *Automatica*, 31:281–286, 1995.

[4] B. A. Foss and T. S. Schei. Putting nonlinear model predictive control into use. In *Assessment and Future Directions Nonlinear Model Predictive Control*, LNCIS 358, pages 407–417. Springer Verlag, 2007.

[5] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I – Nonstiff problems.* Springer-Verlag, 2nd edition, 1993.

[6] A. C. Hindmarsh and R. Serban. *User Documentation for CVODES v2.5.0.* Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 2006.

[7] L. Imsland, P. Kittilsen, and T. S. Schei. Model-based optimizing control and estimation using modelica models. In *Proc. of Modelica'2008*, Bielefeld, Germany, 2008.

[8] J. B. Jørgensen. Adjoint sensitivity results for predictive control, state- and parameter-estimation with nonlinear models. In *Proceedings of the European Control Conference, Kos, Greece*, 2007.

[9] J. M. Maciejowski. *Predictive Control with Constraints.* Prentice-Hall, 2001.

[10] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer-Verlag, New York, 2006.

[11] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11:733–764, 2003.

[12] T. S. Schei. On-line estimation for process control and optimization applications. *Journal of Process Control*, 18:821–828, 2008.

Multiple-Shooting Optimization using the JModelica.org Platform

Jens Rantil* Johan Åkesson[†‡] Claus Führer* Magnus Gäfvert[‡]

** Department of Numerical Analysis, Lund University

[†] Department of Automatic Control, Lund University

*[†] Lund University
Sölvegatan 18
SE-22100 Lund, Sweden
E-mail: claus@maths.lth.se

[‡] Modelon AB
Ideon Science Park
SE-22370 Lund, Sweden
E-mail: info@modelon.se

Abstract

Dynamic optimization addresses the problem of finding the minimum of a cost function subject to a constraint comprised of a system of differential equations. There are many algorithms to numerically solve such optimization problems. One such algorithm is *multiple shooting*. This paper reports an implementation of a multiple shooting algorithm in Python. The implementation is based on the open source platform JModelica.org, the integrator SUNDIALS and the optimization algorithm `scipy_slsqp`. The JModelica.org platform supports model descriptions encoded in the Modelica language and optimization specifications expressed in the extension Optimica. The Modelica/Optimica combination provides simple means to express complex optimization problems in a compact and user-oriented manner. The JModelica.org platform in turn translates the high-level descriptions into efficient C code which can compiled and linked with Python. As a result, the numerical packages available for Python can be used to develop custom applications based on Modelica/Optimica specifications. An example is provided to illustrate the capabilities of the method.

Keywords: optimization; optimal control; parameter optimization; Modelica; Optimica; JModelica.org

1 Introduction

Dynamic optimization problems arise naturally in a wide range of applications and domains. Common examples are parameter optimization problems, design optimization, and optimal control. The key property of a dynamic optimization problem, which also makes such a problem hard to solve, is that it includes a constraint in form of a system of differential equations.

In this paper, we present an implementation of a particular numerical method for solving dynamic optimization problems, namely *multiple shooting*, [4, 20, 5]. In essence, a multiple shooting algorithm consists of an integrator for simulation of the system dynamics and evaluation of the cost function, and an optimization algorithm which tunes the optimization parameters of the problem. The Python language [17] was selected for implementation of the multiple shooting algorithm. Python has several advantages in the context of scientific computing, since there are several packages for high performance computing available, including Numpy [14] and Scipy [6]. Also, the package Matplotlib [9] provides methods for data visualization in a MATLAB-like manner. The main advantage is, however, that Python is a full-fledged high-level programming language offering strong support for generic concepts such as object-orientation and functional programming. Further more, Python has bindings to other languages, e.g, C and Fortran, which are common implementation languages for numerical algorithms. Accordingly, Python is commonly used as *glue language* in applications which integrate different algorithms. A common way to interface C code with Python is ctypes [7], which is based on loading of dynamically linked libraries. For the multiple shooting algorithm, Python is a suitable choice since it allows the majority of the computationally expensive subtasks to be delegated to precompiled C and Fortran codes.

DOI: 10.3384/ecp09430066

The implementation of the multiple shooting algorithm is based on the JModelica.org open source platform, [1]. JModelica.org offers support for dynamic models formulated in Modelica [21] and optimization specifications given in Optimica [2]. JModelica.org also has a Python-based execution API for the evaluation of the model equations, which has been used in this work. The purpose of this work is twofold. Firstly, solution of dynamic optimization problems by means of a multiple shooting algorithm adds to the functionality of the JModelica.org platform. Secondly, the algorithm provides an example of how different algorithms can be integrated with the JModelica.org model interface to create new algorithms.

2 Background

2.1 Modelica and Optimica

Modelica is a high-level language for encoding of complex heterogeneous physical systems, supporting object oriented concepts such as classes, components and inheritance. Also, text-book style declarative equations can be expressed as well as acausal component connections representing physical interfaces. While Modelica offers strong support for modeling of physical systems, the language lacks important constructs needed when formulating dynamic optimization problems, notably cost functions, constraints, and a mechanism to select inputs and parameters to optimize. In order to strengthen the optimization capabilities of Modelica, the Optimica extension has been proposed, [2]. Optimica adds to Modelica a small number of constructs, which enable the user to conveniently specify dynamic optimization problems based on Modelica models.

In the context of dynamic optimization, the use of high-level description formats is particularly attractive, since the interfaces of algorithms for solution of such programs are typically written in C or Fortran. Implementing the optimization formulation for such an algorithm may require a significant effort. In addition, once finalized, the implementation is typically difficult to reuse with another algorithm. The JModelica.org platform offers compilers for transforming Modelica/Optimica specifications into efficient C code which in turn may be interfaced with algorithms for dynamic optimization. The user may then focus on formulation of the actual problem at hand instead of attending to the details of encoding it to fit the requirements of a particular algorithm.

2.2 Dynamic optimization

Dynamic optimization problems may be formulated in many different ways, under different assumptions. In this paper, we assume that the problem is stated on the following form:

$$\begin{aligned}&\min_{p,u} \Phi(x(t_f;u,p),p)\\ &\text{subject to}\\ &\dot{x} = f(x,u,p)\end{aligned} \tag{1}$$

where x is the system state, u are the inputs and p are free parameters in the optimization. The cost function is here assumed to be of Mayer type, that is, a function of the terminal state values and the parameters are minimized[1]. The influence of the control variable u is implicit through x. The state x is governed by an ordinary differential equation (ODE). Indeed, this formulation is somewhat limited. In particular, the dynamics of physical systems are often described by differential algebraic equations (DAEs). Also, inequality constraints representing, e.g., bounds on states and inputs are often present. However, the formulation given above is indeed general enough to demonstrate the concept of the multiple shooting algorithm.

There are three main classes of dynamic optimization problems, namely *parameter optimization*, *optimal control* and *parameter identification*.

In parameter optimization, p is a vector containing a finite number of parameters. The goal is to find a parameter vector p that minimizes $\Phi(x(t_f;p),p)$. p can both contain model parameters and/or initial states of the model. An example of a parameter optimization problem would be to find the optimal wheel radius and tire thickness in a car to minimize the noise in the compartment.

In optimal control, the goal is to find a function $u(t)$ that minimizes $\Phi(x(t_f;u))$. This is usually done by discretization of u, for example by means of piecewise constant profiles or splines. Such an approximation transforms the original problem into a parameter optimization problem. An example of an optimal control problem would be to minimize the fuel consumption for a satellite moving from the moon to the earth.

Thirdly, in parameter identification, the objective is to fit a model to existing measurements. Typically, a perfect fit is usually not possible to obtain, due to the presence of measurement noise. Instead, the best

[1] Notice that a Lagrange type cost function, i.e., a function of the inputs, states and parameters integrated of the optimization interval can be cast into a Mayer cost function by introducing an additional state.

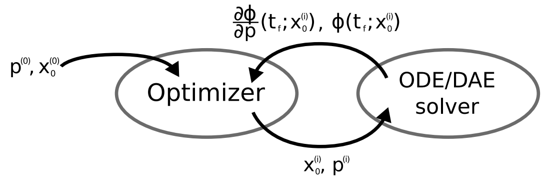

$\frac{\partial \phi}{\partial p}(t_f;x_0^{(i)})$, $\phi(t_f;x_0^{(i)})$
$p^{(0)}, x_0^{(0)}$
Optimizer
ODE/DAE solver
$x_0^{(i)}, p^{(i)}$

Figure 1: Shooting procedure.

fit according to some criteria penalizing the deviation between the model outputs and the measurements is sought.

2.3 Numerical methods for dynamic optimization

There are two main branches within the family of direct methods for dynamic optimization. *Sequential methods* rely on state of the art numerical integrators, typically also capable of computing state sensitivities, and on standard nonlinear programming (NLP) codes. The controls are then usually approximated by piece-wise polynomials (often piecewise constant functions), which render the controls to be parameterized by a finite number of parameters. These parameters are then optimized. *Simultaneous methods*, on the other hand, are based on *collocation*, and approximate *both* the state and the control variables by means of piece-wise polynomials, see [3] for an overview. This strategy requires a fine-grained discretization of the states, in order to approximate the dynamic constraint with sufficient accuracy. Accordingly, the NLPs resulting from application of simultaneous methods are very large, but also sparse. In order to solve large-scale problems, the structure of the problem needs to be explored.

2.3.1 Sequential shooting methods

In a sequential method, the control variables are parameterized by a finite number of parameters, for example by using a piece-wise polynomial approximation of u. Given fixed values of the parameters, the cost function of the optimization problem (1) can be evaluated simply by integrating the dynamic system. The parameters may then, in turn, be updated by an optimization algorithm, and the procedure is repeated, as illustrated in Figure 1. When the optimization algorithm terminates, the optimal parameter configuration is returned. Since the parameters determine the control profiles, which are then used to compute $x(t_f)$, the cost function can be written as $\Phi(x(t_f;u(p),p),p) = \Phi(p)$. The infinite dimensional optimization problem is thus transformed into a finite dimensional problem. For these reasons, sequential methods are also referred to as control parameterization methods. For a thorough description of single shooting algorithms, see [22].

Typically, the convergence of an optimization algorithm can be improved by providing it with gradients of the cost function with respect to the parameters. While finite differences is a simple method for obtaining gradients, it is not well suited for in this particular application due to scaling problems and limited accuracy, [19]. Taking the full derivative of the cost function $\Phi(x(t_f;u(p),p),p) = \Phi(p)$, we obtain

$$\left.\frac{d\Phi}{dp}\right|_{t_f} = \frac{\partial \Phi}{\partial x}^T \left.\frac{\partial x}{\partial p}\right|_{t_f}. \tag{2}$$

While $\frac{\partial \Phi}{\partial x}$ is usually straightforward to compute, the quantity $\frac{\partial x}{\partial p} = x_p$, referred to as the *state sensitivity* with respect to the parameter p, needs attention. A common approach for computing state sensitivities is derived by differentiating the differential equation $\dot{x} = f(x,u,p)$ with respect to p:

$$\frac{d}{dp}\frac{dx}{dt} = \frac{d}{dp} f(x,u,p) \tag{3}$$

$$\Rightarrow \quad \frac{d}{dt}\left(\frac{\partial x}{\partial p}\right) = \frac{\partial f}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial f}{\partial u}\frac{\partial u}{\partial p} + \frac{\partial f}{\partial p} \tag{4}$$

which gives the *sensitivity equations*

$$\dot{x}_p(t) = \frac{\partial f}{\partial x} x_p(t) + \frac{\partial f}{\partial u}\frac{\partial u}{\partial p} + \frac{\partial f}{\partial p}. \tag{5}$$

This suggests a method for computing derivatives by solving results a matrix valued differential equation. If the number of states of the system is n_x and the number of parameters is n_p, then $n_x \times n_p$ additional equations must be integrated. This operation is computationally expensive, although the efficiency of the integration can be increased by exploring the structure of the sensitivity equations. There is also software available which supports integration of the sensitivity equations, for example DASPK, [13] and SUNDIALS [18].

2.3.2 Multiple shooting

An extension of the single shooting algorithm is *multiple shooting*. In a multiple shooting algorithm, the optimization interval $[t_0,t_f]$ is divided into a number of *subintervals* $[t_i,t_{i+1}]$, see Figure 2. New optimization variables corresponding to the initial conditions for the states in each subinterval, are then introduced. This

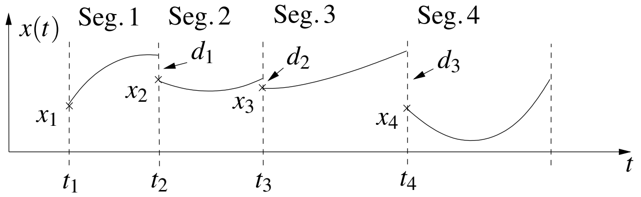


Figure 2: In a multiple shooting method, the control horizon is divided into a number of segments, which are integrated independently.

enables the dynamics, as well as the sensitivity matrices, to be computed *independently* in each segment. In order to enforce continuity of the state profiles, equality constraints are introduced in the optimization problem which ensure that the *defects*, $d_i = x(t_{i+1}^+) - x(t_{i+1}^-)$ are equal to zero.

In an optimization loop some of the intermediate control profiles u and parameter values p may get unphysical values. This can result in stability problems, when numerical integration has to be performed over longer time intervals. Also the computed sensitivity matrices may become unreliable. This is the advantage of multiple shooting algorithms compared to the single shooting algorithm. If the integration is performed over shorter intervals the numerical stability properties of the algorithm are improved. Another advantage of multiple shooting algorithms is that state inequality constraints can be more easily accommodated. Since the initial states in each segment are optimization variables, algebraic inequality constraints can be enforced for the states variables at the segment junctions. However, it has to be emphasized, that only the state variables at the segment junctions, t_i, can be restricted by inequality constraints.

2.4 The JModelica platform

JModelica.org is a novel Modelica-based open source project targeted at dynamic optimization [1]. JModelica.org features compilers supporting code generation of Modelica models to C, a C API for evaluating model equations and their derivatives and optimization algorithms. The compilers and the model C API have also been interfaced with Python in order to enable scripting and custom application development. In order to support formulation of dynamic optimization of Modelica models, JModelica.org supports the Optimica extension [2] of the Modelica language. Optimica offers constructs for encoding of cost functions, constraints, the optimization interval with fixed or free end points as well as the specification of the transcription scheme.

The C API providing functions for evaluating the model equations, cost function and constraints is entitled the JModelica.org Model Interface (JMI). The C code generated by the compiler front-end is compiled with a runtime library into a shared object file which in turn is loaded into Python, using the ctypes library. The JMI C functions can then be conveniently called from a Python shell or script. In addition, the input and return types of the C functions (typically pointers to vectors of *double* type) are mapped onto the types used by the Numpy package. This approach grants for a seamless integration between JMI and algorithms and data structures provided by Numpy and Scipy.

3 Implementation

As described in Section 2.3.2, a multiple shooting algorithm relies on a simulation algorithm, preferable capable of computing sensitivities, and a numerical optimization algorithm for algebraic optimization problems. The remaining part of the multiple shooting algorithm then consists of providing a non-linear program (NLP) to the optimization algorithm and to invoke the simulation algorithm in order to obtain function evaluations and derivative information. This part also includes representation of parameterized control signals, to keep track of optimization parameters, and to interface with the model execution API.

The simulation algorithm SUNDIALS [8, 18] was chosen for integration of the system dynamics. SUNDIALS is a high-quality integration package which is the latest evolution of a branch of ODE and DAE solvers including, e.g., DASSL. SUNDIALS contains a set of integration methods based on variable step size variable order multi-step methods using either the backward differentiation formula (BDF) or the more accurate Adams-Moulton formulae. BDF methods are well known for their good numerical stability properties for highly damped problems. In the context of this work, SUNDIALS has two major advantages. Firstly, it supports computation of sensitivity matrices. Secondly, there is a freely available Python interface for SUNDIALS; PySUNDIALS [16].

As for the optimization algorithm, the method `scipy_slsqp` was used, which is a Python wrap of the sequential quadratic programming algorithm [11]. This method is one of a variety of optimization algorithms interfaced by the Python package OpenOpt [12], which is a package that provides a unified interface to a large number of optimization algo-

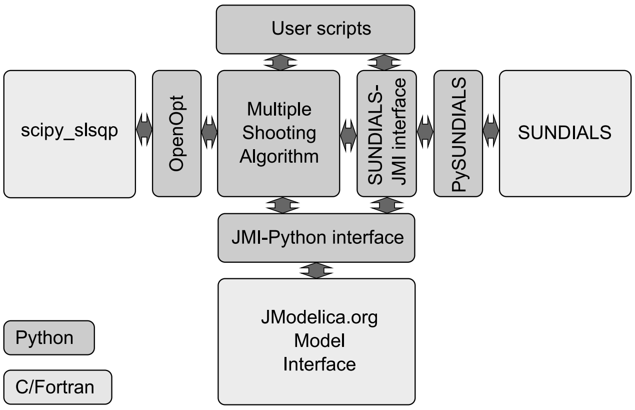


Figure 3: Architecture of the multiple shooting algorithm.

rithms.

The architecture of the algorithm is depicted in Figure 3. SUNDIALS is implemented in C, and interfaced to Python by the PySUNDIALS package. In order to provide convenient means to simulate models compiled with the JModelica.org compilers, an interface between PySUNDIALS and the Python wrappers of the JMI functions has been developed. This interface also supports computation of sensitivities, which is needed by the multiple shooting algorithm. Computation of sensitivities requires a slightly more complex setup than simulation, since in the former case, the sensitivity parameters need to be specified. SUNDIALS can make use of Jacobians provided by a model execution interface. As for simulation, the Jacobian of the right hand side of the ODE with respect to the states is required, and it is also standard for simulation oriented interfaces to provide such a function. In the case of sensitivity computations, however, the Jacobians with respect also to the inputs (in the case of an optimal control problem) and the parameters are required. These Jacobians are also available in the JMI interface. In the first implementation of the multiple shooting algorithm, Jacobians are not propagated to SUNDIALS. Rather, this is left for future improvements.

The main task of the multiple shooting algorithm is to provide call-back functions for evaluation of the cost function and constraints to the optimization algorithm. At this level, the optimization problem is a purely algebraic NLP; the dynamic part is handled by SUNDIALS and is in effect hidden from the optimizer.

OpenOpt provides standardized interfaces to different classes of optimization problems. Amongst them is a class which supports non-linear cost functions, equality constraints, and bounded optimization variables. OpenOpt also interfaces a number of different solvers which support this class of problems. This approach makes it trivial to test different optimization algorithms with very minor changes to the code. In essence, the OpenOpt interface requires Python functions for evaluation of the cost function, the constraints and, if available, their derivatives. These functions, in turn, invoke integration and sensitivity computation by means of SUNDIALS. Also, functions in JMI are directly invoked, e.g., to evaluate the cost function. In effect, the multiple shooting algorithm provides an interface between the optimization algorithm on one hand and the simulation of the dynamic system and associated sensitivities on the other hand.

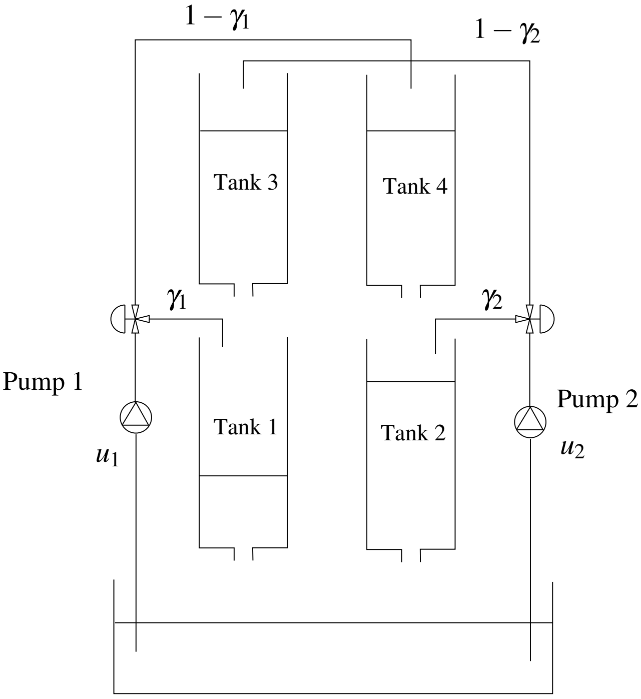


Figure 4: A schematic picture of the quadruple tank process

4 An Example

The quadruple-tank laboratory process, see Figure 4, has been used to demonstrate the multiple shooting algorithm. The model presented here is derived in [10]. The process consists of four tanks, organized in pairs (left and right), where water from the two upper tanks flows into the two lower tanks. A pump is used to pour water into the upper left tank and the lower right tank. A valve with fixed position is used to allocate pump capacity to the upper and lower tank respectively. A

Table 1: Parameter values of the Quadruple Tank

Parameters	Values	Unit
A_1,A_3	2.8e-3	[m^2]
A_2,A_4	3.2e-3	[m^2]
a_1,a_3	7.1e-6	[m^2]
a_2,a_4	5.7e-6	[m^2]
k_1,k_2	3.14e-6, 3.29e-6	[m^3/Vs]
γ_1,γ_2	0.7, 0.7	
g	9.81	[m/s^2]

second pump is used to pour water into the upper right tank and lower left tank. The control variables are the pump voltages. Let the states of the system be defined by the water levels of the tanks (expressed in m) x_1, x_2, x_3 and x_4 respectively. The maximum level of each tank is 20 cm. The dynamics of the system is given by

$$\begin{aligned}
\dot{x}_1 &= -\frac{a_1}{A_1}\sqrt{2gx_1} + \frac{a_3}{A_1}\sqrt{2gx_3} + \frac{\gamma_1 k_1}{A_1}u_1 \\
\dot{x}_2 &= -\frac{a_2}{A_2}\sqrt{2gx_2} + \frac{a_4}{A_2}\sqrt{2gx_4} + \frac{\gamma_2 k_2}{A_2}u_2 \\
\dot{x}_3 &= -\frac{a_3}{A_3}\sqrt{2gx_3} + \frac{(1-\gamma_2)k_2}{A_3}u_2 \\
\dot{x}_4 &= -\frac{a_4}{A_4}\sqrt{2gx_4} + \frac{(1-\gamma_1)k_1}{A_4}u_1
\end{aligned} \tag{6}$$

where the A_i:s and the a_i:s represent the cross section area of the tanks and the holes respectively. The parameters γ_i:s determine the position of the valves which control the flow rate to the upper and lower tanks respectively. The control signals are given by the the u_i:s. Numerical values of the parameters are given in Table 1.

We consider two different stationary operation points corresponding to constant control inputs and where $\dot{x} = 0$. The first operating point, call it A, is defined by the control inputs $u_1^A = u_2^A = 2.0$, and the second, call it B, is defined by $u_1^B = u_2^B = 2.5$. The corresponding stationary state values are $x^A = (0.041, 0.066, 0.0039, 0.0056)$ and $x^B = (0.064, 0.10, 0.0062, 0.0087)$. Based on the operating points A and B, the following optimal control problem is defined:

$$\min_{u(t)} \int_0^{t_f} \alpha \sum_{i=1}^{4}(x_i(t) - x_i^B)^2 + \sum_{i=1}^{2}(u_i(t) - u_i^B)^2 dt \tag{7}$$

where α is a constant weight. Notice that this cost function is not on the form (1) and can therefore not be directly implemented. Instead, an additional state,

```
optimization QuadTank_Opt
  (objective = x_5(finalTime),
  startTime = 0, finalTime = 50)
  import SI = Modelica.SIunits;
  // Process parameters parameter
  SI.Area A1=2.8e-3, A2=3.2e-3,
          A3=2.8e-3, A4=3.2e-3;
  parameter SI.Area a1=7.1e-6, a2=5.7e-6,
                    a3=7.1e-6, a4=5.7e-6;
  parameter SI.Acceleration g=9.81;
  parameter Real k1_nmp(unit="m/s/V") =
                            3.14e-6,
                 k2_nmp(unit="m/s/V") =
                            3.29e-6;
  parameter Real g1_nmp=0.70, g2_nmp=0.70;
  // Initial tank levels
  parameter SI.Length x1_0 = 0.04102638;
  parameter SI.Length x2_0 = 0.06607553;
  parameter SI.Length x3_0 = 0.00393984;
  parameter SI.Length x4_0 = 0.00556818;
  // Reference values
  parameter SI.Length x1_r = 0.06410371;
  parameter SI.Length x2_r = 0.10324302;
  parameter SI.Length x3_r = 0.006156;
  parameter SI.Length x4_r = 0.00870028;
  parameter SI.Voltage u1_r = 2.5;
  parameter SI.Voltage u2_r = 2.5;
  // Tank levels
  SI.Length x1(start=x1_0);
  SI.Length x2(start=x2_0);
  SI.Length x3(start=x3_0);
  SI.Length x4(start=x4_0);
  // Inputs
  input SI.Voltage u1(free=true);
  input SI.Voltage u2(free=true);
  // Cost function weight parameter
  Real alpha = 40000;
  Real x_5(start=0);
equation
  der(x1) = -a1/A1*sqrt(2*g*x1) +
            a3/A1*sqrt(2*g*x3)
            + g1_nmp*k1_nmp/A1*u1;
  der(x2) = -a2/A2*sqrt(2*g*x2) +
            a4/A2*sqrt(2*g*x4) +
            g2_nmp*k2_nmp/A2*u2;
  der(x3) = -a3/A3*sqrt(2*g*x3)
            + (1-g2_nmp)*k2_nmp/A3*u2;
  der(x4) = -a4/A4*sqrt(2*g*x4) +
            (1-g1_nmp)*k1_nmp/A4*u1;
  der(x_5) = alpha*((x1_r - x1)^2 +
                    (x2_r - x2)^2 +
                    (x3_r - x3)^2 +
                    (x4_r - x4)^2) +
                    (u1_r - u1)^2 +
                    (u2_r - u2)^2;
end QuadTank_Opt;
```

Listing 1: An Optimica specification of the quadruple tank optimization problem.

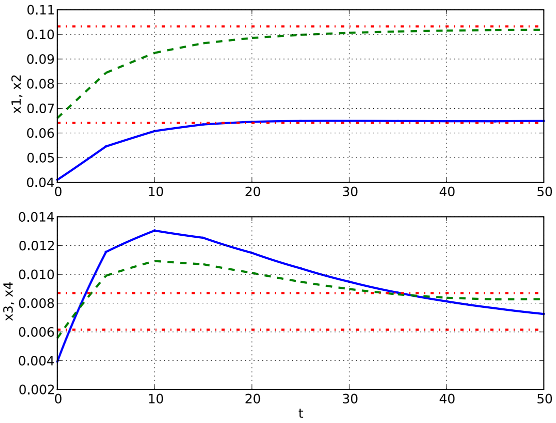


Figure 5: Optimal state profiles

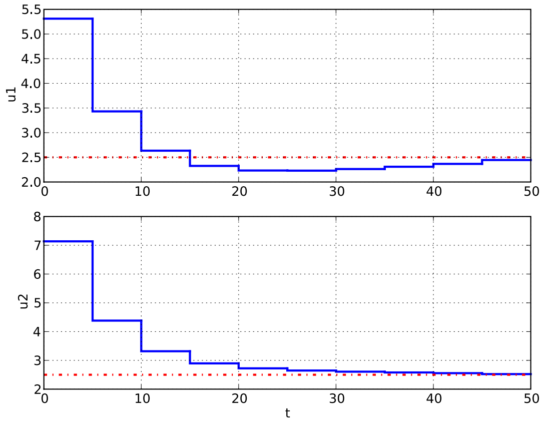


Figure 6: Optimal control profiles

x_5, is introduced. The additional state is governed by the differential equation

$$\dot{x}_5 = \alpha \sum_{i=1}^{4} (x_i(t) - x_i^B)^2 + \sum_{i=1}^{2} (u_i(t) - u_i^B)^2 \qquad (8)$$

The optimization criteria may now be written as

$$\min_{u(t)} x_5(t_f) \qquad (9)$$

The initial state values are assumed to be fixed and equal to x^A. Hence, the optimal control problem is defined as to transfer the state of the system from operating point A to operating point B. The Optimica specification for the optimal control problem is given in Listing 1.

The problem was solved using the multiple shooting algorithm, with the control signal parameterized to be constant over ten interval. Correspondingly, the number of intervals in the multiple shooting algorithm was set to ten. The optimization parameters, i.e., the control variable values in the ten elements and the initial state values in elements 2-9, were initialized in the following way. First, the control inputs were set to $u_1 = u_2 = 2.5$, and the dynamics was simulated over the optimization interval. The control variable values were then all set to 2.5 and the initial state values of each interval were initialized from the simulated state profiles.

The optimal state profiles are shown in Figure 5. The upper plot shows the levels in the lower tanks, where x_1 corresponds to the solid curve and x_2 corresponds to the dashed curve. The lower plot shows the levels in the upper tanks; x_3 in solid and x_4 in dashed. Also, the target values corresponding to operating point `B` are represented by the dash dotted lines. As can be seen, the state profiles approach the target values. The optimal control profiles are shown in 6, where u_1 is given in the upper plot and u_2 in the lower plot. As expected, the control signals approach the stationary values corresponding to operating point `B`.

5 Summary and future work

In this paper, an implementation of a multiple shooting algorithm has been presented. The implementation is done in Python and is based on the JModelica.org open source platform, the numerical integration package SUNDIALS and the optimization algorithm `scipy_slsqp`. It has been shown how the JModelica.org Python interface, providing access to functions for evaluation of the model equations, can be explored in order to develop custom algorithms in Python.

There are several improvements that would increase the applicability of the algorithm. The control variable parameterization is currently limited to one constant value per multiple shooting element. Implementing support for arbitrarily many elements in the parameterization of control variables as well as support for piecewise linear control profiles would be suitable extensions. The performance of the algorithm may be further improved by providing high accuracy Jacobians, available in JMI, to SUNDIALS. In addition, extending the sensitivity analysis of SUNDIALS to support discontinuities, see [15], would enable optimization of hybrid systems.

References

[1] J. Åkesson, T. Bergdahl, M. Gäfvert, and H. Tummescheit. The JModelica.org Open Source Platform. In *7th International Modelica Conference 2009*. Modelica Association, 2009.

[2] Johan Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *In 6th International Modelica Conference 2008*. Modelica Association, March 2008.

[3] L.T. Biegler, A.M. Cervantes, and A Wachter. Advances in simultaneous strategies for dynamic optimization. *Chemical Engineering Science*, 57:575–593, 2002.

[4] H.G. Bock and K. J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In *Ninth IFAC world congress*, Budapest, 1984.

[5] R. Bulirch. Die Mehrzielmethode zur numerischen Lösung von nichtlinearen Randwertproblemen und Aufgaben der optimalen Steuerung. Technical report, Carl-Cranz-Gesellschaft, 1971.

[6] Inc. Enthought. SciPy, 2009. `http://www.scipy.org/`.

[7] Python Software Foundation. ctypes: A foreign function library for Python, 2009. `http://docs.python.org/library/ctypes.html`.

[8] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, 31(3):363–396, 2005.

[9] J. Hunter, D. Dale, and M. Droettboom. matplotlib: python plotting, 2009. `http://matplotlib.sourceforge.net/`.

[10] Karl Henrik Johansson. *Relay Feedback and Multivariable Control*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, September 1997.

[11] Dieter Kraft. TOMP - Fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software*, 20(3):262–281, 1994.

[12] Dmitrey L. Kroshko. OpenOpt Home Page, 2009. `http://www.openopt.org/Welcome`.

[13] T. Maly and L. R. Petzold. Numerical methods and software for sensitivity analysis of differential-algebraic systems. *Applied Numerical Mathematics*, 20(1-2):57–82, 1996.

[14] T. Oliphant. Numpy Home Page, 2009. `http://numpy.scipy.org/`.

[15] A. Pfeiffer. *Numerische Sensitivitätsanalyse unstetiger multidisziplinärer Modelle mit Anwendungen in der gradientenbasierten Optimierung (Numerical sensitivity analysis of discontinuous multidisciplinary models with applications in gradient based optimization)*. PhD thesis, Martin Luther University Halle-Wittenberg, 2008.

[16] Open Source Project. Pysundials. `http://pysundials.sourceforge.net`, May 2009.

[17] Open Source Project. Python programming language. `http://www.python.org`, May 2009.

[18] Open Source Project. Suite of nonlinear and differential/algebraic equation solvers (sundials). `http://www.llnl.gov/casc/sundials/`, May 2009.

[19] O. Rosen and R. Luus. Evaluation of gradients for piecewise constant optimal control. *Comput. chem. Engng.*, 15(4):273–281, 1991.

[20] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, New York and Berlin, 1980.

[21] The Modelica Association. The Modelica Association Home Page, 2007. `http://www.modelica.org`.

[22] V. Vassiliadis. *Computational solution of dynamic optimization problem with general differential-algebraic constraints*. PhD thesis, Imperial Collage, London, UK, 1993.

Symbolic Model Reduction Applied to Realtime Simulation of a Construction Machine

Lars Mikelsons[1] Hongchao Ji[1] Thorsten Brandt[1] Oliver Lenord[2]
[1] Institute for Mechatronics and System Dynamics , University of Duisburg-Essen, Germany
[2] Bosch Rexroth AG, Germany
{mikelsons,ji,brandt}@imech.de

Abstract

The vehicle response of construction machines strongly depends on the tuning of the control system in interaction with the drive system. A compromise between performance and comfort needs to be found to fulfill the operators requirements on a high usability of the machine. In order to achieve an optimal behavior Hardware-in-the-Loop simulation techniques offer a suitable approach to determine the overall behavior in advance. Prerequisition is a realtime capable simulation model of the considered system. Therefore, in this paper the mathematical model of the system is automatically adapted by symbolic model reduction algorithms in order to match real-time requirements on a given hardware. Inputs to the automatic reduction algorithm are the complex mathematical system model, the desired realtime cycle and the number of floating operations per second (flops), which can be realized by the chosen target hardware. The outputs of the algorithm are the automatically reduced model, which is guaranteed to run in realtime on the target hardware and the maximal model error for the test scenario. In this paper, the reduction procedure is demonstrated for the complex hydromechanical model of a so-called skid steer loader. Summarizing, the proposed procedure of symbolic model reduction helps to reduce the developing phase of mechatronic prototypes dramatically as the adaptation of the system model with respect to the target hardware is completely automated.

Keywords: symbolic model reduction, realtime, construction maschines, object oriented modelling

1 Introduction

Nowadays many complex systems are modeled in object oriented simulation tools like for example Dymola [4] or SimulationX, which base on Modelica [5] and hence generate a symbolic representation of the emerging DAE system. Having a symbolic representation at hand, the equations can be manipulated, simplified or even reduced. While algorithms for simplification and index reduction are already implemented in those simulation tools, not much attention has been paid to symbolic reduction techniques [2, 13]. Though, they are a very powerful tool for automated generation of less complex models [9]. Symbolic reduction techniques were first used in analog circuit design [2] and based on the DC-analysis of nonlinear analog circuits. These techniques were extended to the reduction of arbitrary DAE-systems in [12, 13]. Hence, symbolic reduction techniques can be used for the modeling and design of mechatronic systems [10]. Examining a complex physical system like construction maschines in many cases only one model is not sufficient. Often a very accurate model is required in order to analyze certain physical effects, while at the same time a model for realtime simulation is required. Here symbolic reduction techniques come into play. Up to now symbolic reduction techniques lower the complexity (and therefore the level of detail) of the model until a user defined error bound is reached. In this contribution this approach is extended in order to obtain models which are usable for realtime simulation on a given realtime target in a given realtime cycle.

In section 2 symbolic reduction techniques are briefly introduced and extended for realtime reduction. After that the approach is applied to a construction machine called skid steer loader. In section 3 the MathModelica [6] model of the skid steer loader is presented, while in section 4 the reduction results are given. The paper closes with a conclusion and an outlook in section 5.

2 Symbolic Reduction Techniques

The basic idea of symbolic model reduction techniques is to identify those terms of a DAE (or ODE)

DOI: 10.3384/ecp09430136

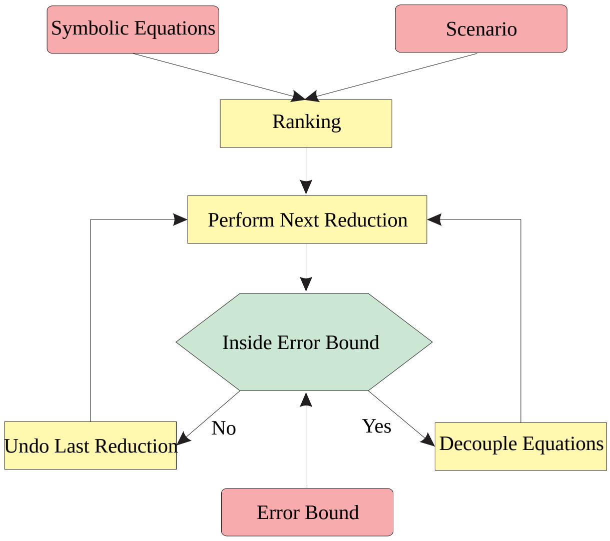


Figure 1: Scheme of the Reduction Algorithm

system, whose influence on the solution of the system is minor, and to perform a reduction on them (e.g. to neglect them). The algorithm consists of two steps, see for example [10] and [13]. First a specific reduction technique is chosen. Afterwards the relevance of each term for the solution of the DAE-System is estimated in the so called "ranking". Then the terms are sorted in increasing order with respect to their influence on the solution in order to perform the reductions as long as the solution of the reduced DAE-System remains within a user-defined error bound ε [13]. This basic idea is extended in section 2.4 in order to obtain reduced models, which can be simulated in realtime on a given realtime target. Possible reduction techniques are neglecting terms, setting terms to constants, linearization of terms or symmetry considerations. While the first three reductions are operations on terms of the DAE-System, the last one operates on variables and is explained later on. A scheme of the symbolic reduction algorithm is shown in Fig.1 for a chosen reduction technique. Given a scenario (system inputs,initial states and parameters) and an error bound, the algorithm starts with the ranking. Afterwards it is checked whether the reductions lead to an error inside the error bounds, beginning with the smallest. Finally, a less detailed model, performing within the prescribed error bounds results.

Let now

$$\mathbf{F} : \Omega \times I \mapsto \mathbb{R}^m \tag{1}$$

be differentiable, where $\Omega \subset \mathbb{R}^n \times \mathbb{R}^n$ is an open set. Then

$$\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = 0 \tag{2}$$

is called DAE-system if $\frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}$ is singular. Furthermore, let $\mathbf{F}$ be given in expanded form

$$\mathbf{F}_i(\mathbf{x}, \dot{\mathbf{x}}, t) = \sum_{k=1}^{l_i^1} t_{k_i}^1(\mathbf{x}, \dot{\mathbf{x}}, t), \quad 1 \le i \le m, \tag{3}$$

where l_i^1 is the number of terms in $\mathbf{F}_i$ and $t_{k_i}^1$ denotes the k-th term in $\mathbf{F}_i$. Each term in the first level $t_{k_i}^1$ may consist of a function $f_{k_i}^1$, whose argument is a sum of $l_{k_i}^2$ second level subterms $t_{k_i}^2$ $(1 \le i \le l_{k_i}^2)$

$$t_{k_i}^1(\mathbf{x}, \dot{\mathbf{x}}, t) = f_{k_i}^1(\sum_{k=1}^{l_{k_i}^2} t_{k_i}^2(\mathbf{x}, \dot{\mathbf{x}}, t)), \tag{4}$$

and so on. Here level indicates the hierarchy of arguments nested into each other in each single summand. Then the set $\mathscr{T}^i$ is the set of all terms in the i-th level. The manipulation of a term is called reduction in the following. Consequently, for the set of all reductions $\mathscr{K}^i$ for one reduction technique in a level i, it holds

$$\left|\mathscr{T}^i\right| = \left|\mathscr{K}^i\right|. \tag{5}$$

For $\kappa \in \mathscr{K}$

$$\mathbf{F}^\kappa = 0 \tag{6}$$

is the DAE-system emerging from the reduction κ. Then for DAE-systems of the form of Eq. 2

$$\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t, \mathbf{u}) = 0 \tag{7}$$

with system inputs $\mathbf{u}$, a scenario is the set of a vector field defined on the interval I for the system inputs, the initial values and the parameters. Furthermore, $\mathscr{N}(F(\mathbf{x}, \dot{\mathbf{x}}, t), \mathbf{u})$ is the solution of Eq. 2 computed by a numerical integrator $\mathscr{N}$ at nodes $t_1, \ldots, t_N$. The solution

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_{out} \\ \overline{\mathbf{y}} \end{bmatrix} = \mathscr{N}(F(\mathbf{x}, \dot{\mathbf{x}}, t), \mathbf{u}) \tag{8}$$

consists of two components. In $\mathbf{y}_{out}$ the n_{out} output variables are contained, while $\overline{\mathbf{y}}$ consists of the remaining internal variables.

2.1 Reduction Techniques

As already mentioned above possible reduction techniques are the neglecting terms ($\mathscr{U}_{neg}$), setting terms to constants ($\mathscr{U}_{const}$), symmetry considerations ($\mathscr{U}_{sym}$) or the simplification of piecewise functions. In this contribution only $\mathscr{U}_{neg}$ is chosen. Certainly, the easiest manipulation of a term is neglecting it. Using $\mathscr{U}_{const}$ for each term a constant has to be chosen. Usually the mean value throughout the simulation is employed. Clearly, this mean value has to be determined before. At first sight, this looks like a drawback, but a reference simulation is essential for the ranking anyway as will be seen in the next section. However, other values than the mean value are thinkable. Choosing $\mathscr{U}_{sym}$ at first variables which have similar values throughout the simulation are sought. Alternatively variables, which are expected to be similar can be flagged. For two similar variables every occurrence of the first variable (or its derivative) is substituted by the second variable (or its derivative). Consequently, now one equation can be canceled. A reasonable choice is that equation which leads to smallest error.

2.2 Ranking

In [12] different ranking algorithms are proposed. In this contribution only the so called One-Step Ranking will be discussed. In general a ranking procedure estimates the influence of a reduction on the solution of a DAE (or ODE) system. A reasonable measure for the influence of a reduction is the error emerging from the reduction. In order to get a good estimate of that error a reference solution $\mathbf{y}^\star$ is required. The crux of the matter is that the quality of the estimate increases with the duration of the ranking procedure. Hence, a ranking procedure should be a good compromise between computation time and accuracy. Mathematically speaking a ranking procedure $\mathscr{R}$ maps two DAE-systems on a real value, estimating the error between their solutions. Apparently, perfect accuracy can be achieved by the use of simulations. Though, this would lead to very high computation costs.

One-Step Ranking Typically, computing the solution of a DAE-system, at each time step a non-linear system of equations is iteratively solved. Usually the solution of the preceding time step is used as the initial value for the solution of the system of non-linear equations at the next time-step. For the computation of the solution of Eq.6, the reference solution $\mathbf{y}^\star$ at the corresponding time steps can be used for the initial values. Now, additionally limiting the iterations to one, a estimate of the solution of Eq.6 $\widehat{\mathbf{y}}$ is obtained. Consequently

$$\mathscr{R}_{step}(\mathbf{F},\kappa) = \|\mathbf{y}^\star_{out} - \widehat{\mathbf{y}_{out}}\| \tag{9}$$

is computed. The one-step ranking usually delivers a good compromise between accuracy and runtime.

2.3 Term Cancellation

In the term cancellation procedure the ranking is used, to perform as many reductions as possible, while preserving the desired accuracy. Hence, reductions are performed as long as the error of the reduced model remains within the error bound ε. The error emerging from the reductions is measured only at the n_{out} output variables. Thus, ε has dimension n_{out}. To perform as many reductions as possible, it is beneficial to start with those reductions, which lead to a small error. Thus, first the set of reductions $\mathscr{K}$ is sorted in ascending order depending on the ranking, resulting in $\mathscr{K}_{sort}$. Now, one possibility is to check one reduction of $\mathscr{K}_{sort}$ after the other. This is done by checking the computed solution of the reduced DAE-system for staying within the error bound ε. However, this method can be accelerated by the use of clusters [11]. Using clusters, the set of reductions $\mathscr{K}_{sort}$ is divided into s disjunct subsets

$$\mathscr{K}_{sort} = \bigcup_{i=1}^{s} \mathscr{S}_i, \tag{10}$$

where

$$\mathscr{S} = [S_1, \ldots, S_s]. \tag{11}$$

Each cluster $\mathscr{S}_i$ contains reductions leading to a similar estimated error (for example up to a factor of 10). Now the clusters are checked one after another, beginning with S_1 containing the reductions leading to the smallest estimated error. Thus, multiple reductions can be verified by one simulation. If a cluster $\mathscr{S}_i$ can not be verified (the reductions of $\mathscr{S}_i$ lead to errors greater than the error bound ε), $\mathscr{S}_i$ is divided disjunct into two clusters $\mathscr{S}_i^1$ and $\mathscr{S}_i^2$. The term cancellation procedure then continues with $\mathscr{S}_i^k$ $(1 \le k \le 2)$. The whole reduction algorithm is shown in algorithm 1 for a reduction technique $\mathscr{U}$, a ranking procedure $\mathscr{R}$, a numerical integrator $\mathscr{N}$ and a certain level k. Here for a reduction $\kappa \in \mathscr{K}$, κ^{-1} undoes the reduction.

2.4 Symbolic Reduction for Realtime Purposes

In this contribution the algorithm described above is extended in order to obtain models, which can be used for realtime simulation on a given realtime target within a given realtime cycle. To simulate a model in realtime it must be guaranteed that one integration step can be computed within a realtime cycle, i.e., the worst case run time for one integration step has to be smaller than the realtime cycle. Hence, two quantities are important. First the maximal number of required floating point operations (FLOPs) for one integration step and second the number of FLOPs, which can be computed on the realtime target in one second (FLOP/s). The number of required FLOPs depends on **F** and the integration method used. Clearly, for realtime purposes a fixed step solver has to be chosen. Then the maximal number of required FLOPs σ_{req} (using a BDF method) can be expressed as

$$\sigma_{req} = n_{iter}^{BDF} \left(n_{eval} (\sigma_F + \sigma_J + \sigma_{dJ}) + \sigma_{LSE} \right) + \sigma_{event}. \tag{12}$$

Here, n_{iter}^{BDF} denotes the maximal number of Newton iterations during one integration step, n_{eval} denotes the number of required function and jacobian evaluations (depending on the order of the method), σ_F denotes the required number of FLOPs for one evaluation of **F**, σ_J denotes the required number FLOPs for one evaluation of $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$, σ_{dJ} denotes the required number FLOPs for one evaluation of $\frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}$ and σ_{LSE} is the number of required FLOPs for the solution of the emerging system of linear equations within every Newton iteration. Furthermore, σ_{Event} denotes the maximal number of required FLOPs for the event-handling (finding new consistent initial values), which has to be considered since an estimate for the worst case runtime is demanded. One common approach to calculate new consistent initial values is the “event iteration” [7]. Having a DAE system with n_{event} zero functions at hand, the maximal number of required FLOPs for σ_{event} then reads

$$\sigma_{event} = 2^{n_{event}} \cdot n_{iter}^{event} \cdot (\sigma_F + \sigma_{J_{event}}), \tag{13}$$

where n_{iter}^{event} denotes the maximal number of Newton iterations during the event-handling and $\sigma_{J_{event}}$ denotes the number of required FLOPs for one evaluation of the jacobian of **F** with respect to the unknowns during the event-handling. The required FLOPs for table lookup are included in σ_F, σ_J, σ_{dJ} and σ_{event}. With this knowledge the maximal number of FLOPs for one integration step can be computed, while the number of FLOP/s of the realtime target can be easily measured. Since no longer an error bound, but an upper bound for the number of FLOPs for one integration step is given, the term cancellation procedure has to be modified. In the modified term cancellation procedure no simulations are performed. After the ranking the reductions are performed as long as the maximal number of required FLOPs is greater than the upper bound for the FLOPs. Hence, this time no clustering is used, since no verification-simulations are performed and thus clustering would be quite inefficient. Clearly, here a very accurate ranking procedure is demanded, otherwise reductions with a small estimated error leading to a high error could be performed. In this contribution the one-step ranking is simply extended to a three-step ranking, which means that three Newton iterations are allowed. Moreover, the computed ranking value is divided by the number of required FLOPs for one evaluation of the term under consideration. Thus, among reductions with a similar ranking value, those which need many FLOPs are favored.

As can be seen in Eq.12 the dimension of **F** has big influence on the number of required FLOPs for one integration step, since the complexity for solving a system of linear equations of dimension k is of order $o(k^3)$. Hence, after each reduction it is checked whether **F** got decoupled. More precisely, it is checked whether the DAE system may be written as

$$\begin{bmatrix} \mathbf{F}_1(\mathbf{x}_1, \dot{\mathbf{x}}_1, t) \\ \mathbf{F}_2(\mathbf{x}_2, \dot{\mathbf{x}}_2, t) \end{bmatrix} = \mathbf{0}, \tag{14}$$

where $\mathbf{y}_{out}$ only depends on $\mathbf{x}_1$. In this case $\mathbf{F}_2$ and $\mathbf{x}_2$ can be canceled out of the DAE system.

3 Modeling of the Skid-Steer Loader

The skid-steer loader (Figure 2) is a small high maneuverable vehicle that is usually used in locations where maneuverability and turning space are severely restricted. The high degree of maneuverability is due to their method of steering which is so-called skid steering. They are typically four-wheel drive vehicles with the left-side drive wheels independent of the right-side drive wheels. By having each side independent of the other, wheel speed and direction of rotation of the wheels determine the direction the loader will turn. The drive system on the skid-steer loaders has no mechanical transmission. Instead it uses a combination of hydraulic pumps and motors, the hydrostatic drive system, to drive the wheels as well as the working hydraulic mechanisms. It generally comprises a diesel engine having its output shaft coupled to a pair of variable displacement pumps. The output of each pump is connected to the respective hydraulic motor, which operates independent chain transmissions and drives on the vehicle. From the modeling point of view, the skid-steer loader comprises mainly the following parts: hydraulic control unit, diesel engine, hydrostatic drive system, working hydraulic mechanisms, tire-road contact and chassis. Due to simplicity there are some limitations concerning the modeling:

Figure 2: Skid-Steer Loader

- the dynamical effects are only considered in the longitudinal direction.
- the working hydraulic mechanisms are simplified as a rigid body.

Based on these two limitations, the chassis together with the working hydraulics is modeled as a sliding mass. All the other parts are introduced in the following.

3.1 Hydrostatic Drive System

The hydrostatic drive system is constructed using a variable displacement pump to drive a constant displacement motor. In this closed circuit, a charge pump is needed to replenish fluids lost and to provide a minimum pressure in the return line. A low-pressure relief valve is used to control the discharged pressure. There are two more relief valves to limit the pressure in the high pressure line. Furthermore, a pair of check valves are used to restrict the flow direction. In order to model the hydrostatic drive system, a simple *hydraulic library* was built in MathModelica. After modeling all the necessary components, the hydrostatic drive system can be easily obtained. Due to space limitations details are neglected here.

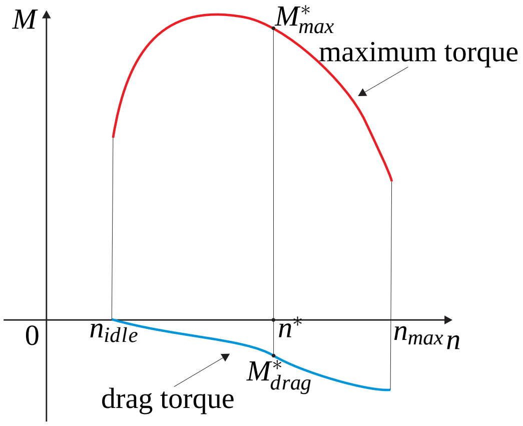


Figure 3: Characteristic Curves of the Engine

3.2 Hydraulic Control Unit

The skid-steer loader is controlled by two joysticks and one foot throttle. The left-hand joystick controls the speed and direction, and the right-hand joystick controls the loader arm. Furthermore, the input signal of the foot throttle can effect the transformation of the input of the two joysticks. The signal itself is only sampled by the trigger in the control unit. In the MathModelica model the left-hand joystick is modeled as two paralleled signal sources namely the driving and steering signal. The right-hand joystick is not necessary to model since the working hydraulic mechanisms are considered as a rigid body. Hence the input signals of the hydraulic control unit are driving and steering signals as well as throttle. The output signals of the control unit control the swivel angles of hydraulic variable pumps in the hydrostatic drive system and the sampled throttle signal to drive the diesel engine. The transformation behavior of the control unit can be described by three characteristic curves. Two curves characterize the relations between swivel angle and the steering and driving signal respectively. Another curve illustrates the effects of driving signals on the steering signals. All these three curves are identified by measurement. In addition, there are some limiters in the control unit to limit the output signals. For example, an amplitude limiter is used to restrict the swivel angle and a rate limiter is used to limit the driving maneuver.

3.3 Engine

The engine used in the skid-steer loader is a diesel engine. It serves to drive the two hydrostatic drive systems. The input of this diesel engine is the foot throttle, which can be normalized in the interval $[0,1]$.

Since there are no different pedal levels, the foot throttle is proportional to the rotational speed of the engine. The idle rotational speed which is the speed when the throttle is 0 is 1000 rpm and the maximum rotational speed which is the speed when the throttle is 1 is 3200 rpm. The relationship between rotational speed and the generated driving torque can be described by two characteristic curves, namely, the maximum and drag torque with respect to the rotational speed. Figure 3 shows the two characteristic curves. The driving torque M^* when the rotational speed is n^* is calculated by

$$M^*(n^*) = M^*_{max} - M^*_{drag} \tag{15}$$

The dynamical behavior of the engine is also approximated by a PT$_2$ system. The speed control is realized by a PID-controller.

3.4 Tire

For the reason of skid steering which causes the high dynamical effects in the lateral direction, the normal tire model of a skid-steer loader can be very complex. As only the longitudinal dynamics is considered here, a simplified one dimensional tire model is sufficient to describe these effects. Figure 4 shows the free body diagram. All the necessary velocities, forces and moments are depicted. The circumferential velocity of the wheel is

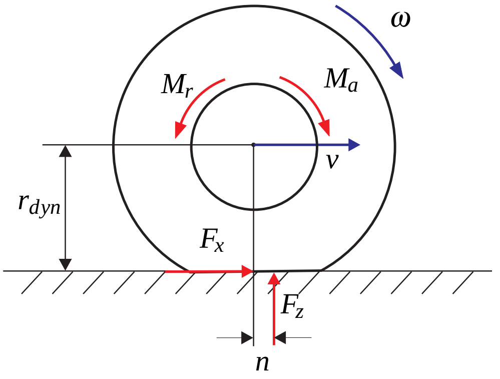


Figure 4: Forces and Moments on the Tire

$$v = \omega \cdot r_{dyn} \tag{16}$$

where ω is the angular velocity of the wheel and r_{dyn} is the effective rolling radius. The rotational motion the wheel can be described by

$$J\,\dot{\omega} = M_a - F_x\,r_{dyn} - F_z\,n \tag{17}$$

The wheels are driven by the driving torque M_a. The distribution of the tire load is normally not unit in the contact patch. Thus the supporting force F_z acted not in the middle and generated a rolling resistance torque M_r. The distance from the acting point to the middle is called pneumatic trail n.

$$M_r = F_z\,n \tag{18}$$

The longitudinal force F_x is calculated with the longitudinal slip s_x. The longitudinal slip is defined by

$$s_x = \frac{v_P}{\omega \cdot r} = \frac{\omega \cdot r - v}{\omega \cdot r} \tag{19}$$

There exist already some tire models describing the mathematical function between these two variables. For example, the magic formula tire model with a pure mathematical description based on the experiment results [1], and the physical HSRI tire model with lower computational efforts. In this work the static HSRI tire model was used. The longitudinal force F_x is described in the following equation.

$$F_x = \begin{cases} \dfrac{C_x\,s_x}{1-s_x} & s_R \le 0.5 \\ \dfrac{C_x\,s_x}{1-s_x} \cdot \dfrac{s_R - 0.5}{s_R^2} & s_R > 0.5 \end{cases} \tag{20}$$

The longitudinal stiffness C_x is a parameter depending on the properties of the tire. It is defined as the linearization of the force-slip relation at $s_x = 0$ and $\alpha = 0$.

$$C_x = \left.\frac{\partial F}{\partial s_x}\right|_{s_x \to 0} \tag{21}$$

The variable s_R is an indicator to identify the linear or non-linear tire behavior, which can be calculated by

$$s_R = \frac{\sqrt{(C_x\,s_x)^2 + (C_\alpha\,\alpha)^2}}{\mu\,F_z(1-|s_x|)} \tag{22}$$

The friction factor μ is defined as

$$\mu = \mu_0(1 - A_s\,v_x\sqrt{s_x^2 + (\tan\alpha)^2}) \tag{23}$$

where, A_s is the adhesion reduction factor, which gives $A_s = 0.011 s/m$ for adhesion coefficients $\mu_0 \in [0.53, 1.05]$. μ_0 can be estimated for different road surfaces. In this section, the introduction of the HSRI tire model enhanced on the mathematical equations. For a more detailed and physical description see [3].

3.5 Driver

The driver modeled here is simply a source of input signals. The output signals from the driver are exactly the same as the inputs of the control unit, namely, steering, driving and throttle signals. Some standard maneuvers were included in this model, such as, the ramp, step and start-stop driving maneuvers.

3.6 Air Resistance

Air resistance describes the influence of the environment. A drag force can be generated by the wind. The equation is as follow.

$$F_{drag} = c_w A \rho \left(v_x - v_{wind}\right)^2 \tag{24}$$

3.7 Overall System

The overall system of the skid-steer loader is obtained by coupling all these sub-systems: drive, hydraulic control unit, engine, hydrostatic drive system, air resistance and the mechanical parts. The acausality of the Modelica language enables the comfort connections between the sub-systems. Figure 5 shows the object diagram of the skid-steer loader in MathModelica.

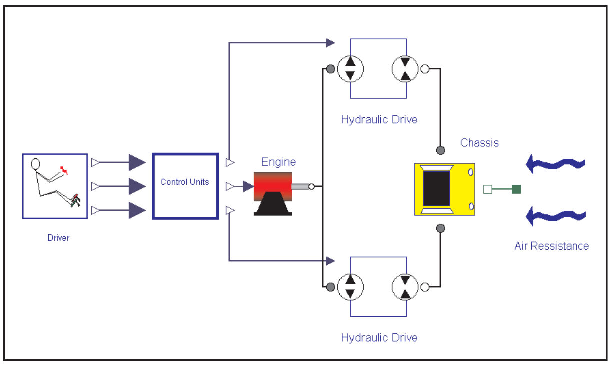


Figure 5: Object Diagram of the Skid-Steer Loader

4 Simulation Results

The symbolic reduction algorithms are implemented in Matlab using the the Maple Toolbox for Matlab. The DAE-system of the previous described model was imported via a MathML interface. Using the Mathematica interface of MathModelica the flat model can be exported to Mathematica and then translated into MathML. The emerging DAE systems are solved using a fixed step BDF method of second order.
In this section, the results for two reductions are given. First the model of the skid-steer loader is reduced using an error bound as stopping criteria. Second the same model is reduced by the extended algorithm for realtime purposes using a maximum number of FLOPs as stopping criteria. Both reductions are performed under a standard start-stop-start-stop straight driving maneuver. Moreover, the longitudinal acceleration is chosen as output variable.

4.1 Reduction with Error Bound

	Original	Error Bound
Number of Equations	69	48
Maximum FLOPs per step	3.42×10^6	1.19×10^6
Maximum absolute error	$\cdots$	0.2851
Maximum relative error	$\cdots$	7.58%

Table 1: Comparison of Original Model and Reduced Model with Error Bound

As presented in Section 2, an error bound for the output variable must be provided for the reduction. Here an error bound of $0.3\,m/s^2$ is set for the longitudinal acceleration a_x. The ranking is computed using the one-step ranking procedure. Negligence of terms is the only reduction technique chosen in this example. Figure 6 shows the nearly overlaying curves of the longitudinal acceleration a_x of the original and the reduced model from the reduced model. It can be seen that the maximum error of $0.2851\,m/s^2$ occurs at the acceleration peaks, where the longitudinal acceleration of the reduced model is slightly higher than the acceleration of the original model. The reduction of complexity of the DAE-systems can be seen in Table 4.1. The number of equations is reduced from 69 to 48, corresponding to a reduction of approximately 30%. Moreover, the maximum required FLOPs for one integration step is reduced by approximately 65%. Thus, the computation time is accelerated by a factor

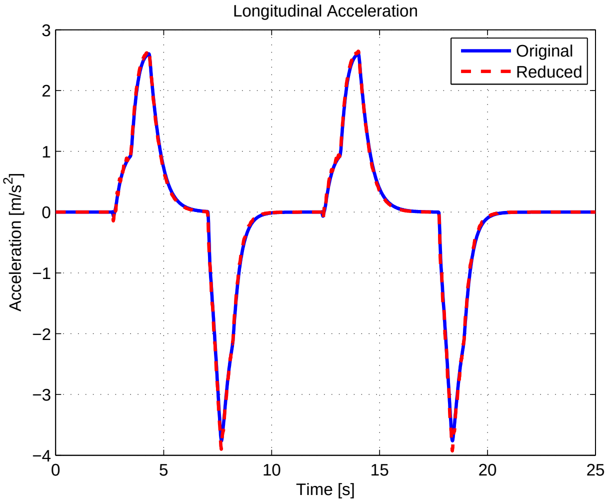
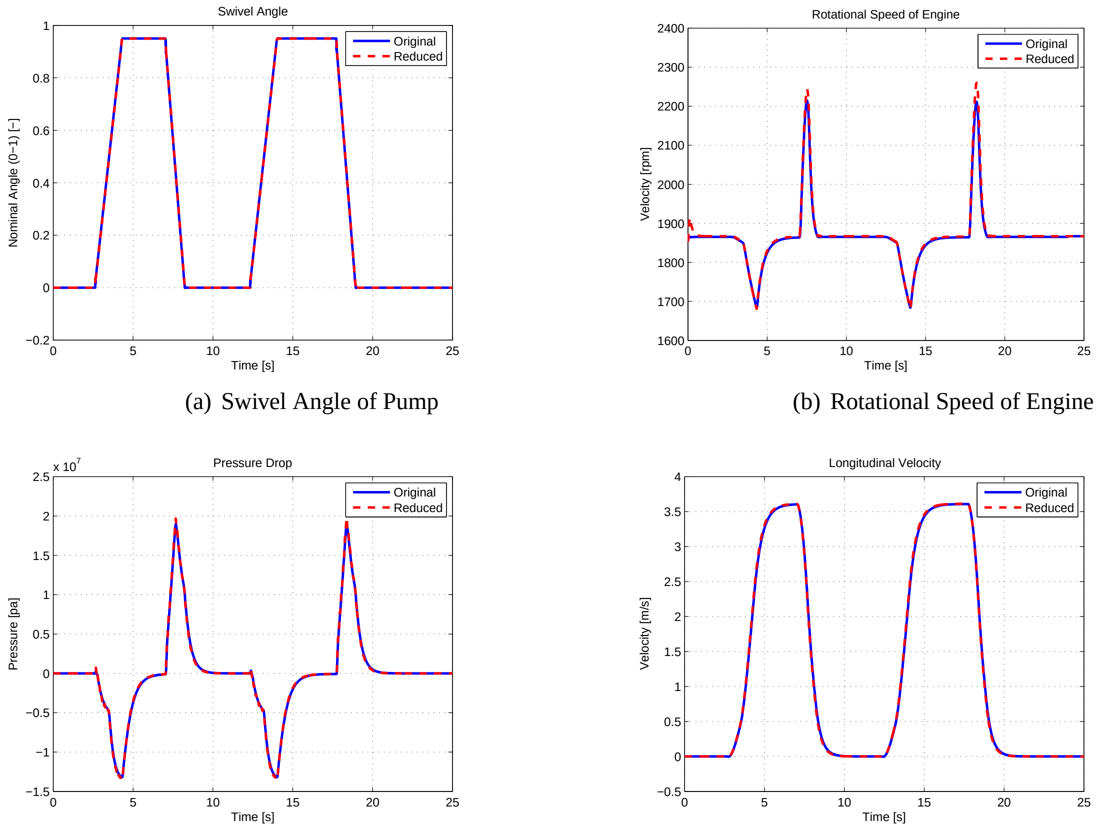


Figure 6: Simulation Results of Output Variable in Reduction with Error Bound



(a) Swivel Angle of Pump

(b) Rotational Speed of Engine

(c) Pressure Drop

(d) Longitudinal Velocity

Figure 7: Simulation Results of System Variables in Reduction with Error Bound

of three.

According to the reduction algorithms described before, only the error of the output variable is considered during the reduction. In Figure 7, some other system variables are plotted. The simulation results of the reduced model of those variables are also very close to the original model. That implies that not only the longitudinal acceleration but also another important dynamical effects are conserved during the reduction.

4.2 Realtime Reduction

In the previous section the original model was reduced until a given error bound was (nearly) reached. Thus, simulating the reduced model will require less time than simulating the original model. In practice models often have to run in realtime environments. For such applications the previously obtained model is more or less worthless, since no worst case runtime for one integration step is known. In this section the original model is reduced in order to obtain a model, which can be simulated in realtime on a given realtime target in a given realtime cycle. Hence, instead of providing an error bound, a realtime target as well as a realtime cycle is provided for the realtime reduction. The realtime target is assumed to be able to perform 1×10^9 FLOP/s, which corresponds roughly to Pentium III. The realtime cycle is chosen as $2\,ms$. Thus, 2×10^6 are the maximal available FLOPs for one integration step. The results from the reduced model for

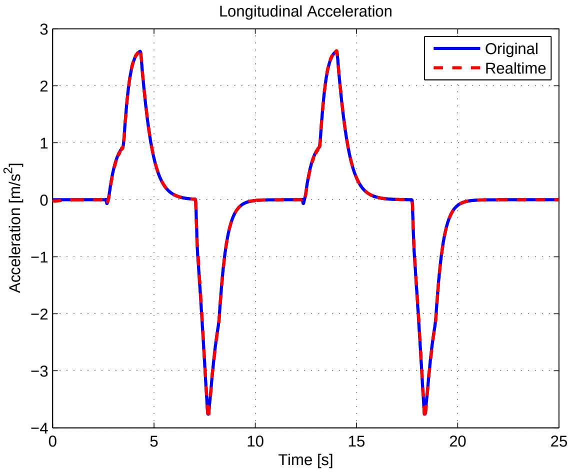
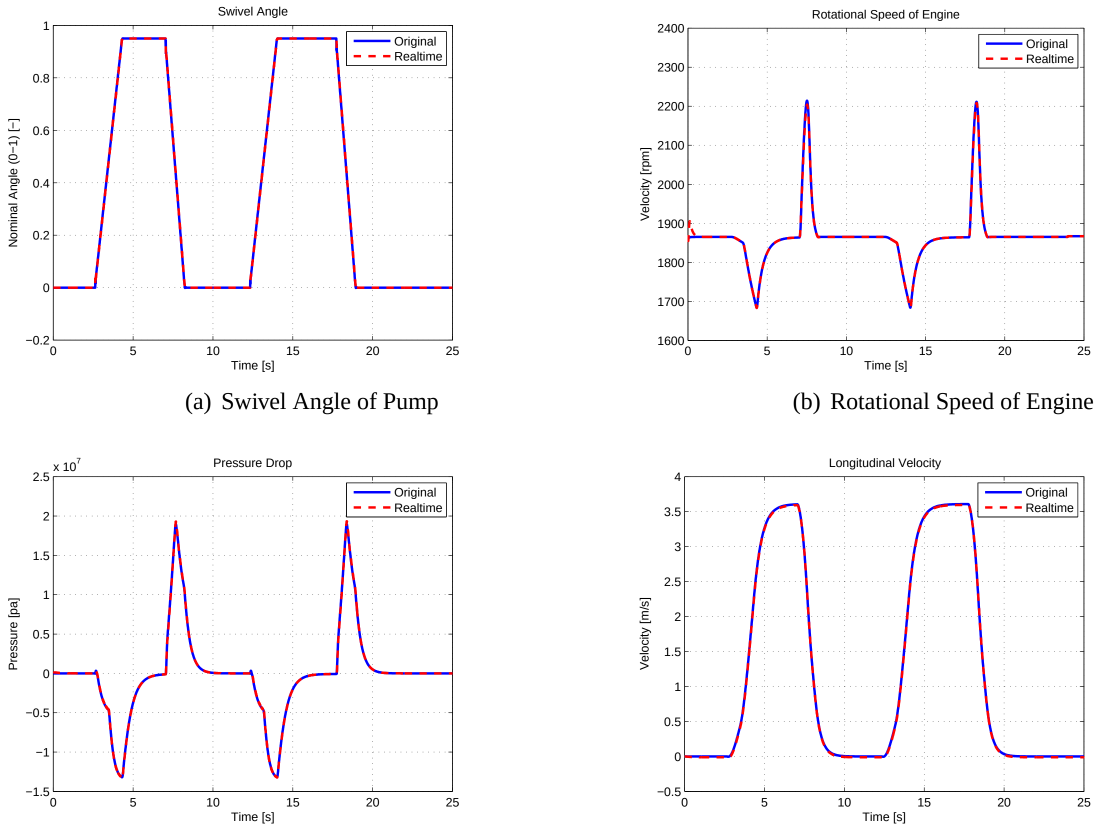


Figure 8: Comparison of Realtime Reduction

realtime purpose are shown shown by the almost iden-



(a) Swivel Angle of Pump

(b) Rotational Speed of Engine

(c) Pressure Drop

(d) Longitudinal Velocity

Figure 9: Simulation Results of System Variables in Reduction for Realtime

tical curves in 8. Nevertheless they are quite different from the previous reduced model. Again the maximal absolute error of $0.1111\,m/s^2$ occurs at the the acceleration peaks, but this time the longitudinal acceleration of the reduced model is slightly lower than the acceleration of the original model. The FLOPs for one integration step is reduced by a factor of 1.8 to 1.92×10^6. Noteworthy, reducing the original model by a factor of 3 is possible without significant loss of accuracy as can be seen in the previous section. Figure 9 shows other relevant system variables. It can be observed that the reduced model is in very good agreement with the original model for all shown system variables. Therefore, again the relevant physical effects are conserved.

5 Conclusion and Outlook

In this contribution a reduction algorithm is extended in order to generate models for realtime purposes. While up to now an error bound was used as stopping criteria, the extended algorithm uses a maximum number of flops for one integration step as stopping criteria. Furthermore, in this contribution the new approach is applied to the model of a construction machine. The generated model is in quite good agreement with the original model at a computational effort, which is considerably lower. In this contribution only the longitudinal dynamics is considered. In the near future the model will be extended by lateral and vertical motion. Moreover, the implementation of the generated models on a realtime target is part of current work.

The presented reduction method takes into account only one scenario. This strongly limits the guaranteed validity of the model. In [8] it has been tried to overcome this drawback by using interval arithmétics. Unfortunately, this approach works only for rather simple systems. Therefore, the scenario has to be chosen quite carefully and can thus be a worst case scenario for example. In future works it shall be investigated how one or multiple scenarios can be chosen systematically such that the desired effects remain.

Since a quite accurate ranking is required for the realtime reduction, here the one-step ranking was extended to a three step ranking. Currently, a reliable ranking procedure based on a sensivity analysis is developed. The new ranking procedure is expected to be

	Original	Realtime
Number of Equations	69	57
Maximum FLOPs per step	3.42×10^6	1.92×10^6
Maximum absolute error	$\cdots$	0.1111
Maximum relative error	$\cdots$	2.58%

Table 2: Comparison of Original Model and Reduced Model for Realtime Purpose

more time efficient, since modern solvers like DASPK offer a sensitivity analysis during the integration and hence the ranking can be computed together with the reference solution.

References

[1] E. Bakker, L. Nyborg, and H. Pacejka. Tyre modelling for use in vehicle dynamics studies. In *Society of Automotive Engineers international congress and expo*, volume 23, 1987.

[2] C. Borchers. Symbolic behavioral model generation of nonlinear analog circuits. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, 45(10):1362–1371, 1998.

[3] H. Dugoff, P.S. Fancher, and L. Segel. Tire Perfomance Characteristics Affecting Vehicle Characteristics to Steering and Braking Control Inputs. Technical report, Highway Safety Research Institute, University of Michigan, 1969.

[4] H. Elmqvist, D. Brück, and M. Otter. Dymola-User's Manual. *Dynasim AB, Research Park Ideon, Lund, Sweden*, 1995.

[5] P. Fritzson and V. Engelson. Modelica-a unified object-oriented language for system modeling and simulation. *Lecture Notes in Computer Science*, 1445:67–90, 1998.

[6] P. Fritzson, J. Gunnarsson, and M. Jirstrand. MathModelica-an extensible modeling and simulation environment with integrated graphics and literate programming. *Proceedings of the 2nd International Modelica Conference*, pages 18–19, 2002.

[7] H. Lundvall, P. Fritzson, and B. Bachmann. Event Handling in the OpenModelica Compiler and Runtime System. *Linköping University Electronic Press*, 2006.

[8] L. Mikelsons and T. Brandt. Symbolic Model Reduction for Interval-Valued Scenarios. *To appear in Proceedings of the ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2009.

[9] L. Mikelsons, M. Unterreiner, and T. Brandt. Generation of Continuously Adjustable Vehicle Models using Symbolic Reduction Methods. *To appear in ECCOMAS Multibody Dynamics*, 2009.

[10] R. Sommer, T. Halfmann, and J. Broz. Automated behavioral modeling and analytical model-order reduction by application of symbolic circuit analysis for multi-physical systems. *Simulation Modelling Practice and Theory*, 2008.

[11] T. Wichmann. Computer aided generation of approximate DAE systems for symbolic analog circuit design. *Proc. Annual Meeting GAMM*, 2000.

[12] T. Wichmann. Transient Ranking Methods for the Simplification of Nonlinear DAE Systems in Analog Circuit Design. *PAMM*, 2(1):448–449, 2003.

[13] T. Wichmann. Symbolische Reduktionsverfahren für nichtlineare DAE-Systeme. *Berichte aus der Mathematik. Shaker Verlag, Aachen, Germany*, 2004.

Modelica and Dymola for education in vehicle dynamics at KTH

J. Edrén, M. Jonasson, A. Nilsson, A. Rehnberg, F. Svahn, A. S. Trigell
KTH Vehicle Dynamics, SE-100 44 Stockholm, Sweden
edren@kth.se, mjonass2@volvocars.com, andnil@kth.se,
adamrb@kth.se, fsvahn@kth.se, annika@kth.se

Abstract

Dymola and Modelica have been used in research at KTH Vehicle Dynamics since 2000. With the introduction of new templates and standard components, Modelica has become increasingly accessible for researchers and students in the field of vehicle dynamics and therefore a project was initiated to evaluate its usefulness as an educational tool. The general idea is to introduce Dymola in smaller assignments, aiming to familiarise the students with the basic functions of the tool while demonstrating its use through simulation examples. Four exercises were studied: a truck braking system model, a friction disc clutch model, a controllable torque clutch model and a combined driver and vehicle model. The conclusions from this pilot study were that Dymola offers several ways to enhance the quality of vehicle engineering education, mainly due to the ease of modelling and the rich animation functions which make the simulation exercises more interesting.

Keywords: Vehicle Dynamics; Education; Dymola; Modelica.

1 Introduction

Dymola and Modelica have been used at KTH Vehicle Dynamics for research work since 2000, see for example [1]. With the Vehicle Dynamics Library [2] (VDL) available, Modelica has become far more accessible for both researchers and students in the field of vehicle dynamics. Because of this, a project was initiated in order to evaluate the current state of Dymola and Modelica as tools for wider use in education at the division. The work presented in this paper was realized as a part of a PhD course, where one of the tasks was to design dedicated exercises to illustrate fundamentals of vehicle dynamics for students.

The strategy for educational introduction of Dymola is to introduce it in different student assignments during the final year of education. Four such assignments are described. Generally, the exercises focus on a specific vehicle component or function and uses the features of Dymola to expand the modelling to a wider perspective, thus providing a systems view of vehicle elements while also demonstrating the use of model-based design in vehicle engineering. The specific exercises are described in greater detail in the following sections.

2 Simulation of an automobile clutch

In an introductory course in vehicle engineering given at KTH, the students are analysing an automobile friction disc clutch during a vehicle start procedure. The purpose is to study the basic behaviour of the clutch and its effects on the dynamics of the vehicle during vehicle start. The fundamentals of the dynamic system that is analysed are shown in figure 1.

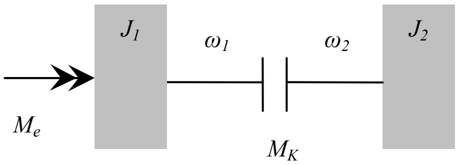


Figure 1: Fundamental driveline model.

Here, J_1 represents the inertia of the car engine, accelerated by the constant engine torque M_e. The inertia J_2 represents the translating mass of the vehicle, transformed here to a rotating inertia. Thus, the rotation of J_2 corresponds to the motion of the car. The clutch itself is described by the constant torque M_K, applied between the inertias. External loads such as aerodynamic or rolling resistance are neglected. In the initial position, J_1 is rotating at a given angular velocity while J_2 is stationary. At start-up, slipping

DOI: 10.3384/ecp09430112

will occur in the clutch until the angular velocity of J_2 matches that of J_1.

The equations of motion of this basic setup are straightforward and can be solved analytically. The angular velocities of J_1 and J_2 during the start procedure can be seen in figure 2.

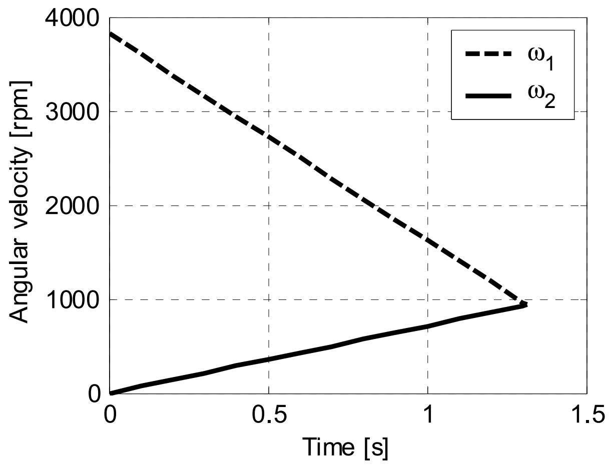


Figure 2: Angular velocities at vehicle start.

The initial, analytical study of the simplified baseline helps to provide a starting point and a basic understanding of the clutch and driveline dynamics. This also establishes good simulation practice as the analytical results serve as a base for comparison with later, more complex simulation models.

After analysing the fundamentals of the clutch and driveline, the students move on to simulating the system using rotational components from the Modelica standard library. This way, a simple but expandable model is built from the ground using physical components. Figure 3 shows the system described in figure 1, as modelled in Dymola.

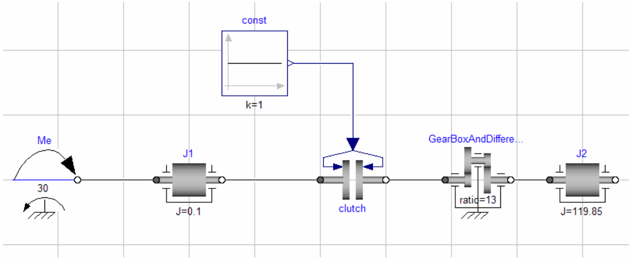


Figure 3: Basic driveline model in Dymola.

With the basic system set up, the analysis can be expanded to cover more complex aspects of the driveline dynamics. Firstly, the clutch model is modified so that the clutch torque is not constant, but ramped from zero to the maximum value in a given time, thus more accurately representing the behaviour of a driver. This makes it possible to study the effects on torque ramping time on the vehicle start, by means of engine rpm and vehicle motion.

Another interesting aspect is the effect of driveline flexibility. Because of the elasticity of the driveshaft, oscillations in the powertrain will occur during vehicle start, leading to a longitudinal vibration in the vehicle known as “shuffle”, considered detrimental to ride comfort. Using Modelica, adding a flexible element to the driveline is straightforward and makes it possible to study shuffle vibrations. This expands the initial analysis to include ride comfort aspects as well as vehicle performance.

In the driveline, nonlinearities will also occur in the form of backlashes in the vehicle transmission. Because of backlash, the vehicle will experience a sharp peak in angular acceleration at start up. This is known as “shunt” and is experienced as a longitudinal jerk by vehicle occupants. As the next step in modelling, a backlash element is added to the model, in series with the gearbox and driveline components. With this added functionality, shunt effects in the driveline can be studied and the effect of backlash parameters on shunt magnitude can be studied. The complete model can be seen in figure 4.

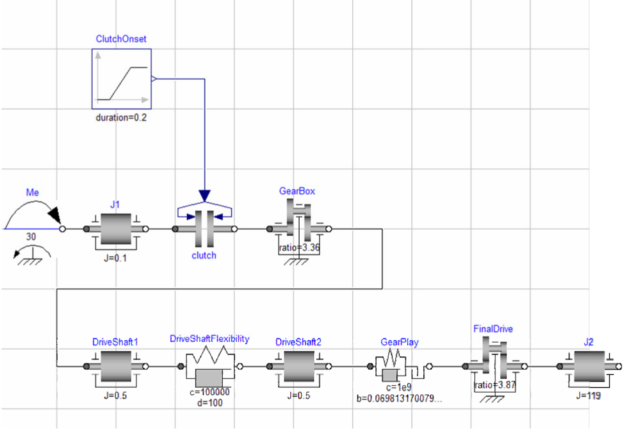


Figure 4: Extended driveline modelling in Dymola.

Figure 5 shows a simulation of clutch engagement with elasticity and backlash included. Compared to figure 2, the ramped onset of clutch torque is seen in the gradual retardation of ω_1 beginning at 0.5 seconds. Also, some ripple is visible in ω_2 at the same instant, caused by the backlash and flexibility.

The driveline shunt and shuffle can be seen more clearly by further examining the angular acceleration of J_2. This is shown in figure 6, where the shunt acceleration and subsequent oscillation can be observed, in this case occurring at the time of clutch lockup.

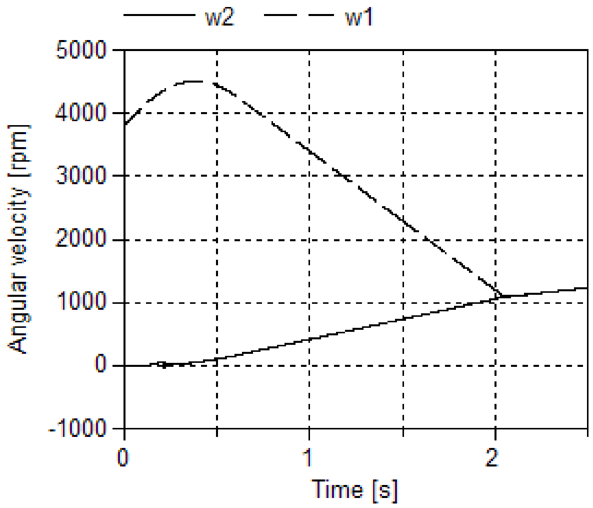


Figure 5: Simulation output from extended model.

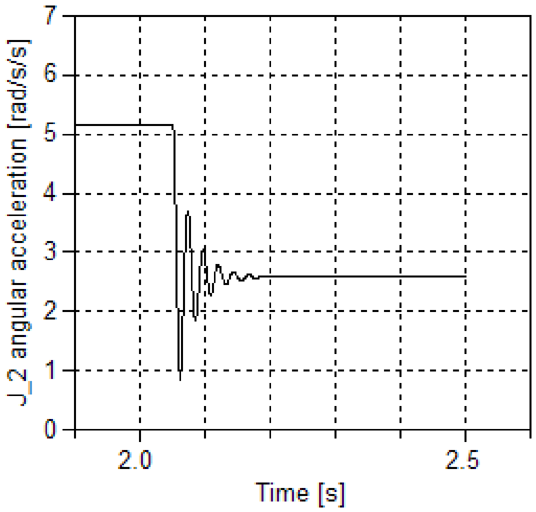


Figure 6: Angular acceleration of J_2.

This example of a student exercise shows one possibility to use Modelica in education of vehicle engineering. Using the Modelica open library, added complexity can be added to the driveline model without the tedium of lengthy derivations. This makes it possible to simulate a more complete model and thereby provide a deeper insight in the actual driveline dynamics and how it affects vehicle behaviour. However, the equations and algorithms of the model remain accessible because of the open source code and straightforward design of the components. Thus, an understanding of the underlying equations can be maintained although the model itself is made more complex.

3 Simulation of a truck braking system

The task in this assignment, given in the basic ground vehicle engineering course at KTH, is to design the braking system for a truck. It has previously been solved analytically with a tool selected by the student, typically Matlab or Excel. Dymola in addition gives the students a virtual testing environment for verification of their result as well as improved understanding.

The goal of the exercise is to design a braking system for a regular 4x2 truck to pass legal requirements. To handle both loaded and un-loaded condition, the brake system will need to have a brake pressure regulating valve. Individual vehicle parameters are set for each student. Solving this assignment is of course possible to do with Matlab alone, but together with Dymola the problem is more easily grasped. The idea is to utilize Dymola for virtual testing to visualize for example why it is unwanted to have the rear wheels to lock up.

The whole simulation experiment is based upon the vehicle dynamics library, and is assembled at the highest level. To set up the model all the parameters needed is propagated to be accessible at this level, as seen in figure 7.

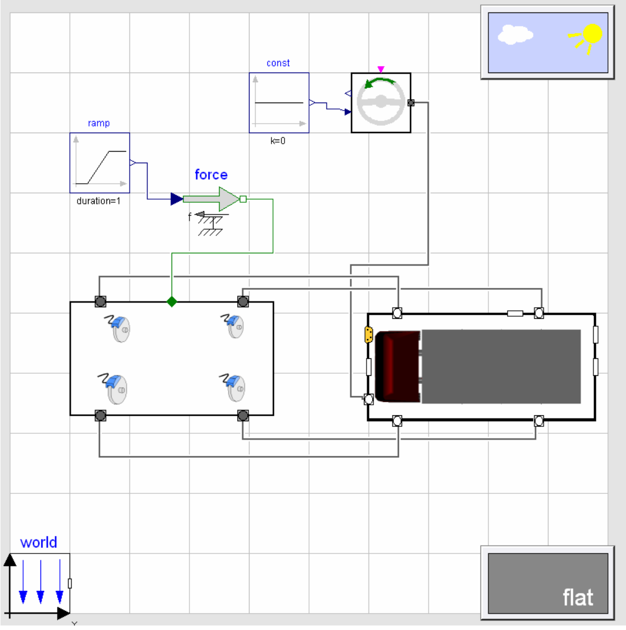


Figure 7: Top level view of the braking experiment.

The Dymola experiment template consists of a truck chassis together with a simple braking system. However using the Dymola model will require some additional calculations, setting up the model to the correct mass and weight distribution. These calculations are fairly basic but essential. The Dymola model as a multi body model is very close to the real world as to estimate the total mass and centre of gravity. The idea is to measure axle weights during standstill and at a constant acceleration or deceleration using the multibody model shown in figure 8. From these measurements the total mass and centre of gravity position can be calculated and the model fine tuned. The brake system is put together with exactly the same simple calculations as given in the main task, using gain-blocks. After this point the student will have a good reference to compare against when designing the brake system. One can also see the complexity and difficulties that follow, when beginning the study of more and more complex models that are closer to reality.

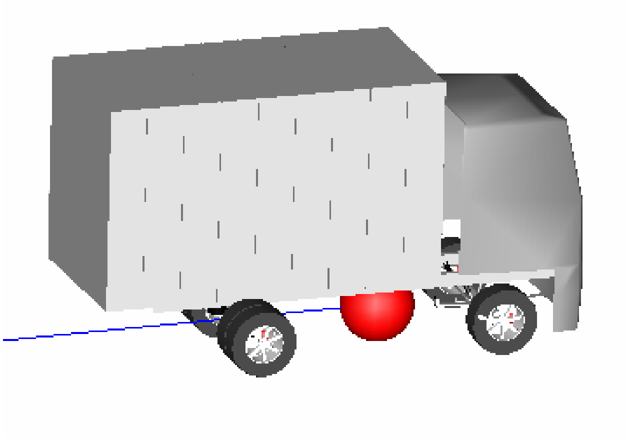


Figure 8: Simulation model used to calculate mass and centre of gravity position.

4 Torque on demand unit

This exercise aims to give an insight in advanced modelling and simulation software and its usage as a powerful tool during the development of a vehicle transmission. The vehicle component selected for the exercise, referred to as a torque-on-demand unit (TOD), is a controllable coupling, which is used to distribute propulsion torque between front and rear axle and simultaneously allow for a difference in angular front and rear axle rotational speeds. Classically, TOD systems have gained a lot of attention due to their ability to improve traction, see fore example [3,4]. This exercise describes the fundamental functions provided by the TOD and put particular attention to energy consumption and vehicle stability. Emphasise is put on simulation rather than modelling.

The underlying didactic strategy behind this TOD exercise is based on three steps; "Explore", "Execute" and "Evaluate". Necessary qualifications required are basic knowledge in vehicle dynamics and Dymola. Additionally, before the students start the exercise, they need to read a short description of the TOD and how it functions.

4.1 Explore

This step aims to provide an understanding of the physics behind the models and students are tempted to open and explore vehicle and subsystem models in Dymola. Emphasis is put on understanding the flow of torque from combustion engine to the hubs. In particular, the students are asked to study the equations for the differentials, which constitute a clear example of that only a few equations are able to describe a relatively complex component. Finally, the students briefly explore the devises modelled for measuring torque, angular speeds and power. This part brings an extra understanding of the physics of the system.

4.2 Execute

This step lets the students execute simulations in existing experiments in Dymola. The experiment is based of an urban driving cycle of 18 minutes. The driver gets instructions to drive through the cycle, which means that the driver is forced to follow a desired steering wheel angle and a vehicle velocity. At a particular global position, the vehicle is exposed to split friction while cornering (see figure 9), which threaten vehicle stability. The simulation is repeated with and without the TOD engaged.

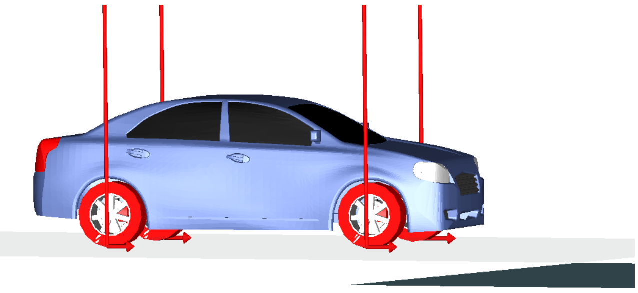


Figure 9: Test drive on split-μ with a controllable clutch.

4.3 Evaluate

The evaluation concerns the energy consumption of the driveline, and in particular, the extra energy consumption caused by friction losses of the four wheel driveline. Power losses are analysed for the individual driveline parts. Finally, an investigation in depth of the split friction event is also made for driveline torque distribution and vehicle behaviour. Since one of the most important TOD control goals is to maintain a relatively similar front and rear axle speeds, those will be checked (see figure 10).

5 Model reduction

This exercise focuses on analysis by means of linearisation and model-reduction. A complex vehicle model, with and without driver in the loop, is linearised and the results are analyzed using Bode diagrams. Additionally, virtual experiments are performed to find parameter values for simplified models. The nonlinear model is modelled in Dymola, using the Vehicle Dynamics Library (VDL), and the frequency analysis is done in Matlab.

The exercise lets the students learn about how to use the linearisation function in Dymola and also performing virtual experiments to find parameter values of a simple planar single track (bicycle) model. The first part consists of linearising a complex vehicle model from VDL and importing the resulting state space form into Matlab for Bode plot analysis. In the second part the VDL model is simu

lated during specific manoeuvres to determine parameter values for the bicycle model. The bicycle model is then implemented in Matlab and a comparison is made with the complex model through Bode plots. The last part concerns linearisation of a driver-vehicle combination in Dymola where some parameters of the driver model are varied and the results analyzed in Matlab, again with the use of frequency response diagrams.

The emphasis of this exercise is analysis of vehicle dynamics and not modelling in Dymola. The students are therefore not required to have extensive knowledge of Dymola beforehand. It will, however, give the students an introduction into some advanced usage of the program. To make the exercise as efficient and equal as possible for the students it is suggested that the teachers prepares the required models in a package in advance, and that a tutorial is produced with detailed guidance through the different steps. The students should at this stage not be hampered by difficulties of finding and utilizing the different features of the program. Questions about the vehicle dynamical aspect of the exercise should be interspersed throughout the tutorial, which the students present to the teacher afterwards.

5.1 Linearisation of vehicle model

The first task is to use Dymola to linearise a complex vehicle model and import it into Matlab in the form of a state space model. Of interest in characterizing the handling properties of a vehicle is typically to study the response of the lateral acceleration and yaw angular velocity (yaw rate) to inputs from the steering wheel. The model is therefore going to be analyzed through a frequency response diagram.

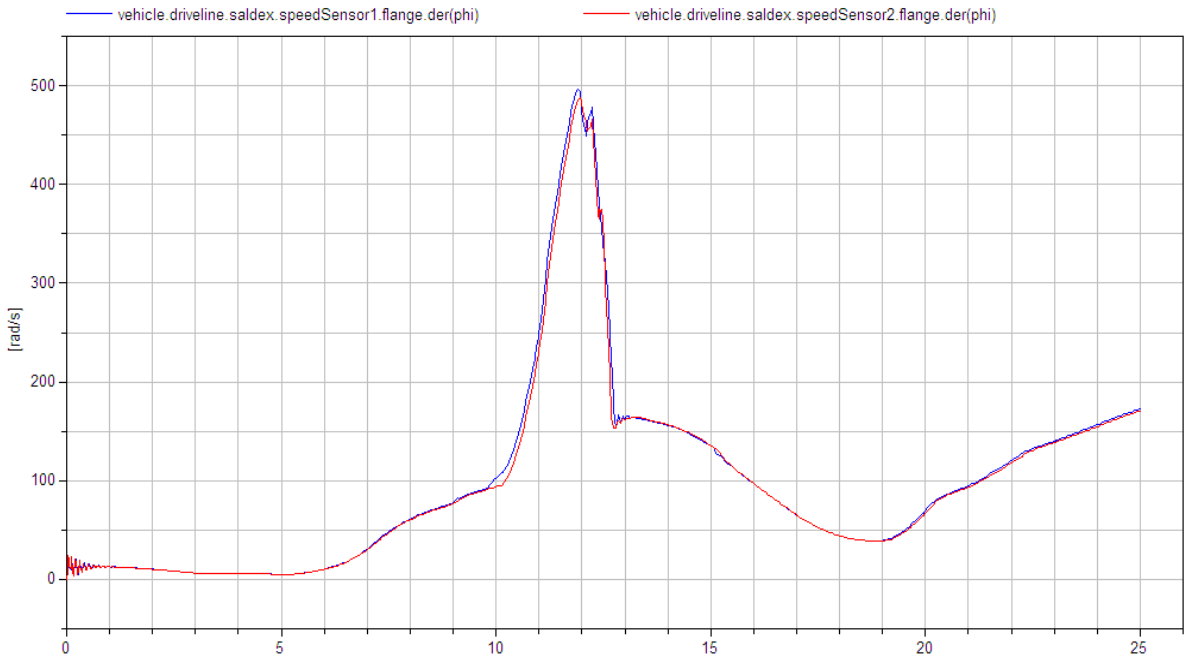


Figure 10: Input and output speeds of the controllable clutch as the vehicle is passing a split-μ section

From the VDL the model *LEK Pacejka02* is chosen. It represents a small compact car and has 134 states in its dynamical description. The kinematics of the suspension is detailed, including variable toe- and camber angles. The tyre model is a Pacejka Magic Formula model which is suitable for handling analysis.

An input and output has to be defined for the vehicle model to enable frequency response analysis. Based on the *OpenLoopDriver* model of the VDL, two version of the model are prepared with steering wheel angle as input and lateral acceleration and yaw rate as output respectively. Figure 11 shows one of the models. As can be seen in the figure a constant term is added to the steering angle input to account for an offset in the neutral position of the steering wheel caused by an asymmetry in the steering system.

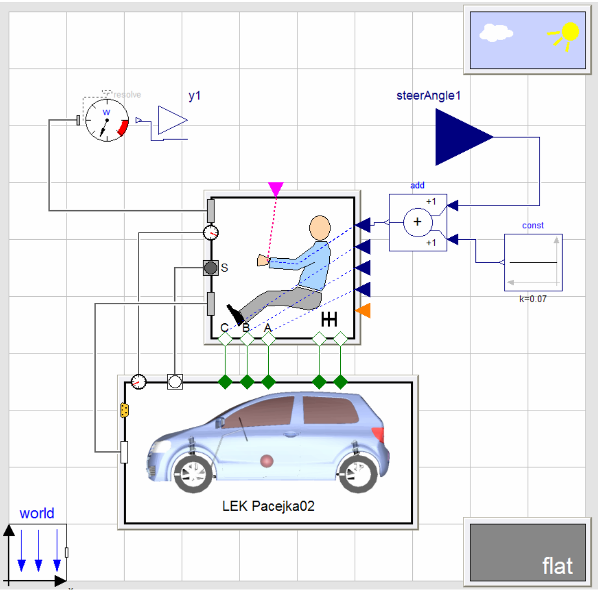


Figure 11: Driver model with defined input/output.

The *Linearize* command in Dymola uses the initial values of a simulation to linearise about. Therefore it is suggested that a short straight ahead driving simulation is performed where after the command *importInitial();* on the command line loads the terminal values of the simulation as initial conditions. Thus linearisation around a steady state condition is assured. The *Linearize* command is then found from the Simulation menu. In this process, a file named *dslin.mat* is generated that contains the linearised model. Use *tloadlin.m* found in the Dymola search path to import the linearised model into Matlab:

```
> [A,B,C,D,xName,uName,yName] =
tloadlin('dslin');
```

State space systems of the imported matrixes are made with the ss-function:

```
> ss_sys=ss(A,B,C,D);
```

If the system has several outputs and only one of them is to be used for the state space system, limit C and D to one of the rows (output parameter index can be identified with *yName*).

After linearisation and creation of state space systems, Bode plots can be generated and used for analysis, e.g. with the Matlab function:

```
> bode(ss_sys).
```

5.2 Determination of parameters

The second task is to find parameter values for a two-degree-of-freedom bicycle model which corresponds to the complex model. The governing differential equations are seen in equation 1,

$$\begin{pmatrix} \dot{v}_y \\ \ddot{\psi} \end{pmatrix} = \begin{pmatrix} -\dfrac{C_{12}+C_{34}}{mv_x} & -v_x + \dfrac{-fC_{12}+bC_{34}}{mv_x} \\ \dfrac{-fC_{12}+bC_{34}}{J_z v_x} & -\dfrac{f^2 C_{12}+b^2 C_{34}}{J_z v_x} \end{pmatrix} \begin{pmatrix} v_y \\ \dot{\psi} \end{pmatrix} + \begin{pmatrix} \dfrac{C_{12}}{mi_s} \\ \dfrac{fC_{12}}{J_z i_s} \end{pmatrix} \delta_R$$

(1),

where v_y is the later velocity, ψ the yaw angle, C_{12} and C_{34} the cornering stiffness front and back, m the total mass of the vehicle, v_x the longitudinal speed, f and b the distance from centre of gravity to front and rear axle respectively, J_z the moment of inertia around the vertical axis of the vehicle, i_s the steering ratio and δ_R the steering wheel angle.

The value of *L, i.e. f+b,* is given to the students. J_z of the sprung mass of the vehicle is also given. However, the students are asked to make back-of-the-envelope calculations of the additional moment of inertia of the unsprung masses. To find the rest of the parameter values a couple of simulations are performed with the vehicle model. First a straight ahead drive is used to find the normal forces on each tyre, from which f and b can be calculated. Then, a steady state cornering simulation is performed with a small steering angle. From the lateral forces and body slip angle the cornering stiffness coefficients can be calculated. The effective steering ratio is measured by making a sinusoidal steering simulation with the ground friction set to a small positive number (zero does not work numerically) and then comparing the steering wheel angle to the wheel angles.

These virtual experiments can principally be done with a real vehicle as well. The measurement equipment needed is four scales, accelerometers and a device to measure body slip angle. The lateral tyre forces can be calculated from the lateral acceleration.

5.3 Analysis of linearised models

Identifying the A, B, C and D matrices of the bicycle model makes the two models ready for a comparative analysis. The frequency response of the two models can be compared for different speeds and the students should note for what frequency range the simple model is a good approximation for the complex model. Figure 12 shows an example of a Bode diagram of the two models with the yaw rate as the output. It can be noted that the bicycle model starts to deviate from the complex model in phase already at 1 rad/s and in magnitude at about 4 rad/s, but that the shape of the curves are quite accurate up to about 70 rad/s. Students should also note that the simple bicycle model is unable to capture the local peak in phase at about 100 rad/s.

5.4 Driver model analysis

The objective of the last exercise is to study effects of changed parameters in a driver model. A linearisation of an existing nonlinear driver-vehicle model is done in order to analyze Bode diagrams of the generated linear model. None of the existing driver model experiments in the current VDL is suitable for analysis without modifications, but the main part of these modifications can be done in advance to shift focus from modelling to analysis. Here a tutorial is suggested to guide the students for the remaining modelling and analysis, since this can improve the understanding of the driver model components with only a small amount of additional effort. A tutorial approach should, compared to an exercise where the student has to solve the modification problem by him or her self, not only be quicker but should also result in identical baseline models for analysis. This gives the students equal opportunities to do analysis with the models.

5.5 Assembling the driver-vehicle model

One of the existing experiments in VDL, the *ClosedLoopDriver*, is used here after it has been modified to enable input of the preview point position and output of (at least) the position of the vehicle. The model is later linearised around zero steering wheel angle, so that the longitudinal distance from the vehicle to the preview point can be assumed to be relatively equal for both global and vehicle coordinates.

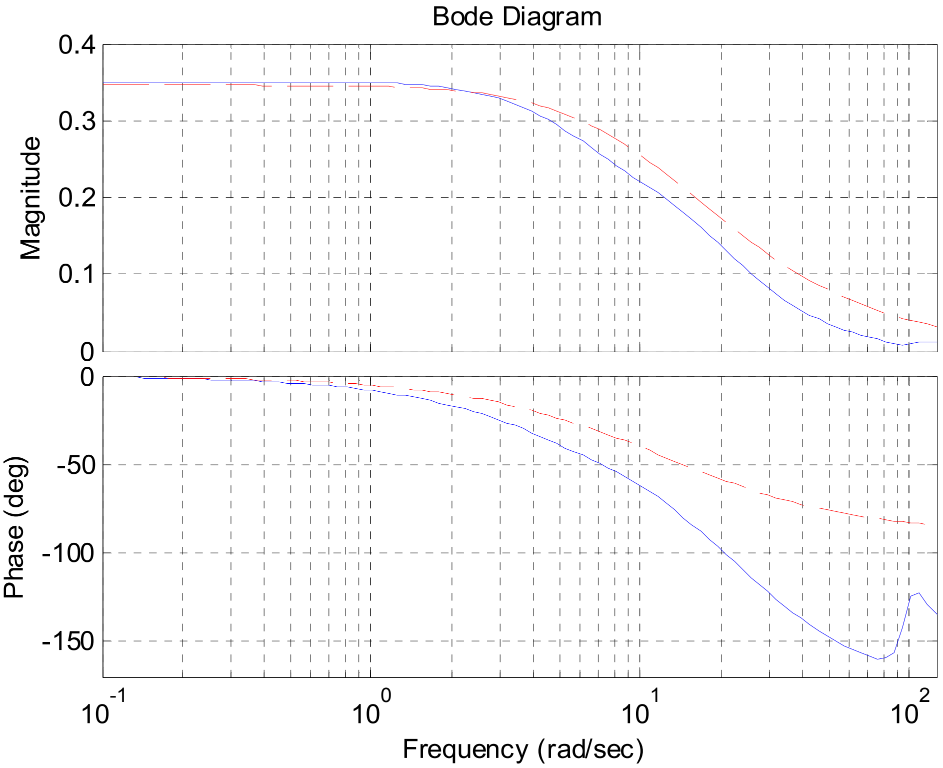


Figure 12: Bode diagram of the complex linear model (solid) and simple bicycle model (dashed). The output is here the yaw rate and input is the steering wheel angle.

Hence we can use only the lateral component of the preview point and vehicle position as input and output respectively, and thus limit the number of inputs and outputs to two. The use of global reference for both input and output enable analysis of how well the driver-vehicle system will follow the path described by the preview point. To simplify the model, the driver in the *ClosedLoopDriver* is replaced with a constant speed driver based on the *ClosedLoopLateral* template. The tracker in the template is modified to allow direct input of lateral position into the steering control calculation since the standard path input taken from spatial time independent coordinates. Direct input of preview point position in vehicle reference frame into the tracker is required for active modification of the preview point position.

At the experiment level, the *LEK Pacejka02* vehicle is selected again. In the chassis, the parameter summary resolve point is changed from the centre of the front axis to the vehicle centre of gravity. It is also necessary to specify an approximation of the steering wheel gear ratio to be used by the driver model, and this can be done in the experiment with given data or experimental results.

For the correct input to the tracker block in the driver model, a new text layer block is added with the coordinate transformation from global reference to the vehicle front axis, which requires adding sensors to the top level of the experiment (see figure 13). Possibility to change the preview time is also added as an input to the transformation block. One input has to be added to the experiment, the lateral position of the preview point. Several outputs can be added, i.e. the lateral position of the vehicle that will be used here for analysis of how the driver-vehicle system responds to movement of the preview point.

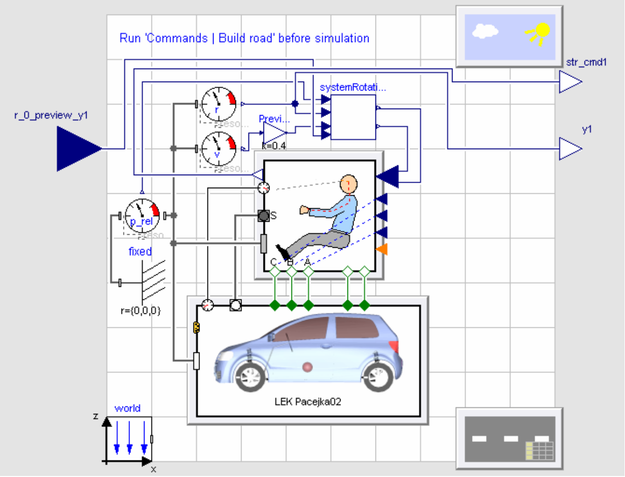


Figure 13: Top level view of driver-vehicle experiment.

5.6 Analysis of linearised model

With the model modified it is now possible to export the model for analysis. The system can now be set to different states in the experiments, e.g.:

- In the vehicle, set the velocity to a chosen value to use for the linearisation.
- In the top level of the experiment, change the preview time to analyze the effect of different path tracking strategies.
- Change the steering wheel gear ratio in the driver model to analyze effects of driver interpretation of the vehicle characteristics.

The model can be linearised and exported after the experiment simulation has been executed in the same way as described in Section 2.1. The students should analyze the default setup of the driver model first, e.g. with accurate information about the vehicle at a selected speed, and then try different values to study the effects of the parameters. Bode plots of three different selected preview times are presented in figure 14. The resonance peak is seen to grow for decreasing preview time, which would indicate that the system becomes more nervous.

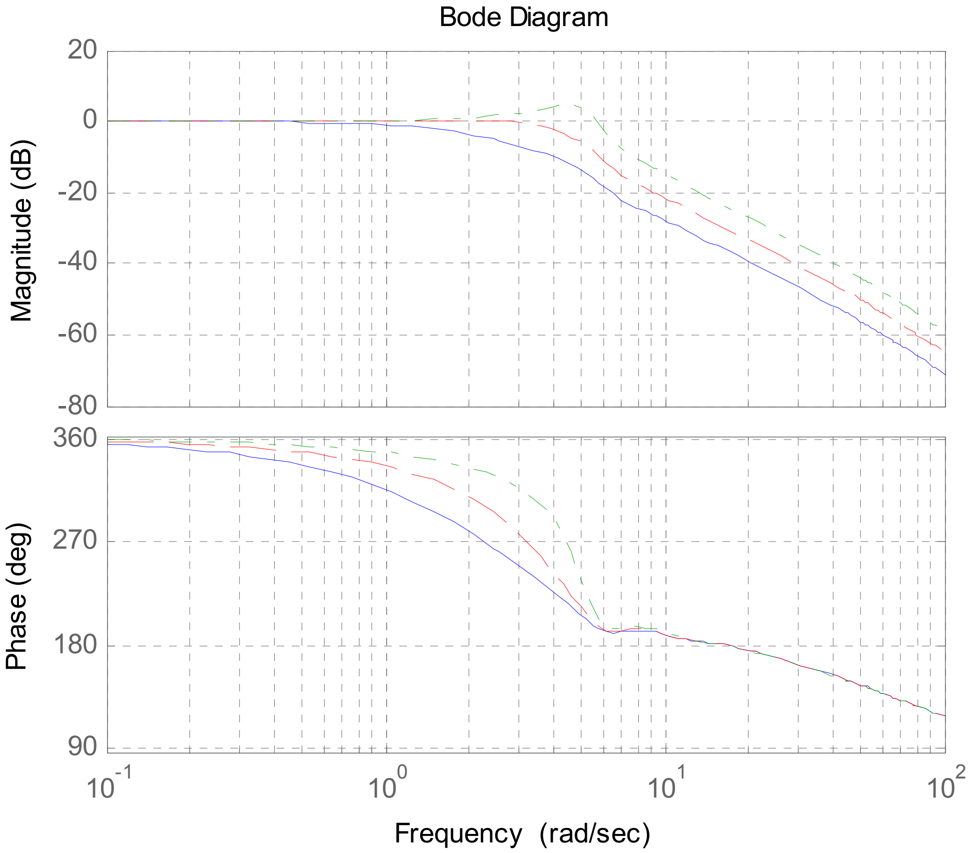


Figure 14: Bode diagram of linearised model, using steering wheel gear ratio 19, speed 30 m/s, preview time 0.8 (solid), 0.4 (dashed) and 0.2 s (dash-dotted).

6 Conclusions

The evaluation during the course of this project has shown that Dymola and VDL offer many ways to enhance the learning of vehicle engineering. The physical based modelling makes it easy to replace or add components, while the hierarchical structure and icon based model layout give the user a clear overview of the different subsystems and their interconnections. Animations also help to visualise results and clearify physical behaviourThe VDL also contains example models which can be used to perform virtual experiments as a substitute for real life experiments that might be too expensive or time consuming for a student course. Modifying the example models and experiments is also easy, making it straightforward to design student exercises showing various aspect of vehicle dynamics.

The problems with Dymola include the somewhat crude tools for analysis, sometimes making it necessary to export models or simulation results to other tools such as Matlab. Also, the documentation can be found inadequate when confronted with certain problems such as the linearisation process.

References

[1] Andreasson J., Möller A., Otter M.: Modeling of a Racing Car with Modelicas Multibody Library, in Proceedings of the Modelica Workshop, 2000.

[2] Andreasson, J., Gäfvert M.: The Vehicle Dynamics Library – Overview and Applications, in Proceedings of the 5th international Modelica Conference, 2005.

[3] Mäki R.: Wet Clutch Tribology - Friction Characteristics in Limited Slip Differentials, doctorial thesis, Luleå University of Technology, Sweden, 2005.

[4] Piyabongkarn D., Lew J.Y., Rajamani R., Grogg J.A. and Yuan Q.: On the use of Torque-biasing Systems for Electronic Stability Control, IEEE transaction on control systems technology, Vol. 15, No. 3, pp. 581-589, 2007.

On the link between Architectural Description Models and Modelica Analyses Models

Damien Chapon Guillaume Bouchez
Airbus France
316 Route de Bayonne 31060 Toulouse
{damien.chapon,guillaume.bouchez}@airbus.com

Abstract

When designing complex systems such as Aircraft, harmonizing the way we describe and analyze systems physical architectures is important in order to reduce costs, lead-time, and to increase systems maturity at entry into service. As Modelica has interesting multi-domain modeling capabilities, we define a harmonization approach that is based on the use of Modelica in an Integrated Development Environment.

Keywords: Modelica; UML; Domain specific language; Topcased; Physical Architecture;

1 Introduction

Airbus is a designer and integrator of systems. Airbus imagines the system' concepts, designs, and specifies the system so that suppliers can manufacture it. Those systems have the granularity levels as those considered within the ATA aeronautical classification of aircraft systems (e.g. ATA 27: flight control system, ATA 24: Electrical power generation system, ATA 32: Landing gear). Following a System Engineering Process (SEP), the stakeholders' needs, requirements, and constraints are transformed into a system architecture solution. For purpose of illustration, a generic SEP, the IEEE 1220 standard [1] is used in this document. The figure 1 depicts the sub processes of the IEEE 1220 SEP and shows how they iterate to produce a consistent set of requirements, functional arrangements, and design solutions. As depicted in this figure, one of the main outputs of this SEP is the verified physical architecture. Indeed during the synthesis and design verification phases, the functional architecture is translated into a physical architecture that provides an arrangement of system elements, their decomposition, interfaces (internal and external), and design constraints, to satisfy stakeholder expectations as defined in the requirements baseline.

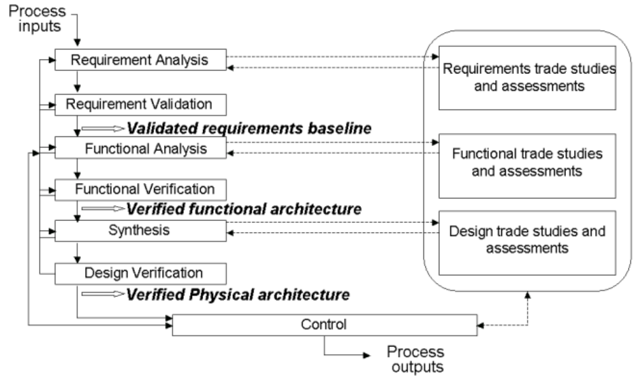

Process inputs
Requirement Analysis
Requirement Validation
Validated requirements baseline
Functional Analysis
Functional Verification
Verified functional architecture
Synthesis
Design Verification
Verified Physical architecture
Control
Process outputs
Requirements trade studies and assessments
Functional trade studies and assessments
Design trade studies and assessments

Figure 1 - IEEE 1220 Systems Engineering Process

Harmonizing the way we define and analyze system physical architectures is obviously a recurrent preoccupation when designing complex and critical systems because this is a mean to reduce costs, lead-time, and to increase systems maturity at entry into service. Modelica [2, 3] is a multi-domain and multi-paradigm modeling language for component-oriented modeling of complex systems that is well suited for physical and multi-domain physical analyses.

In this article we want to define recommendations in order to bring harmonization to the system physical architecture description and analyses process by promoting a model-based approach. Therefore this document is organized as follows. The second section is devoted to a deep explanation of the system physical architecture description and analyses phases in order to understand their particularities and to define recommendations for our model-based approach. In the third part we present some related works within the Modelica community and comment them with regards to our recommendations. Then in the fourth section we present the capabilities that need to be part of an Integrated Development Environment (IDE) in order to support our harmonization initiative. We also introduce Topcased, an integrated open source System/Software engineering toolkit that is a good candidate to support our work. Finally in the fifth section we present an example of a domain specific UML profile that we have created with the meta-modeling capabilities of Topcased.

2 System physical architecture description and analyses process

The figure 2 depicts more precisely the system physical architecture description and analyses process and exhibits the relations between the main products of this process.

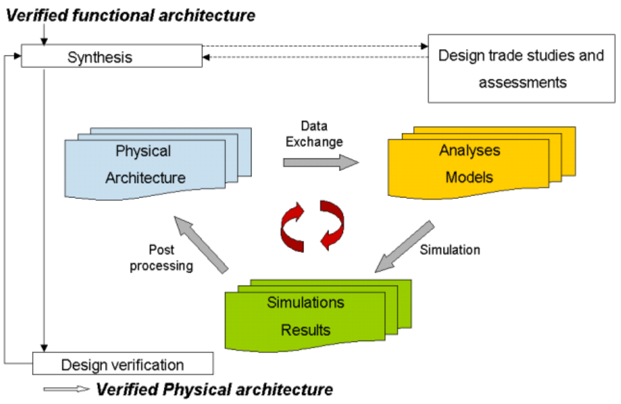


Figure 2 - System Physical Architecture description and analyses process

2.1 Separation between the system physical architecture description and analyses steps

As depicted in the figure 2 the physical architecture provides information (e.g. parameter, topology) to create the analyses models. Then simulations are run on the analyses models to get simulations results and exploit them to draw conclusions about the physical architecture. This process is iterated until the physical architecture satisfies all the requirements captured from the operational perspective (users' needs) and also defined in the functional architecture (e.g. safety, reliability, reuse, maintainability, performance, design constraints), with all the constraints imposed by the physical laws (e.g. aerodynamic forces, thermal dissipation, power consumption.) and with cost reduction constraints. Therefore the link between the architectural description step and the analyses step is very close. However it is better not to mix these two steps into the same modeling language or tools. Indeed their intended goals are not the same. The architectural description step defines the topological arrangement of the system components, how they are connected, and stores data, e.g. parameters, about the systems being designed whereas during the analyses step the information stored in the architecture is used to build analyses models in order to run simulations and to draw conclusions about the system physical architecture. This separation is sometimes very explicit; e.g. these two tasks are sometimes performed by different people or even subcontracted.

2.2 Physical architecture description

In order to identify good practices for the physical architecture description we use the IEEE Std 1471 [4]. This standard provides definitions and a metamodel for the description of architecture. The Figure 3 depicts the part of the metamodel that is relevant for our needs.

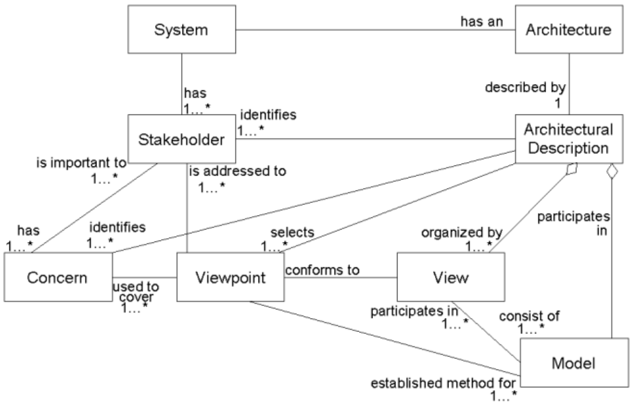


Figure 3 - IEEE 1471 conceptual model for architectural description

This metamodel state that a system architecture is described by an architectural description organized into one or more views and that each of them are conformed to a viewpoint. The central notion of viewpoint is defined as "the conventions by which a view is created, depicted, and analyzed. The viewpoint determines the languages (including notations, model, or product types) to be used to describe the view, and any associated modeling methods or analyses techniques to be applied to these representations of the view. These languages and techniques are used to yield results relevant to the concerns addressed by the viewpoints". This standard also recommends using the considerations and the concerns of the stakeholders as the inputs for the viewpoints selection. Obviously the systems designers are among the system stakeholders and one of their considerations is to have languages for describing the views that are customized for their job. A way of doing so is to encapsulate within the language some part of the domain knowledge. The domain knowledge is the good rules, good practices, and the experiences of the system designers in a technological field, a domain (e.g. a domain specific iconography).

2.3 Modeling languages for describing systems physical architectures

As explained above, modeling languages are needed in order to describe systems physical architectures. Depending on the systems being designed, the stakeholders and their concerns, different alternatives are available:

1. To use existing modeling languages. Among the most popular is UML, [5] (Unified Modeling Language), a standardized general-purpose modeling language in the field of software engineering. UML includes a set of graphical notation techniques to create abstract models of specific systems. Another solution when designing complex system is SysML[6] (Systems Modeling Language), a UML profile for systems engineering.
2. To customize existing modeling languages. When a specific syntax or a more precise semantic is needed for a specific system being designed, a solution is to use the extension mechanism of UML to create a domain specific UML profile. A profile is used to extend the UML metamodel by using three basic mechanisms: stereotypes, tagged values and constraints added in OCL (Object Constraint Language).
3. To create new Domain Specific Languages (DSLs). This solution allows customizing the modeling languages to cope totally with the needed specificity of each viewpoint. Creating a DSL can be worthwhile if the language allows expressing a particular type of problems or solutions more clearly than pre-existing languages would allow, and if the type of problem in question reappears sufficiently often.

However as defined in the IEEE 1471, the stakeholders to whom the architectural description is addressed are responsible for the choice of the modeling languages to be used. Whatever this choice is, what is really important is to have the modeling and meta-modeling framework to create, customize and/or use the chosen architectural modeling languages.

2.4 Languages and tools to support the analyses steps

Different analyses are performed during this process. We can classify these analyses into two categories:

- Trade studies: to evaluate different architectural alternatives against performance, cost, schedule, and risk implications. These analyses include parametric sensitivity analyses, Monte-Carlo analyses or optimizations.
- Verification activities: to ensure the conformity of the design with respects to functional architecture and the higher-level requirements. These analyses include direct simulations, formal proofs or robustness analyses.

In order to support all the analyses performed during the design of all the ATA systems of an aircraft, there exists a huge variety of system modeling languages and tools. We consider Modelica only as one of these modeling languages. We want to use the multi-domain modeling capabilities of Modelica everywhere it is possible in order to bring some harmonization to the wide variety of languages and tools. However let’s remark that Modelica cannot obviously replace all the system modeling languages and tools used when designing complex and critical systems because it cannot support all the different analyses required when designing complex systems.

2.5 Recommendations for harmonizing this process

During the four previous subsections we exhibit some features that allow us to identify recommendations in order to define a model-based approach in an Integrated Development Environment (IDE) for harmonizing the physical architecture description and analyses process. The IDE should include modeling and meta-modeling capabilities to create, customize and/or use the chosen architectural modeling languages. Then, as Modelica can bring some harmonization, the IDE should include the capabilities to use Modelica model, either by including a textual editor, a graphical editor, and a simulator into the IDE or by allowing the use of an external Modelica tool. The IDE should also include the capabilities to use other analyses languages and tools that are relevant for the system being designed, or in other words the IDE should not be only Modelica-centric. Finally the physical architecture description and analyses steps should not be mixed. The IDE should rather provide or allow defining some links (e.g. Model transformation, data exchange) between the physical architecture description step and analyses step.

3 Related Work

Several attempts to integrate or link Modelica with UML or SysML have been launch within the Modelica community:

- ModelicaML by A. Pop and P. Fritzson [7];
- A mapping between SysML and Modelica by T. Johnson and C. Paredis [8];
- UmlH by C. Nytsch-Geusen [9].

These are three very interesting and promising works but from our viewpoint they don't yet satisfy all the needed features that we exhibit in the section 2.5. Moreover the authors of ModelicaML argue in the following paper [10], for a development direction of ModelicaML that creates a small core with well-defined semantics, instead of the current version that is based on an extension of SysML. We agree with the fact that UML is too big and semantically unsound to be used to describe efficiently systems physical architectures. We differ from the vision of ModelicaML because we do not think that an Integrated Whole Product Modeling based on Modelica is the right solution. Modelica should be considered as an analyses tool among others and we propose rather to have an IDE that contains the capabilities to define customized physical architectures languages (e.g. notations, data model, semantics, and domain rules) that could be formally link with analyses activities based on models expressed with Modelica and other analyses tools, in a model-based fashion.

4 Using the model-based capabilities and services of Topcased

Topcased [11] is an integrated open source System/Software engineering toolkit compliant with the requirements of critical and embedded applications. It covers the stages from requirements analysis to implementation, as well as some transversal activities like anomaly management, version control, and requirements traceability. Topcased principles are based on Model Driven Engineering and therefore provide:

- Meta-modeling capabilities to describe all the modeling languages in a common framework;
- A model bus to access easily to the various tools;
- Model transformations capabilities to relate the various models and adapt models to the various tools involved in a project;
- Generative programming capability to easily produce both textual and graphical model editors.

We focus in this article on Topcased because it owns all the model-based capabilities and services that are expected to deploy our harmonization initiative. What is important is the use of these capabilities and services, not the use of Topcased. However Topcased is a good solution because it provides a system-engineering platform that owns expected services for critical embedded systems, like configuration management and traceability.

4.1 Topcased as a Meta-modeling tool

Topcased relies on the Eclipse platform [12]. With regard to the four-layer meta-model architecture of the OMG's Meta-Object Facility [13], the M3 meta-modeling language used in Topcased is Ecore, provided by the EMF [14] (Eclipse Modeling Framework) project. Topcased is therefore strongly model-oriented. Indeed TOPCASED provides model editors, model checkers and model transformations capabilities, but is also itself based on modeling and code generation. With Topcased it is possible to use existing modeling language such as UML or SysML, customize domain specific UML profile, or create new DSL with their graphical editors using the EMF and the Graphical Modeling Framework (GMF)[15]. For illustrating the meta-modeling capabilities of Topcased, an example of a domain specific profile for the description of aircraft on-board power systems architecture is given in the section five of this article.

4.2 Providing a Modelica enabler and system physical analyses capabilities in Topcased

The meta-modeling capabilities can also be applied to provide editors for some of the analyses tools needed to perform trade studies, verification activities or other specific analyses on the systems physical architectures. Providing an enabler to Modelica in Topcased with graphical and textual editors and a Modelica simulator is a way to bring some harmonization to the physical architecture description and analyses process. In this perspective an interesting work has been done for the Modelica Development Tooling (MDT) [16]. MDT is a collection of plug-ins for Eclipse and can therefore be plugged into Topcased. It provides an environment for working with Modelica projects and integrates the OpenModelica compiler to provide support for various features, for example package and class browsing and code completion. However the MDT lacks for the moment a graphical editor for Modelica models.

4.3 Linking the physical architectural models to the analyses models

As defined earlier, we do not recommend mixing the physical architecture description step and the analyses step because these are separated tasks with different objectives. However as depicted in the figure 2 these two tasks are closely linked because the physical architecture provides information (e.g. parameter, topology) to create the analyses models and in return, simulations are run on analyses models during the analyses step to get simulations results and draw conclusions about the physical architecture. This process is iterative and therefore the link between the architectural models and analyses models is crucial. In this perspective Topcased provides a model bus that could be useful to access easily to the various analyses tools. Another potential idea is close to the mapping between SysML and Modelica realized by T. Johnson and C. Paredis. But as we mentioned earlier we don't want to impose in our framework a predefined architectural modeling language such as SysML. So the idea is rather to use the meta-modeling capabilities of Topcased in order to create domain specific UML profile. Then a mapping between these architectural modeling languages and analyses languages (e.g. Modelica) could be realized and finally the model transformation capabilities of Topcased could be used to link the physical architecture description step and the analyses step.

5 Illustrating the meta-modeling capabilities of Topcased - A UML profile for the description of aircraft on-board electrical power systems

In order to illustrate the meta-modeling capabilities of Topcased we have customized a UML profile for the description of aircraft on-board power systems architecture. The role of this profile is to store topological and parametric data that are relevant for the system stakeholders. This domain specific UML profile encapsulates a small part of the needed domain specific knowledge. Please remark that this profile is not intended to be usable as it is. It has only been created in order to illustrate how to customize a domain specific UML profile with the meta-modeling capabilities of Topcased and should be improved greatly to be usable. Our profile extends the UML metamodel by using three basic mechanisms:

- Stereotypes. These are extensions of already existing elements in order to define new types specific to the domain.
- Tagged values. These are extra properties that can be added to UML elements in order to specify additional information.
- Constraints. These enable to specify well-formedness rules and restrictions on model elements.

We define new types for the elements of the aircraft on-board power systems architecture by stereotyping the Class metaclass. Following the same idea we define new types for the topological connections between the elements of the architecture by stereotyping the Association metaclass. The figure 4 depicts the stereotypes used in our profile.

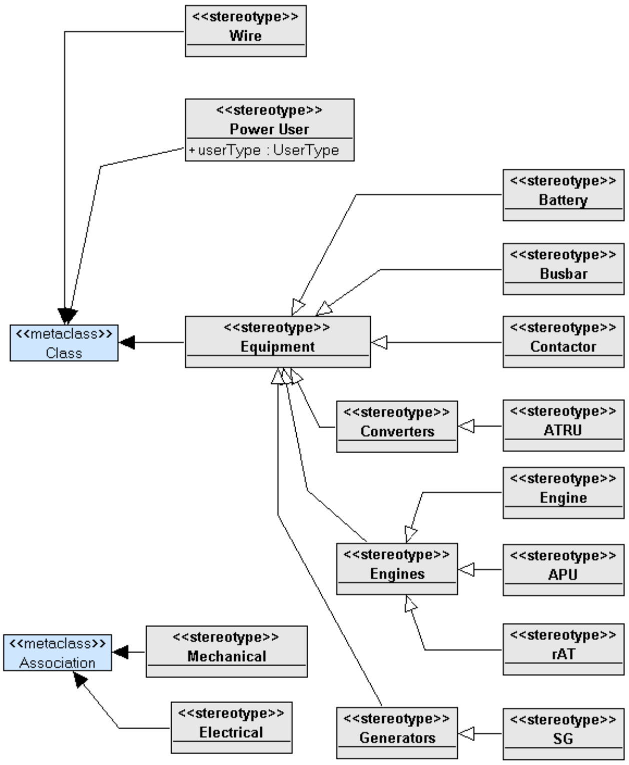


Figure 4 – New stereotype for the Aircraft On Board Electrical Power Systems architecture

Black arrows denote an extension of the targeted metaclass and white arrows denote inheritance between the stereotypes. A tagged value has been added to the Power User in order to add a new property for this new stereotype. Several others could have been added for the other stereotypes and shared by the inheritance mechanism between stereotypes. Then constraint written in OCL can be used to specify well-formedness rules. An example of well-formedness rule could be that you cannot draw an association between two classes of stereotype <<Engines>>. OCL can also be used in various ways to specify the stereotypes more precisely such as constraining property values or specifying dependencies between values of different properties of elements. Another mean in order to customize a domain specific UML profile is to adapt the visual aspects of the different elements with icon and symbol that are familiar and more intuitive for the systems stakeholders.

6 Conclusion

In this document we have focused on the system physical architecture description and analyses process. We have defined recommendations in order to build a model-based approach for harmonizing this process. As Modelica has interesting multi-domain modeling capabilities, this approach is based on the use of Modelica model in an Integrated Development Environment. However this IDE should also include the capabilities to use other analyses languages and tools that are relevant for the system being designed and should not been only Modelica-centric. We recommend also that the IDE include modeling and meta-modeling capabilities to create, customize and/or use architectural modeling languages in a model-based fashion with models transformations or model bus services in order to provide links between the physical architecture description step and the analyses step. As a consequence of these recommendations we have presented Topcased, an integrated open source System/Software engineering toolkit compliant with the requirements of critical and embedded systems. Indeed Topcased owns all the model-based capabilities and services that are expected to deploy our harmonization initiative. Finally we have illustrate the meta-modeling capabilities of Topcased with customized domain specific UML profile for the description of Aircraft On-Board Power Systems architecture.

References

[1] T. Doran. IEEE 1220: for practical systems engineering. IEEE Computer, Vol.39, No. 5, May 2006.

[2] P. Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, 940 pp., Wiley-IEEE Press, 2004.

[3] The Modelica Association. http://www.modelica.org/

[4] R. Hilliard, IEEE-Std-1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE. 2000.

[5] OMG. UML. http://www.uml.org/

[6] OMG. SysML. http://www.sysml.org/

[7] A. Pop, D. Akhlevidiani, and P. Fritzson. Towards Unified System Modeling with the ModelicaML UML Profile. EOOLT'2007. July 2007.

[8] T. Johnson, C.J.J. Paredis, R. Burkhart, Integrating Models and Simulations of Continuous Dynamics into SysML. Modelica 2008.

[9] Christoph Nytsch-Geusen. The use of the UML within the modelling process of Modelica-models. EOOLT.

[10] J. Guy Süß, P. Fritzson, A. Pop. The Impreciseness of UML and Implications for ModelicaML. EOOLT

[11] P. Farail, P. Gaufillet, A. Canals, C. Le Camus, D. Sciamma, P. Michel, X. Crégut, M. Pantel. The TOPCASED project: a Toolkit in Open source for Critical Aeronautic SystEms Design. ERTS 2006.

[12] Eclipse. http://www.eclipse.org/

[13] OMG's Meta Object Facility. http://www.omg.org/mof/

[14] Eclipse Modeling Framework Project. http://www.eclipse.org/modeling/emf/

[15] Graphical Modeling Framework. http://www.eclipse.org/modeling/gmf/

[16] H. Tummescheit. Design and Implementation of Object-Oriented Model Libraries using Modelica. Lund, Sweden: PhD thesis, Department of Automatic control, Lund Institute of Technology, 2002.

Model based Virtual Startup of Automation Systems

Uwe Schob (uwe.schob@iwu.fraunhofer.de), Ralf Böttcher,
Fraunhofer IWU,
Reichenhainer Straße 88, 09126 Chemnitz

Torsten Blochwitz, Olaf Oelsner, ITI GmbH Dresden
Marek Winter, USK Karl Utz Sondermaschinenbau GmbH Limbach-Oberfrohna

Abstract

This paper deals with the model generation for automation systems. A generic solution is presented, which analyses existing Computer Aided Engineering (CAE) documents and thereof automatically generates simulation models. They are described as Modelica models and use a special library, customized for the automation branch. It contains common elements which can be parameterized with very few efforts. Additional components for the communication to the control are available. They support several widely accepted technologies like OPC, Profibus, Profinet, analog and digital signals.

The generation process focuses on being as automated as possible. It is a general approach, which could be applied to various types of simulation with different levels of detail. Its integration into a software prototype shows the feasibility and provides first practical feedback.

Keywords:

Automation; Model Generation; Machine-simulation; Modeling; CAE-analysis; Virtual Startup

1 Introduction

Global product diversification increases the demand for more flexible and thus more complex manufacturing plants. Reducing their time to market is of utmost priority in order to remain competitive. Developing and deploying highly automated machines is a most demanding task requiring specialists from different domains. Plans for the mechanical construction, the electrical diagrams as well as control programs are created under deadline pressure followed by the assembly and startup phase.

Several trends arose to meet the above mentioned requirements. The introduction of high level paradigms to restructure the development process itself may be considered as the most future-oriented solution as seen in [1, 2, 3]. Software tools implementing these ideas are based on unified data models to simplify the inter-domain associations [4, 5]. Another method of supporting the machine development is the extensive use of simulation to improve the design quality. Simulations may be applied to verify early design decisions and help finding flaws [6].

2 Motivation

A common practice in plant development is simulating the machines or parts thereof. Such virtual machines are used to verify control programs and thus increase their overall quality in an early stage. This method is called virtual startup and may already be performed without real machine hardware, leading to a reduction of the time needed for the real machine deployment as stated by Wünsch [7].

This assumption is only true, if the time needed to create simulation models does not surpass the expected gain at a later phase. Especially in the field of special purpose machines, where only few repetitive systems are constructed, the task of model generation is too time consuming. This is also stressed by Bergert in [8]. The task of creating the machine model is regarded as additional work to the normal development and creates error prone solutions.

As stated above, the idea of virtual startup lacks optimizations allowing its application to broader areas. To meet the problem of the time consuming model generation task, one has to be reminded that most of the required information already exists. It is contained in the construction and manufacturing plans, e.g. Computer Aided Design (CAD) drawings and circuit diagrams, which are available in digitalized form. They are stored and managed in various CAE, Enterprise Resource Planning and office applications.

DOI: 10.3384/ecp09430120

One feasible solution is to use readily available interfaces of the domain-specific applications as data access. They provide the data in digitalized and reusable form. As stated in [9], the interoperability between software systems is a matter of designing appropriate adapters, which translate one set of data elements into another one.

3 State of the art

3.1 Machine development

The process of developing a machine or a whole plant consists of several phases as seen in Fig. 1. These are mainly, the mechanical construction, the electrical, pneumatic and/or hydraulic design as well as control programming. According to actual guidelines, the phases should be performed in a parallel way to reduce overall-time and inter-domain errors [1]. Despite the obvious advantages, the idea of a holistic mechatronic design is far away from being common knowledge or a widely accepted method. Each domain-specialist performs his respective tasks for himself with his favored software tools. His results remain independent unless they are communicated to other engineers, often in an informal manner. The outcome is a set of different electronic CAE-documents, for example CAD-drawings, circuit diagrams, pneumatic plans as well as control programs, whose association is as good as the previous inter-domain communication was.

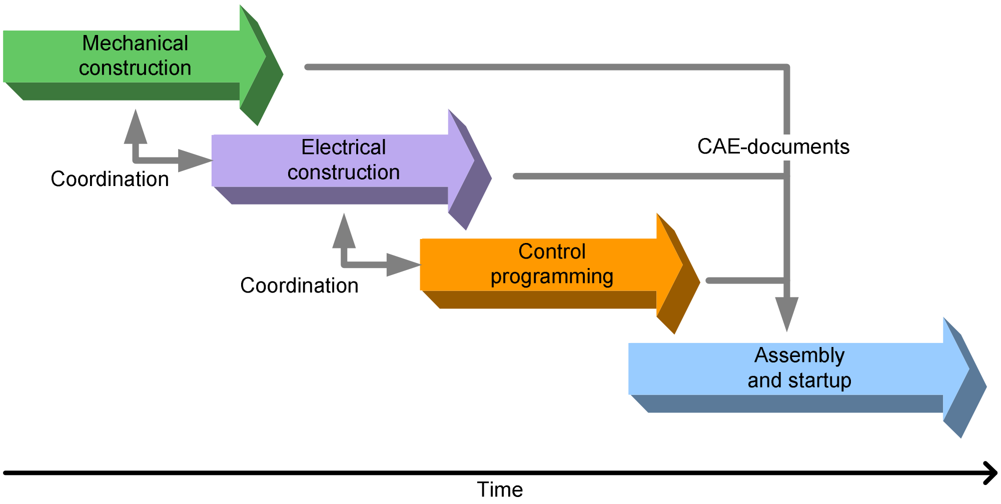


Figure 1: Machine development process

3.2 Assembly and startup

By using the generated assembly documents, the commissioning of the required components and products begins. Depending on the product delivery times, the machine is being assembled more or less in time. As soon as the first functional parts are assembled and electrical connected, its startup begins. Design flaws, wiring mistakes, non-documented changes and even programming mistakes mostly manifest in the startup phase. Beginning at this step, the domain specific decisions are forced to work together and thus often highlight inconsistencies. A missing sensor for process-control might lead to changes in the mechanical structure of the machine as well as its wiring and the control program. Most of all, these late corrections take a lot more time than they would have required during the design-time.

3.3 Simulation

A commonly accepted method to reduce design flaws and inter-domain problems is to simulate the desired system or parts thereof. Depending on envisaged results, different kinds of simulation are used. They range from finite-elements-method to test mechanical strain over multi-body-systems to calculate kinematic behavior of geometric bodies and up to behavior-simulation of whole machines. Simulation may take place as soon, as the first specifications are available. Based on parts of the assembly documents, simulation models may be defined. Results are used for dimensioning machine parts or to verify its functionality.

3.4 Virtual startup

A behavioral model of a machine used for simulation is called a virtual machine. Virtual startup is the process of driving a virtual machine or plant model with real control hardware, as defined by [8]. Currently available commercial tools like WinMOD or Virtuos [10, 11] emphasize the importance of early control program tests as their main goal. One of the main hindrances of a wide acceptance is the additional effort of creating the simulation models [12]. Repeating machine parts, the heavy use of predefined library components and modification of existing simulation models allow savings in the modeling effort.

The remaining effort is still considerably and requires engineers, which are familiar with all machine-aspects expected within the simulation results. This counts especially in the field of special purpose machines, where the repeated construction of the same machine is more than unusual.

Unfortunately, no currently available tool allows a sophisticated reuse of CAE-documents from all design phases as a base for simulation models.

4 Solution

4.1 Starting point

The software tools currently used for documenting the different design phases are manifold. They do not share a common data depository nor are they able to export their content into a universal data format. However, most of them include application programming interfaces (API), which provide a limited access to their internal data structures [13, 14]. This can be used to algorithmically acquire a subset of the available information, which is contained in the CAE-documents. Each design domain provides a different set of information.

The mechanical construction defines, of which parts a machine is composed and how they are interconnected. It focuses on the geometric design of bodies. Additionally, their assembly hierarchy is defined by various constraints.

The electrical construction determines, by which means electrical energy is transformed into useful energy. Actuators manipulate the work piece and sensors provide feedback about it. Every machine component that needs electrical energy has to be documented in circuit diagrams.

Similar to the electric are the pneumatic and hydraulic constructions. They also transform source energy into useful energy. Only the sources, pressurized air respective pressurized fluid, are different. The resulting diagrams and schematics contain all machine parts, which are connected by tubes.

By relating these data sources to each other, the major part of the machine behavior can be estimated. Nevertheless, these data source are not sufficient to form a complete simulation model. Some associations are often undocumented, as they are trivial noticeable with a human understanding of the machine. For instance, the relation of a linear drive in a circuit diagram and its geometric representation in a CAD-drawing is algorithmically not obvious. Even the positioning of sensors within the machine is seldom to be found in CAE-documents. In order to being able to generate a meaningful simulation model, additional data needs to be supplied manually.

Relating the different data sources takes place as part of a data analysis resulting in groups of logical connected objects. Each of them is then to be translated into a behavioral identical simulation element. By this translation, simulation models representing the supplied data sources are created. Fig. 2 shows this translation process in a simplified form.

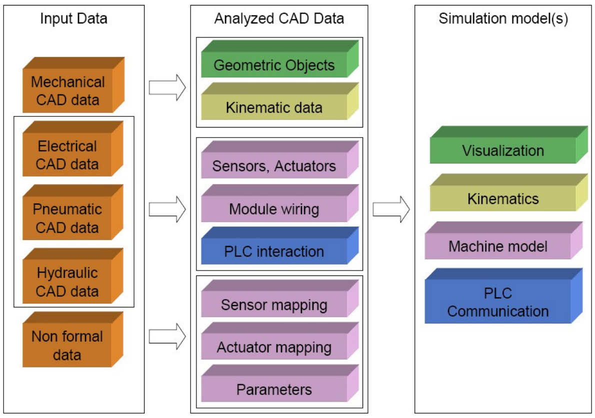


Figure 2: Transforming data sources

4.2 Transformation process

Looking into the transformation process in more detail reveals four separate steps as shown in Fig. 3. Their starting point are the different data sources, which are preferably supplied automatically. Partial information needs to be fed manually.

1) The data acquisition targets the available CAE-documents. A program module uses the APIs of each involved software tool to read out the raw information, a user has previously entered during his normal workflow. The output represents the actual state of the machine being developed.
2) During the mapping, the raw information parts are analyzed and grouped based on their resource mark, defined by [15]. Through them, logical connected units can be recognized. For instance, an electric relay in the circuit diagrams is represented by a conductor and one or more switches which all share a similar resource mark.

3) The manual input is a necessary step which requires information by the user. The informal data exchanged through the inter-domain coordination, as shown in Fig. 1, has to be formalized. A functional decomposition of the machine is provided as the desired machine development state. This will be supplemented by simulation-parameters not otherwise available. Additional associations are created between actuators and their respective geometric object, thus allowing a 3D-visualization during simulation based on the CAD-drawings.
4) The final step in creating a simulation model is the consistency check. It is performed based on the nominal and the actual machine development state. A comparison shows common mistakes such as:
   a) specified but not yet realized machine functions
   b) designed but unspecified functions
   c) wiring and tubing changes not consistent with the specification
   d) possible communication / wiring problems between control program and connected actuators and sensors.

   The results are reported to the user as recommendation. Updating the data sources in this development stage is optional regarding the generation of a simulation model. Nevertheless would a later error search be simplified, if obvious design flaws are removed.

In addition, the consistent logical elements are mapped to ready Modelica elements. By trying to apply rules of a larger set to an analyzed logical group, a corresponding Modelica library element can be found. The rule set to be used depends on the complexity of the logical group and the available element library. Different kinds of mapping rules may apply:

a) Exactly one circuit element is matched with exactly one simulation element.
b) Exactly one circuit element is matched with two or more simulation elements. This helps expressing more complex circuit elements through a combination of several simple simulation elements.
c) Two or more electric (or pneumatic) elements are mapped to a single simulation element.
d) A set of circuit elements is matched by another set of simulation elements.
e) Defining the rule set responsible for a mapping is a one-time task. It might be reused for following machine developments, as long as no additional unmapped electrical components exist in the CAE documents.

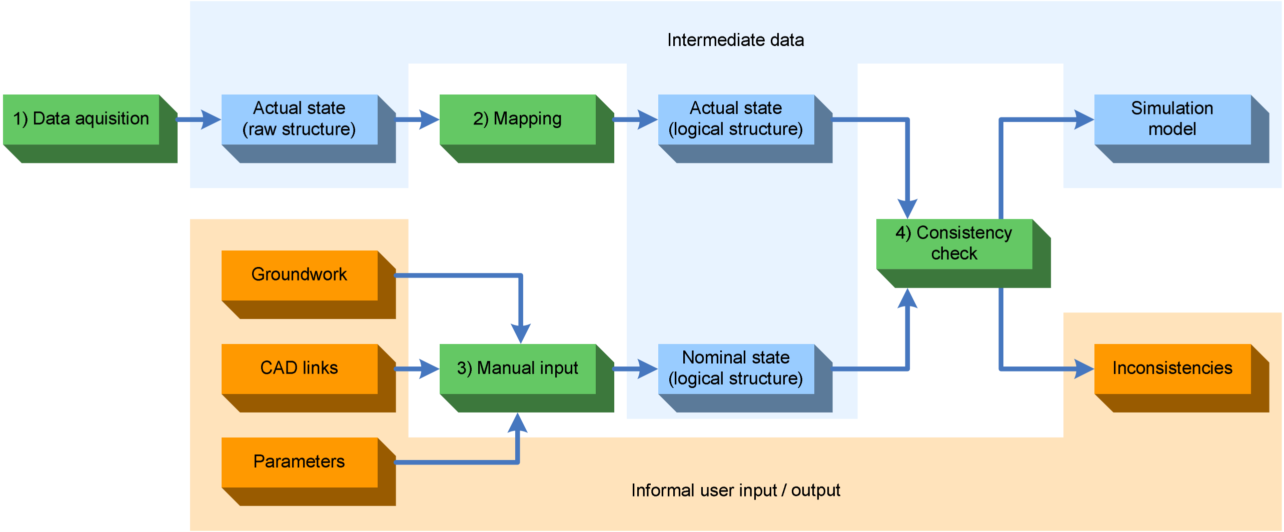


Figure 3: Transformation process in detail

4.3 Simulation usage

An important impact on the allowed simulation model complexity is the application purpose. The main goal of this work is the virtual startup of a machine by using its real control program. Although the programming of programmable logic controls (PLC) is standardized in [16] not all control vendors conform to it. Additionally, various hardware dependant features are proprietary, thus making it difficult to simulate the control itself in software. Based on this assumption, hardware-in-the-loop is the means of choice for the simulation usage. The interface between the virtual machine and the real control hardware are the PLCs input and output signals. An exemplary signal flow through a circuit diagram is shown in Fig. 4.

In order to feign a real machine to the control, the simulation has to meet several requirements:

1) All output signals written by the control need to be used within the simulation. Respective need all input signals read by the control to be provided as actual simulation outputs.
2) The simulation must not progress faster than the real control.
3) The complexity of the simulation model has to be chosen to be as time-efficient as possible.
4) The signal-transmission to and from the control must not influence the control program

Although the control hardware is a hard real-time system, the virtual startup itself does not strictly requires it. Of course, if the simulation is working in a soft real-time mode, its average calculation time should not exceed the controls cycle time. Exceeding the cycle time is critical for the validity of the simulation results only if the control is performing operations, depending on strictly timed feedback.

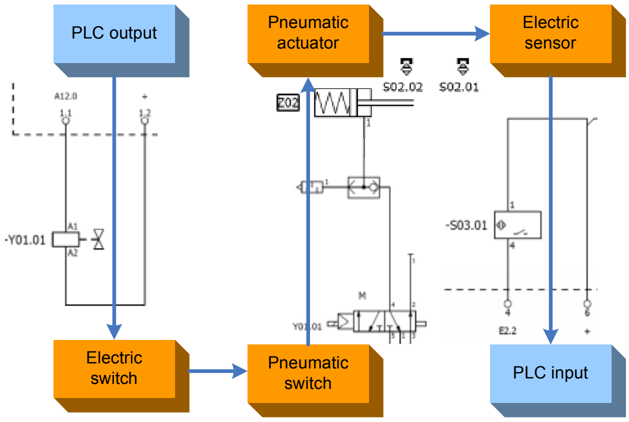
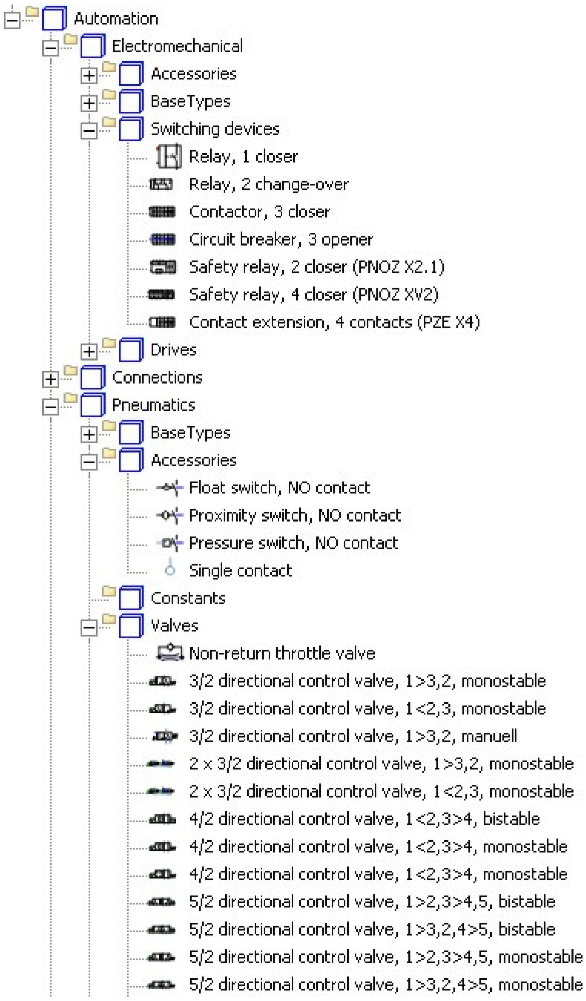


Figure 4: Signal flow

4.4 A customized Modelica library

Although various Modelica libraries exist, the above mentioned requirements of runtime optimized simulation elements need to be met. Respecting and modeling all physical effects is possible, but not necessarily relevant. A new library was created to incorporate machine devices modeled more behavioral than physical. It was named after and designed for the special needs of the automation branch. An excerpt is shown in Fig. 5.

The desired level of detail of each model element was chosen element-wise, depending on the devices real functionality. By initially creating all basic elements, such as relays and switches, more complex devices could be built upon them. A later described machine example was created through the normal development process and taken as reference for filling the automation library. It contains pneumatic and electric elements and is still being extended.



- Automation
  - Electromechanical
    - Accessories
    - BaseTypes
    - Switching devices
      - Relay, 1 closer
      - Relay, 2 change-over
      - Contactor, 3 closer
      - Circuit breaker, 3 opener
      - Safety relay, 2 closer (PNOZ X2.1)
      - Safety relay, 4 closer (PNOZ XV2)
      - Contact extension, 4 contacts (PZE X4)
    - Drives
  - Connections
  - Pneumatics
    - BaseTypes
    - Accessories
      - Float switch, NO contact
      - Proximity switch, NO contact
      - Pressure switch, NO contact
      - Single contact
    - Constants
    - Valves
      - Non-return throttle valve
      - 3/2 directional control valve, 1>3,2, monostable
      - 3/2 directional control valve, 1<2,3, monostable
      - 3/2 directional control valve, 1>3,2, manuell
      - 2 x 3/2 directional control valve, 1>3,2, monostable
      - 2 x 3/2 directional control valve, 1<2,3, monostable
      - 4/2 directional control valve, 1<2,3>4, bistable
      - 4/2 directional control valve, 1<2,3>4, monostable
      - 4/2 directional control valve, 1<2,3>4, monostable
      - 5/2 directional control valve, 1>2,3>4,5, bistable
      - 5/2 directional control valve, 1>3,2,4>5, bistable
      - 5/2 directional control valve, 1>2,3>4,5, monostable
      - 5/2 directional control valve, 1>3,2,4>5, monostable

Figure 5: Overview of the automation library

5 Implementation

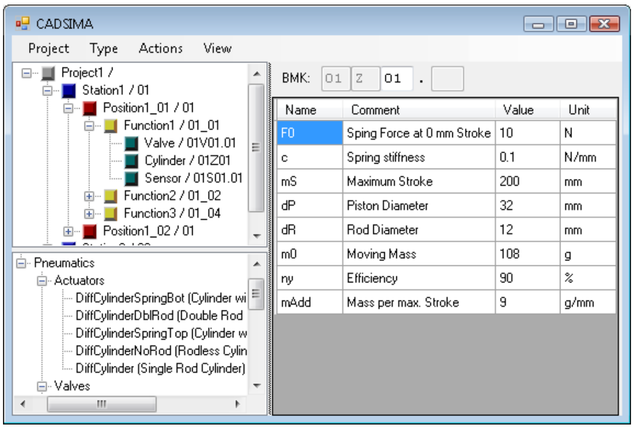
5.1 Connected tools

During a current research project, the mentioned resolution is being implemented. A difficult decision had to be made, which software tools are to be used. A market survey revealed several CAE-tools, whose features were roughly similar, concerning the domain specific development requirements. By examining the available API-functionality according to their flexibility and complexity, the choice felt on the most promising tools:

1) Autodesk Inventor as CAD software,
2) EPlan Electric P8 and its Fluid-addon as tool for designing circuit and pneumatic diagrams and

3) ITI SimulationX as modeling and simulation tool [17].

The missing connection between all these software tools was a program module, which managed the transformation process and its underlying data sources. In consequence, a standalone application (CADSIMA, Fig. 6) was developed, that was able to connect and handle the data flow from and to each API. In addition it can be used as coordination tool between the domain specific design steps, through which necessary informal data is acquired.



Project1 /
Station1 / 01
Position1_01 / 01
Function1 / 01_01
Valve / 01V01.01
Cylinder / 01Z01
Sensor / 01S01.01
Function2 / 01_02
Function3 / 01_04
Position1_02 / 01

Pneumatics
Actuators
DiffCylinderSpringBot (Cylinder wi
DiffCylinderDblRod (Double Rod
DiffCylinderSpringTop (Cylinder w
DiffCylinderNoRod (Rodless Cylin
DiffCylinder (Single Rod Cylinder)
Valves

Name	Comment	Value	Unit
F0	Sping Force at 0 mm Stroke	10	N
c	Spring stiffness	0.1	N/mm
mS	Maximum Stroke	200	mm
dP	Piston Diameter	32	mm
dR	Rod Diameter	12	mm
m0	Moving Mass	108	g
ny	Efficiency	90	%
mAdd	Mass per max. Stroke	9	g/mm

Figure 6: Screenshot of newly tool Cadsima

The nominal state can be entered by the user and is stored in a database. This allows the distributed access to the project data. CADSIMA creates a SimulationX model based on the results of the mapping and consistency checks.

5.2 Analysis and mapping example

The general data analysis and mapping into model elements have been implemented for the selected software tools. Their internal data structures showed a diversity of objects, whose properties need to be evaluated in detail during the analysis. The resulting mapping mechanism and corresponding rules are derived thereof and explained in the following paragraph.

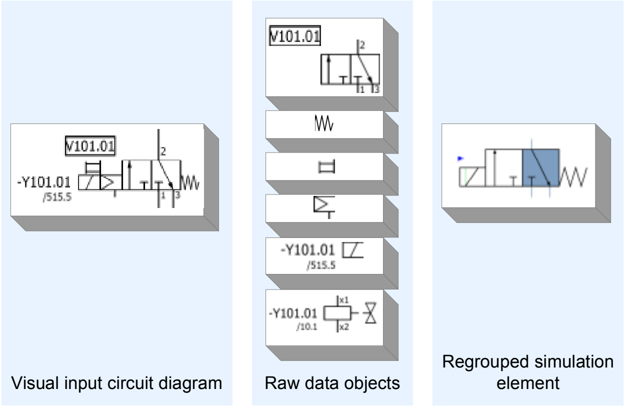


Figure 7: Mapping example

Fig. 7 shows the analysis and mapping of a pneumatic monostable 3/2-way valve. A visual inspection of the corresponding circuit diagram page displays one element with two visual resource marks. The internal data model reveals that it contains 5 separate objects:

1) the valve itself with resource mark V101.01,
2) a spring on the right side,
3) a manual reset at the left upper side,
4) a pneumatic trigger on the left side,
5) an electric trigger at left side with the different resource mark Y101.01 and
6) a reference to an inductor at another page, which represents the wired element.

A rule to find such valves looks for each of the 6 elements and if found, signals the creation of the corresponding simulation element. Wires and tubes are stored as connections between pins in the circuit diagrams. They are also mapped to pins of the simulated pneumatic valve.

5.3 A first use case

Current work focuses on providing the resources to transform small machines or parts thereof. The resources meant are appropriate library elements and defining rule sets for them. This work is a necessary step prior to transforming actual CAE documents. The reference project of a pick and place handling machine, as shown in Fig. 8, contains 3 separately controlled axes and a not displayed conveyer. Looking at the mechanical characteristic, it contains about 30 mechanical parts. Its circuit and pneumatic diagrams include on 87 pages roughly 170 articles represented by more than 1300 separate symbols. About 25% of these symbols are only for a display purpose and not actually wired to other components. Unfortunately, as worst case every remaining symbol needs to be transformed into a simulation element.

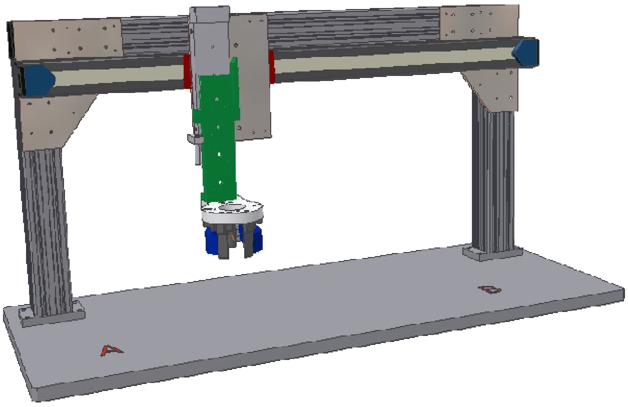


Figure 8: Handling machine for pick and place

5.4 Virtual startup

A setup for a virtual startup contains a real control, without additional peripheral equipment and a pc running the generated simulation model. The pc will communicate with the control hardware via OPC, overwrites the controls inputs and reads its outputs. No additional hardware as the above mentioned is required, although actual displays or human machine interfaces can be connected to the control hardware, if desired. The control program itself needs a minor change to accept the missing peripheral equipment. The programs write and read access is not influenced by it. Despite that, the program remains identical to the one used in the real machine.

During the simulation run, the control programmer is connected to the real control hardware as well. He performs the same startup steps as with a real machine. Its current state can be inspected through the simulator and its built-in signal-visualizations. Interactive changes are available through overwriting runtime parameters.

6 Conclusions and Outlook

This article described a method, which transforms an input data set, e.g. the circuit diagram of a manufacturing system, to an output data set, e.g. a simulation model of the same machine. It detailed the structure of the source data model, the components to which they will be transformed and finally, how a simulation run can be performed. The target model library consists of Modelica elements combined with communication components which are used for a hardware-in-the-loop simulation.

This approach drastically reduces the effort in creating machine models and thus enables even the special purpose machine manufacturer to use the virtual startup. A system test for validating the control program may be undertaken prior to assembly of the real machine.

Actual outcomes for the startup phase, e.g. higher program quality or shortened startup time, and for the machine development process as whole are outstanding. Hence, future works should focus on providing reliable feedback about the methods effectiveness in the field of application.

Experiments on measuring the numerical performance of larger machine models should be undertaken prior to extending the automation library to support more complex development projects. The usage of a faster communication protocol and appropriate hardware, e. g. Profibus or Profinet, would allow a better integration of feedback controlled systems working with smaller cycle times.

This work originated in the cooperation of the companies USK Karl Utz Sondermaschinenbau GmbH Limbach-Oberfrohna, ITI GmbH Dresden and the Fraunhofer Institute for Machine Tools and Forming Technology IWU during the joint research project “Depromes”, funded by the Sächsische Aufbaubank.

References

[1] Verein Deutscher Ingenieure: VDI 2206 - Entwicklungsmethodik für mechatronische Systeme, Juni 2004.

[2] J. Bathelt: Entwicklungsmethodik für SPS-gesteuerte mechatronische Systeme, ETH Zürich, 2006.

[3] CADsys: FOD - Prozesskettenübergreifende Produktkonfiguration, online, 2006.

[4] Verein deutscher Maschinen- und Anlagenbauer: Baukastenbasiertes Engineering mit Föderal, 2004.

[5] B. Grimm et al.: Universelles Datenaustauschformat, A&D Kompendium, 2008.

[6] M. Ehrenstraßer et al.: Virtuelle Werkzeugmaschinen für die Simulation, wt Werkstattstechnik online, Jahrgang 92 (2002)

[7] G. Wünsch: Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme, Technische Universität München, 2007.

[8] M. Bergert and C. Diedrich: Durchgängige Verhaltensmodellierung von Betriebsmitteln zur Erzeugung digitaler Simulationsmodelle von Fertigungssystemen, in Automation Kongress, 2008.

[9] U. Schob: Werkzeuge und Methoden zur mechatronischen Modellierung von Produktionsanlagen, Technische Universität Chemnitz, 2007

[10] Mewes & Partner GmbH: WinMOD, online, http://www.mewes-partner.de/www/eng/, 2009

[11] ISG Industrielle Steuerungstechnik GmbH: ISG-virtuos, online, http://www.isg-stuttgart.de/virtuos.html?&L=1, 2009

[12] G. Reinhart: Teilautomatisierter Aufbau von Simulationsmodellen, wt Werkstattstechnik online, Jahrgang 97 (2007)

[13] EPLAN Software & Service GmbH & Co. KG.: EPLAN API 1.0, application documentation, 2006

[14] Autodesk Inc.: Autodesk Inventor API, application documentation, 2004

[15] Deutsches Institut für Normung: DIN EN 61346-2 Industrielle Systeme, Anlagen und Ausrüstungen und Industrieprodukte - Strukturierungsprinzipien und Referenzkennzeichnung, 2000

[16] International Electronical Commission: IEC 61131-3 Programmable controllers - Part 3: Programming languages, 2003

[17] ITI GmbH: SimulationX, online, http://www.iti.de/cms/en/simulationx.html, 2009

Advanced simulation methods for heat pump systems

Kristian Huchtemann Dirk Müller
Institute for Energy Efficient Buildings and Indoor Climate
E.ON Energy Research Center, RWTH Aachen University
Jägerstraße 17/19, 52066 Aachen, Germany
khuchtemann@eonerc.rwth-aachen.de

Abstract

Within the use of renewable energies in buildings several dynamic influences and interactions have to be regarded. For example, the performance of heat pump systems depends sensitively on weather or geological effects as well as on building and user behaviour. An energetic optimization requires an intelligent coupling and control of all components of the building services installation, the building envelope and the influences mentioned.

The Modelica libraries developed at the Institute for Energy Efficient Buildings and Indoor Climate allow a detailed analysis of the overall system including the various influences and interactions.

Keywords: building simulation; heat pump; building services installation

1 Introduction

The building sector is getting more and more into the spotlight of energy efficiency programs considering its great portion to the overall energy consumption and its huge potential for energy savings. As every existing and new building is a unique system, adaptable simulation tools are required to calculate and predict its energy demand.

For example heat pumps offer a huge potential of energy savings compared to classic heating devices in buildings. Possible savings depend on the performance of the heat pump, which is described through the Seasonal Performance Factor β:

$$\beta = \frac{Q_{use}}{W_{el}} \tag{1}$$

Q_{use} is the generated heat during one year and W_{el} is the electrical energy demand of the compressor and the auxiliary drives [1]. In practice, identical heat pump types often achieve very different performance factors. The reasons for that are generally known - the control system and the ground source heat exchanger have a significant impact on the system. For example, heat source temperatures or needed supply temperatures of the heating system influence the performance. Thus an analysis of the heat pump performance requires an analysis not of the heat pump alone but of the overall system. A sample scheme in figure 1 shows the most important components of heat pump systems.

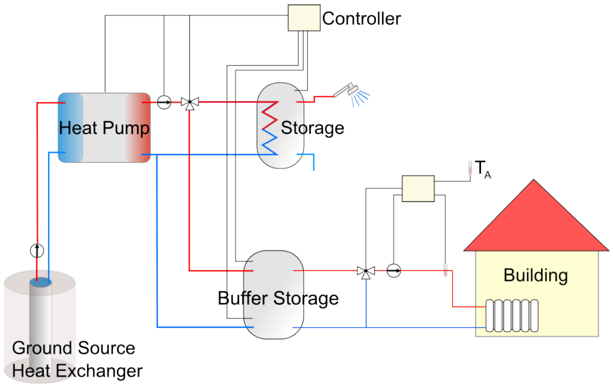


Figure 1: Sample scheme of a heat pump system.

The Modelica libraries developed at the Institute for Energy Efficient Buildings and Indoor Climate allow a detailed modelling of the whole thermo-hydraulic system including the heat source, heat pump, water storages and heat sink - the building. User behaviours are being implemented as well as weather data, control and geologic models. In this way, all relevant influences on the heat pump performance are regarded, sensitivity analyses are accomplished and control strategies are examined.

DOI: 10.3384/ecp09430043

2 Building Library

The building models represent structural effects and user influences [2]. They are compatible with the building services installation models described below. The library contains multilayer walls, windows and doors including the phenomena involved such as heat conduction, convection and radiation. A room model is shown in figure 2. The air volume of the room is calculated by the medium models of the Modelica.Media library. User influences are described by internal heat loads and variable air exchange. The library includes an extensive weather model based on test reference year data of the German Meteorological Service, offering boundary conditions for the simulation. The model calculates the ambient temperature, air pressure, humidity and solar exposure rates on arbitrarily sighted surfaces.

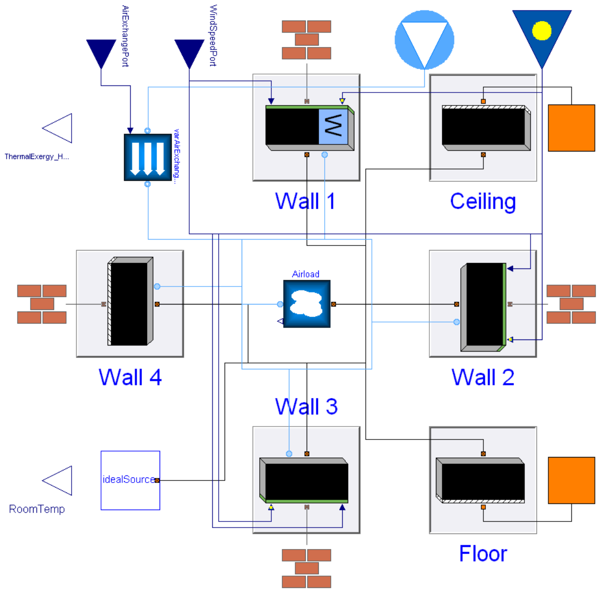


Figure 2: Model of a sample room.

3 Building Services Installation Library

The building services installation library contains basic components of building services installations, such as pumps, pipes, boilers, heaters and valves. It uses medium models of the Modelica.Media and components of the Modelica_Fluid libraries. Simple components calculate state changes of fluid by look-up tables, more complex models use finite volume methods and empiric correlations [2]. Pipes, heat exchangers and thermally activated building parts are implemented in the latter way. Figure 3 shows a simple heating circuit containing some models of the library.

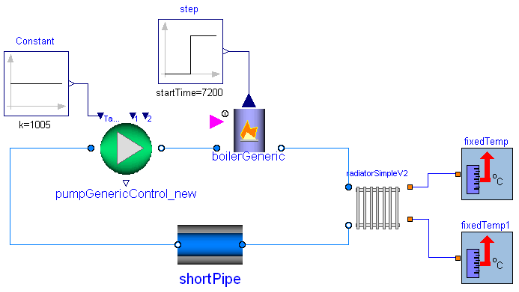


Figure 3: Model of a simple heating circuit.

4 Heat Pump System Library

For the purpose of modeling heat pump systems, the existing libraries are complemented by a new library. It includes different heat pump models, storage models, ground source heat exchangers and ground models as well as controllers.

4.1 Heat Pump Model

The heat pump model is implemented as a black box model consisting of two heat exchangers that are connected to a module that calculates the heat flows and compressor power by look-up tables using manufacturer data. Generally this data is given at working points standardised by [3]. The more working points are given the better the real heat pump's dynamic behaviour is reproduced. The basic working scheme of this black-box model is shown in figure 4.

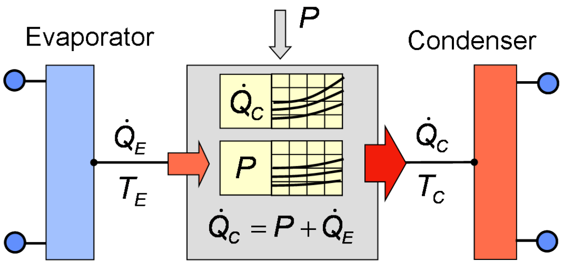


Figure 4: Scheme of the table based heat pump model.

A more detailed model implementing the refrigerant circuit is being developed using external fluid properties by the ExternalMedia library [4].

4.2 Ground Source Heat Exchanger Model

Ground source heat exchangers serve as heat source for the heat pump system. They are using either the upper ground's or the relatively deep ground's (up to 200 m depth) heat capacity.

The models describe coaxial pipes as well as u-formed pipes discretised in axial direction connected to a ground model. The ground model is an axially and radially discretised volume enclosing the borehole. At the borders of the simulated ground volume a boundary condition is assumed. In figure 5 a coaxial pipe is modeled and a constant temperature is assumed as boundary condition in a certain radius to the borehole. The pipe model can also be initialised with an increasing temperature representing the geothermal coefficient. The boundary condition is then adapted accordingly.

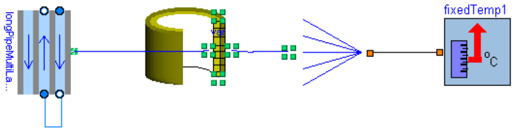


Figure 5: Model of a coaxial pipe connected to a ground model with constant temperature boundary condition.

The simulation of such a model with a given heat extraction time series taken from field test data leads to a funnel-shaped distribution of temperature in the surrounding ground (see figure 6). Ground-water flow can strongly influence such distributions of temperature. That is why more complex ground models are being developed following the example of the software SHEMAT [5].

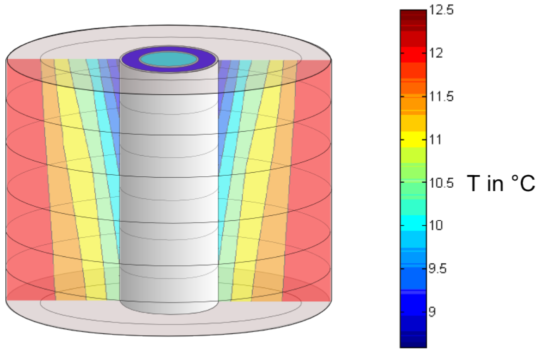


Figure 6: Simulation results: Temperature field in the profile of the ground model.

4.3 Stratified Storage Model

Storages are important components of the heat pump system. They are used to handle variable heating demand or serve as drinking water storages. A wide range of different storage types exist, such as storages with heat exchangers, combined buffer and drinking water storages as well as stratified water storages. Basically, every storage has to be modeled separately.

However, a unified modeling can be done for simple buffer storages, that generally consist of a fluid volume with several fluid inlets and outlets. To implement stratification inside this volume, the buffer storage model consists of several fluid volumes representing fluid layers (see figure 7).

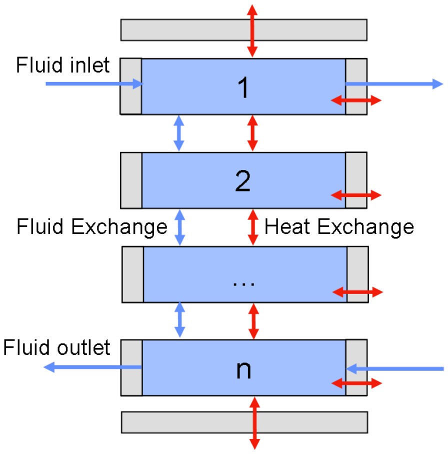


Figure 7: Model of the stratified buffer storage.

The layers are connected to each other allowing heat and fluid flow. Buoyancy effects are regarded by an effective heat conductance λ_{eff} depending on the temperature differences between the layers:

$$\lambda_{eff} = \lambda_{water} \text{ if } T_n > T_{n+1} \tag{2}$$

$$\lambda_{eff} = \lambda_{water} + \lambda_{turbulent} \text{ if } T_n \leq T_{n+1} \tag{3}$$

The turbulent heat conductance is a function of the layer thickness, its temperature and the temperature difference to the above layer and is based on the work of Viskanta et al. [6]. The buffer storage model has fluid inlets and outlets in the top and bottom layers. The quality of stratification in the specific storage can be fitted by choosing the number of layers in the model.

4.4 Validation of Models

A validation of library components is done using the data from a field study in Germany recorded by the Fraunhofer Institute for Solar Energy Systems in Freiburg, which was initiated by the E.ON Energie AG. The field study is recording time series of temperature and volume flows at several points of the regarded heat pump systems. This data is taken as input to the components or combined components. This way the model's reaction can be compared to the behaviour of the real component.

This is done for the table-based heat pump model in the set-up shown in figure 8. Figure 9 shows the results for two different heat pump types, that means two different data sets. The error gets up to 10 % in few models under certain circumstances, but generally the heat pump behaviour can be simulated well by the manufacturers' data.

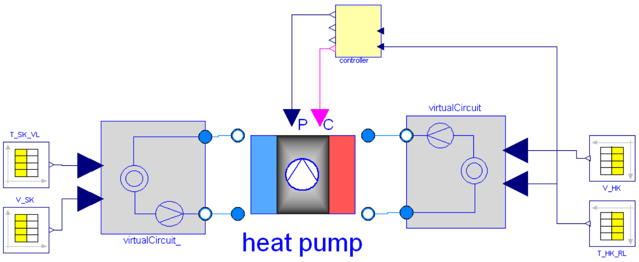


Figure 8: Validation of the heat pump model.

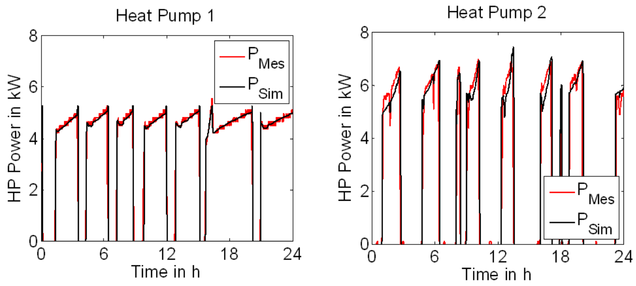


Figure 9: Comparison of simulated heat pump power to field test data.

Other components are validated analogously. For the validation of ground source heat exchanger models the field test data is complemented by thermal response test data.

5 Combined Simulation

As mentioned in the beginning, the combined simulation of the thermal building behaviour, the hydraulic components, weather- and user influences and, in case of geothermal heat pumps, geothermal heat exchangers and ground models is required to describe the main influences on the performance of the heating system. Figure 10 shows a model according to the sample heat pump system in figure 1.

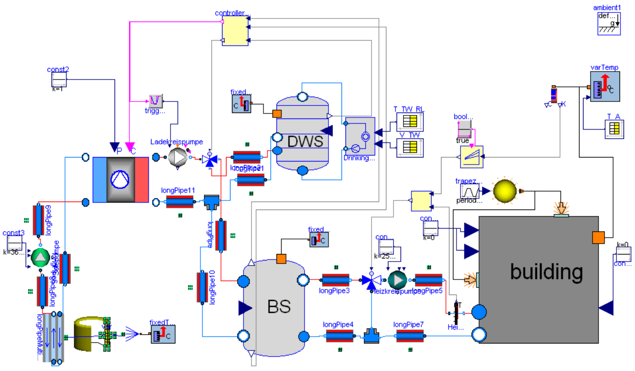


Figure 10: Model of the total system.

The simulation results of different temperatures occurring in the system are displayed in figure 11. The ambient temperature (T_A) is a given time series from field test data. At ambient temperatures above 15 °C the heating is turned off by the controller, so that the buffer storage is not discharged. During that time the top and bottom buffer storage temperatures ($T_{BS,t}$ and $T_{BS,b}$) decline because of storage heat losses. The reaction of the evaporator and condenser temperature on the water side (T_E and T_C) are shown, too. The highest peaks of the condenser temperature occur when the drinking water storage is charged. The brine temperatures (T_{Flow} and T_{Return}) show an effect of short term rebound of ground temperature during the turn-off interval of the heat pump. A room temperature (T_R) of 20 °C can be ensured throughout the whole day.

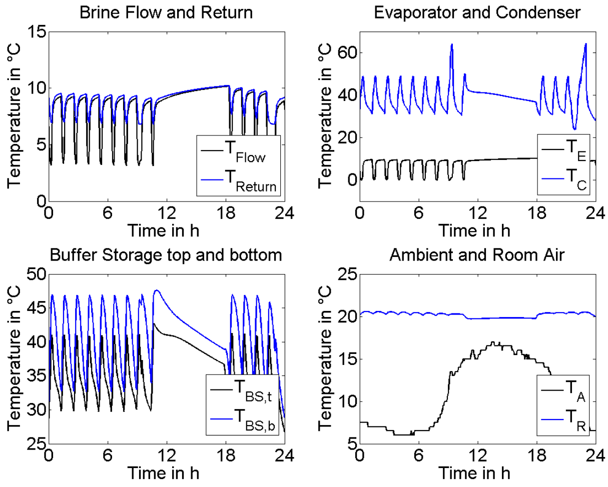


Figure 11: Simulation results of different temperatures of the total system.

6 Analyses of the heat pump system

A sensitivity analysis is done, varying the volume of the buffer storage. Results are shown in figure 12. The heat pump power is displayed for the time of 24 hours each for a January and April day using a small and a big buffer storage. Small volumes mean smaller heat losses but also a high number of operating intervals. This has to be avoided to ensure the heat pump's lifetime.

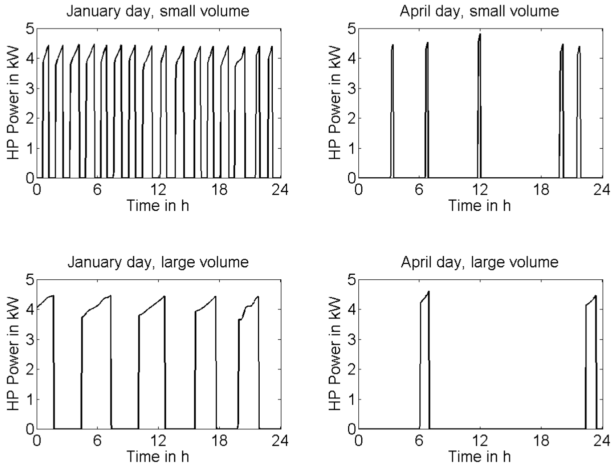


Figure 12: Simulation results: Heat Pump electric power.

For a January day with a high and constant heating demand a small buffer storage cannot ensure the temperatures required by the heating system. The temperature in the buffer storage gets low faster than the heat pump is allowed to turn on again by the controlling system. Minimum turn-off intervals are implemented in the controls. With a large buffer storage the required supply temperatures can be ensured and a reduced amount of operating intervals can be achieved.

These results are summarized in figures 13 and 14. They show the number of operating intervals and the heat pump work for one month for different tested volumes. The reference volume (750 l) is set to 100 % in each chart and represents the volume of a reference field test object's buffer storage. The results show a lower heat pump work using smaller buffer storages, because of storage heat losses being smaller. But a strong increase of operating intervals can be observed, too.

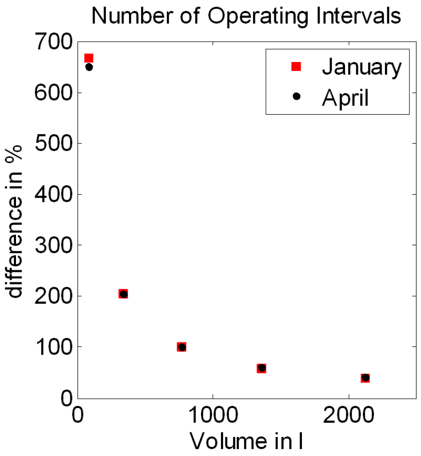


Figure 13: Simulation results: Heat pump working intervals (four week simulation each).

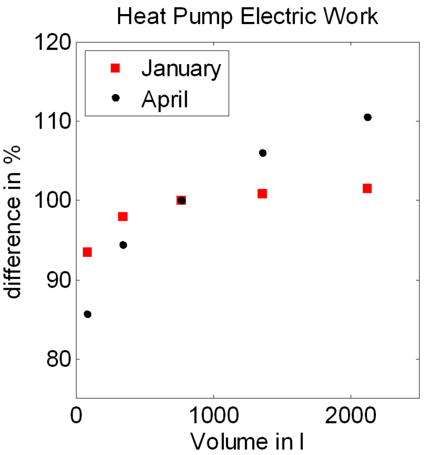


Figure 14: Simulation results: Heat pump electric work (four week simulation each).

7 Summary and Outlook

The system models regarded in this paper describe the heat source, the heat pump system including valves, storages and controls and the building as heat sink, including the building envelope, air volume and user influences. Analyses are made to detect possible enhancements for heat pump systems. The focus lies on the analyses of different system arrangements and control strategies.

The presented libraries and components allow a detailed energetic modeling and simulation of single and multi-family houses. User defined buildings and building services installations can be modeled taking into account the diversity of such systems in reality.

The heat pump system library is currently being extended by more detailed models. A detailed heat pump model will describe the heat pump refrigerant circuit.

The ground source heat exchanger models will be enhanced, allowing the simulation of ground source heat exchanger fields and of ground-water flow.

8 Acknowledgement

The authors would like to thank the E.ON gGmbH for the project's promotion. Thanks are also given to E.ON Energie and the Fraunhofer Institute for Solar Energy Systems for cooperation and transmission of field test data.

References

[1] VDI-Richtlinie 4650: *Calculation of heat pumps - simplified method for the calculation of the seasonal performance factor of heat pumps.* Verein Deutscher Ingenieure, Düsseldorf, 2009.

[2] A. Hoh, T. Haase, T. Tschirner, D. Müller. *A combined thermo-hydraulic approach to simulation of active building components applying Modelica.* In Proc. of 4th International Modelica Conference, Hamburg, March 2005.

[3] DIN EN 255-3 *Luftkonditionierer, Flüssigkeitskühlsätze und Wärmepumpen mit elektrisch angetriebenen Verdichtern - Heizen.* Deutsches Institut für Normung e.V., Berlin, 2007.

[4] F. Casella, C.Richter. *ExternalMedia: A Library for Easy Re-Use of External Fluid Property Code in Modelica.* In Proc. of 6th International Modelica Conference, pages 157-161, Bielefeld, March 2008.

[5] C. Clauser *Numerical Simulation of Reactive Flow in Hot Aquifers Using Shemat/Processing Shemat.* Berlin, Heidelberg, Springer Publishing, 2003.

[6] Viskanta et al. *Interferometric Observations of the Temperature Structure in Water Cooled or Heated from Above*, Advances in Water Resources, Vol. 1, No. 2, 1977

Thermal Separation: An Approach for a Modelica Library for Absorption, Adsorption and Rectification

Andreas Joos[*] Karin Dietl[†] Gerhard Schmitz[‡]
Hamburg University of Technology
Institute of Thermo-Fluid Dynamics,[§] Applied Thermodynamics
21071 Hamburg, Germany

Abstract

Due to its objected-oriented design Modelica is predestinated to describe chemical engineering unit operations. Still only few activity on this field is published. This paper introduces a library which covers three of four major thermal seperation processes: absorption, adsorption and rectification.

Additionally an extension of the ExternalMedia library is presented that allows the connection to thermodynamic and physical property packages for two phase mixtures. Also a possible forecast to an implementation of a CAPE-OPEN interface is made, allowing the use of a plurality of established chemical data packages.

Keywords: Thermal Separation, Separation Column, ExternalMedia, Heat and Mass Transfer

1 Introduction

Global warming has become a major issue in society and politics. The reduction of CO_2-emission of power plants plays an important role when talking about reduction of CO_2-emission. One possibility is to remove the CO_2 from the exhaust gas via absorption, liquefy it and store it underground. In order to optimize this process dynamic modeling is very important since for example the composition of the exhaust gases vary during operation. Whereas there exist already libraries for the dynamic modeling of power plants (like for instance the free Modelica library ThermoPower [2]), Modelica models to model the CO_2 - separation process have not been found in literature.

However not only CO_2 - separation is an interesting topic of dynamic modeling but also other separation processes like adsorption or rectification. A dynamic analysis of such processes gains in importance as dynamic process strategies becoming more popular, be it in batch processing or start-up strategies in continuous processing. Since ad- and absorption as well as rectification have much in common from a modeling point of view, the development of a combined separation library is proposed. The fourth common separation process, extraction, differs from the other processes and is therefore not considered up to now.

2 Modeling

2.1 Library Structure

The library structure can be seen in figure 1 which shows a class diagram of the library. There exist three different models: one of the packed column, one of the tray column and one of the spray column. They all extend from `BaseStageVL`. This base class contains the mole and energy balances for *n* stages, an instance of the `PhaseBoundary`-model and of the medium models as well as the instances of the connector classes. The extending classes supply the geometry, the instances of the pressure loss model, the heat transfer model between the two phases and the mass transfer model. Each column type is structured the same; but only the structure of the packed column is shown in the diagram. For each column type a different class exists. They differ in the models which they allow to replace their base classes. This prevents the user to choose for example in a tray column a pressure loss model developed for a packed column.

2.2 Interfaces

Columns can be connected to other columns or to fluid sources or sinks using connectors. The connector for the vapour flow takes the volume flow rate as flow vari-

[*] email: andreas.joos@tu-harburg.de, Tel.:+49 40 42878 3079
[†] email: karin.dietl@tu-harburg.de
[‡] email: schmitz@tu-harburg.de, Tel.:+49 40 42878 3144
[§] `http://www.tu-harburg.de/tt`

DOI: 10.3384/ecp09430102

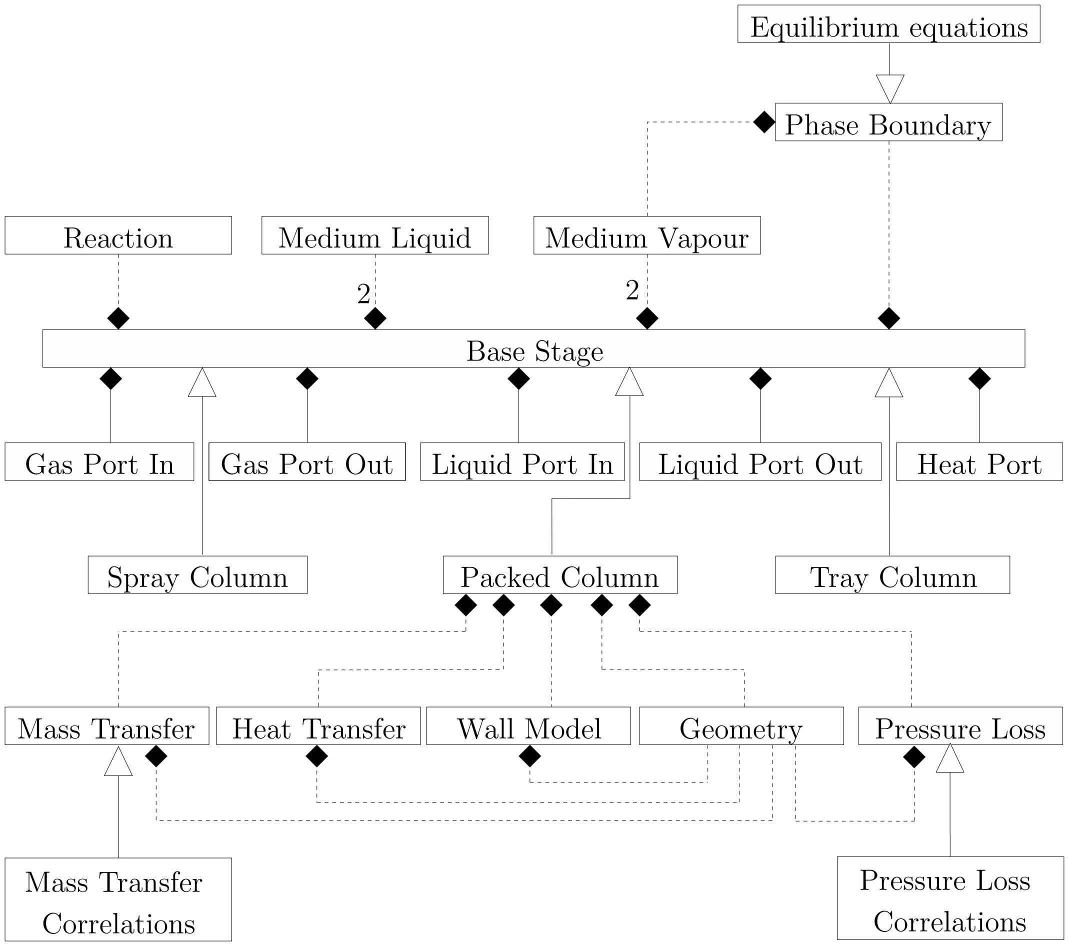


Figure 1: UML class diagram of an absorption or rectification process. The arrow denotes inheritance, the line with the diamond denotes composition. Dotted lines mean that the object is replaceable. The composition of spray column and tray column are analogue to the composition of the packed column. However, this is not shown in the class diagram due to readability.

able and the pressure as potential variable and temperature, molar concentration and composition as in-/outputs. The connectors for the liquid flow are the same, only the pressure as potential variable is replaced by the height. Each column also contains a heat port to account for heat losses to ambiance.

2.3 Assumptions

The following assumptions hold for the modeling:

- discretization only in axial direction
- counterflow of vapour and liquid flow
- no entrainment of the liquid with the vapour
- tray columns: no *raining* through the plates
- reaction, if any, takes place in the liquid phase
- spray columns: drops move only in axial direction
- spray columns: no coalescence or splitting of drops
- no accumulation in gas and liquid films
- no second liquid phase possible

2.4 Modeling of one discrete element

One major design criterion of the library is to take into account heat and mass transfer between the phases; i.e. consideration of non-equilibrium states; equilibrium is only attained at the phase boundary. This is illustrated in figure 2: Therefore the balance equations for heat as well as for mass have to be written in each discrete element for every phase, so a molar flux tends to eliminate the difference between actual composition x_A and equilibrium composition x_A^* and a heat flux tends to equal the vapor temperature T_v and the temperature of the liquid phase T_l. Even though in general the temperature of vapour and liquid phase are almost the same, the introduction of a heat flow rate $\dot{Q}_t$ between the two phases makes sense, since it simplifies the calculation.

The `BaseStage`-model contains n discrete elements for which mole and energy balances are established.

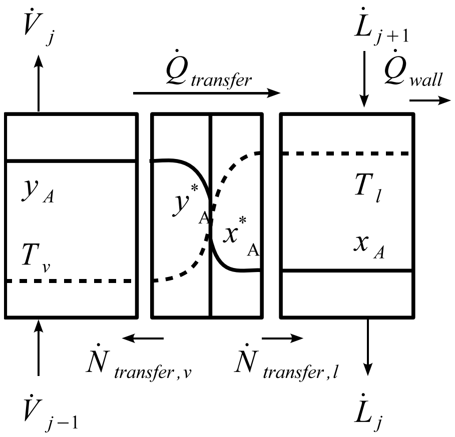


Figure 2: Sketch of the control volumes and the phase boundary

The mole balance for the stage j and the component i of the vapour phase and the liquid phase respectively are as follows:

$$V\cdot\varepsilon\cdot\frac{d}{dt}(\varepsilon_{v,j}\cdot c_{v,i,j}) = \dot V_{j-1}\cdot c_{v,i,j-1} - \dot V_j\cdot c_{v,i,j} + \dot N_{v,t,i,j} \qquad (1)$$

$$V\cdot\varepsilon\cdot\frac{d}{dt}(\varepsilon_{l,j}\cdot c_{l,i,j}) = \dot L_{j+1}\cdot c_{l,i,j+1} - \dot L_j\cdot c_{l,i,j} + \dot N_{l,t,i,j} + \dot N_{reaction,i,j} \qquad (2)$$

Since no mass is accumulated in the gas or liquid film, the molar fluxes sum up to zero:

$$\dot N_{v,t,i,j} + \dot N_{l,t,i,j} = 0 \qquad (3)$$

The vectors $\dot N_{v,t,j}$ and $\dot N_{l,t,j}$ contain the molar flux of each component. If the component is entering the control volume, the sign is positive, otherwise negative.

The energy balance is also established for the two phases separately. Solid material in the column (trays and packing material) are supposed to have the same temperature as the liquid phase. Heat transfer to ambiance also takes place via the liquid phase. Thus for the element j the energy balance for the liquid phase becomes:

$$V\cdot\varepsilon\cdot\frac{d}{dt}(\varepsilon_{l,j}\cdot u_{l,j}) + V\cdot\rho_s\cdot c_s\cdot\frac{dT_l}{dt} = \dot L_{j+1}\cdot h_{l,j+1} - \dot L_j\cdot h_{l,j} - \dot Q_{wall,j} + \dot Q_t + \dot H_{\text{fromV},j} - \dot H_{\text{fromL},j} \qquad (4)$$

and for the vapour phase:

$$V\cdot\varepsilon\cdot\frac{d}{dt}(\varepsilon_{v,j}\cdot u_{v,j}) = \dot V_{j-1}\cdot h_{v,j-1} - \dot V_j\cdot h_{v,j} + \dot Q_t + \dot H_{\text{fromL},j} - \dot H_{\text{fromV},j} \qquad (5)$$

Using the medium models, the molar specific inner energies and enthalpies are calculated for a certain temperature, pressure and mole fraction. The volume specific inner energies and enthalpies are then calculated using the concentration, for example:

$$u_{l,j} = \sum_{i=1}^{nS} c_{l,i,j}\cdot u^m_{l,j} \qquad (6)$$

Together with the molar flux over the phase boundary there is also an enthalpy flux over the phase boundary. Since obviously the composition of the condensing vapour stream differs from the composition of the vapour bulk phase a second instance of the vapour medium, called `mediumVapourTransfer` exists. In there the thermodynamic properties of the transfer vapour stream are calculated, using temperature and pressure of the bulk phase but the composition of the transfer stream. The liquid is treated accordingly. The specific enthalpy obtained by `mediumVapourTransfer` and `mediumLiquidTransfer` is used to calculate the two enthalpy flows, $\dot H_{\text{fromV},j}$ and $\dot H_{\text{fromL},j}$.

Additionally to the differential equations above, there exist also a differential equation for the liquid mass of each element:

$$V\cdot\frac{d}{dt}(\varepsilon_{l,j}\cdot\rho_{l,j}) = \dot L_{j+1}\cdot\rho_{l,j+1} - \dot L_j\cdot\rho_{l,j} + \sum_{i=1}^{nS}\dot N_{l,t,i,j}\cdot M_{l,j} \qquad (7)$$

2.5 Mass Transfer and Phase Equilibrium

The molar flow rate which transfers the phase boundary is calculated using the mass transfer resistance on the liquid and the vapour side and the equilibrium concentration at the phase boundary. However the determination of the mass transfer coefficient and of the mass transfer area (especially for tray columns) can be very difficult and the result very inaccurate. Therefore often equilibrium is assumed on every stage. The number n of discrete elements for spray and packed column is then calculated as $n = H/\text{HETP}$, where HETP is the height equivalent to one theoretical plate. For the tray columns, equilibrium is assumed on every tray. The deviation from equilibrium is then taken into account using the Murphree tray efficiency $\eta_{Murphree}$:

$$\eta_{Murphree,i,j} = \frac{y_{i,j} - y_{i,j-1}}{y^*_{i,j} - y_{i,j-1}} \tag{8}$$

The thermodynamic equilibrium at the phase boundary is generally calculated using

$$y^*_{j,i} \cdot p_j \cdot \phi_{v,j,i} = x^*_{j,i} \cdot \gamma_{j,i} \cdot \phi_{l,j,i} \cdot p^{sat}_{j,i} \tag{9}$$

This equation is valid in case the pressure is not extremely high, since the Poyinting-factor is neglected. For the activity coefficient γ up to now Margules two-suffix equation, Wilson-equation and NRTL-equation are implemented. If the fugacity coefficient ϕ_v is not supposed to be 1, it can be calculated using the virial equation or a cubic equation of state after Redlich-Kwong (see for example [10]). In absorption processes the simpler Henry's law is often used. The equilibrium using Henry's law becomes:

$$y^*_{j,i} \cdot p_j \cdot \phi_{v,j,i} = x^*_{j,i} \cdot \text{He}_{j,i} \tag{10}$$

The temperature dependency of the Henry-constant He is taken into account. The user can decide for each component whether equation (9) or (10) is used.

If mass transfer is taken into account the transfer molar flow rate is calculated using the difference between the concentration on the phase boundary and the bulk concentration. The molar vapour flow becomes:

$$\dot{N}_{v,t,i,j} = -k_{v,i,j} \cdot A \cdot (c^{bulk}_{v,i,j} - c^*_{v,i,j}) \tag{11}$$

For the liquid molar flow rate additionally an enhancement factor is introduced:

$$\dot{N}_{l,t,i,j} = -E_{i,j} \cdot k_{l,i,j} \cdot A \cdot (c^{bulk}_{l,i,j} - c^*_{l,i,j}) \tag{12}$$

The enhancement factor E can be used if chemical reaction occurs. If for example the absorbed gas reacts with one or more liquid components, the mass transfer is enhanced due to the reaction. This influence is very important for fast reactions, where the reaction takes place in the liquid film. For slow reactions, which take place in the liquid bulk phase, the mass transport is only influenced indirectly and the enhancement factor becomes 1. The determination of the enhancement factor can become very complex, especially if parallel reactions interact with each other. An overview can be found in [13]. In this library the equations for the following cases are implemented: irreversible first order reaction, pseudo-irreversible first order reaction, reversible reactions of type A ↔ P, A ↔ 2P and A+B ↔ P. Also parallel, non-interacting reactions can be taken into account.

The liquid and vapour mass transfer coefficient for tray columns is calculated by correlations proposed by [12]. The interfacial area can be calculated for drop regime and emulsion regime; for the regime in between an interpolation, also proposed by [12], is used, using the ratio vapour load / maximum vapour load. For packed columns the interfacial area is simply the area of the packing material. The liquid and vapour mass transfer coefficients can either be calculated by the correlation from Onda or by using the Sherwood-number from Wesselingh (see for example [7]).

2.6 Liquid and Vapour Flow Rate

For different column types different equations for the liquid volume flow rate have to be used. The equations are directly implemented in the column model. Tray and packed column account for the case where at simulation start no liquid is in the column. In this case the outgoing liquid flow rate $\dot{L}$ is zero, unless the height of the liquid on the tray gets higher than the weir height (tray column) or the packing material is fully wetted (packed column). For a spray column it is assumed that if liquid enters the column, instantaneously liquid is also leaving. Hereby the liquid volume flow rate may change over the column height, but the number of drops per second remains the same. Therefore the liquid volume flow rate for the spray column is simply:

$$\dot{L}_j = v_{\text{drop},j} \cdot A \cdot \varepsilon_l \tag{13}$$

The velocity of one liquid drop can be obtained by a simple balance of forces on one liquid drop [8] and depends on vapour and liquid density, vapour viscosity, vapour velocity and the diameter of the liquid drop at the inlet. For the tray column the equation from [12] was used:

$$\dot{L}_j = \begin{cases} l_w \cdot \varepsilon_{l,2ph,j} \cdot \left(\frac{(h_j - h_w)\cdot g^{1/3}}{1.45}\right)^{3/2} & \text{if } h > h_w \\ 0 & \text{else} \end{cases} \tag{14}$$

l_w denotes the weir length and $\varepsilon_{l,2ph}$ the liquid fraction in the two-phase regime on the tray.

For packed columns the liquid volume flow rate is 0, as long as the packing material is not yet wetted, otherwise

$$\dot{L}_j = \sqrt{\left(\frac{\varepsilon_{l,j}}{0.555}\right)^3 \cdot \frac{g \cdot \varepsilon^{4.65}}{a}} \cdot A \cdot \varepsilon \cdot \varepsilon_{l,j} \cdot 50 \qquad (15)$$

As for the liquid flow rate, also for the determination of the vapour flow rate different equations for the different column types have to be used. In all cases, the momentum balances are formed across the segment boundaries (staggered grid), as was proposed by [3]. The used pressure drop correlations for the three column types are written below:
Tray column:

$$p_j - p_{j+1} = \left(\frac{\dot{V}_j}{A_{\text{free}}}\right)^2 \cdot \zeta \cdot \frac{\rho_{v,j}}{2} + h_j \cdot \varepsilon_{l,2ph,j} \cdot \rho_v \cdot g \qquad (16)$$

Spray column:

$$p_j - p_{j+1} = \left(\frac{\dot{V}_j}{A_{\text{free}}}\right)^2 \cdot \frac{\rho_{v,j}}{2} \cdot \lambda \cdot \frac{h_j}{d} \qquad (17)$$

Packed column:

$$p_j - p_{j+1} = \frac{1}{8}\zeta \left(\frac{\dot{V}_j}{A_{\text{free}}}\right)^2 \cdot \left(\frac{6\varepsilon_l}{d_L} + a\right) \cdot \frac{\rho_{v,j} \cdot h_j}{(\varepsilon - \varepsilon_l)^{4.65}} \qquad (18)$$

Since the staggered grid approach is used, for the pressure difference $p_{in} - p_1$ and for the pressure difference $p_n - p_{n+1}$ the height of the discrete element, h, is divided by 2.

In order to make things easier for the nonlinear solver, the equations are written in the form $\dot{V}_j = ...$; only for the inlet volume flow rate the equation remains $p_{in} = fkt\left(\dot{V}_{in}, p_1, etc.\right)$.

3 Media Modeling

3.1 Physical and Thermodynamic Property Package

A very important part when modeling separation processes are the medium models. As shown in figure 1, models for a liquid and a gas phase as well as a model for the equilibrium at the phase boundary are necessary.

The *Modelica Standard Library* provides the `Modelica.Media` package. This package provides medium models of ideal mixtures of ideal gases and liquids. If this phase behavior is not adequate for the modeled process other medium models are needed. Since implementing the mediums of interest in Modelica is very time consuming and complex, an external media interface to existing physical and thermodynamic property packages is preferable in such a case. Casella and Richter proposed the ExternalMedia library in order to include external fluid property code, [4]. As an example this library provides the interface to FluidProp of TU Delft [6] which is itself an interface to different medium databases.

The ExternalMedia library however only provides access to multi-phase pure substances but not for multi-phase-multi-component mediums. Since the authors of the ExternalMedia library provided access to the C++ interface, this interface was changed in order to have access to the mixtures also.

Figure 3 shows a class diagram of the modified interface layer consisting of C++ objects. Beside the `TwoPhaseMedium` a `TwoPhaseMixture` object was introduced to handle the extra functionality that is needed to compute the properties of a mixture in contrast to a pure substance. To make the new object to fit in the library structure and to preserve the present interface some of the existing classes had to be changed. Most of these changes were expansions of the former function templates to accept parameter for the mixtures like `nComp` for the number of components. The additional parameters were located at the end of the argument list of each functions and given default values. For example the class `TwoPhaseMedium` could remain unchanged, although arguments were added to functions in `BaseTwoPhaseMedium` allowing `TwoPhaseMixture` to inherit. To access the mixture properties via the `TwoPhaseMixture` object functions were added to the Modelica *ExternalMediaLib* front-end and the C interface layer.

Using two phase mixtures from the *NIST Reference Fluid Thermodynamic and Transport Properties Database* (REFPROP) [9] with the ExternalMedia library by the FluidProp interface shaped up to be quite inappropriate. First the calculation of properties from pressure, enthalpy and composition are quite slow, about 50 ms per function call. This leads due to several hundred to thousand function calls per time step to very large simulation times. Second some functions seem to be discontinuous, which can cause the simulation to crash. Both of these facts are disadvantageous above all in iterations of the non-linear solver.

However it would be a very useful tool not only for this library to have an interface to external media packages which are designed for process simulation.

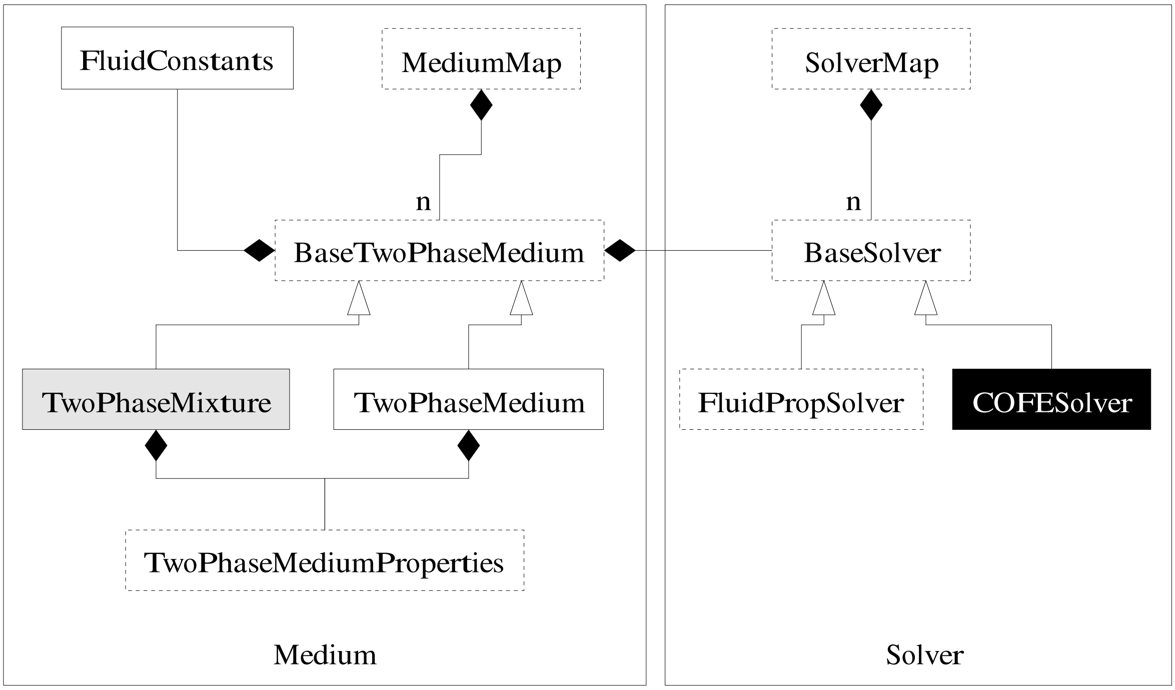


Figure 3: UML class diagram of the changed ExternalMedia C++ layer. A dotted box indicates a rewritten class. A new class is marked with a gray filled box or a black box, if it is in the planning stages.

For this purpose the interface standard *CAPE-OPEN* [1] has been developed by the *CAPE-OPEN Laboratories Network* at the beginning of this decade. [11] describe the use of such an interface to use thermodynamic and property data from an external tool in a Modelica model of a distillation column.

There are several commercial process engineering tools, which use this interface to exchange unit operation models or thermodynamic and physical property packages. One interesting tool is the free-of-charge *COCO* simulation environment [5], which contains *TEA* (*COCO's Thermodynamics for Engineering Applications*). *TEA* contains routines to calculate multiphase mixture properties. It also can access other property packages by the *CAPE-OPEN* interface. So there is the plan to implement a new solver object `COFESolver` to access *TEA* from the ExternalMedia library. This would allow Modelica models to access a wide range of simulation approved thermodynamic and physical property packages.

3.2 Non-equilibrium Modeling

One major problem using property packages of mixtures is that thermodynamic equilibrium is always assumed. That is for a certain composition x_{mix} (see figure 4) the medium is decomposed in two phases with the equilibrium mole fractions x^*_{A} and y^*_{A}. Therefore the thermodynamic properties like for instance the specific enthalpy for liquid and for vapor phase correspond the equilibrium compositions. However as in non-equilibrium models mass transfer is taken into account in one stage the medium is not decomposed in a liquid phase with x^*_{A} and a vapor phase y^*_{A} but in a liquid phase with x_{A} and a vapor phase with y_{A}.

But most media models are not able to provide for instance the specific enthalpy at a temperature ϑ^* and a liquid composition x_{A}, since in equilibrium the medium would not be single phase at these conditions.

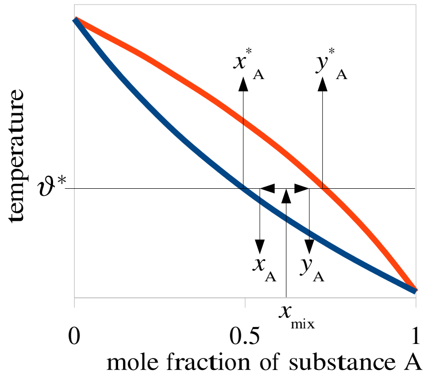


Figure 4: Decomposition of a mixture with mole fraction x_{mix} into two phases with the composition x^*_{A} and y^*_{A} (mass and thermal equilibrium) or x_{A} and y_{A} (only thermal equilibrium between the two phases)

For the current library the problem was solved in a way that the mixing of the different media is performed in the medium interface, so it is independent whether the properties are calculated in a Modelica model or external code. The mixing is performed ideal, where in case of a liquid phase for all components, whose bubble temperature is below the actual temperature, the enthalpy at boiling point is used. i.e. the enthalpy of liquid phase with a composition corresponding to a point in the two phase region x_A is computed the following way:

$$h_l = x_A \cdot h'_A + (1 - x_A) \cdot h_B \tag{19}$$

If the enthalpy of a vapor phase is computed, for all components, whose dew temperature is above the actual temperature, the enthalpy at dew point is used. So the vapor composition results in:

$$h_v = y_A \cdot h_A + (1 - y_A) \cdot h''_B \tag{20}$$

This introduces however an error in the thermophysical properties and a careful failure analysis has to be done before using the results.

4 Example of Use

In this section one example of use is presented. As an example the purification of flue gas of a waste incineration plant was chosen, since in this case media from `ModelicaMedia` can be used after some small modifications. The flue gas consists of a mixture of N_2, O_2, H_2O, CO_2, SO_2, HF and HCl. In a first absorber this flue gas is brought in contact with water in order to primarily remove HF and HCl. This absorber is modelled. The flue gas enters the absorber at the bottom, with a temperature well above 100 °C; the water enters the column at the top, with a temperature around 50 °C. The absorber is a spray absorber, i.e. the necessary contact area is achieved by spraying small droplets of liquid in the air stream. The gases are then absorbed by the water, whereas water evaporates. At the column outlet a part of the liquid is purged, the rest is mixed with pure water and re-used. Such a system was for example also investigated by [8].

All gaseous components, beside N_2 and O_2 dissociate in the liquid phase. In this example, only the dissociation of SO_2 is considered, since here the dissociation is strongest. The reaction equation is as follows:

$$SO_{2aq} + H_2O \Longleftrightarrow H^+ + HSO_3^- \tag{21}$$

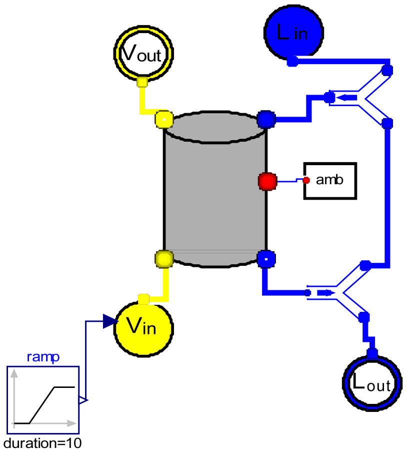


Figure 5: Modelling of a HF-HCl - absorber

For all gaseous components besides H_2O equation (10) is used for the calculation of the phase equilibrium (where the Henry-coefficient was calculated temperature dependent), for water equation (9) is used. The components H^+ and HSO_3^- do only exist in the vapour phase. Please note: not much emphasis was put on the task to gather all necessary parameters of all components (for example the necessary coefficients in order to calculate the temperature dependency of the Henry-constant). In this case reasonable assumptions were used. This example shall show in the first place that the simulation works and that it gives reasonable results.

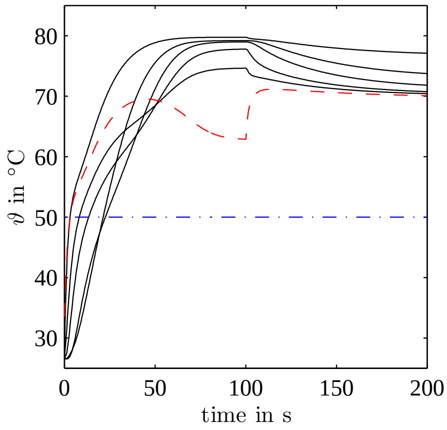


Figure 6: Vapour temperatures in the column (black lines) and temperature of fresh water (blue dot-and-dashed line) and water at column inlet (red dashed line)

In this example the vapour flow inlet is halved at $t = 100\,\mathrm{s}$ once steady state is obtained. Some results are shown in the following figures. Figure 6 shows the vapour temperature some of the stages. The highest

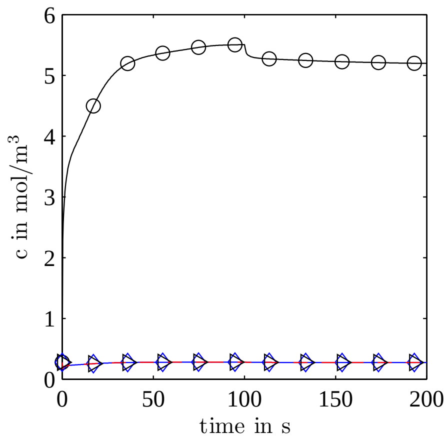


Figure 7: Liquid concentration of SO_2 (black ○), H^+ (blue ◇) and HSO_3^- (red ▷)

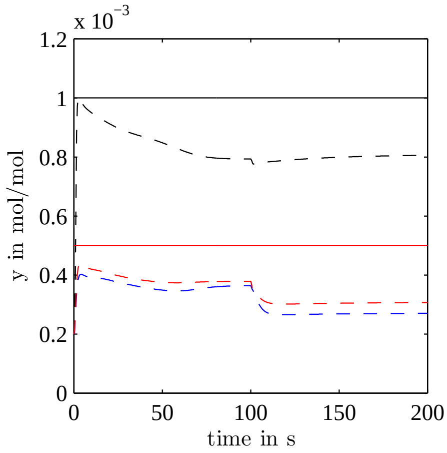


Figure 8: Vapour concentration of SO_2 (black), HCl (blue) and HF (red) at column inlet (solid) and column outlet (dashed)

temperature correspond to stage 1. The vapour enters the column at 144 °C, the fresh liquid water has a temperature of 50 °C (blue dot-and-dashed line) the temperature of the recycled liquid water is denoted by the red dashed line (here, 80 % of the water is recycled). Obviously the vapour temperature goes down when the vapour flow is reduced.

In figure 7 the concentration in the liquid of SO_2, H^+ and HSO_3^- is shown. As can be seen in the reaction equation, the concentration of H^+ and HSO_3^- are equal. The ratio of $c_{H^+} \cdot c_{HSO_3^-}$ to c_{SO_2} correspond to the equilibrium constant of the reaction.

Figure 8 finally shows the vapour concentration of the components to separate at the column inlet (solid line) as well as at the column outlet (dashed line). The concentration of HCl and HF at the column inlet are equal.

5 Conclusion & Outlook

This paper proposed a Modelica-library for the modeling of simulation processes. The different column types can be used: spray, packed and tray column. All columns extend from the same base class where the mass and energy balance is established for the liquid as well as for the vapour phase. This also allows to calculate non-equilibrium states, where the thermodynamic equilibrium only exists at the phase boundary. The molar flow rate which results due to the difference between bulk concentration and the concentration at the phase boundary is calculated using empirical equations for mass transfer coefficients and interfacial area. However it is also possible to use an equilibrium model. In this case the Murphree tray efficiency can be used in order to describe the deviation from equilibrium in tray columns.

In order to avoid the implementation of medium models in Modelica, the `ExternalMedia` from [4] was adapted to account also for multi-phase - multi-component mixtures. Since it turned out that the databases which can be easily accessed by the `ExternalMedia` are not suitable for process simulation, it is proposed to allow the use of the CAPO-OPEN interface standard since this would give access to thermodynamic and physical property packages more suitable for process simulation.

Nomenclature

A	inner cross-sectional area of the column in m^2
A_{free}	solid-free cross-sectional area in m^2
a	specific area in m^2/m^3
c	concentration in mol/m^3
c	heat capacity in J/kg/K
d	diameter of the column in m
E	enhancement factor in
g	gravitational acceleration in m/s^2
$\dot{H}$	enthalpy flow rate in W
h	height of one discrete element in m
h	specific enthalpy in mol/m^3
k	mass transfer coefficient in m/s
$\dot{L}$	liquid volume flow rate in m^3/s

M	molar mass in kg/mol
$\dot{N}$	molar flow rate in mol/s
n	number of discrete elements
nS	number of substances
p	pressure in Pa
$\dot{Q}$	heat flow rate in W
T	temperature in K
t	time in s
u	specific inner energy in mol/m^3
$\dot{V}$	vapour volume flow rate in m^3/s
V	volume of one element in m^3
v	velocity in m/s
x	liquid composition in mol/mol
y	vapour composition in mol/mol

Greek symbols

ε	hold up in mol/mol
ε	void fraction
$\eta_{Murphree}$	tray efficieny of Murphree
γ	activity coefficient
ϕ	fugacity coefficient
ρ	density in kg/m^3
ϑ	temperature in °C
ζ	drag factor

Subscripts

fromL	coming from liquid phase
fromV	coming from vapour phase
i	component i
j	stage j
l	liquid
s	solid
t	transfer
v	vapour

Superscripts

$bulk$	bulk phase
m	molar
sat	saturation
$'$	property at bubble point
$''$	property at dew point
$*$	equilibrium

References

[1] *The CAPE-OPEN Laboratories Network.* `http://www.colan.org/`, visited on August 2009.

[2] Casella, Francesco and Alberto Leva: *Modelica open library for power plant simulation: design and experimental validation.* In *Proceedings Modelica Conference 2003*, pages 41–50, 2003.

[3] Casella, Francesco, Martin Otter, Katrin Prölß, Christoph Richter, and Hubertus Tummescheit: *The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks.* In *Proceedings Modelica Conference 2006*, pages 631–640, 2006.

[4] Casella, Francesco and Christoph Richter: *ExternalMedia: A Library for Easy Re-Use of External Fluid Property Code in Modelica.* In *Proceedings Modelica Conference 2008*, pages 157–161, 2008.

[5] *COCO: cape open to cape open simulation environment.* `http://www.cocosimulator.org`, visited on August 2009.

[6] *FluidProp: Software for the calculation of thermophysical properties of fluids.* `http://fluidprop.tudelft.nl/`, visited on January 2009.

[7] Goedecke, Ralf (editor): *Fluid-Verfahrenstechnik.* Wiley-VCH, 2006.

[8] Malzkorn, Rainer: *Simulation eines Rauchgassprühwäschwers mit aufwärts gerichteten Düsen.* PhD thesis, Ruhr-Universität Bochum, 1999.

[9] *NIST Reference Fluid Thermodynamic and Transport Properties Database*. `http://www.nist.gov/srd/nist23.htm`, visited on June 2009.

[10] Reid and Prausnitz: *The Properties of Gases & Liquids*. McGraw-Hill, 4th edition, 1987.

[11] Sandrock, Carl and Philip L. de Vaal: *Dynamic simulation of Chemical Engineering systems using OpenModelica and CAPE-OPEN*. In Jezowski, J. and J. Thullie (editors): *19th European Symposium on Computer Aided Process Engineering - ESCAPE19*. Elsevier B.V./Ltd., 2009.

[12] Stichlmair, Johann: *Grundlagen der Dimensionierung des Gas/Flüssigkeits-Kontaktapparates Bodenkolonne*. Verlag Chemie, 1978.

[13] Swaaij, W.P.M. van and G.F. Versteeg: *Mass Transfer Accompanied With Complex Reversible Chemical Reactions In Gas-Liquid Systems: An Overview*. Chemical Engineering Science, 47:3181–3195, 1992.

Modeling of Rotary Kilns and Application to Limestone Calcination

Uwe Küssel[1] Dirk Abel[1] Matthias Schumacher[2] Martin Weng[2]
[1]RWTH Aachen University, Institute of Automatic Control
Steinbachstraße 54A, D-52074 Aachen
[2]Aixprocess, Process and Fluid Engineering
Alfonsstrasse 44, D-52070 Aachen

Abstract

This paper presents the one dimensional modeling of rotary kilns used for energy intensive production processes. Raw material is fed into an inclined rotating kiln and heated by counter current gas flow. Chemical reactions take place in the bed of raw material as well as in the gas phase. Heat and mass transfer between the bed and the gas phase are implemented. Also the heat transfer to the environment is taken into account. As a benchmark, the process of limestone calcination is chosen. Results are compared with computational fluid dynamic simulations.

Keywords: $CaCO_3$, calcination, CFD comparison, kilns, limestone, rotary kilns, simulation

1 Introduction

Unhydrated lime is used as a raw material for many products in chemical industry. The limestone calcination, as an energy intensive production process for unhydrated lime, is often performed in continuously operating rotary kilns. Until today, the process is manually operated and despite existing approaches the use of automatic control is very uncommon. Due to the hot and dusty atmosphere inside the drum, thermodynamic states used as controlled variables by an expert system are hardly measurable in a reliable way. In heat driven chemical production processes such as limestone production measured data like a temperature as one thermodynamic variable of the process (often most easily to measure) is not sufficient to estimate the chemical rate of degradation of $CaCO_3$. To overcome this problem dynamic physical and chemical models can be applied to predict the behavior of dependent measures. By comparing the results of the modeling to current plant measurements the states of independent variables become available to close the control loop. To be applicable to an expert system, the very complex model of the process has to be capable of real-time operation anyway and thus, several assumptions and simplifications have to be done.

As a first step to the automatic control of continuously operating rotary kilns, a detailed model of this process is developed. The model abstracts from reality by assuming one dimensional (1-D) counter current flow of the raw material phase (bed phase) and the gas phase. The bed and gas phase are surrounded by a combination of isolating refractory and steel shells. Two reactions are implemented, one for the production of heat and one for the calcination itself. Methane is oxidized in the gas phase in order to supply the necessary energy to the process and limestone is calcinated in the bed phase consuming energy due to an endothermic degradation process. Heat and mass transfer take place between the counter current flows and the environment. The paper is structured as follows: In chapter 2 the abstraction to 1-D counter current flow is shown. Additionally, the mechanisms of reactions and heat transfer are described. In chapter 3 the computational results with Dymola are analyzed in detail. For validation, the results are compared to computational fluid dynamics (CFD) simulations in chapter 4. Chapter 5 concludes the paper with an outlook of future investigation.

2 Modeling of rotary kilns

The rotary kiln is modeled using a 1-D approach. The flows of gas and bed phase are counter current. The rotary kiln itself consists of an isolation and a steel shell. This abstraction is depicted in Figure 1.

DOI: 10.3384/ecp09430084

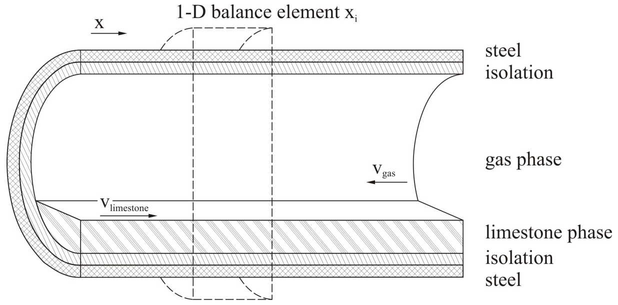


Figure 1: Rotary kiln with 1-D balance element

2.1 Modeling of the gas phase

The gas phase model is based on the DynamicPipe model of the Modelica fluid library [1]. It is extended by dynamically changing cross sectional areas due to bed heights. Replaceable models of radiative and convectional heat transfer are integrated and will be discussed in section 2.6. In addition, the gas phase is equipped with a replaceable chemical reaction model in order to cover the oxidation of mixed gaseous hydrocarbons (i.e. in this case Methane only), details are covered in section 2.5.

The balancing equations with a dynamic momentum balance are formulated in [2]. Using a finite volume approximation and setting the momentum balance to be static, the equations (1)-(3), corresponding to a single slice used in the DynamicPipe model, are derived. To avoid obscurity, additional equations covering multi-component media are not given in this formulation.

$$
\begin{aligned}
\frac{dm_i}{dt} &= \dot m_{i+\frac{1}{2}} + \dot m_{i-\frac{1}{2}} + \cdots && (1)\\
0 &= \frac{\dot m^2_{i+\frac{1}{2}}}{A_{i+\frac{1}{2}} \rho_{i+\frac{1}{2}}} - \frac{\dot m^2_{i-\frac{1}{2}}}{A_{i-\frac{1}{2}} \rho_{i-\frac{1}{2}}} && (2)\\
&\quad + A_i \left(p_{i+\frac{1}{2}} - p_{i-\frac{1}{2}} \right) \\
&\quad - F_{F_{FV}} - A_i \rho_i g \delta z \\
\frac{dU_i}{dt} &= \dot H_{i+\frac{1}{2}} + \dot H_{i-\frac{1}{2}} && (3)\\
&\quad + v_i A_i \left(p_{i-\frac{1}{2}} - p_{i+\frac{1}{2}} \right) + \cdots \\
&\quad \text{for slice } i = 1, \ldots, n
\end{aligned}
$$

Source terms for heat, namely radiational and convectional contributions, are added to the right hand side (RHS) of the energy balance in equation (3). Mass transfer between bed and gas phase due to degradation of $CaCO_3$ (releases CO_2) and changes in the composition of the gas phase due to the oxidation of gaseous hydrocarbons are added to the RHS of the mass balance in equation (1). Including the heat of formation of all components implicitly adds the contribution of chemical reactions to the energy balance while only considering mass balance changes. For the longitudinal gas flow and the heat transfer standard connectors are used. For the gaseous exchange between gas and bed signal oriented connectors for each direction are designed using the same exchange medium (O_2, CO, CO_2, H_2O).

All cross sectional areas, and hence corresponding volumes, in these equations are dynamically changed due to changes in bed height. Therefore, the information of the bed's cross sectional area is transmitted via input output relation to the gas phase. Finally, data on gas phase properties are provided for other components by an output connector.

2.2 Modeling of the bed phase

The bed phase is modeled by mass and energy balances.

$$
\begin{aligned}
\frac{dm_i}{dt} &= \dot m_{i+\frac{1}{2}} + \dot m_{i-\frac{1}{2}} + \cdots && (4)\\
\text{with} &\quad \dot m_{i+\frac{1}{2}} = \rho_i \dot V_{i+\frac{1}{2}} \\
\frac{dU_i}{dt} &= \dot H_{i+\frac{1}{2}} + \dot H_{i-\frac{1}{2}} + \cdots && (5)
\end{aligned}
$$

Heat and mass transfer is included similarly to gas phase modelling via source terms to the RHS of the balance equations. The transport of material through the rotary kiln is specified by equation (6), which is commonly known as Kramers equation [5].

$$
\dot V_{i+\frac{1}{2}} = \frac{4}{3} \pi \omega R^3 \left(\frac{tan\alpha}{sin\beta} - \frac{dh_i}{dx} cot\beta \right) \left(2\frac{h_i}{R} - \frac{h_i^2}{R^2} \right)^{\frac{3}{2}} \quad (6)
$$

The volume flow rate is a function of the rotation frequency ω, the inner radius of the pipe R, the inclination of the kiln α, the materials angle of repose β, the height of material h and the corresponding gradient $\frac{dh}{dx}$. This differential equation in terms of the height is simplified by rearranging and approximating the height and the corresponding gradient with the equations (7) to (9).

$$
\begin{aligned}
h_i &= R - (R \cos(\varphi_i/2)) && (7)\\
\frac{dh_i}{dx} &= (R/2) \sin(\varphi_i/2) \frac{d\varphi_i}{dx} && (8)\\
\frac{d\varphi_i}{dx} &\approx (\varphi_{i+1} - \varphi_i) / (L/n) && (9)
\end{aligned}
$$

The mid point angle of the bed phase is denoted by φ, the length of the rotary kiln by L and n is the number of slices in the rotary kiln. All geometric details for gas and bed phase are calculated using the mid point angle. The calculation of $\varphi_i = f(A_i)$ is presented in section 2.4.

2.3 Modeling of the shells

The isolation and steel shells are modeled by an energy balance for the corresponding material volume resulting from the 1-D balancing elements. Heat transfer, and hence the temperature profile, is realized using Fourier's relation for heat conduction in solid material. In principle, the shells are structured as grids consisting of heat capacitors and thermal conductors likewise modeled in the modelica standard package. Heat transfer is possible in two directions, namely longitudinally and transversely to the flow direction in the rotary kiln. Geometric and material properties are parameters of the model. Standard heat connectors are used.

2.4 Mid point angle approximation

All geometric properties for gas and bed phase can easily be calculated using the mid point angle φ of the bed phase. Nevertheless, the angle has to be calculated from the cross sectional area of the bed phase, which is a direct result of the mass balance in every bed slice of the rotary kiln. This is shown in equation (10).

$$A_i = (m_i/\rho_i) \,/\, (L/n) \tag{10}$$

The relation between the cross sectional area A_i of a single slice and the mid point angle φ_i is implicitly given with the equations (11)-(12).

$$\begin{aligned} 0 &= -P + (\varphi_i - \sin\varphi_i) && (11)\\ P &= 2A_i/R^2 && (12) \end{aligned}$$

In order to avoid additional nonlinear equations, the relationship given with equation (11) is numerically solved for φ in a range of P between $1.25 \cdot 10^{-5}$ and 2π. The corresponding pairs are then used for a least square fit in order to find a good approximation for $\varphi = f(P)$. The function $\varphi = f(P)$ is given with equation (13).

$$\begin{aligned} \varphi &= a_0 \log(P) + a_1\,(1/P) + a_2 P^{\frac{1}{2}} \\ &\quad + a_3 P + a_4 P^2 + a_5 P^3 + a_6 P^4 \\ &\quad + a_7 P^5 + a_8 P^6 + a_9 \end{aligned} \tag{13}$$

The error between the approximation and original data pairs is given in Figure 2.

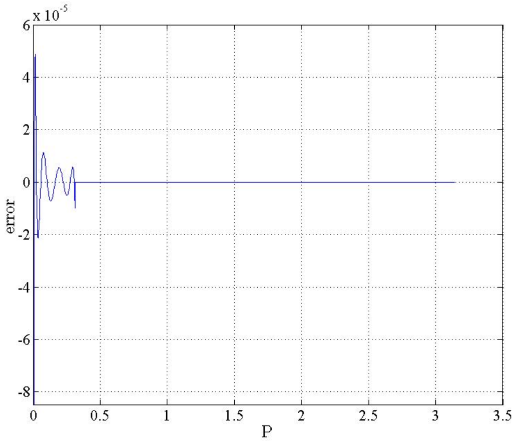


Figure 2: Approximation error for $\varphi = f(P)$

The adapted relationship $\varphi = f(P)$ is implemented as a function and integrated in both the gas and bed phase in order to calculate the mid point angle and subsequently all geometric characteristics of the rotary kiln.

2.5 Mechanism of chemical reactions

The replaceable chemical reaction model is designed as a reactant limited elementary reaction using an Arrhenius type approach for the reaction kinetics. The reaction kinetics approach is shown with the equation (14).

$$k = B \cdot T^n \cdot e^{-\frac{E_a}{R \cdot T}} \tag{14}$$

The preexponential coefficient B, the activation energy E_a and n are free parameters which will be used to fit the model to CFD simulation data. Generally speaking, this fit is necessary since, on the one hand, no reliable chemical reaction data is available for such a set up of coupled ideal reactors and, on the other hand, the model is not able to cope with local conditions due to low resolution and the lack of adequate models. For example, the oxidation of mixed gaseous hydrocarbons (i.e. in this case CH_4 only) is mass transfer limited and therefore, an ideal mixed reactor needs to be adapted using the averaged chemical reaction kinetics.

The rate of degradation (massflow) for each component is calculated using the relationship given with equation (15).

$$\dot{m}_j = kV\lambda_j M_j \sum_{l}^{k=1} C_k^{\lambda_k} \tag{15}$$

In this relationship, the velocity of reaction k is coupled with the volume of the reaction element V, the molar mass M and the stoichiometric cofficient λ for each component (index j over all components). The approach is reactant limited assuming an elementary reaction order. Hence, the molar density C to the power of the stoichiometric coefficient λ sums up over all reactants k of a reaction.

For the bed phase, the degradation of $CaCO_3$ is implemented. The chemical reaction is given with equation (16).

$$CaCO_3 \Longrightarrow CaO + CO_2 \tag{16}$$

Energy is chemically provided by the combustion of CH_4, equation (17) holds.

$$CH_4 + 2O_2 \Longrightarrow CO_2 + 2H_2O \tag{17}$$

In principle, various reactions are possible, as long as the defined medium covers all components and the stoichiometric coefficients are defined.

2.6 Mechanisms of heat transfer

For the heat transfer between the gas and bed phase as well as the isolation shell convectional and radiational mechanisms are implemented. For convectional heat transfer between the gas and its corresponding exchange partner, the heat transfer coefficient α is dependent on the Reynolds number Re, the Prandtl number Pr and the Nusselt number Nu. The general convectional heat transfer equation in terms of α is shown with equation (18).

$$\dot{Q}_{conv\ 1\leftrightarrow 2} = \alpha_{12} A_{12} (T_1 - T_2) \tag{18}$$

Various heat transfer models for the gas phase solid interaction are implemented in the Modelica fluid library [1]. Additionally, two relationships for Nusselt numbers are given in [10]. The equations are shown with (19) and (20).

$$\begin{aligned} Nu &= \frac{\xi/8\,(Re-1000)\,Pr}{1+12.7\sqrt{\xi/8}\,\left(Pr^{2/3}\right)}\left[1+\left(\frac{D_h}{L}\right)^{\frac{2}{3}}\right] && (19) \\ \xi &= (1.82\cdot\log_{10}(Re)-1.64)^{-2} && (20) \\ & \quad 2300 < Re < 10^6,\ \ D_h/L < 1 \end{aligned}$$

Within small deviations in the range of Pr, $0.5 < Pr < 1.5$, the simplified equation (21) can be used.

$$Nu = 0.0214\left(Re^{0.8}-100\right)Pr^{0.4}\left[1+\left(\frac{D_h}{L}\right)^{\frac{2}{3}}\right] \tag{21}$$

All the models calculate α in a similar range from $8-10\ \mathrm{W/m^2K}$.

Convectional heat transfer between the bed phase and the isolation is also realized with the heat transfer coefficient α and equation (18). α is calculated from various process parameters using two different approaches which are described in detail in [8]. Typical values are said to be $50 \le \alpha \le 200\ \mathrm{W/m^2K}$

Also, models with constant α are possible to choose. The heat transfer models are designed to be replaceable in order to enable the choice between different heat transfer mechanisms as described above.

Radiation between the bed, the gas and the isolation is modeled using equation (22).

$$\dot{Q}_{rad\ 1\leftrightarrow 2} = \varepsilon_{12} A_{12} \sigma \left(T_1^4 - T_2^4\right) \tag{22}$$

The referenced area for radiational heat transfer between transfer object 1 and 2 is given with A_{12}. The emissivity coefficient ε_{12} relates the visibilty between area A_{12} and A_{21} and the emissivity of corresponding objects. There are three emissivity coefficients to be calculated, namely wall → bed (wb), wall → gas (wg) and bed → gas (bg). The formulas for calculating these coefficients are given with equation (23) to (26).

$$\begin{aligned} \varepsilon_{wb} &= \frac{\varepsilon_w \varepsilon_b \left(1-\varepsilon_g\right)}{U} && (23) \\ \varepsilon_{wg} &= \frac{\varepsilon_w \varepsilon_g \left(1+\Phi\left(1-\varepsilon_g\right)\left(1-\varepsilon_b\right)\right)}{U} && (24) \\ \varepsilon_{bg} &= \frac{\varepsilon_b \varepsilon_g \left(1+\Phi\left(1-\varepsilon_g\right)\left(1-\varepsilon_w\right)\right)}{U} && (25) \\ U &= 1-\left(1-\varepsilon_g\right)\left(1-\varepsilon_w\right) \\ & \quad \left(1-\Phi\left(1-\left(1-\varepsilon_b\right)\left(1-\varepsilon_g\right)\right)\right) && (26) \end{aligned}$$

The variable Φ is defined as the ratio between the free bed area (surface) and the free isolation area. Details about the derivation can be found in [4]. The values of emissivity are almost constant over length x and temperature T with $\varepsilon_{wb} = 0.184$, $\varepsilon_{wg} = 0.695$ and $\varepsilon_{bg} = 0.615$.

3 Computational results

The model is designed and numerically integrated within the Dymola modeling environment. The set up is depicted in the Figure 3.

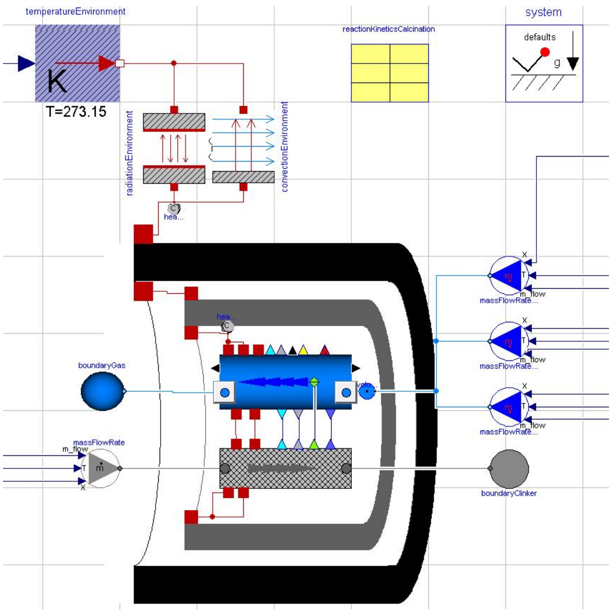


Figure 3: Dymola model of a rotary kiln

The black and the grey objects denote the steel and isolation shell, respectively, with different geometric and material parameters. Inside the pipe, there is counter current flow of the gas phase (blue element) and the bed phase (limestone material, grey element). Different mechansims for heat and mass transfer as well as chemical reactions are chosen as described earlier in this paper. Further components, e.g. crust of semi liquefied limestone, can be added but are not modeled in this contribution.

The calcination process (degradation of $CaCO_3$) is shown in Figure 4.

The system is initialized at 273.15 K and atmospheric pressure (time [s]: 0-3000). In a next step, the system is brought to the operating point by increasing temperature of the incoming mass flows of the bed and the gas phase (time [s]: 3000-13000). This procedure is followed by the intial increase of the combustion gas massflow (time [s]: 13000-250000, massflow step [kg/s]: 0.1-1.2692). It is clear to see that the rate of $CaCO_3$ degradation intensifies as the temperature increases. A second step to the combustion gas massflow is applied after 300000 seconds (massflow step [kg/s]: 1.2692-2.5392). Again, the degradation of $CaCO_3$ increases. The increase of degradation, while applying these steps to the combustion gas flow, is due to the increase of temperature and the endothermic character of the calcination reaction. The increase of temperature for the main components of the rotary kiln (steel, isolation, gas and bed phase) is depicted in Figure 5

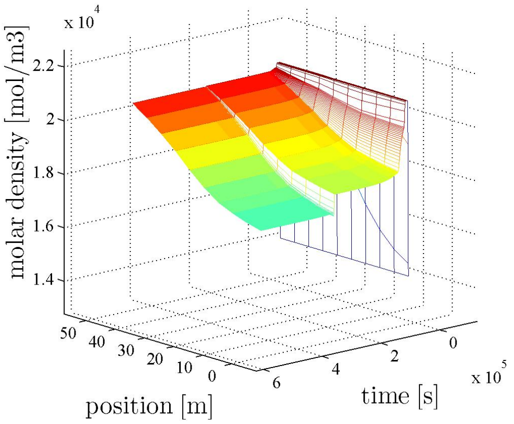


Figure 4: Degradation of $CaCO_3$ through calcination

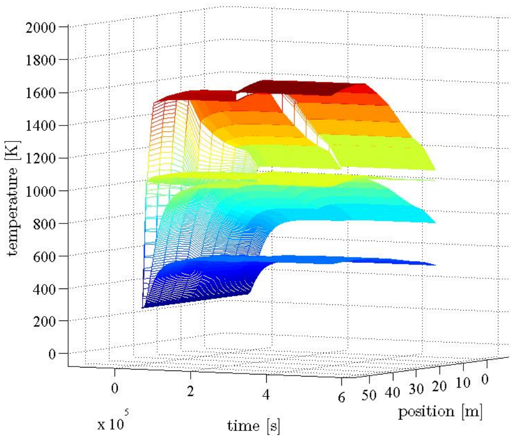


Figure 5: Temperature of steel, isolation, bed and gas phase

While the gas phase (red) responds extremely fast to the step in combustion gas massflow, the bed phase (yellow) is slower. The isolation (turquoise) and steel (blue) shell are even slower in their step response. The different time constants of the process can be recognized by an exemplary inspection of Figure 6 to 7.

It is easy to observe the small time constant of the gas response lying in the range of 10-200 seconds.

For the isolation shell, the time constant of reponse lies in the range of 20000-30000 seconds. This explains the stiffness of the system and the necessity of stiff numerical solvers for simulations. Furthermore, these time constants correspond to the known time constants from process industry, which implies that

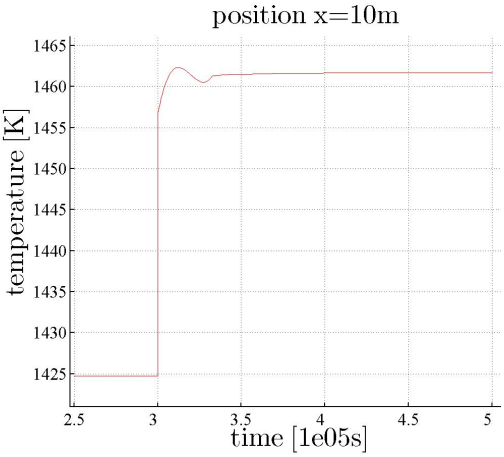


Figure 6: Temperature response of gas phase due to combustion gas step

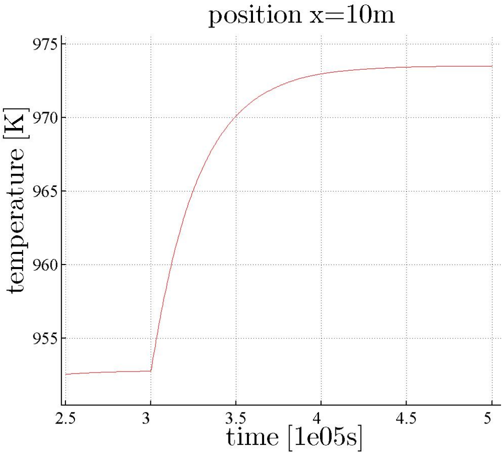


Figure 7: Temperature response of isolation due to combustion gas step

the dynamical behaviour of the model reliably represents the calcination process. Although the steady state values of the operating point also correspond to the known values from process industry, this modeling approach is justified even more by comparing its values to CFD simulation results in the following chapter.

4 Comparison to CFD simulations

Computational fluid dynamics (CFD) is a widely accepted tool for the detailed description of gaseous combustion phenomena occurring within a rotary kiln [6],[3]. Unfortunately, it does neither allow for the modeling of transport of the solid bed nor for the theoretical description of the chemical reactions therein. In the current work, different sub-models have been integrated in the commercial CFD code Fluent in order to model the dynamics of the granular flow of the limestone particles and the coupling of gas and solid phase. Since this work mainly focuses on the 1-D model of the limestone calcination process, only a short summary of the models used in the CFD simulations will be given here.

4.1 Introduction of the CFD rotary kiln model

The three dimensional computational domain of the freeboard in the kiln is bounded by the refractory and the surface of the solid bed. The methane burner extends into the kiln on one end of the drum (Figure 8).

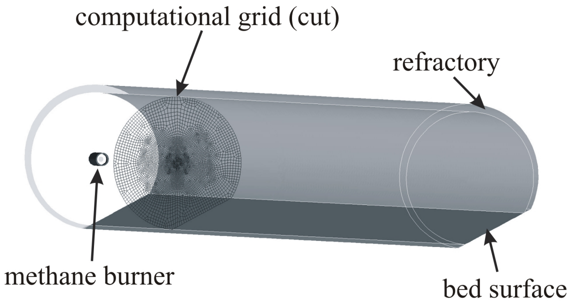


Figure 8: 3-D computational domain of the limestone rotary kiln

Steady state conservation equations of the compressible gas flow are solved considering the realizable k-ε-model for turbulent effects in the gas phase. Transport of four separate species (O_2, CH_4, CO_2 and H_2O) is calculated explicitly in the gas phase, while the fifth one (N_2) sums to unity. Combustion rates of methane oxidization are assumed to be limited by the mixing of turbulent eddies and thus, the eddy-dissipation model is used to predict reaction rates of the volumetric reactions in the gas phase. Energy transport phenomena include conduction, convection and radiation (P-1 model), while the latter plays the major role in rotary kiln processes. Absorption of the gas mixture is mainly affected by the product constituents of the gas (CO_2 and H_2O). Therefore the wsggm-cell-based model is applied to calculate the local absorption coefficient of the gas mixture. A significant amount of energy discharges into the environment. Conduction driven heat transfer through the refractory is being calculated explicitly by applying a computational grid to this region also. On the outer surface of the kiln two heat transfer mechanisms are considered: convection and radiation.

Thermal coupling between gas and solid phase is realized by setting a temperature profile on the bed surface. For this purpose mass, species and energy conservation equations within the particulate solid bed are solved by implementing a mathematical submodel for the solid bed. Since it has recently been proven, that axial mixing can be neglected in industrial rotary kiln processes, transport of the solid bed can be simplified by assuming a plug flow [9]. Additionally the bed is assumed to be well mixed locally in any given cross section.

Limestone calcination can be modeled as a shrinking core process with surface reaction control according to equation (27) [7].

$$\frac{dm_{CaCO_3}}{dt} = -k_0 \exp\left(-\frac{E_A}{RT_p}\right) \cdot 4\pi r_p^2 N_p M_{CaCO_3} \tag{27}$$

Herein r_p and N_p represent the mean particle radius and the total number of limestone particles, respectively. CO_2 released by the calcination reaction is assumed to be transported instantaneously to the hot flow by neglecting any transport resistance in the solid bed. It is released in the computational space of the hot gas flow by adding mass and species sources in the cells adjacent to the bed surface.

In Figure 9 the contour plot of the local gas temperature on the center plane is shown as an exemplary result of the CFD simulations.

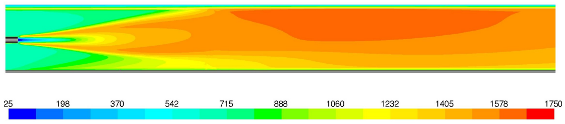


Figure 9: Contour plot of the local gas temperature on the center plane in [K]

In order to justify the 1-D abstraction of the process modeled within the Dymola environment, the computational results are compared to the highly resolute results of the CFD simulations. For this purpose, volume based average values $\bar{\Theta}_{V,j}$ of the CFD simulation data are calculated for a discrete number of slices in the gas phase. Due to the strongly anisotropic flow field in the vicinity of the burner mass and velocity weighted average has to be applied for the potential variables. This is shown in equation (28).

$$\bar{\Theta}_{V,j} = \frac{\sum_{N_{cells}} \Theta_i \rho_i A_i v_{i,ax}}{\sum_{N_{cells}} \rho_i A_i v_{i,ax}} \tag{28}$$

In equation (28), V_i as the volume of each single cell inside the discrete slice and $v_{i,ax}$ as the local axial velocity of the gas mixture are used to calculate the volume based average of slice j for each scalar Θ.

Since plug flow has been assumed in the clinker bed, equation (28) can be simplified to the volume weighted average for all potential variables in the solid phase as given in equation (29).

$$\bar{\Theta}^s_{V,j} = \frac{\sum_{N_{cells}} \Theta_i V_i}{\sum_{N_{cells}} V_i} \tag{29}$$

In contrast, flow variables such as reaction rates need to be integrated within each single control volume in the gas and in the solid domain, respectively. For these data equation (30) is valid.

$$\Theta_{V,j} = \sum_{N_{cells}} \Theta_i \tag{30}$$

4.2 Comparison

Geometrical parameters and boundary conditions are chosen to be equal in both simulation environments. In addition, the free chemical reaction parameters (B, n, E_a) for adapting the model are fitted to CFD simulation data by applying a least square fit using equation (15). Therefore, the CFD simulation data are concentrated in larger 1-D cells by applying the above described weighting functions. After model adaption via least square fit, a Dymola simulation to steady state is performed. The simulated data from the Dymola simulation environment are then compared to the weighted, concentrated CFD simulation data. The essential values of the process are compared. This includes temperature profile of gas and bed phase, molar densities of O_2 and CH_4 in the gas phase as well as molar density of $CaCO_3$ in the bed phase (controlled variable for future model predictive control).

Figure 10 shows the temperature profiles of the gas phase for Dymola and CFD simulation.

In the next figure (Figure 11), the consumption of O_2 and CH_4 is depicted.

The figures show a similar behaviour of the gas phase in both simulation environments. For the error e holds $e < 2\%$. On first inspection, this seems to be a good result. Nevertheless, the more relevant data for intended controlling of the rotary kiln process is the temperature and the concentration of components in

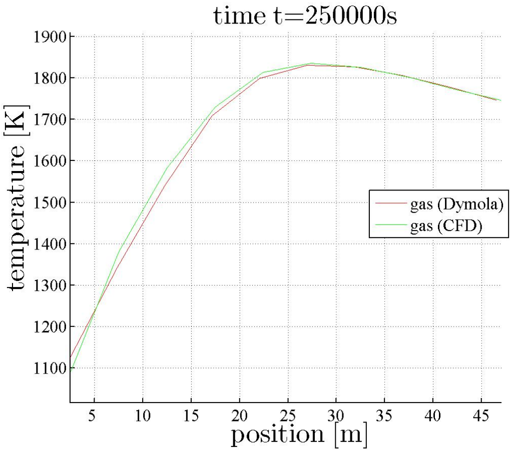


Figure 10: Comparison of gas phase temperature

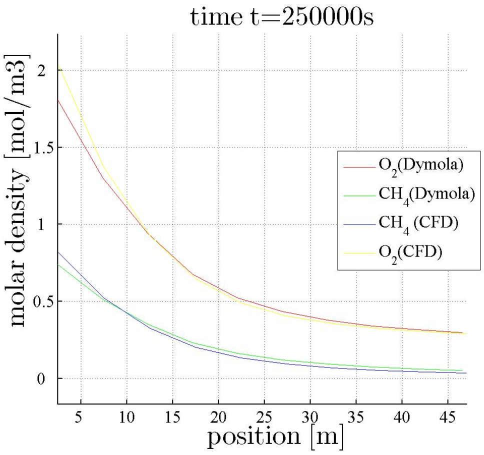


Figure 11: Comparison of gas phase molar density of O_2 and CH_4

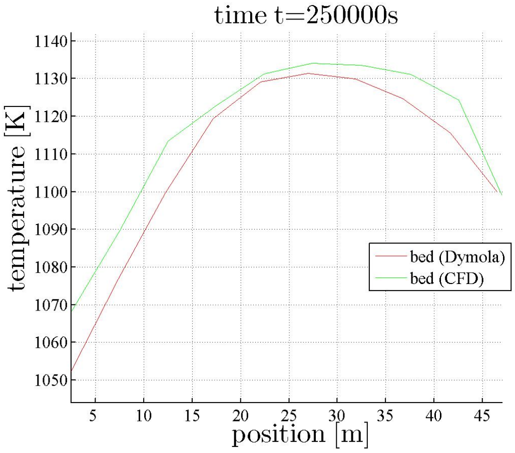


Figure 12: Comparison of bed phase temperature

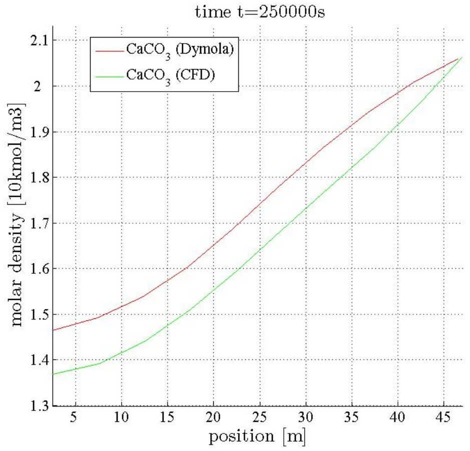


Figure 13: Comparison of bed phase molar density of $CaCO_3$

the bed phase. Figure 12 shows the temperature of the bed phase.

Despite the different scaling of the graph, for the error e holds $e < 3\%$. The next figure (Figure 13) shows the degradation of $CaCO_3$ in the bed phase using the molar density as depicted variable. The maximum error for the molar density of $CaCO_3$ is $e = 6.6\%$. Although this value is slightly higher than the previous errors, the 1-D modeling approach is successfully applied. Various alternative values of the process were compared, yielding to the same outcome. The main advantage of the 1-D modeling approach is the possiblity of faster simulations while deviations are kept small enough in the scope of robustness in terms of observer based model predictive control. While the simulation until convergence in the CFD environment takes up to several days, the simulation of the Dymola model takes seconds to minutes depending on the change of input variables. Another enormous advantage of the 1-D modeling approach is the use of the 1^{st} principles for physical abstraction. Since the identified data based models are only valid in the range of measured data, the nonlinear 1-D model of the rotary kiln will have a wider range of validation. Furthermore, the simulated data sets are always open to physical interpretation and hence, are easier to check for plausibility.

5 Outlook

In a first step, the complex counter current flow process of rotary kilns is modeled and applied to the limestone degradation. First results show reasonable behaviour of the process physics. The comparison to CFD simulation data confirms these results. The essential process parameters are compared and the errors are small enough to allow for application in observer based model predictive control application. In addition, the modeling approach has a lower computational burden and results in linearizations which are capable of real-time application in an expert system.

For the future, details in both models will be refined in order to get even better results. Furthermore, the model is going to be enhanced with a particle burner and applied to the chemically more complex cement production process. The task of automatic least square fit for the free chemical parameters of the model will be enhanced in order to cover the more complex cement production process. A detailed report on both topics will soon be published. Alongside with the enhanced model, linearizations of the nonlinear model in desired operating points will be derived. Using these linearizations, an observer for immeasurable process values will be established. Furthermore, a model predictive control will be designed in order to hold a desired operating point of the plant.

References

[1] F. Casella, H. Tummescheit, and M. Otter. Modelica fluid library www.modelica.org/libraries/ modelica_fluid/releases/1.0, 2009.

[2] H. Elmqvist, H. Tummescheit, and M. Otter. Object-oriented modeling of thermo-fluid systems. Modelica Association, November 2003.

[3] M. Georgallis, P. Nowak, M. Salcudean, and I.S. Gartshore. Modelling the rotary lime kiln. *Canadian Journal of Chemical Engineering*, 83(2):212–223, 2005.

[4] R. Jeschar, R. Alt, and E. Specht. *Grundlagen der Wärmeübertragung*. Viola-Jescher Verlag, 1990.

[5] H. Kramers and P. Croockewit. The passage of granular solids through inclined rotary kilns. *Chemical Engineering Science*, 1(6):259–265, 1952.

[6] F. Marias, H. Roustan, and A. Pichat. Modelling of a rotary kiln for the pyrolysis of aluminium waste. *Chemical Engineering Science*, 60:4609–4622, 2005.

[7] K.S. Mujumdar, K.V. Ganesh, S.B. Kulkarni, and V.V. Ranade. Rotary cement kiln simulator (rocks): Integrated modeling of pre-heater, calciner, kiln and clinker cooler. *Chemical Engineering Science*, 62:2590–2607, 2007.

[8] A. Queck. *Untersuchung des gas- und wandseitigen Wärmetransportes in die Schüttung von Drehrohröfen*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, 2002.

[9] R.G. Sheritt, J. Chaouki, A. Mehrotra, and L. Behie. Axial dispersion in the three-dimensional mixing of particles in a rotating drum reactor. *Chemical Engineering Science*, 58(2):401–415, 2003.

[10] VDI-Gesellschaft Verfahrenstechnik und Chemieingeniuerwesen, editor. *VDI-Wärmeatlas*. Springer, 8th edition, 1997.

6 Acknowledgement

The authors gratefully thank the Federal Ministry of Education and Research for funding BMBF 3257 MoProOpt. Furthermore, we highly appreciate the supervision by the Project Management Agency Forschungszentrum Karlsruhe (PTKA).

Implementation of an Extended Vehicle Model Architecture in Modelica for Hybrid Vehicle Modeling: Development and Applications

John Batteh Michael Tiller
Emmeskay, Inc.
Plymouth, Michigan USA
jbatteh@emmeskay.com mtiller@emmeskay.com

Abstract

This paper outlines the development and implementation of a vehicle model architecture for hybrid vehicle modeling. The architecture is based on the VehicleInterfaces library with significant extensions to enable more flexible, configurable implementations for hybrid vehicle applications. Additional elements are added to the interfaces and architecture to allow more flexible electrical system modeling and more detailed thermal modeling. Four different hybrid vehicles are implemented as sample applications using the newly-developed architecture. The scheme and canonical library structure for the component, subsystem, and system models is also discussed to document a mechanism for user-friendly handling of parameterized models and fully-implemented models in a complex model architecture with extensive model data. Models and simulation results are shown for the Toyota Prius, Lexus RX400h, a concept hybrid sedan, and a concept hybrid sport utility vehicle (SUV). Extensions to VehicleInterfaces are also proposed to enhance the library to include additional features to improve support for future conventional and hybrid vehicle modeling efforts.

Keywords: hybrid vehicles; vehicle modeling; model architecture; VehicleInterfaces

1 Introduction

Since the introduction of the Toyota Prius in the U.S. in 2000, hybrid vehicles have been gradually gaining acceptance in the U.S. as more consumers become aware of fuel economy and the effect of atmospheric CO_2 on climate change. While existing tax credits and government incentives have provided some stimulus for hybrid vehicle purchases, the overall share of hybrid vehicles in the light duty segment is still less than 2% as shown in Figure 1. However, hybrid vehicle share is expected to increase substantially over the next 10 years as more manufacturers introduce hybridized vehicles. The share of hybrid vehicles is projected to reach nearly 9% in 2015 as shown in Figure 1.

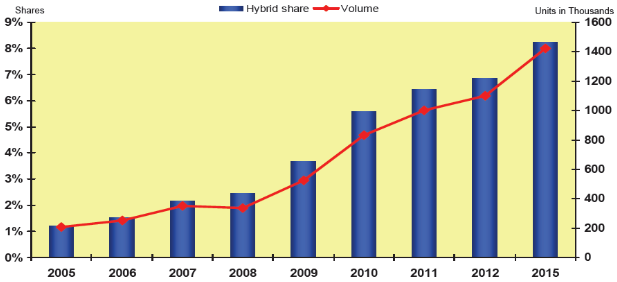


Figure 1. Projected US hybrid vehicle sales

Given the accelerated introduction of hybrid vehicle models over the next several years, there is an increasing need to develop analytic tools to reduce development time for these vehicles which are significantly more complex than conventional vehicles. These analytic tools can be used to assess the impact of different hybrid architectures, size/design the components, perform tradeoff and robustness studies, provide component specifications based on vehicle targets, and develop/optimize the control strategy and subsequent calibration to balance vehicle attributes.

Modelica has been used extensively for vehicle system modeling [2]-[6]. With a growing list of commercial, free, and internally-developed OEM proprietary model libraries, the need for a unifying vehicle model architecture was quickly realized. The purpose of a standardized model architecture is to provide consistent interfaces and system decomposition to promote plug-n-play interoperability between libraries. The first vehicle modeling architecture in Modelica was VMA [7]. Released in 2003, VMA was based on a Ford-internal architecture. After additional feedback from library vendors and end users, VMA was subsequently modified and released as the VehicleInterfaces library in 2006 [8]. The objective of VehicleInterfaces is to provide an open architec-

DOI: 10.3384/ecp09430109

ture to support configurable modeling of both conventional and hybrid vehicles. The library has been used as the starting point for several vehicle modeling applications [6] and is still under development.

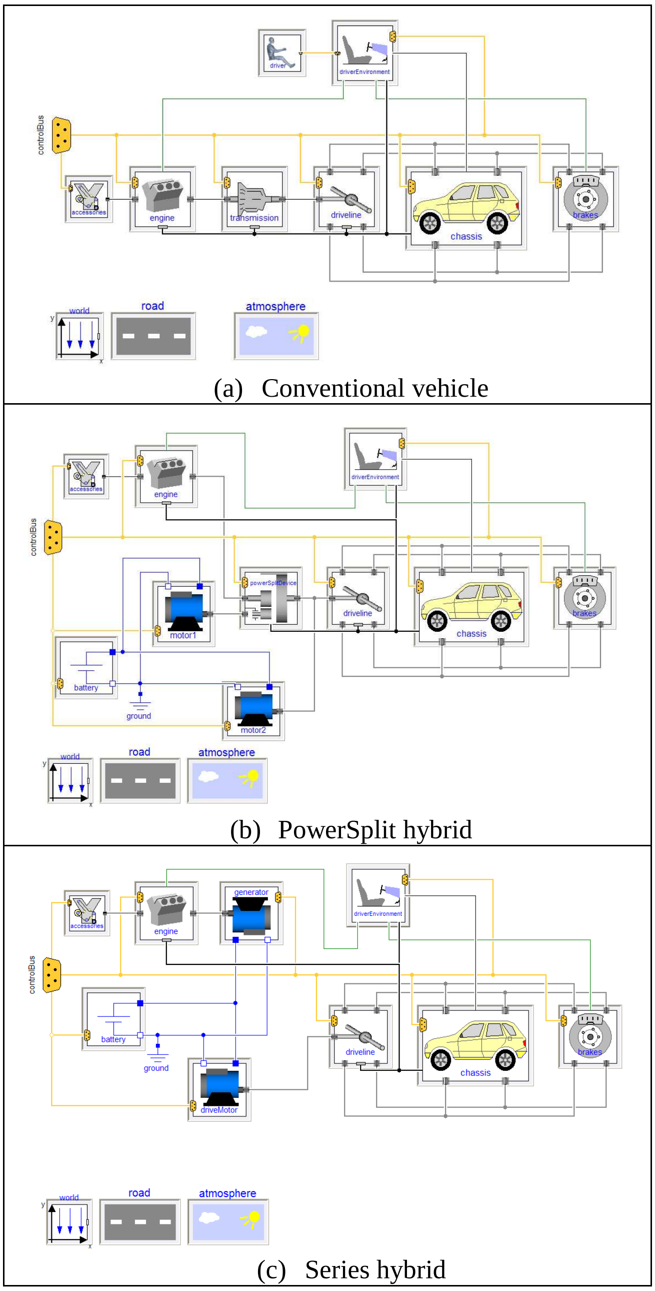
This paper outlines the development and implementation of an extended vehicle model architecture based on VehicleInterfaces with additional enhancements to better support hybrid vehicle modeling. Extensions have been made to the interfaces and additional components added to the architecture to enable more flexible, configurable implementations for hybrid vehicle applications. Four different hybrid vehicles, namely the Toyota Prius, Lexus RX400h, a concept hybrid sedan and SUV, are implemented as sample applications using the newly-developed architecture. Sample drive cycle simulations are shown for the four vehicles. The scheme and canonical library structure for the component, subsystem, and system models is also discussed to document a mechanism for user-friendly handling of parameterized models and fully-implemented models in a complex model architecture with extensive model data. Finally, extensions to VehicleInterfaces are proposed to enhance the library for future conventional and hybrid vehicle modeling efforts.

2 Architecture Development

2.1 VehicleInterfaces Examples

The VehicleInterfaces library includes example model architectures for many different types of vehicles, including conventional and hybrid vehicles. Example architectures from VehicleInterfaces 1.1 are shown in Figure 2 for a conventional (a), PowerSplit hybrid (b), and series hybrid (c) vehicle.

While the conventional vehicle architecture seems quite suitable, the two hybrid vehicle architectures do not appear to offer a similar system decomposition to enable modeling flexibility at the system level. In particular, these example hybrid architectures do not appear to implement a formal electrical subsystem nor are the elements of the hybrid drivetrain grouped at the subsystem level. These features are required to support plug-n-play modeling at the system level with model components of varying level of detail. It should be noted that the hybrid vehicle architectures are appropriate for some model implementations but simply may not provide enough flexibility for models of varying level of detail with minimal changes to the top-level architecture.



(a) Conventional vehicle

(b) PowerSplit hybrid

(c) Series hybrid

Figure 2. Example architectures for conventional and hybrid vehicles from the VehicleInterfaces library

2.2 New Architecture

Given the observations noted in the previous section regarding the example architectures in Vehicle Interfaces 1.1, a new architecture was developed based on the following design criteria:

- Extension from VehicleInterfaces design to maximize compatibility with existing model libraries
- Single model architecture that supports both conventional and hybrid vehicle models
- Additional support for electrical and thermal systems

To meet the design criteria above, the extended vehicle architecture shown in Figure 3 was developed. There are several interface models from the VehicleInterfaces library which required little or no modification. These models include the driver, world, road, and atmosphere components. The remaining interfaces are either modified or newly-added and will be discussed in detail next.

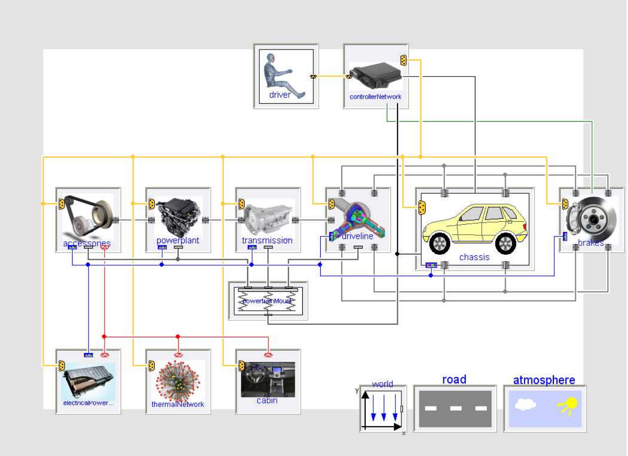


Figure 3. Model architecture

To support the proliferation of electrical components throughout modern vehicle subsystems, an electrical bus connector was added to the accessories, powerplant, transmission, driveline, chassis, and brakes subsystems. The electrical bus is an expandable connector that supports both single and multivoltage representations of the vehicle electrical system. Note that it is not required to terminate the electrical connection in component implementations which do not interact with the electrical system. As a result, no special provisions must be made for handling electrical connections in subsystem models that do not interact with the electrical bus.

In an effort to formalize the electrical subsystem, a new component is added for the electrical power network. The electrical power network is meant to represent the source of electrical power for the vehicle. Implementations of this subsystem could include a single battery, multiple batteries, power converters, and other components that provide and transform electrical power for use by the other subsystems.

A thermal bus was added to several components in the architecture. The thermal bus is also implemented as an expandable connector. The thermal bus was added to the electrical power network and accessory subsystems to facilitate modeling of the HVAC system for both vehicle and electrical system cooling. It should be noted that the thermal bus could also be added to the other vehicle subsystems to support thermal modeling of the engine, transmission, driveline, chassis, and brakes as shown in Section 6. A new cabin component was added to support thermal modeling of the cabin environment. A new thermal network component was added to provide the thermal linkages between the various interacting thermal components. These linkages could include cooling provided from the HVAC components in the accessories to the electrical power network and cabin components, thermal pathways between the electrical power network and the cabin, and thermal linkages between the vehicle and external environment. The addition of the thermal network component provides additional flexibility to modify the thermal routing between components without requiring modification of the models that implement the thermal capacitances.

The design of the electrical and thermal networks decouples the mechanical, electrical and thermal architectures. In this way, the electrical power and thermal network subsystem models allow complete different architectures for those subsystems to be implemented in a way that is orthogonal to the mechanical architecture.

With the ability to internally ground the reaction torques in the various component models in the Modelica Standard library, the impact of the various models on the powertrain mounts is often easily overlooked. Thus, powertrain mounts were also added to the vehicle architecture to encourage consideration of the impact of the drivetrain on the mounting system.

To support modeling of the vehicle control strategy, a controller network component was added to the vehicle architecture. The controller network can support both a single and distributed controller architecture as shown in the interfaces in Figure 4. Note the vehicle system controller which interacts with the driver interface and component controllers. Sample component controllers are engine, transmission, battery, driveline, climate control, motor, generator, *etc.* depending on the vehicle architecture. The functional form of these controllers is flexible enough that they can be mapped to hardware control units if desired. The controller network interface is flexible and configurable to allow the addition of other controllers, implementation of controllers of varying levels of detail, and controller implementations natively in Modelica along with external implementations such as C code and Simulink.

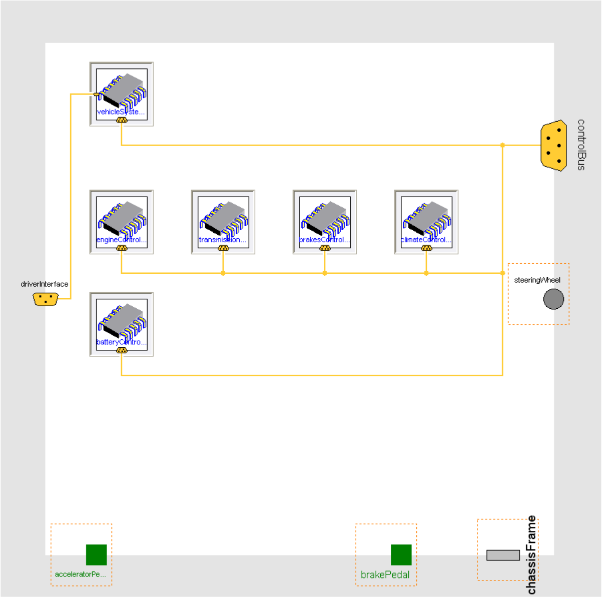


Figure 4. Distributed controller network interfaces

3 Canonical Library Structure

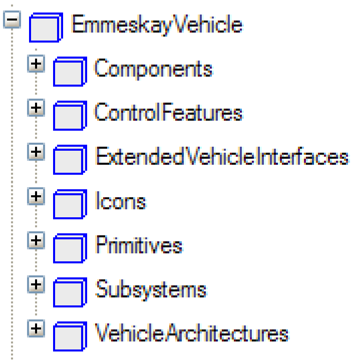
Despite the formal Modelica language features for model configuration, managing model variants and parameter data is a challenge in complex, hierarchical models. The challenge exists not only for the initial library developer but also subsequent model developers and end users. This section describes a canonical library structure implemented as part of the vehicle architecture and implementation effort. This structure was implemented in an effort to satisfy the needs of the model developer while balancing usability concerns for the end user. The guiding principles behind this structure are as follows:

- Promote object-oriented modeling of plant and controller subsystems by composition from reusable, parameterized components
- Provide a model package structure consistent with the model architecture and within which it is easy to find existing models and place new models
- Parameterize models at all levels (subsystem, component, and primitive) to promote model reuse
- Clearly separate generic, parameterized models from specific model implementations
- Implement a data model that preserves the integrity of parameter data throughout the model life cycle

The key design element of the canonical library structure is the separation and clear distinction between parameterized models and model implementations. Parameterized models include all relevant equations for simulation but do not specify any design parameter values. Model implementations extend from the parameterized models and provide the parameter design values. In this structure, explicit model implementations exist as named, fully-specified entities in the package hierarchy rather than *ad hoc* implementations created by specifying parameter values at instantiation. The advantages of named model implementations are as follows:

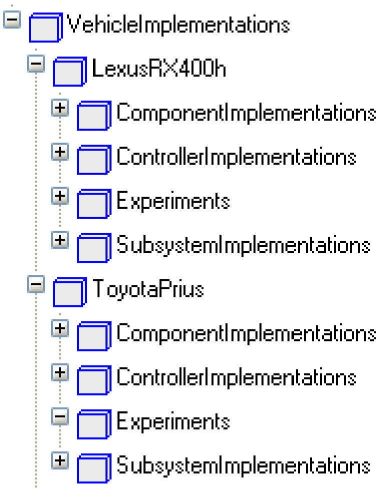
- No need for separate data package hierarchy as parameter data is specified directly in model implementations
- Implemented models clearly separated for model users
- Fully specified implementations consistently used in architecture, component tests, *etc.* without requiring any additional data to be provided by user
- Parameterization clearly identified as a task at creation of implementation model and not model instantiation
- Integrity of parameter data in model implementations can be maintained based on design choices initiated by model developer
- Implementations offer true plug-n-play capability in architecture without requiring subsequent modifications, thus integrating nicely with the replaceable concept in Modelica and tool implementations including multiple redeclares

The following figures show a sample implementation of the canonical library structure. Figure 5 shows the top level package structure which contains the interfaces package and the packages for the parameterized models. These packages can include additional subpackages to further classify the parameterized models. Note that these packages do not contain any implementations. Figure 6 shows the vehicle implementations package with implementations for the Prius and Lexus RX400h. An exploded view of the Prius implementation package is shown in Figure 7.



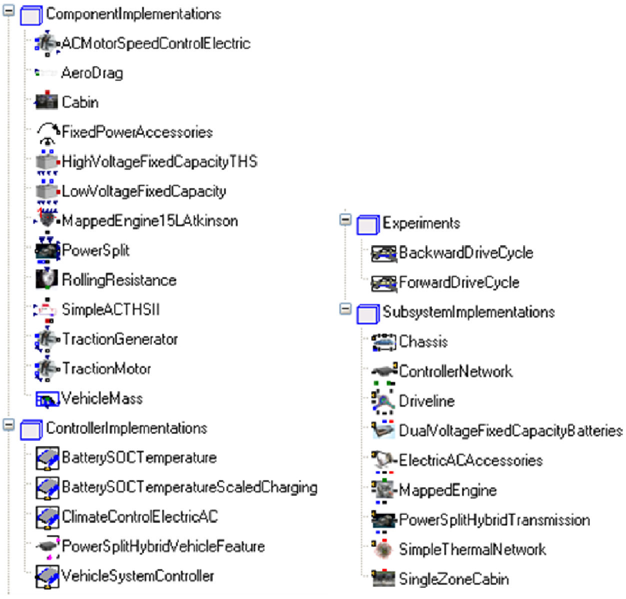
⊟ EmmeskayVehicle
- ⊞ Components
- ⊞ ControlFeatures
- ⊞ ExtendedVehicleInterfaces
- ⊞ Icons
- ⊞ Primitives
- ⊞ Subsystems
- ⊞ VehicleArchitectures

Figure 5. Top level package structure



- VehicleImplementations
  - LexusRX400h
    - ComponentImplementations
    - ControllerImplementations
    - Experiments
    - SubsystemImplementations
  - ToyotaPrius
    - ComponentImplementations
    - ControllerImplementations
    - Experiments
    - SubsystemImplementations

Figure 6. Vehicle implementations package structure for Toyota Prius and Lexus RX400h



- ComponentImplementations
  - ACMotorSpeedControlElectric
  - AeroDrag
  - Cabin
  - FixedPowerAccessories
  - HighVoltageFixedCapacityTHS
  - LowVoltageFixedCapacity
  - MappedEngine15LAtkinson
  - PowerSplit
  - RollingResistance
  - SimpleACTHSII
  - TractionGenerator
  - TractionMotor
  - VehicleMass
- ControllerImplementations
  - BatterySOCTemperature
  - BatterySOCTemperatureScaledCharging
  - ClimateControlElectricAC
  - PowerSplitHybridVehicleFeature
  - VehicleSystemController
- Experiments
  - BackwardDriveCycle
  - ForwardDriveCycle
- SubsystemImplementations
  - Chassis
  - ControllerNetwork
  - Driveline
  - DualVoltageFixedCapacityBatteries
  - ElectricACAccessories
  - MappedEngine
  - PowerSplitHybridTransmission
  - SimpleThermalNetwork
  - SingleZoneCabin

Figure 7. Toyota Prius implementation

To support this library structure, two different types of parameterization are defined: parameterized model and configurable models. The characteristics of a parameterized model are defined as follows:

- Parameters declared in public section of Modelica model
- Can include instantiation of non-replaceable models
- Model can be instantiated
- Parameter values provided at instantiation and parameters can be propagated to higher level model
- Parameter values can be modified by higher level component
- Parameter values can be modified after compilation

The characteristics of a configurable model are as follows:

- Parameters declared in protected section of Modelica model
- Can include instantiation of replaceable components
- Model denoted as "partial" to indicate that it is not complete and can only be instantiated as a replaceable component in another model
- Explicit model implementations which are stored in the package hierarchy are required
- Model implementations are created by extending from the configurable model, providing parameter data, and selecting implementations for other configurable models
- Model implementations can be used directly in other models or tests
- Parameter values cannot be modified by higher level components at instantiation
- Parameter values can be modified after compilation

Ultimately, the type of parameterization used is defined by the model developer when the model is created. Some factors to be considered are the complexity of the parameter data, desired integrity of the parameter data, and the anticipated usage of the model. It should be noted that virtually all the models in the Modelica Standard Library are parameterized models according to the characteristics above. Figure 8 shows a sample configurable transmission subsystem model. This model is comprised of two replaceable configurable models for the `torque_converter` and `gearbox` components and one parameterized model for the inertia component.

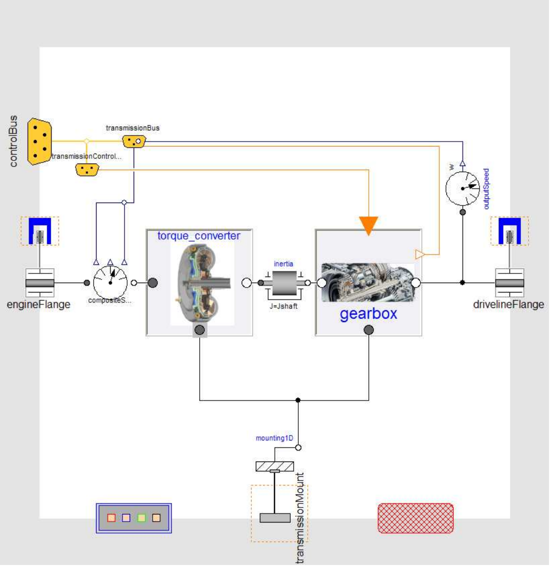


Figure 8. Sample configurable model

4 Hybrid Vehicle Implementations

Using the newly developed architecture, four sample vehicle implementations were created. The implemented models include the Toyota Prius, Lexus RX400h, and concept versions of a hybrid sedan and SUV. While the vehicle model architecture obviously supports models of varying level of detail and a wide range of engineering analyses, these implementations were focused on drive cycle simulations for fuel economy. The parameterization data for these models was collected from available publications in the open literature and from the last publicly-available version of ADVISOR [9].

The vehicle model implementations include the following subsystem representations:

- Accessories including performance-oriented model of vehicle air-conditioning system
- Mapped engine model
- Various implementations of conventional and hybrid transmissions with motors, gear seats, clutches, *etc.*
- Rigid front wheel drive (FWD) drivelines
- Vehicle chassis with lumped vehicle inertia, no-slip tires, and loads for aerodynamic drag and rolling resistance
- Simple brakes with prescribed actuation
- Dual voltage electrical power networks with fixed capacity battery models including battery thermal response
- Thermal networks including routing for battery and cabin cooling
- Lumped cabin models for vehicle cooling
- Controller network implementations including vehicle system, engine, transmission, battery, motor, generator, climate, and brake controllers
- Driver models based on drive cycles with capability to run both forward and backward models

The acausal nature of the Modelica modeling language enables several nice features of the model architecture:

- Ability to run both forward and backward drive cycle simulations with change only to the driver model (assuming underlying model is invertible)
- Ability to use model inversion to implement control features
- Ability to re-use physical, validated models across subsystems and applications
- Ability to plug-n-play models of varying level of detail to enable a wide range of engineering analyses to support model-based engineering over the entire product development process

4.1 Toyota Prius

The vehicle model implementation for the parallel hybrid Toyota Prius is shown in Figure 9. The implementation of the transmission subsystem for the PowerSplit transmission contains the motor, generator, and gearing components consistent with the hybrid transmission delineation in the Toyota drivetrain schematic [10] shown in Figure 10.

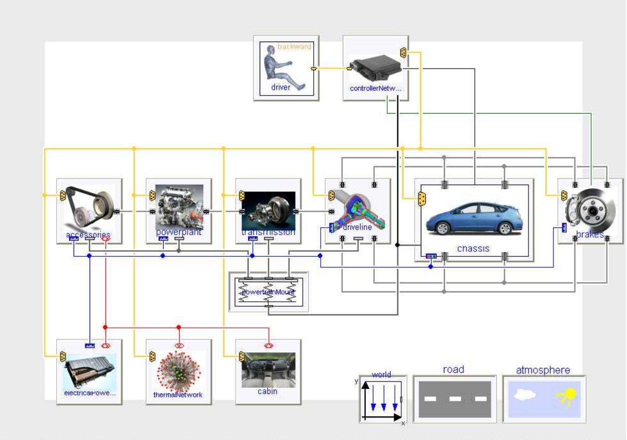


Figure 9. Toyota Prius model

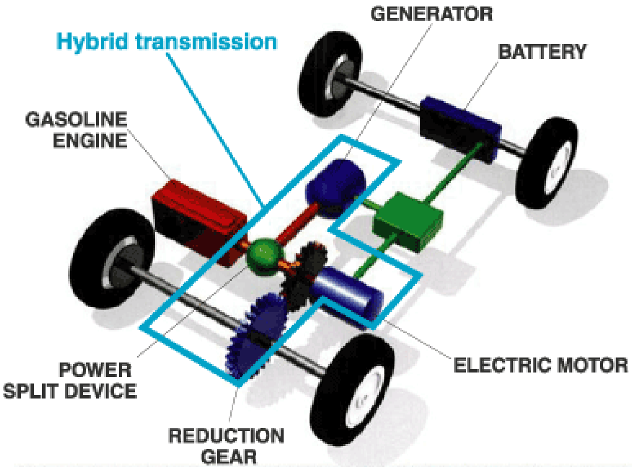


Figure 10. Toyota Prius drivetrain schematic [10]

4.2 Lexus RX400h

The vehicle model implementation for the Lexus RX400h is shown in Figure 11. Like the Toyota Prius, the Lexus RX400h is a parallel hybrid vehicle with a PowerSplit transmission. The drivetrain schematic in Figure 10 is applicable to the Lexus RX400h as well.

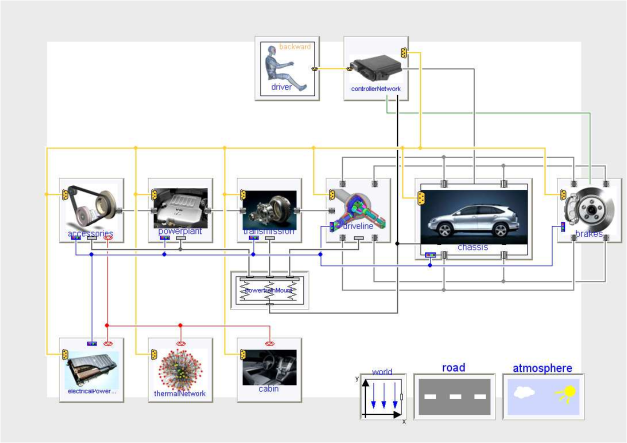


Figure 11. Lexus RX400h model

4.3 Concept Hybrid Sedan

The vehicle model implementation for a concept hybrid sedan with a parallel hybrid architecture is shown in Figure 12.

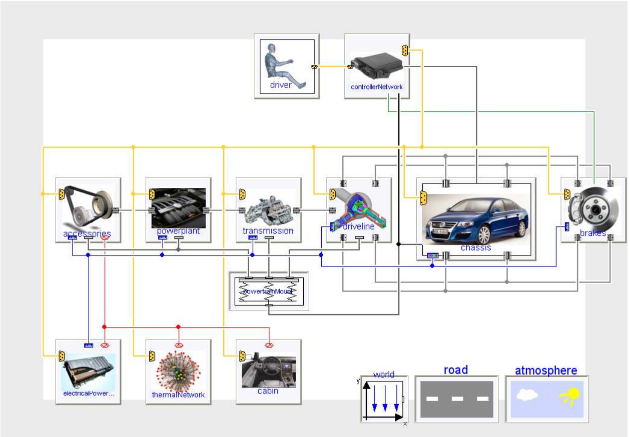


Figure 12. Concept hybrid sedan model

4.4 Concept Hybrid SUV

The vehicle model implementation for a concept hybrid SUV with a parallel architecture is shown in Figure 13.

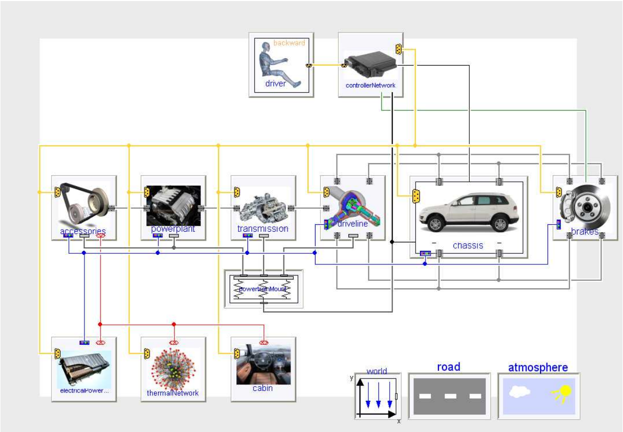
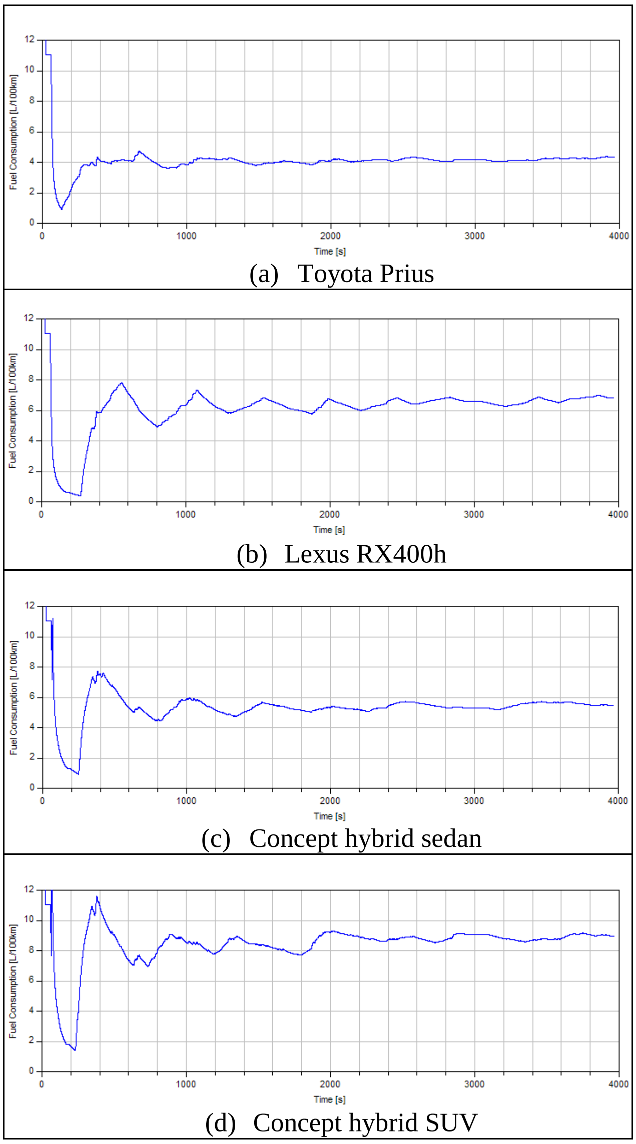


Figure 13. Concept hybrid SUV model

5 Drive Cycle Simulations

Sample drive cycle results from the four vehicle implementations are shown in this section. Fuel consumption data in L/100km is shown in Figure 14 for the four vehicles. The drive cycle is a proprietary cycle developed based on real-world driving over a range of conditions of interest to hybrid vehicle development.



(a) Toyota Prius

(b) Lexus RX400h

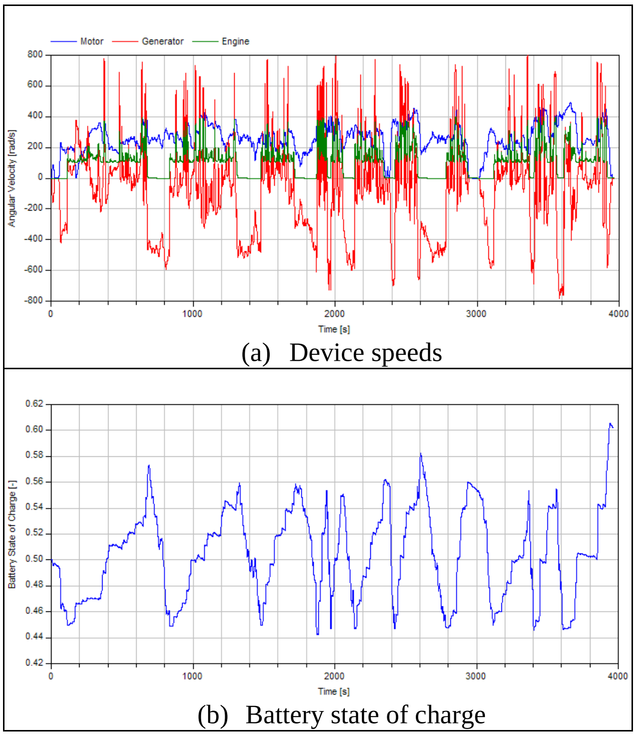
(c) Concept hybrid sedan

(d) Concept hybrid SUV

Figure 14. Fuel economy simulations

While every attempt was made to incorporate actual vehicle parameter data into the simulations, certain key parameters and component specifications were not available and thus were implemented based on the authors' best engineering judgment or based on appropriate scaling from existing data. In addition, drive cycle fuel consumption is highly dependent on the implementation and calibration of the vehicle control strategy. While control strategies were implemented for all four vehicles, these strategies may not be representative of the actual, proprietary control strategies for the production vehicles. Thus, the fuel economy results should be viewed as representative only. Furthermore, it should be noted that there is no experimental data with which to compare the model as these vehicles either do not exist yet in hardware or were not actually driven over this drive cycle. However, the results appear reasonable and follow the expected trends.

Figure 15 shows some additional signals from the Prius drive cycle simulations. The top graph shows the speeds of the engine, motor, and generator during the drive cycle. The bottom graph shows the state of charge (SOC) in the high voltage battery. The resulting battery dynamics include the contributions of the vehicle system and battery control characteristics and charge/discharge due to driving requirements and regenerative braking.



(a) Device speeds

(b) Battery state of charge

Figure 15. Prius drive cycle results: device speeds and battery state of charge

6 Extensions to VehicleInterfaces

The VehicleInterfaces library [8] provides a solid architecture to support vehicle system modeling. The library offers substantial flexibility in modeling the mechanical (both 1D and 3D) and control system interactions in the vehicle. Based on the extensions to the library implemented as part of this work, this section proposes additions to VehicleInterfaces to enable improved support for future vehicle modeling efforts.

6.1 Electrical Modeling

Electrification of nearly all major vehicle subsystems in both conventional and hybrid vehicles necessitates system leveling modeling of electrical systems. Currently VehicleInterfaces does not include electrical connectors and interactions at the subsystem level. The following extensions to VehicleInterfaces would improve the library's ability to support vehicle modeling including electrical system effects:

- Addition of expandable electrical bus to all major vehicle physical subsystems as shown in Figure 16
- Addition of electrical power network subsystem to serve as architecture placeholder for electrical energy sources, converters, *etc.* which distribute electrical power via the expandable electrical bus to other vehicle subsystems

These extensions eliminate the need to extend the existing interfaces in VehicleInterfaces simply to add an electrical bus. In addition, the formal inclusion of an electrical system will natively allow modeling of hybrid vehicle architectures in a standardized architecture as shown in Figure 16 without having to add electrical components in an *ad hoc* way to the top-level architecture.

6.2 Thermal Modeling

System level thermal modeling is another key element of vehicle system modeling. Currently VehicleInterfaces does not include thermal interactions at the subsystem level. The extended vehicle model architecture shown in Figure 3 includes the addition of an expandable thermal bus to a few top level subsystem components. As mentioned in Section 2.2, the most flexible implementation would include the addition of the thermal bus to all major vehicle subsystems as shown in Figure 16. Including an expandable thermal bus eliminates the need to extend the existing interfaces in VehicleInterfaces simply to accommodate thermal modeling. The thermal network and cabin subsystems are an integral part of the thermal architecture for the simulations shown in Section 5 but could be omitted from a standard architecture in an effort to minimize top level subsystems.

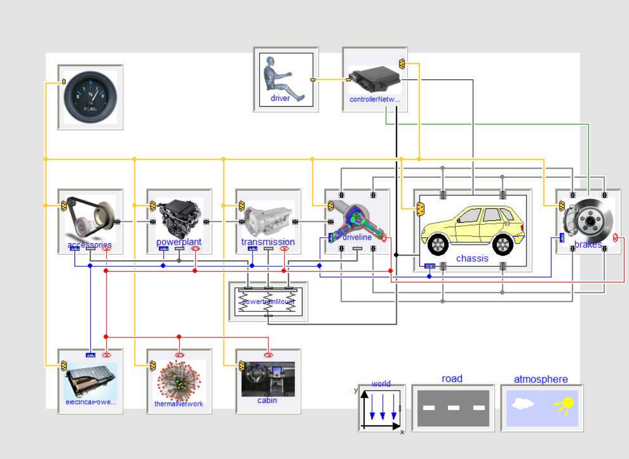


Figure 16. Sample extended architecture

7 Conclusions

This paper documents the development and implementation of an extended vehicle model architecture for hybrid vehicle modeling. This architecture is based on VehicleInterfaces and more easily enables flexible, configurable modeling of different hybrid vehicle configurations without the need for several different architectures. Additional elements have been added to the interfaces and architecture to allow more flexible electrical system modeling and more detailed thermal modeling. To illustrate the usage of this architecture, four different hybrid vehicles have been implemented and sample drive cycle simulations results shown. The canonical library structure implemented in this work has proven very capable of handling model development and implementation of model variants in a user-friendly way that integrates well with the formal model configuration language elements in Modelica. The canonical library structure has been discussed in detail along with a sample package implementation for the vehicle implementations shown in this work. Extensions to VehicleInterfaces have been proposed to improve the library for future vehicle modeling efforts.

Acknowledgements

The authors would like to acknowledge Hubertus Tummescheit and Magnus Gafvert from Modelon for initially proposing the scheme and canonical library structure implemented in this paper. Their contributions and insights were extremely valuable and are gratefully acknowledged.

References

[1] J.D. Power Automotive Forecasting, "US Hybrid-Electric Vehicle Sales Forecast Q3 2008", 2008.

[2] Tiller, M., Tobler, W.E., and Kuang, M., "Evaluating Engine Contributions to HEV Driveline Vibrations", Proceedings of 2nd International Modelica Conference, pp. 19-24, 2002. http://www.modelica.org/events/Conference2002/papers/p03_Tiller.pdf

[3] Laine, L. and Andreasson, J., "Modelling of Generic Hybrid Electric Vehicles", Proceedings of 3rd International Modelica Conference, pp. 87-94, 2003. http://www.modelica.org/events/Conference2003/papers/h26_Laine.pdf

[4] Hellgren, J., "Modelling of Hybrid Electric Vehicles in Modelica for Virtual Prototyping", Proceedings of 2nd International Modelica Conference, pp. 247-256, 2002. http://www.modelica.org/events/Conference2002/papers/p32_Hellgren.pdf

[5] Simic, D., Giuliani, H., Kral, C., Gragger, J., "Simulation of Hybrid Electric Vehicles", Proceedings of 5th International Modelica Conference, pp. 25-31, 2006. http://www.modelica.org/events/modelica2006/Proceedings/sessions/Session1b1.pdf

[6] Simic, D., and Bauml, T., "Implementation of Hybrid Electric Vehicles Using VehicleInterfaces and the SmartElectricDrives Libraries", Proceedings of the 6th International Modelica Conference, pp. 557-563, 2008. http://www.modelica.org/events/modelica2008/Proceedings/sessions/session5c.pdf

[7] Tiller, M., Bowles, P., and Dempsey, M., "Development of a Vehicle Modeling Architecture in Modelica", Proceedings of 3rd International Modelica Conference, pp. 75-86, 2003. http://www.modelica.org/events/Conference2003/papers/h32_vehicle_Tiller.pdf

[8] Dempsey, M., Gafvert, M., Harman, P., Kral, C., Otter, M., and Treffinger, P., "Coordinated Automotive Libraries for Vehicle System Models", Proceedings of 5th International Modelica Conference, pp. 33-41, 2006. http://www.modelica.org/events/modelica2006/Proceedings/sessions/Session1b2.pdf

[9] National Renewable Energy Laboratory (NREL), "Advisor documentation", www.ctts.nrel.gov, 2002.

[10] Toyota Motor Corporation, "Toyota Hybrid System II: Hybrid Transmission", 2009.

http://www2.toyota.co.jp/en/tech/environment/ths2/hybrid.html

Interfacing Abaqus with Dymola: A High Fidelity Anti-Lock Brake System Simulation

Brad Schofield Harish Surendranath Magnus Gäfvert Victor Oancea
Modelon AB, Ideon Science Park, SE-22370 Lund, Sweden
{brad.schofield, magnus.gafvert}@modelon.se
Dassault Systèmes Simulia Corp., 166 Valley Street, Providence, RI 02909
{harish.surendranath, victor.oancea}@3ds.com

Abstract

Accurate simulation of anti-lock braking systems (ABS) requires detailed models of several subsystems in different physical domains. The most important subsystems are the hydraulic brake system, the tire and the control algorithm. The creation of detailed models of each subsystem in a single modeling tool may be difficult if not impossible. To overcome this, co-simulation may be used to combine the strengths of different tools. In this article, co-simulation between Dymola and Abaqus is used to investigate the performance of an ABS algorithm with a highly detailed finite-element tire model. The brake system hydraulics along with the control algorithm are simulated in Dymola while the tire model, the wheel, the braking caliper and the contact with the road are simulated in Abaqus.

While computationally more expensive than a traditional modeling approach when a semi-analytical tire model (such as the Magic Formula model) may be used to model the tire and tire road interaction, the approach described in this paper includes a fair amount of details when modeling of the tread, the tire plies, the wire reinforcements in the tire and the contact with the road. The necessary data is exchanged between the two applications using the co-simulation capabilities available in Abaqus and the .DLL option in Dymola. Sensors in Abaqus provide information about the mechanical state of the system such as forward translational velocity, angular velocity/acceleration and the free rolling effective radius. This information is communicated to Dymola at frequent simulation time intervals at runtime. Dymola uses this information as inputs and computes the brake caliper clamp force. This force is communicated in turn to Abaqus which determines the force on the brake rotor.

Keywords: Anti-lock Brake Systems, Dymola, Abaqus, Hydraulics, Finite elements, Automotive Control, Co-simulation

1 Introduction

Modeling automotive systems very often necessitates modeling in multiple physical domains. The Modelica language is a natural choice for such multi-domain modeling, but for situations in which very high resolution is required, or where physically-derived models are not available, other tools such as finite-element modeling may be more appropriate. In such situations, co-simulation may be used to combine the strengths of different modeling tools to achieve the desired level of accuracy. In this paper, co-simulation between Dymola and Abaqus is used to investigate the effects of ABS braking with a highly detailed finite-element tire model.

In the investigation, a single wheel setup was used. For simplicity, suspension components are not modeled. Dymola was used for the implementation of the brake system hydraulics model as well as the control algorithm, and Abaqus was used for the tire and brake rotor models. The signals necessary for ABS control are passed from Abaqus to Dymola, and the Dymola model provides a brake caliper clamp force output.

2 Brake System Modeling in Dymola

For the co-simulation, the hydraulic braking system as well as the control algorithm were implemented in Dymola. Hydraulic components from the HyLib library are used. The braking system consists of a single brake caliper cylinder, connected to a master cylinder via a three port valve. The three port valve is set up to have three modes of operation: a pressure increase mode in which the master cylinder is connected to the slave,

DOI: 10.3384/ecp09430110

a hold mode in which all ports are disconnected, and a pressure decrease mode in which the slave cylinder is connected to a tank. A return circuit is not modeled. This system represents the simplest form of production ABS implementation in which brake fluid is not returned to the master cylinder during braking, but rather after the braking event has taken place.

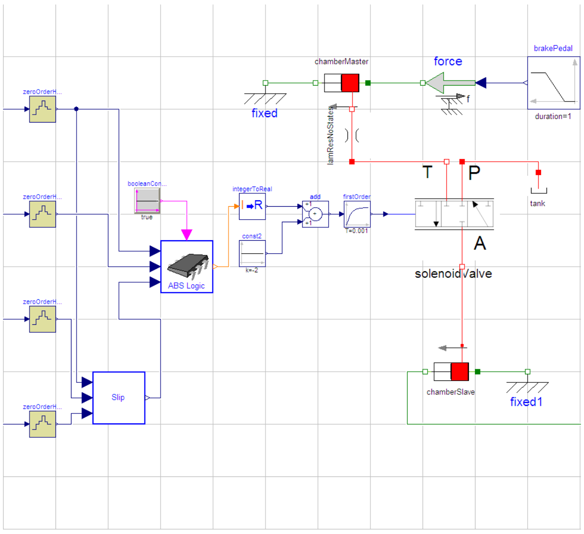


Figure 1: Diagram view of the brake system hydraulics and control model in Dymola

2.1 Control Algorithm

Control is performed by modulating the position of the three port valve depending on wheel acceleration and slip. The algorithm is based on that described in the Bosch Automotive Handbook [1], and is modeled as a state machine. The required inputs to the controller are wheel angular velocity ω, angular acceleration $\dot{\omega}$, rolling radius r and hub longitudinal velocity v_x. The input signals to the controller are sampled with period $T_s = 1ms$. The longitudinal slip is calculated as

$$\lambda = \frac{v_x - r\omega}{v_x}$$

The ABS is triggered when the wheel deceleration falls below the prescribed threshold $-a$. Pressure is then held until the slip exceeds a threshold λ_T, at which point pressure is dropped for a given time. Pressure is the held until a positive acceleration A is reached, at which point pressure is increased until the acceleration drops to a. At this point pressure is increased slowly via alternate hold and increase commands. This allows the 'peak' of the friction characteristic to be traversed slowly, before the unstable side is reached. The cycle begins again once the $-a$ acceleration threshold is crossed. The controller is deactivated for longitudinal velocities under a prescribed level. In addition, a timeout parameter is used to reset the ABS algorithm if it remains in any one state for extended periods. This is necessary to prevent the controller becoming 'locked' in any state when ABS action is no longer required.

The aim of the algorithm is essentially to move quickly away form the region of the slip characteristic corresponding to unstable slip dynamics, and to maximize the time spent near the the friction peak (to improve braking performance).

Rule-based control algorithms such as the one described here often suffer from problems such as large numbers of tuning parameters, lack of robustness to process variations and difficult stability and performance analysis. Model-based approaches often lead to more manageable systems with fewer tuning parameters and greater reusability. Nevertheless, rule-based algorithms are widespread in production systems. A model-based approach to ABS control design is described in [3].

3 Tire, road and brake model in Abaqus

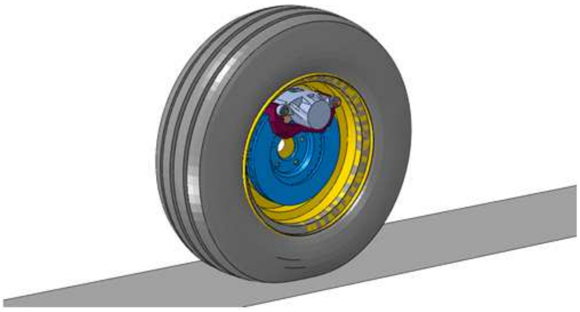


Figure 2: Animation of tire, road and brake model in Abaqus

The wheel, the tire, and the braking caliper-rotor subassembly are modeled in the finite element software Abaqus. There most important ingredient in the Abaqus model is the detailed modeling of the tire. The Abaqus software is used by a large number of tire manufacturers to study tire responses spanning steady state, braking or abuse type loading conditions such as curb hitting or driving over a pothole. The finite element tire models the tread, plies and wire reinforcements in the actual tire. The tire is first pressurized and placed in contact with the road under the weight corresponding to that wheel (quarter car). While a flat road was depicted above, Abaqus modeling of more complex road geometries such as uneven cobblestone roads is readily available. A steady state transport analysis is performed next in Abaqus/Standard (implicit integration) to compute the state of stress and deformation in the tire corresponding to a given forward velocity before any braking is applied. Optionally, a cornering radius can be specified as well. The contact pressures between the tire and the road corresponding to this configuration are depicted below.

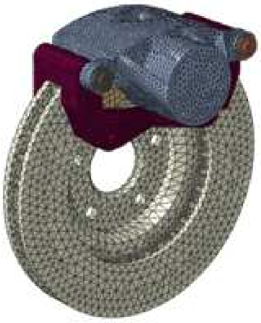


Figure 3: Brake disc model in Abaqus

The deformations and stresses in the tire are then imported in Abaqus/Explicit (explicit integration) which is used for the actual braking analysis via co-simulation with Dymola. Abaqus/Explicit solves for most of the mechanics in the analysis to update the current state of stress and deformation in the tire. Sensor information such as wheel angular velocity/accelerations are communicated at runtime at frequent simulation intervals to Dymola which computes the brake caliper cylinder pressure for effective braking. The braking pressure is then communicated back to Abaqus/Explicit which applies this pressure to the brake caliper cylinder depicted below. The brake pads are pressed against the brake rotor to produce a breaking torque that decelerates the single wheel-brake subassembly.

The total number of degrees of freedom in the Abaqus model is about 125000 which are basically nodal translations and rotations.

4 Simulation of Single-Wheel Braking in Dymola

In order to test the ABS system in Dymola, a single-wheel test rig was constructed using components from the VehicleDynamics library, see Figure 4. Only longitudinal dynamics were modeled. A 'Magic Formula' tire model was used.

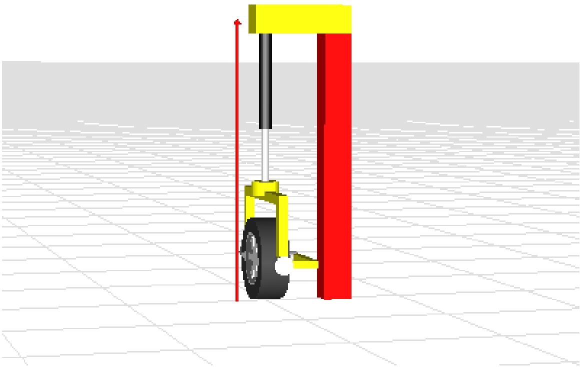


Figure 4: Animation of single-wheel test rig used for simulation in Dymola

The rig was simulated with an initial velocity of $10m/s$, and a brake pedal force ramp was applied. Figure 5 shows the state of the controller during the simulation. The rapid switching represents the slow build-up mode, achieved by alternating between hold and increase positions of the three port valve. The brake caliper clamp force is illustrated in Figure 6. In Figure 7 the hub fore-aft velocity and wheel circumferential velocity are shown. The wheel circumferential velocity is computed by multiplying the wheel angular velocity by the effective rolling radius.

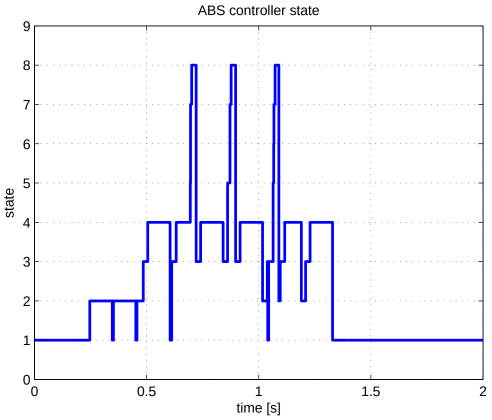


Figure 5: The ABS controller state during simulation of single-wheel braking in Dymola.

5 Co-simulation Setup

The physical system modeled with Abaqus is furnished (in Abaqus) with sensors and actuators to create what is commonly referred to as the Plant. The

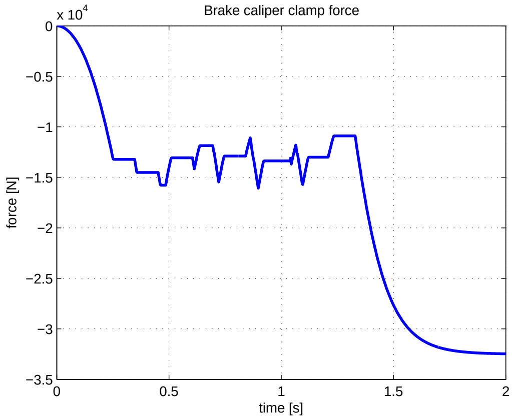


Figure 6: The brake caliper clamping force during simulation in Dymola.

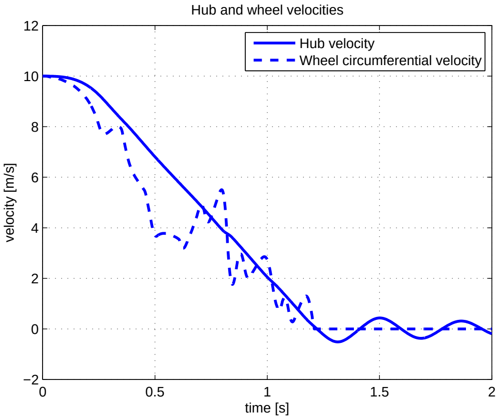


Figure 7: Hub velocity and wheel angular velocity during simulation in Dymola.

sensor data computed by Abaqus is passed to Dymola (the controller) which computes the needed actuation to drive the physical system in the desired state. The translational velocity, angular velocity, angular acceleration and the free rolling radius of the tire are defined as the sensors in Abaqus. The actuator (computed in Dymola during co-simulation) is the clamp force to push the brake pads against the brake disc. The solution in Abaqus/Explicit uses a central difference method to compute the equilibrium condition of the physical system at time $t + \delta t$ based on the equilibrium conditions at time t. One of the fundamental requirements of the central difference scheme is that the time step δt be smaller than a critical value, known as the stable time increment. In this particular application, the value of stable time increment is close to 3 microseconds. Note that this is significantly below the required time increment in Dymola which is on the order of milliseconds. The co-simulation procedure analysis requires significantly more computational resources than a standalone Dymola run. This is primarily due to the fact that finite element computations in this application are computationally very intensive given the required small time increment size and the number of solution variables involved. Consequently, given that the number of unknowns is roughly 125000 in the Abaqus model and roughly 585000 increments are required to complete the 2.0 seconds of simulation time, the computational cost of the co-simulation is dominated by far by the cost of the finite element analysis. In this particular case, the analysis was run using 8 CPUs on a HP BL460 Intel Xeon Dual Core 3Ghz Linux 64 bit computer and required 237 minutes to complete. By contrast, the Dymola job required less than a minute of actual computational time to complete on a 32 bit Windows computer using only one CPU. Dymola with the .dll option is used. A C++ driver uses the API for the .dll to advance the computations in the ABS model.

The Abaqus co-simulation capabilities include various co-simulation rendezvousing schemes. It is not the purpose of this paper to review the algorithmic nature of these schemes as they are described in [2]. Given the time incrementation in Abaqus/Explicit requires a far smaller time increment than Dymola and that the required 3 microseconds time increment is far smaller than the sampling frequency of sensors used in ABS systems, a simple non-iterative co-simulation scheme is used:

- Each Abaqus increment sensor information is computed (Hub velocity, free rolling radius, angular velocity and angular acceleration) and then communicated to Dymola via a socket-based interface.

- Dymola (with the .dll option) is run via a driver that uses the API to the .dll to advance the simulation time in Dymola. The sensor information from Abaqus is read in as inputs to Dymola. Dymola integrates in time with a time step size equal to the Abaqus/Explicit time increment (roughly 3 microseconds) which is far smaller then what Dymola would require for an accurate integration. Actuators computed in Dymola (brake caliper pressure) are communicated back to Abaqus which applies this freshly computed load as the load for the next increment.

- The process is repeated until the simulation time is exhausted.

The co-simulation process was also run with an even smaller time increment (2 microseconds) to check whether the results change. The results were virtually identical.

6 Co-simulation Results

In this section the results of the co-simulation using are presented, and compared to the results obtained using the simple tire model in Dymola. It should be noted that no re-tuning of the ABS control algorithm was performed for the co-simulation. It is highly likely that performance could be improved considerably by re-tuning the controller with the Abaqus wheel model.

6.1 Parameter selection

The primary aim of the investigation was to test the co-simulation interface between Dymola and Abaqus on a relevant industrial example. In particular, due to the extremely large differences in complexity of the tire models in Dymola and Abaqus, it was not expected that the co-simulation results would be quantitatively very similar. To this end only a rudimentary synchronization of parameter values between the Dymola and Abaqus brake and wheel models was performed. Wheel and brake rotor diameters are similar but not exactly equal. Tire force-slip characteristics are close, but the stiffnesses differ slightly.

In the Dymola brake model, the braking moment is obtained using the clamping force, an effective radius and a friction model. The Abaqus brake model uses a nonuniform pressure distribution on the brake pads, making a comparison with the effective radius of the Dymola model difficult.

Figure 8 shows the hub and circumferential velocities during the co-simulation. The slip is initially large, but the controller is able to prevent wheel lock. Figure 9 shows the brake caliper clamping force during co-simulation. The rapid force build-up and release phases are clearly visible.

6.2 Comparison between co-simulation and Dymola

Figure 10 shows the hub and circumferential velocities in the co-simulation and Dymola-only simulation respectively. The qualitative behavior is clearly similar. The controller is more effective in the Dymola-only simulation, maintaining lower slip. This is to be expected as the controller was tuned using the Dymola tire model.

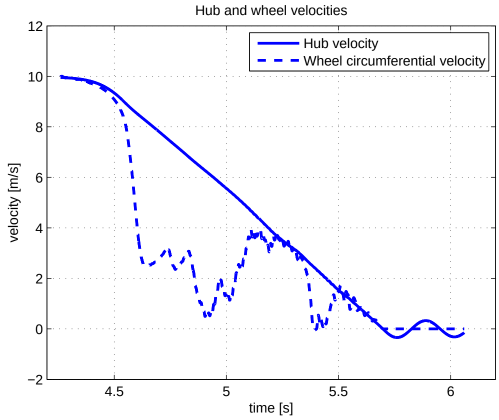


Figure 8: The hub velocity and wheel circumferential velocity during co-simulation.

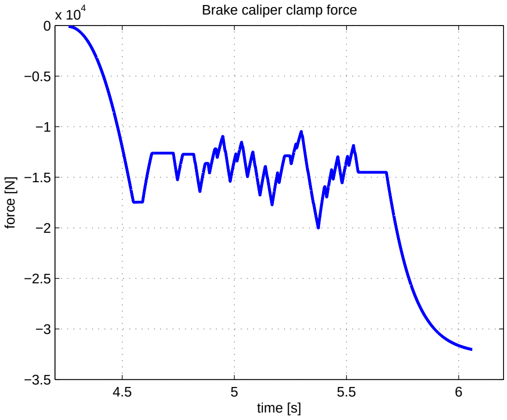


Figure 9: The brake caliper clamping force during co-simulation.

Figure 11 shows a comparison of the brake caliper clamping forces in the co-simulation and Dymola-only simulation. The forces are very similar. The resultant longitudinal tire forces are compared in Figure 12. The tire force results from the co-simulation had considerable high-frequency content, as may be expected from a high-fidelity model. For the comparison the results were low-pass filtered with a bandwidth of 100Hz.

During the initial braking the tire forces are very similar. A larger slip is developed in the co-simulation leading to lower tire forces. The oscillations seen after roughly 1.3 seconds are due to the test rig coming to a standstill. This behavior is observed in both the co-simulation and the Dymola-only simulation.

7 Conclusions

In this article, co-simulation is used to study anti-lock braking control using a high-fidelity tire model. Dy-

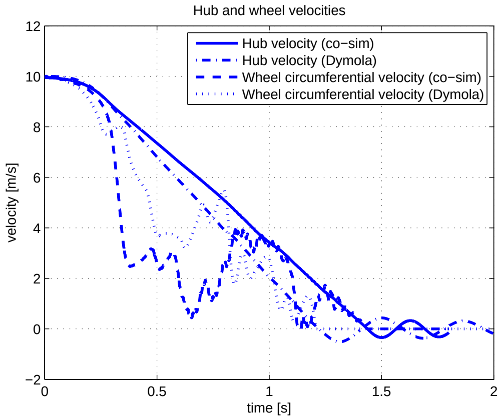


Figure 10: The hub velocities and wheel circumferential velocities during co-simulation and Dymola-only simulation.

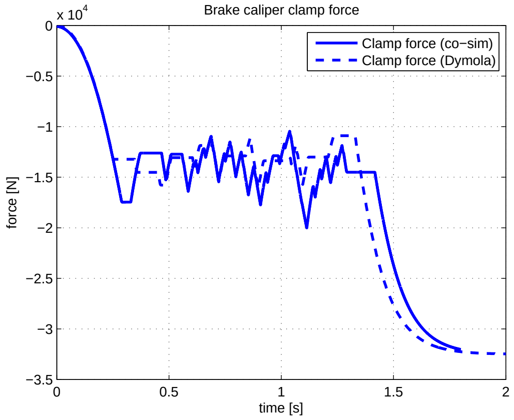


Figure 11: The brake caliper clamping forces during co-simulation and Dymola-only simulation.

mola is used to implement a realistic hydraulic braking circuit using existing components from the HyLib library. The control algorithm is also implemented in Dymola. Abaqus is used to implement very high fidelity tire, road and brake rotor models.

The co-simulation results presented here are preliminary in the sense that controller tuning was performed using a much simpler tire model in Dymola. No retuning was performed for the co-simulation. As previously mentioned, the tire characteristics and brake model parameters were not exactly the same for the Dymola and Abaqus models. Nevertheless, the ABS controller was capable of preventing wheel lock in the co-simulation with the high-fidelity wheel model.

This example illustrates how co-simulation between different packages may extend the realism of system-level simulations. The multi-domain modeling strengths of Dymola are combined with the computational power of Abaqus to provide a level of detail which would be extremely time-consuming to achieve with a single tool.

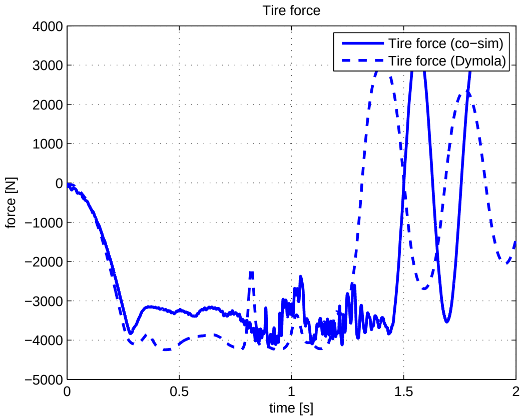


Figure 12: The tire longitudinal forces during co-simulation and Dymola-only simulation. The result for the cosimulation was low-pass filtered with 100Hz bandwidth.

As this was a preliminary investigation, there is considerable scope for future work. In particular, tuning of the ABS controller using the Abaqus is expected to yield large performance increases. In addition, simulation with a full vehicle model in Abaqus is also planned. Successful co-simulation of an ABS system with a full vehicle model would then allow high-fidelity simulations of vehicle dynamics controllers to be performed.

References

[1] Horst Bauer, editor. *Bosch Automotive Handbook*. Robert Bosch GmbH, 4th edition, October 1996.

[2] Dassault Systèmes Simulia Corp. *Abaqus Analysis User's Manual*, 2009. Version 6.9.

[3] Stefan Solyom, Anders Rantzer, and Jens Lüdemann. Synthesis of a model-based tire slip controller. *Vehicle System Dynamics*, 41(6):477–511, June 2004.

Real-time Drive Cycle Simulation of Automotive Climate Control System

Anand Pitchaikani, Kingsly Jebakumar S, Shankar Venkataraman, S. A. Sundaresan
Emmeskay, Inc
47119 Five Mile Road, Plymouth, MI 48170, USA
anandp@emmeskay.com kingsly@emmeskay.com sasundaresan@emmeskay.com

Abstract

Technologies like hardware-in-the-loop simulation (HILS) can play a significant role in reducing the product development life cycle. An essential requirement for HILS is the availability of system models that are capable of running in real-time. This paper demonstrates a closed loop real-time simulation of a vehicle with a climate control system model developed in Modelica. This paper details the modeling aspects of the vehicle system and climate control system using the object-oriented, acausal modeling language Modelica. To demonstrate the real-time simulation capabilities of the developed models, they were integrated with a controller model developed in the Simulink environment and the integrated model was then simulated in an Opal-RT real-time environment. This paper describes the models developed and the tool-chain used to achieve real-time simulation. The real-time simulation results as well as the performance results are also presented. This paper demonstrates Modelica's capabilities in creating models for real-time vehicle climate control system simulations.

Keywords: climate control system; vapor cycle; closed-loop simulation; real-time simulation;

1 Introduction

The development of system simulation models for vehicle as well as the air-conditioning system is extensively described in various previous works [1, 2 and 4]. The challenge lies in developing these models so as to balance the need for real time capability and reasonable accuracy. The real-time closed-loop simulation of the plant model along with the controller model is a major validation milestone in the controller development process.

A conventional internal combustion engine powered vehicle is considered in this work. The system includes an engine driven compressor and a vapor compression refrigeration cycle based Heating, Ventilation and Air-Conditioning (HVAC) system. Models for vehicle subsystems and climate control system subsystems are required to study the impact of the vehicle climate control system over a simulated drive cycle. The vehicle subsystems (engine, transmission, drive train, chassis and accessory drive) and the HVAC components (compressor, condenser, evaporator and expansion valve) need to be modeled. The literature reports the development of models for HVAC components [1, 2]. The accuracy levels of these reported models differ as they address different needs like design of climate control system or simulation of overall vehicle with climate control system. Though the real-time simulation of Modelica® models was demonstrated successfully for vehicle system models [4], it has not been reported for a vehicle model with climate control system. The real time simulation of vehicle model with climate control system components will help the developers of the climate system controllers by enabling Hardware-in-the-Loop simulation of varying complexities. Different controller hardware can be tested with these real time models on an HILS platform.

The controller model was available as Simulink® S-function along with the object code. The assembled vehicle and climate control system models in Modelica were converted to a Simulink C-file S-function using Dymola® [9]. The plant and controller S-function models were integrated in the Simulink environment. This closed loop model consisting of the two S-functions was downloaded to Opal-RT's [11] target processor. The controller model was simulated in the target itself due to non-availability of the controller hardware. This real time simulation verified the real time capability of the developed plant models as well as ability of the controller to achieve the control targets. This takes the overall controller development process one step closer to closed loop HIL simulation.

Though literature is available for such real time simulations using Modelica models [5] in other do-

DOI: 10.3384/ecp09430007

mains, there is very little reported in the area of HVAC system. A HVAC system is a multi-domain system straddling mechanics, fluid mechanics and thermal engineering.

In this paper the details involved in modeling the vehicle as well as the climate control system components are described. The simulation was carried out by parameterizing the vehicle and climate control system to represent a conventional mid sized sedan.

2 Tool-chain

The plant models were developed using the Modelica language in the Dymola 6.1 environment. The development re-used most of the components from the Modelica Standard Library (MSL). The special needs were satisfied by creating additional custom models in Modelica. The Dymola-Simulink interface is used to create the S-function of the developed plant models. The controller S-function and plant model S-function are integrated in MATLAB® R2006b version. The integrated model is converted to a real-time model by using RT-LAB™ 8.1.7. This real-time model is downloaded into the TestDrive™ system that operates on QNX® 6.3. The real-time simulation is carried out in the above mentioned TestDrive system. The above tools were used based on their applicability to the automotive domain.

3 Plant Models

The goal of this work was to model a conventional engine driven vehicle with engine driven HVAC system. The vehicle subsystems (driver, controller, engine, transmission and chassis) and the vehicle air-conditioning subsystems (compressor drive, HVAC systems, cabin and atmosphere) are modeled in this work. Most of these models are developed using simple formulations keeping the real-time requirements in mind. The overall plant model architecture (Figure 1) in Modelica is made such that all the main subsystems are replaceable. This architecture provides flexibility to choose different implementations for each subsystem. In this way, the same model can be configured to model different vehicle systems easily. However, only the real-time capable configuration is described in this paper.

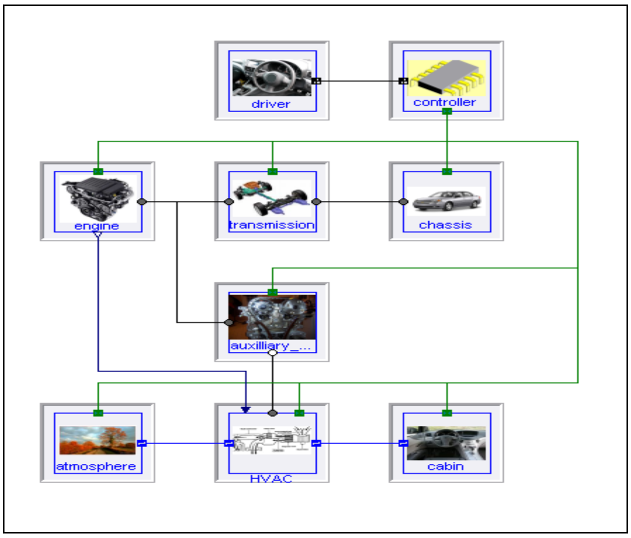


Figure 1 System Architecture

3.1 Vehicle System Models

3.1.1 Driver

The driver model generates the input signals like throttle, brake and clutch commands. The driver model takes advantage of the acausal nature of the Modelica language for setting up the inverse problem: taking the drive cycle vehicle speed command as input and then computing the clutch, accelerator pedal and brake pedal commands required to follow the desired drive cycle input. In other words, the model is “backward driven” (at least with respect to the longitudinal vehicle dynamics). The model is built with the capability of idling the engine when the vehicle is at rest. These driver command signals are sent to the controller through the driver bus.

3.1.2 Controller

The controller model processes the driver commands and plant sensor outputs and generates control signals for the engine, transmission and the HVAC system. As a simple engine model is used in this study, the engine control logic is a simple pass through in the controller model. The climate system controller is a part of the controller model. The control signals are sent to the plant through the system bus. The system bus is an expandable connector from which all other components can access the required control signals. In the case of a “forward-driven” model, the controller model would quite be complex and play an important role in driver inputs like throttle, brake, clutch and gear command. Because of the “backward driven” driveline model, most of the complexity in the controller model is related to control of the HVAC components.

3.1.3 Engine

The engine model delivers torque based on commanded throttle from the driver. This model incorporates performance maps in the form of look-up tables. The torque-speed map which provides a minimum and maximum limit of the torque for every speed is used where minimum corresponds to 0 throttle and maximum corresponds to 1 throttle. The fuel consumption map is based on the torque and speed. This formulation will be very useful for fuel economy studies. Also such minimal data will be very easy to get for parameterizing any type of engine.

3.1.4 Transmission

The transmission model is a six-speed manual type encompassing clutch, synchronizers and final drive. The gear, clutch and inertia models of MSL are used to construct the transmission model. The current gear selection is given by a gear-shift map based on throttle command and engine speed. A simple gear selector model is included in the transmission model. This model includes a shift map that consists of threshold speed limits for upshifts and downshifts, at the given throttle. Based on the throttle and engine speed, the gear selector will select a gear number and engage the appropriate synchronizer. The final drive is modeled as a simple gear. A simple clutch model is used to represent the clutch that engages and disengages the engine from the driveline. This model will capture the torque ratio introduced by the transmission and effective inertia of the transmission based on the gear engaged.

3.1.5 Chassis

The chassis model handles the longitudinal dynamics of the vehicle by modeling vehicle mass, aerodynamic drag and rolling resistance. The driver brake command is converted to brake torque and is sent to the wheels. Drag and rolling resistance are modeled using representative equations. The chassis model gets the torque generated in the engine model through the transmission and final drive. The vehicle speed from the chassis model is put on to the system bus.

3.2 Climate Control System Models

The climate control model includes the cabin, atmosphere, auxiliary drive and HVAC components as shown in Figure 2. The climate control components handle air flow as well as refrigerant flow. Expandable HVAC connectors are used in the subsystem level so that the separate interfaces used for air and refrigerant are not visible at the top-most level.

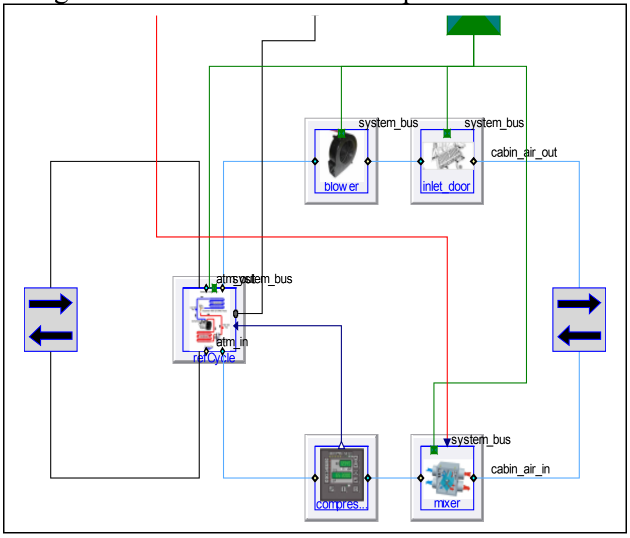


Figure 2 Climate Control System Models

3.2.1 Cabin

The single zone cabin is modeled as a closed thermal volume surrounded by a metal frame mass. Solar heating effects are included in the model. The amount of solar heating will vary with the ambient conditions chosen for the simulation. The heat capacity of the cabin metal mass is captured in the model. The heat transfer from metal frame to the interior is through convection. For typical closed cabin climate control systems, it was felt that relative humidity effects will not be significant and hence were not modeled. Air flow inside the cabin is modeled using custom made air flow connectors.

3.2.2 Atmosphere

The atmosphere is modeled as an infinite source-sink model which fixes the vehicle ambient conditions like temperature, pressure, air density etc. This reference data is used in the HVAC models to compute the heat transfer and related variables associated with air flow over the condenser and evaporator. For the sake of simplicity, the mass flow rate of air that blows over the condenser is set as a constant in the atmosphere model itself. The mass flow rate of air that blows over the evaporator will be fixed by the blower model and the atmosphere or the cabin will supply that air flow based on the recirculation command.

3.2.3 Auxiliary Drive

The auxiliary drive model captures the gearing and clutch between the engine shaft and the HVAC compressor drive shaft. It represents the engine-driven compressor configuration in which a portion of the engine power is used to drive the HVAC compressor. The compressor off signal from the climate system controller will disengage the clutch modeled in the auxiliary drive. This model captures the load applied by the HVAC system on the engine which is crucial in estimating the fuel economy accurately.

3.2.4 HVAC

The HVAC components form the core of the climate control system. They include models for refrigeration cycle, refrigerant properties, blower, inlet door and heater as shown in Figure 2 and Figure 3. The refrigeration cycle models include the vapor-cycle refrigeration components like evaporator, expansion valve, condenser and compressor. The refrigerant expansion, compression and heat transfer processes have been captured in the models. The refrigerant flow and air flow interact thermally by using thermal connectors of MSL.

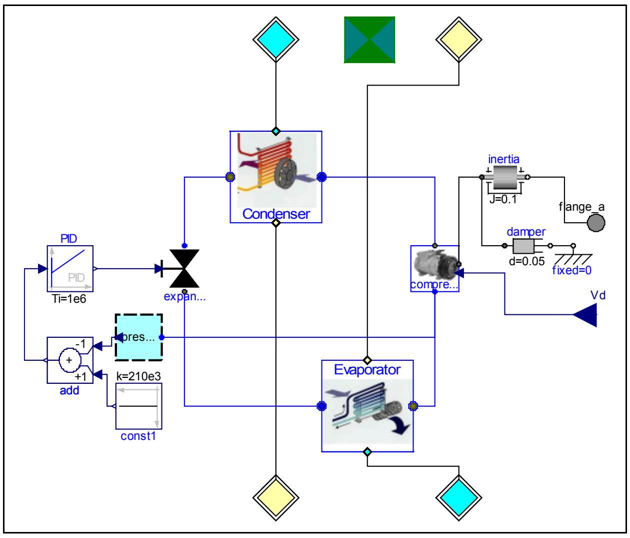


Figure 3 HVAC Components

3.2.4.1 Compressor

HVAC compressor is modeled using isentropic relationships as shown below. Compressor is characterized by its volumetric efficiency and isentropic efficiency. Provision has been given to vary its displacement volume as required.

For model robustness we ensured that the compressor model could handle the following special conditions:

- Suction pressure greater than discharge pressure
- Negative compressor speed

$$\dot{m} = \frac{\omega}{2\pi}\eta_v V_d \rho_{suction}$$

$$M = \frac{\gamma \eta_v P_{suction} V_d}{2\pi \eta_{isen} \eta_{mech} (1-\gamma)} \left(r^{\frac{\gamma - 1}{\gamma}} - 1 \right)$$ Where

$\dot{m}$ - *Refrigerant flow rate (kg/s)*
ω - *Compressor rotational speed (rad/s)*
$\rho_{suction}$ – *Refrigerant density at suction side (kg/m3)*
M – *compressor torque (Nm)*
r – *Pressure ratio*
$P_{suction}$ – *Suction pressure (N/m2)*
γ – *Ratio of specific heat capacities of the refrigerant*
V_d – *Displacement volume (m3)*
η_v – *Volumetric efficiency*
η_{isen} – *Isentropic efficiency*
η_{mech} – *Mechanical efficiency*

3.2.4.2 Condenser and Evaporator

The condenser and evaporator are modeled as lumped volume elements to emulate the heat exchange between the refrigerant medium and air (see Figure 4). This model incorporates the two important heat exchange mechanisms as given below.

- Convection between refrigerant and pipe wall
- Convection between pipe wall and air

The convective heat transfer coefficients on the refrigerant side are estimated based on the refrigerant quality and other thermal coefficients [7]. The pipe wall heat capacity is captured in the model. The convective heat transfer coefficient is modeled to have two values – one for low air mass flow rates and another for high air mass flow rates. The pressure loss inside the heat exchangers is assumed to be negligible and hence the entire heat exchanger volume is lumped into a single volume. We accepted these assumptions as reasonable in order to achieve real-time performance.

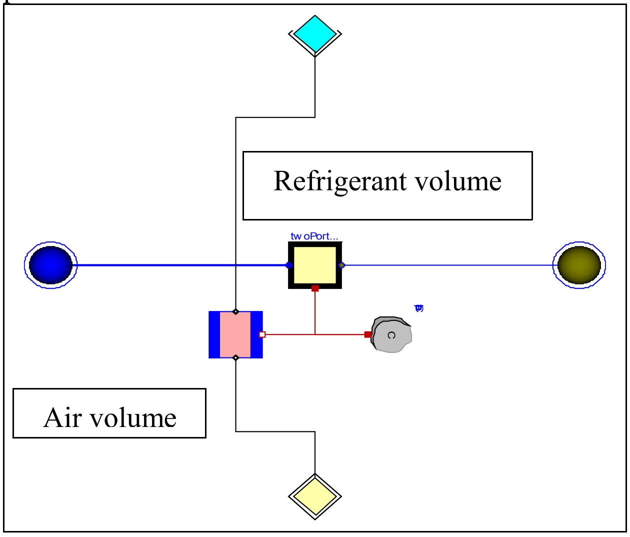


Figure 4 Condenser and Evaporator

The processes modeled in the condenser and evaporator volume models can be summarized as below.

$$\frac{dm_{vol}}{dt} = \dot{m}_i - \dot{m}_o$$

$$\frac{d\left(m_{vol}\left(h_{vol} - P_{vol}v_{vol}\right)\right)}{dt} = \dot{m}_i h_i - \dot{m}_o h_o + U_i A\left(T_w - T_{vol}\right)$$

$$C\frac{dT_w}{dt} = U_o A\left(T_{amb} - T_w\right) - U_i A\left(T_w - T\right)$$ Where

$\dot{m}_{vol}$ – *Mass of refrigerant in the volume (kg)*

$\dot{m}_i$ – *Refrigerant inlet mass flow rate (kg/s)*

m_o – *Refrigerant outlet mass flow rate (kg/s)*

h_{vol} – *Specific enthalpy of refrigerant in the volume(J/kg)*

P_{vol} – *Pressure of refrigerant in the volume (N.m^2)*

v_{vol} – *Specific volume of refrigerant in the volume (m3/kg)*

h_i – *Specific enthalpy of incoming refrigerant flow (J/kg)*

h_o – *Specific enthalpy of outgoing refrigerant flow (J/kg)*

U_i – *Refrigerant side convection heat transfer coefficient (W/m2K)*

U_o – *Air side convection heat transfer coefficient (W/ m2K)*

T_{vol} – *Refrigerant temperature in the volume (K)*

T_w – *Pipe wall temperature (K)*

T_{amb} – *Ambient air temperature (K)*

C – *Specific heat capacity of pipe (J/K)*

3.2.4.3 Expansion Valve

The expansion valve is modeled based on empirical relationship for flow as a function of pressure drop [3]. It can handle the choked mass flow based on sub-cooled entry conditions. The expansion valve is modeled as a variable area flow device. The valve area is an input to the model. A PID controller is used to control the flow area to maintain near constant suction pressure. This will ensure a proper functioning of the compressor model.

3.2.4.4 Refrigerant Properties

The refrigerant type considered in this work is R134a. The refrigerant properties are generated using property relationship curves available in the literature [6] and used in the evaporator and condenser models. The refrigerant properties were represented by data tables created from the complex empirical relationships (see Figure 5). The tables specify the enthalpy, pressure and quality (mass fraction) for the given density and temperature. The resulting model is computationally less intensive and hence capable for real-time simulation.

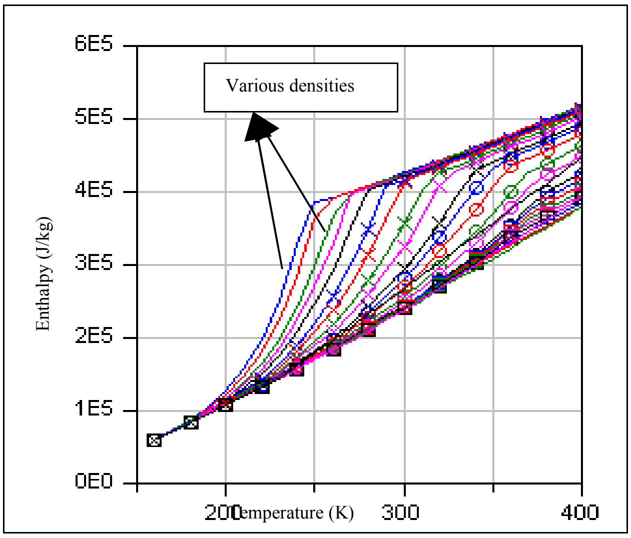


Figure 5 R134a Enthalpy Vs Temperature

3.2.4.5 Blower

The blower delivers the air required by the cabin. Blower mass flow rate is computed by a 2-D look-up table based on fan setting and outlet door position. Both of these inputs are set by the climate system controller. The blower blows the air over the evaporator. The data driven model for the blower is chosen keeping the real time requirements in mind.

3.2.4.6 Inlet door

The inlet door can re-circulate air from the cabin or take fresh air intake. The inlet door model takes in a Boolean signal which indicates the recirculation setting ON or OFF. Based on the re-circulation command from the climate system controller, this door model allows the fresh air to enter into the cabin or re-circulates the cabin air back to the cabin through the blower.

3.2.4.7 Heater

The heater core element of the HVAC system takes heat from the engine coolant and heats the air going into the cabin. The amount of heating can be regulated by adjusting the mixer door that controls the amount of air flow that gets exposed to the heater. The exposed and unexposed air mixes in the region next to the heater. The heating effect of the heater core is implemented using an efficiency term. This heat flow efficiency is given by a look-up table based on the mixer door open ratio. The heater core will be at a temperature dictated by the coolant temperature. The rate at which the air flow gets heated by the heater is controlled by a time constant and the heater efficiency term.

3.3 Testing of Models

The vehicle subsystems and the climate control system models are integrated in the Dymola environment and fundamental validation tests checks were performed. The vehicle was run at a constant speed and the HVAC compressor was switched on and off. The effect of this switching was observed in the cabin temperature. In another test, the time required by the cabin to reach a target temperature at a particular ambient condition with the vehicle being stationary was determined. These tests helped to ascertain the fidelity of the models developed and the corresponding parameterization made.

4 Real-time Adaptations

The HVAC components models can become very complex. The calculation of the refrigerant properties in the HVAC component models is itself a computationally expensive calculation. The equations used in the reference [6] are highly non-linear. When the property calculations are written as non-linear functions, differentiating these functions becomes a significant issue. To overcome this problem, tables were made that take density and temperature of the refrigerant as inputs and output the enthalpy, pressure and vapor quality. The condenser and evaporator models can be constructed using an array of refrigerant volumes and refrigerant flow resistance models. Such models were found to be computationally very complex. To avoid this complexity and the associated computational effort, the condenser and evaporator were modeled as single volumes after it was found that the results that were produced with the vehicle model were satisfactory. It was ensured that all the vehicle subsystem models were of the appropriate level of detail to characterize the actual vehicle.

5 Climate System Controller Model

The climate system controller model was available as a Simulink S-function. The controller exercises its control action over various components of the HVAC system to maintain the cabin at a set temperature. As the controller is of production standard, only the object code of the source was available. A successful integrated simulation of the developed plant models with the controller itself will mark a milestone. There were no details available about the precise control logic used in the controller. The integration is done based on the available information like input/output signals list of the controller. In many cases, the plant model developer may not get access to the source code of the controller. This method of using the object code for the controller will be very useful in such cases. Except for a few definition files (headers), the object code (binary) was sufficient to download the controller model along with the plant models into the target processor/ machine.

6 Real-time Simulation

The first level target of achieving a closed loop simulation of the developed models is accomplished by performing a dynamic simulation of plant and the controller in Simulink. The next level will make sure that the plant models developed in Modelica can be directly used in any Hardware-in-the-Loop testing. There are many practical scenarios where the climate system controller hardware needs to be tested in a real vehicle environment which might not always be possible. This validation testing will trigger the need for HIL simulation where the developed climate system controller hardware will run with the vehicle and climate system components as models in a real-time environment.

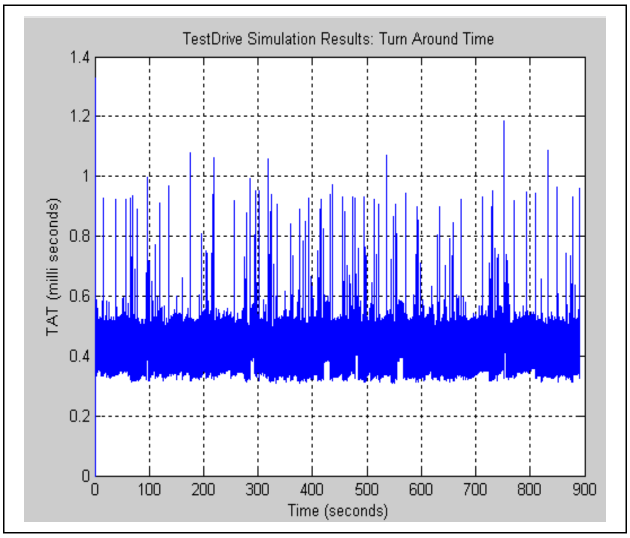


Figure 6 Turn around time of the model

The simulations were carried out for Japanese 10-15 mode drive cycle data. The most important metric to be measured for judging the real time capability of models is turn around time. The turn around time for this simulation is well below the chosen sample time of 10 ms as seen in Figure 6. The plant model results are shown for two different ambient conditions of 40°C and -10°C. In both the conditions, the controller is able to bring the cabin to the set temperature of 25°C. In the 40°C case, the HVAC compressor is switched on and off to control the temperature to 25°C (See Figure 7).

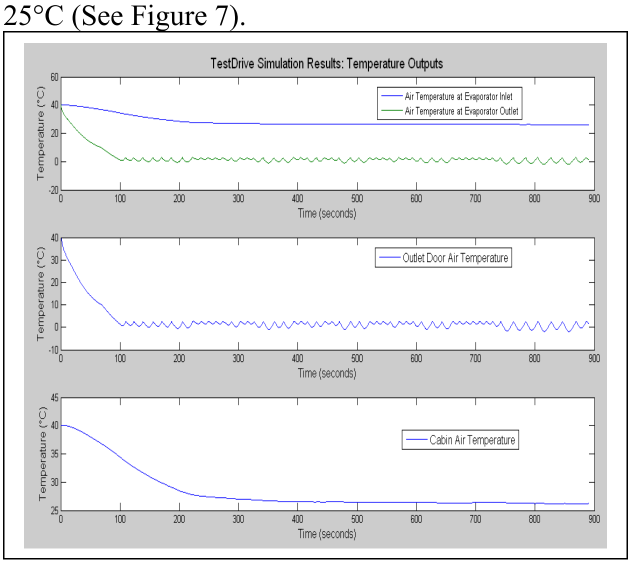


Figure 7 Results for 40°C ambient case

In the -10°C case, the heater comes in to play and heats the ambient air to the required 25°C. In this case, the HVAC compressor was hardly switched on by the controller (see Figure 8).

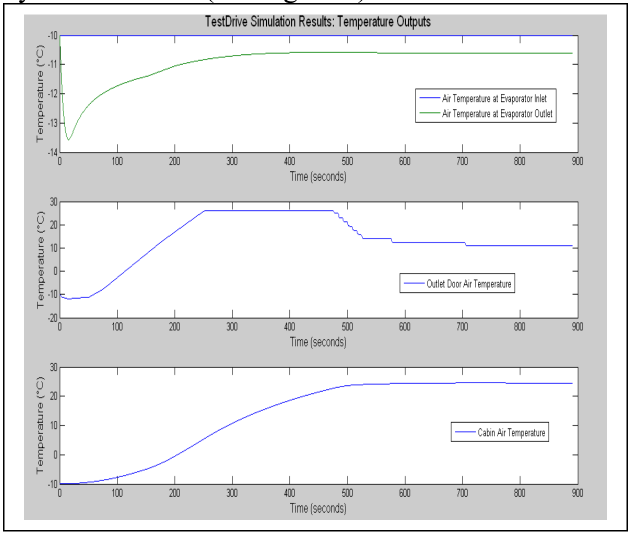


Figure 8 Results for -10°C ambient case

7 Conclusions

The exercise of running the closed loop model establishes the compatibility between the developed Modelica plant models with the chosen controller. The closed loop simulation is very beneficial for the following reasons:

- Validation of control strategies and functionality of control elements like heater, inlet door, *etc*.
- Comparison of different control strategies
- Fuel economy prediction

The real-time simulation of this integrated model in the TestDrive platform confirms that the developed models are amenable to Hardware-in-the-Loop simulations. It can be said that the level of fidelity that is captured in these plant models is sufficient for testing climate control logic, fuel economy studies and alternate vehicle configuration sensitivity. These models in the TestDrive hardware can be tested with any climate system controller with suitable input/output connections. In terms of plant modeling, the scope can be increased to add an e-HVAC powered by a battery instead of an engine. The impact of e-HVAC on conventional and hybrid vehicles can also be studied.

8 Acknowledgment

This work in parts was supported by Mr. Yutaka Kato of NEAT Co. Ltd., Japan and Dr. Yasunori Yokojima. Authors wish to thank Dr. Michael Tiller for his guidance and support throughout this work.

References

[1] Öner ARICI, Song-Lin YANG, Daniel HUANG and Emin ÖKER. Computer Model for Automobile Climate Control System Simulation and Application. Int.J. Applied Thermodynamics, Vol.2, (No.2), pp. 59-68, June-1999.

[2] Torge Pfafferott and Gerhard Schmitz. Implementation of a Modelica Library for Simulation of Refrigeration Systems. pp. 197-206. Proceedings of the 3rd International Modelica Conference, Linköping, November 3-4, 2003.

[3] Kim, Y. 1993. Two-Phase Flow of HCFC-22 and HFC-134a Through Short-tube Orifices. Ph.D. dissertation, Texas A&M University.

[4] Hilding Elmqvist, Sven Erik Mattsson, Hans Olsson, Johan Andreasson, Martin Otter, Christian Schweiger and Dag Brück. Real-time Simulation of Detailed Automotive Models. pp. 29-38. Proceedings of the 3rd International Modelica Conference, Linköping, November 3-4, 2003.

[5] J. Bäckman and M. Edvall. Using Modelica and Control Systems for Real-time Simulations in the Pulp. pp. 579-583. Proceedings of the 4th International Modelica Conference, Hamburg, March 7-8, 2005.

[6] Tillner-Roth, R., Baehr, H.D. An international standard equation of state for the thermodynamic properties of 1,1,1,2-tetrafluoroethane (HFC-134a) for temperatures from 170 K to 455 K at pressures up to 70 MPa. J. Phys. Chem. Ref. Data 26 (1994) 657-729.

[7] Corberan, J.M. and Melon, M.G. 1998. Modeling of plate finned tube evaporators and condensers working with R134a, Int. J. Refrigeration, Vol. 21, No. 4, pp. 273-284.

[8] M. Tiller, "Introduction to Physical Modeling with Modelica", Kluwer Academic Publishers, ISBN 0-7923-7367-7

[9] Dymola. Dynamic Modeling Laboratory, Dynasim AB, Lund, Sweden, http://www.-Dynasim.se

[10] Modelica, http://www.Modelica.org.

[11] OPAL-RT, http://www.opal-rt.com.

The AdvancedMachines Library: Loss Models for Electric Machines

Anton Haumer Christian Kral Hansjörg Kapeller Thomas Bäuml Johannes V. Gragger
Austrian Institute of Technology
Giefinggasse 2, 1210 Vienna, Austria
anton.haumer@ait.ac.at

Abstract

This paper presents how losses of electric machines are modeled as an extension of the `Modelica.Electrical.Machines` library. The theoretical background of the different loss models is elaborated and a Modelica implementation – the AdvancedMachines Library – is presented. Additional examples demonstrate the usage of the library.

Keywords: electric machines, losses, loss models

1 Introduction

Especially for simulations with the goal to determine energy consumption of a system – e.g. an electric vehicle – over a given load cycle, consideration of losses as well as the variation of losses with respect to load, speed and temperature is indispensable. Since the Modelica Standard Library (MSL) `Modelica.Electrical.Machines` provides only basic machine models which only take copper losses caused by constant winding resistor models into account, an extension of these machine models is desired.

The different losses that have to be considered are described in [5], [6], [7]:

- Copper losses in stator and rotor windings respectively rotor cage: These losses are coupled with voltage drops; they vary with the current flowing through the winding and are temperature dependent. Though the rotor cage of an asynchronous induction machine or even the winding may be built of another conductor material like brass or aluminum, these losses are usually called “copper losses”.
- Brush losses model the losses caused by the voltage drop across brushes, as needed for DC machines and slipring motors.
- Core losses in stator and rotor iron core: They vary with the quality of the used iron sheets as well as with magnetic flux and frequency of the magnetic field.
- Friction losses summarize friction at the surface of the rotor, at the bearings as well as windage losses caused by cooling fans that are mounted on the machine’s shaft. They are dependent on speed.
- Stray load losses are difficult to compute and/or measure with reasonable effort [9]. Therefore their magnitude is defined in standards ([6], [7]), but no hints are given for their variation with speed. In [8] the different sources of stray load losses are explained, which allows to define the variability of these losses with respect to current and speed.

Since copper losses are temperature dependent, it is necessary to provide the actual operation temperature of the windings. This can be done by connecting a thermal ambient model which calculates the operating temperatures depending on the actual losses [2]. The simplest thermal ambient model considers constant operating temperatures, however.

Although other than copper losses are not temperature dependent, thermal connectors for all losses are included in order to provide a proper implementation which dissipates all losses, enabling a correct energy balance. They are needed for coupling a detailed thermal model to the electrical machine model.

2 Loss Models

2.1 Copper Losses

These losses are modeled by temperature dependent resistances:

$$R_{Operation} = R_{ref} \cdot \left(1 + \alpha_{ref} \cdot \left(T_{Operation} - T_{ref}\right)\right) \qquad (1)$$

DOI: 10.3384/ecp09430103

where α_{ref} identifies the linear temperature coefficient of the specific material, with respect to the reference temperature T_{ref} :

$$\alpha_{ref} = \frac{\alpha_{20°C}}{1+\alpha_{20°C}\cdot\left(T_{ref}-293.15\right)} \tag{2}$$

The losses are calculated by $p_{loss} = v \cdot i$ where i identifies the current flowing through the resistor and $v = R_{Operation} \cdot i$ indicates the voltage drop across the component. Losses are dissipated to the component's thermal connector.

2.2 Brush Losses

The voltage drop *v* across brushes is considered to be independent of the current *i*. Nevertheless, the voltage drop changes its direction according to the direction of the current flow. In order to avoid numerical problems, we have to define a linear transition around zero according to Fig. 1. The voltage drop *v* is shown as a multiple of the nominal voltage drop *V*, whereas the parameter *I* defines the transition range of the current *i*.

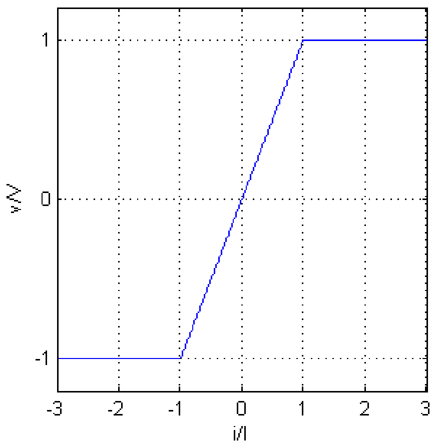


Fig. 1 Characteristic of the voltage drop across brushes

2.3 Core Losses

Changes of the magnetic field cause losses in the iron core which can be separated into hysteresis losses and eddy current losses. In order to avoid excessive eddy currents, the iron stack is built from sheet iron.

Assume we know the ratio of hysteresis losses r_H with respect to the total core losses for a reference point of operation. According to [14] the core losses can be expressed depending on angular frequency, *ω*, and flux respectively voltage, *v*:

$$p = p_{ref} \cdot \left(r_H \cdot \frac{\omega_{ref}}{\omega} + 1 - r_H \right) \cdot \left(\frac{v}{v_{ref}} \right)^2 \tag{3}$$

Therefore core losses can be modeled as a frequency dependent conductor:

$$G = \frac{p_{ref}}{v_{ref}^2} \cdot \left(r_H \cdot \frac{\omega_{ref}}{\omega} + 1 - r_H \right) \tag{4}$$

The core loss model can be connected either to the airgap model, or to the terminals.

Since the frequency of remagnetization, i.e. the velocity of the changes of the magnetic field, cannot be detected in Modelica so easily, the hysteresis losses are neglected in the first implementation ($r_H = 0$).

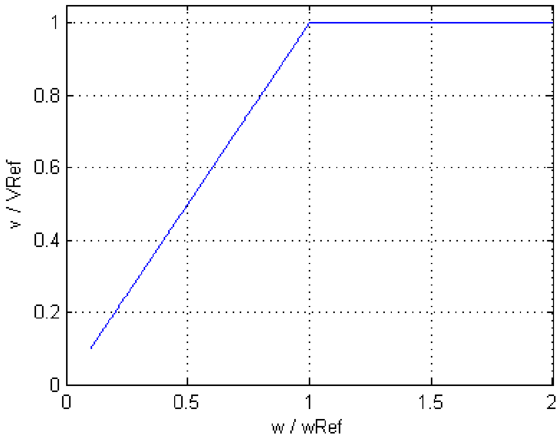


Fig. 2 Voltage versus angular velocity

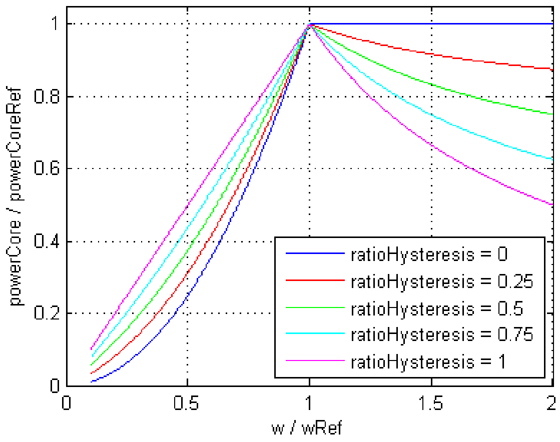


Fig. 3 Core losses versus angular velocity with parameter r_H (`ratioHysteresis`).

However, we should know the influence of this simplification. Fig. 2 shows the typical dependency of voltage on angular velocity for a variable speed drive. For such a voltage/frequency relationship the characteristic of core losses dependent on angular velocity is shown in Fig. 3. In the region of constant flux (`w/wRef<1`), the core losses are underestimated, and in the region of field weakening (`w/wRef>1`), the core losses are overestimated.

However, determining the correct velocity of the changes of the magnet field will be subject for further investigations.

2.4 Friction Losses

Friction losses are caused by different phenomena:

- The rotor surface rotates relative to the surrounding medium, normally air.
- The moving parts of the bearings cannot rotate frictionless.
- Cooling fans mounted on the machine shaft require torque respectively power to drive the medium, normally air. This amount of power is called windage loss.

All friction losses depend on speed, therefore they are calculated by the following equations:

For $|\omega| > \omega_{Linear}$:

$$\tau = sign(\omega) \cdot \frac{p_{ref}}{\omega_{ref}} \cdot \left|\frac{\omega}{\omega_{ref}}\right|^{power_{\omega}-1} \tag{5}$$

For $-\omega_{Linear} \leq \omega \leq +\omega_{Linear}$:

$$\tau = \frac{p_{ref}}{\omega_{ref}} \cdot \left(\frac{\omega_{Linear}}{\omega_{ref}}\right)^{power_{\omega}-1} \cdot \left(\frac{\omega}{\omega_{Linear}}\right) \tag{6}$$

A linearization around zero speed is defined by ω_{Linear} for stability reasons, according to Fig. 4.

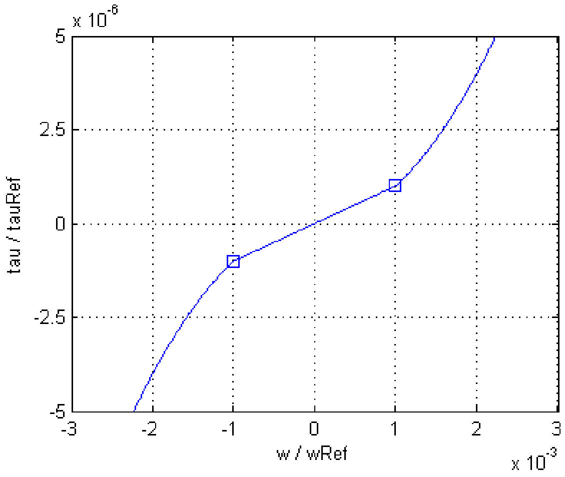


Fig. 4 Friction torque versus angular velocity

2.5 Stray Load Losses

Since stray load losses cannot be computed or measured with reasonable effort, their magnitude is defined in standards ([6], [7]). Unfortunately these standards deal with machines connected to a constant grid, operating at nearly constant speed. Therefore the results presented in [8] were taken as a basis to define the variability with respect to speed:

$$\tau = \frac{p_{ref}}{\omega_{ref}} \cdot \left(\frac{i}{I_{ref}}\right)^2 \cdot \left(\frac{\omega}{\omega_{ref}}\right)^{power_{\omega}-1} \tag{7}$$

The stray load losses are defined by reference power at reference conditions I_{ref} and ω_{ref}. For induction machines i respectively I_{ref} designate the magnitude of the current phasor divided by $\sqrt{2}$.

The dependency of stray load losses on speed is modeled with the exponent $power_{\omega}$.

Since a voltage drop associated with the stray load losses seems to be unphysical, they are modeled as a braking torque according to [7] acting on the shaft. Again, the stray load losses $p = \tau \cdot \omega$ are dissipated to the component's thermal connector.

For stability reasons, a similar linearization as for the friction losses (6) can be implemented additionally.

2.6 Thermal Connectors and Ambients

In order to provide operation temperatures to copper loss models, as well as to establish a proper power balance, all loss models are provided with thermal connectors `Modelica.Thermal.HeatTransfer.Interfaces.HeatPort`.

Each ready-to-use machine model instantiates a machine-specific super-port, containing heat ports for all loss models which are in turn connected to that super-port. From outside, this super-port has to be connected to an ambient model, which collects all losses as well as provides all operation temperatures. This implementation allows comfortable and flexible usage.

The simplest ambient models provide constant operating temperatures, as set by the user via parameters. More sophisticated ambient models contain detailed thermal model of the corresponding machine – simulating the actual component temperatures dependent on losses and cooling conditions. Such sophisticated models are planned for future releases.

3 The AdvancedMachines Library

3.1 Structure of the Library

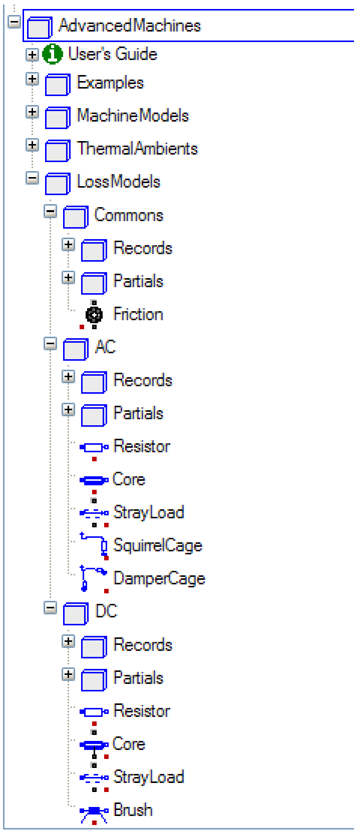


Fig. 5 Structure of the Advanced Machines Library

Besides a User's Guide and Examples, ready-to-use machine models with appropriate ambients as explained in 3.2 are provided. The loss models instantiated by the ready-to-use machine models are structured in 3 packages:

- Common to AC and DC machines
  - Friction losses
  - Parameter records
- Used by 3-phase AC machines
  - Temp. dependent resistor
  - Core loss model
  - Stray load loss model
  - Symmetrical squirrel cage
  - Asymmetric damper cage
  - Parameter records
- Used by DC machines
  - Temp. dependent resistor
  - Core loss model
  - Stray load loss model
  - Brush loss model
  - Parameter records

3.2 Ready-to-use Machine Models

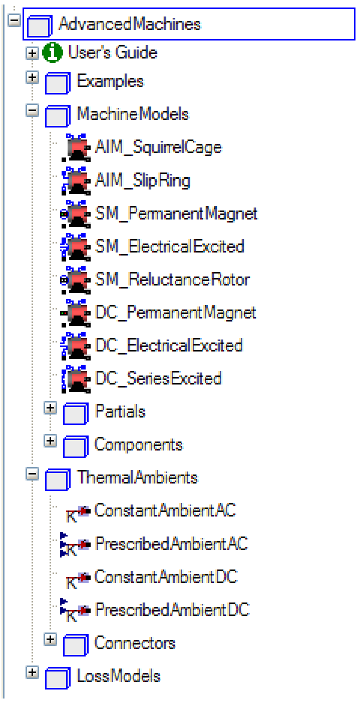


Fig. 6 Ready-to-use machine models and ambients

The AdvancedMachines library implements models for the same machine types as `Modelica.Electrical.Machines`, additionally taking losses into account:

- Asynchronous induction machines:
  - with squirrel cage
  - with slipring rotor
- Synchronous induction machines:
  - with permanent magnets
  - with electrical excitation
  - with reluctance rotor
- DC machines:
  - with permanent magnets
  - with electrical excitation
  - with series excitation

The ambient models provide either constant or prescribed temperatures with signal inputs, both for AC induction machines and DC machines. The user has to set the appropriate machine type by means of Boolean parameters, e.g. for an asynchronous induction machine with squirrel cage:

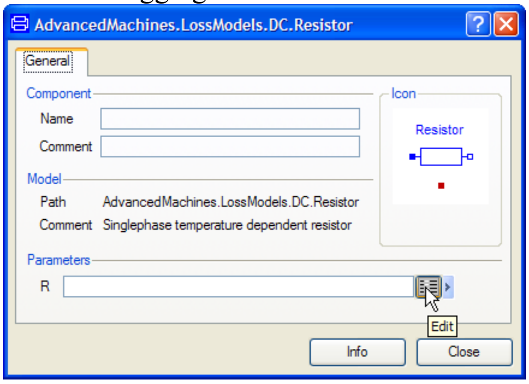
- `useRotor=true`
- `useRotorCage=true`
- `useExcitation=false`

to enable the appropriate heat ports in the super-port.

Future releases might split up these ambient models for more convenient usage, providing a specific ambient model for each machine type.

3.3 Parameterization

In order to avoid name conflicts respectively clumsy naming for the loss parameters, like `Ra_Rref` or `Rref_Ra`, all parameters needed for a specific loss model are aggregated in records:

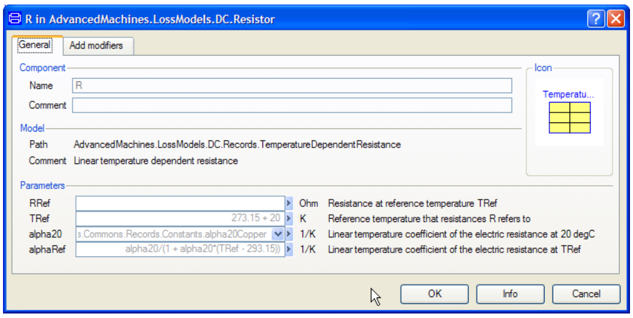

AdvancedMachines.LossModels.DC.Resistor

Component
Name:
Comment:

Model
Path: AdvancedMachines.LossModels.DC.Resistor
Comment: Singlephase temperature dependent resistor

Parameters
R:

Fig. 7 Parameter record of temperature dependent resistor


R in AdvancedMachines.LossModels.DC.Resistor

Component
Name: R
Comment:

Model
Path: AdvancedMachines.LossModels.DC.Records.TemperatureDependentResistance
Comment: Linear temperature dependent resistance

Parameters

RRef		Ohm	Resistance at reference temperature TRef
TRef	273.15 + 20	K	Reference temperature that resistances R refers to
alpha20	s.Commons.Records.Constants.alpha20Copper	1/K	Linear temperature coefficient of the electric resistance at 20 degC
alphaRef	alpha20/(1 + alpha20*(TRef - 293.15))	1/K	Linear temperature coefficient of the electric resistance at TRef

Fig. 8 Parameters of the temperature dependent resistor

These records allow to access parameters in an object oriented way, e.g. as `ra.R.Rref`.

Additionally, all parameters for a loss model are propagated by a single propagation of the appropriate parameter record, which allows convenient exchange and testing of different parameter settings.

The parameters of many loss models (FrictionLosses, CoreLosses, StrayLoadLosses) require to specify a reference value for the losses and corresponding reference conditions (like reference speed).

In many cases the user knows these values from manufacturer data or from test protocols, but wants to specify consistent parameter sets [12]. Consistent parameter sets means that the specified reference losses are dissipated exactly in the reference point of operation. Unfortunately, the specification of the reference point of operation (like nominal load) might be incomplete, e.g. with unknown voltage at the core loss model.

To help the user to define consistent parameter sets, a future release of the library will provide a parameter record for each ready-to-use machine model, calculating the missing reference values initially.

4 Simulation Examples

4.1 DC Permanent Magnet Machine

This example investigates the impact of losses on the behavior of a DC permanent magnet machine, based on the default machine data used in the MSL:

	Standard	Advanced	
	Machine	Machine	
Armature voltage	100	100	V
Armature current	100	100	A
Nominal speed	1425.0	1417.5	rpm
Inner voltage	95	94.5	V
Armature resistance	0.05000	0.03864	Ω
Temperature coefficient	n/a	0.00392	1/K
Reference temperature	95	20	°C
Operation temperature	n/a	95	°C
Brush Voltage drop	0	0.5	V
Linear transition current	n/a	1	A
Core Losses	0	200	W
Reference voltage	n/a	94.5	V
Reference speed	n/a	1417.5	rpm
Stray Load Losses	0	50	W
Reference current	n/a	100	A
Reference speed	n/a	1417.5	rpm
Friction Losses	0	100	W
Reference speed	n/a	1417.5	rpm
Electrical Input	10,000.00	10,000.00	W
Armature Losses	500.00	500.00	W
Brush Losses	0.00	50.00	W
Core Losses	0.00	200.00	W
Stray Load Losses	0.00	50.00	W
Friction Losses	0.00	100.00	W
Mechanical Ouput	9,500.00	9,100.00	W
Nominal Torque	63.66	61.30	Nm

Table 1 Parameters of both DC PM machines

The inner voltage at the airgap can be calculated as:

$$V_i = V_a - V_{Brush} - R_{a,operation} \cdot I_a = k \cdot \omega \qquad (8)$$

The magnet design of both models is considered to be the same. Since the inner voltage V_i of the AdvancedMachine is lower than that of the StandardMachine due to the brush voltage drop, the nominal speed of the AdvancedMachine has to be lower than that of the StandardMachine. Note that the nominal torque of the AdvancedMachine is lower than that of the StandardMachine at the same electrical input, due to the losses.

Both machines are started on voltage ramp (Fig. 9) with a duration of 0.8 s, starting at 0.2 s. At t=1.5 s a torque step (respective nominal torque for each model) is applied.

The final stationary resulting speed (Fig. 10) meets the values of Table 1. Both armature currents shown in Fig. 11 reach 100 A according to Table 1, but differences can bee seen during transient operation.

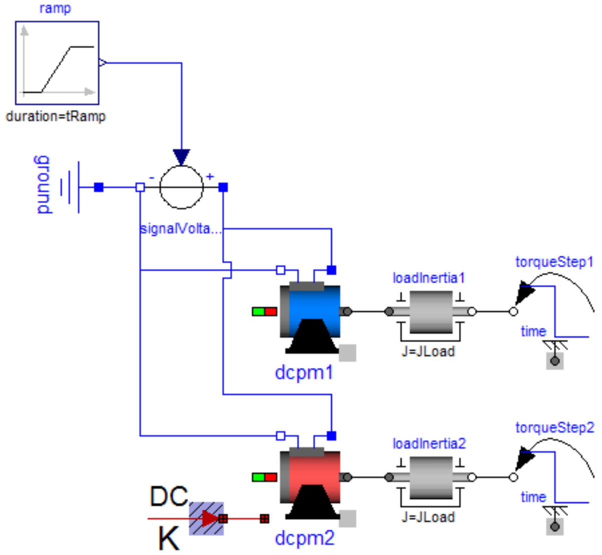


Fig. 9 Starting both DC permanent magnet machines

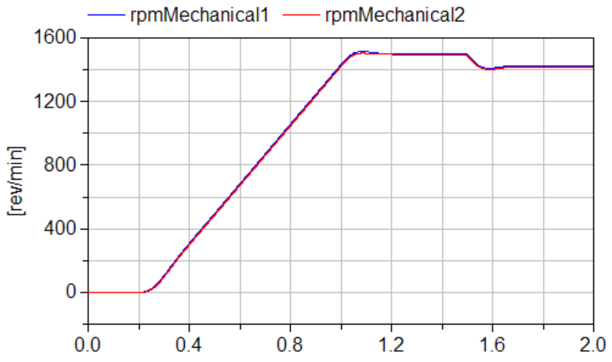


Fig. 10 Comparing speed of both machines

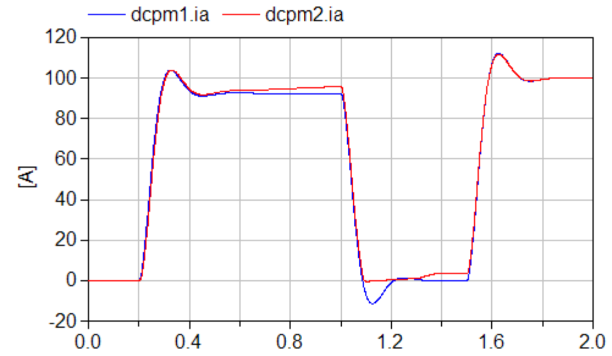


Fig. 11 Comparing armature currents of both machines

4.2 Asynchronous Induction Machine with Squirrel Cage

In order to validate the loss models, an asynchronous induction machine with squirrel cage (AIMC) is simulated for different partial loads. The results are compared with measurements of a 18.5 kW 4-pole standard motor (Table 2). The motor is of totally enclosed fan cooled design (Fig. 12), the rotor cage is made of aluminum (Fig. 13).

Nominal output	18,500	W
Nominal voltage	400	V
Connection	delta	
Nominal freuqency	50	Hz

Table 2 Nominal parameters of the AIMC

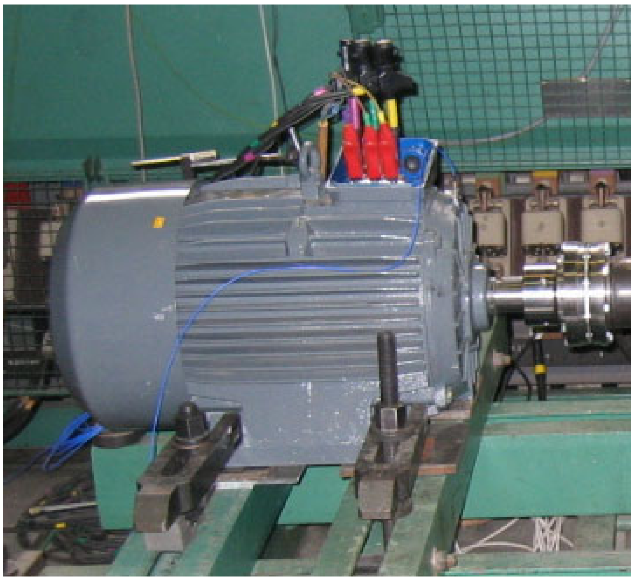


Fig. 12 The AIMC at the test bench

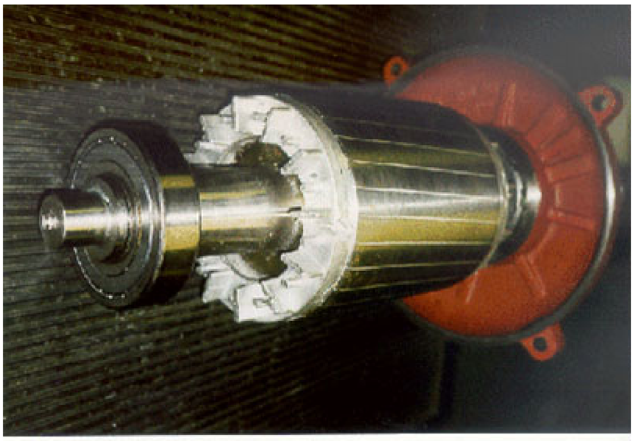


Fig. 13 Die cast aluminium rotor of the AIMC

Stator resistance / phase	0.560	Ω
Temperature coefficient	0.00392	1/K
Reference temperature	20	°C
Operation temperature	90	°C
Stator leakage reactance	1.520	Ω
Main reactance	66.400	Ω
Rotor leakage reactance	2.310	Ω
Rotor resistance / phase	0.420	Ω
Temperature coefficient	0.00400	1/K
Reference temperature	20	°C
Operation temperature	90	°C

Table 3 Impedances of the AIMC
rotor impedances with respect to the stator

The impedances (Table 3) were taken from the results of a conventional design program, the losses (Table 4) were taken from a type test protocol.

The voltage drop across the core loss conductance for nominal operation (V_{core} = 375.7 V) was iterated with the model shown in Fig. 14, which was used to obtain the results, too.

Stator current	32.85	A
Power factor	0.898	
Speed	1462.5	rpm
Electrical input	20,443.95	W
Stator copper losses	770.13	W
Core losses	410.00	W
Rotor copper losses	481.60	W
Stray load losses	102.22	W
Friction losses	180.00	W
Mechanical output	18,500.00	W
Efficiency	90.49%	
Nominal Torque	120.79	Nm

Table 4 Nominal operation of the AIMC

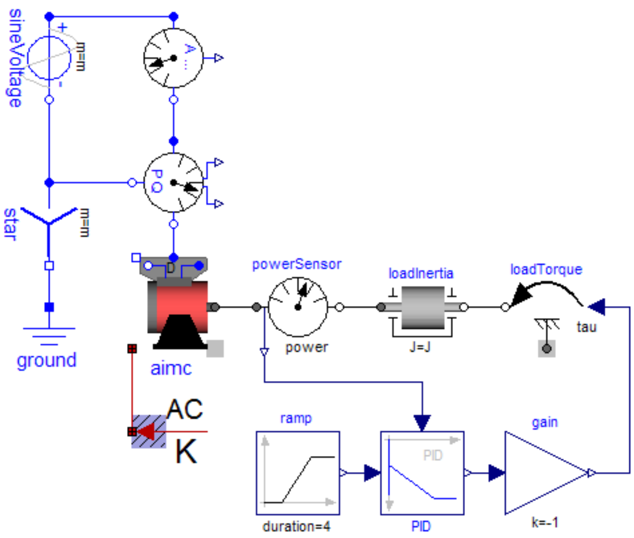


Fig. 14 Simulation model of the AIMC

The asynchronous induction machine is started at nominal speed and fed from a constant grid with nominal voltage and frequency. The load torque is controlled to achieve the desired mechanical power. After the initial transients have vanished, the set point for the mechanical power is raised with a very slow ramp to achieve quasi-stationary operation.

The comparison of measurement and simulation results (Fig. 15 – Fig. 17) shows very good coincidence which proofs the validity of the presented load models. Remaining differences can be explained due to measurement uncertainty, the fact that the calculated impedances used for parameterization do not match that of the real machine exactly which gives rise to deviations mainly in reactive power and last but not least the fact that the test protocol calculates the losses slightly different from the presented loss models.

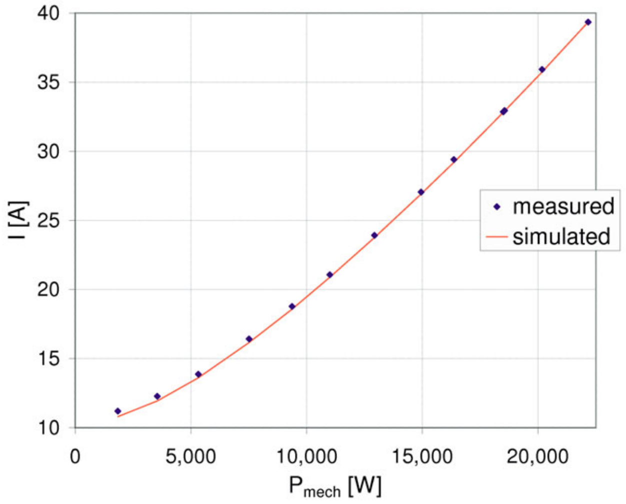


Fig. 15 Current of the AIMC, simulation results compared with measurements

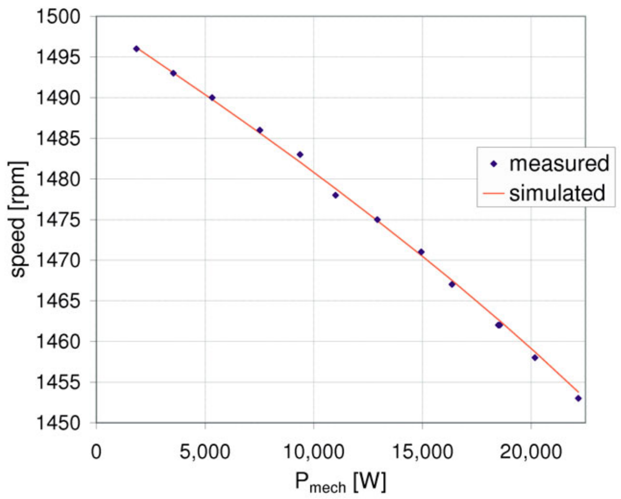


Fig. 16 Speed of the AIMC, simulation results compared with measurements

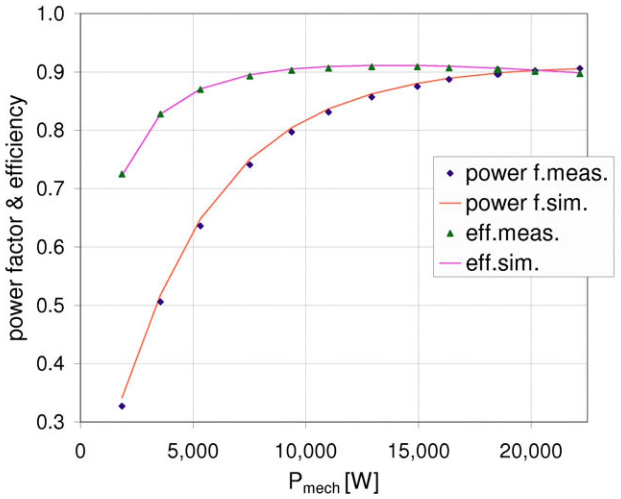


Fig. 17 Power factor and efficiency of the AIMC, simulation results compared with measurements

5 Conclusions and Outlook

The design of a Modelica library for electric machines with detailed loss models has been presented. The structure of the different loss models – copper losses, brush losses, core losses, friction and stray load losses – has been discussed in detail. Furthermore, the usage of the library has been presented with two examples, one of them providing a comparison between simulated and measured losses.

It is planned to release the library as a supplement for the SmartElectricDrives Library [4]. In this context also quasi-stationary models are of interest. As soon as it is possible to achieve a stable implementation of quasi-stationary machine models – which depends on the implementation of complex numbers in Modelica – as described in [3], the loss models will be adapted for the quasi-stationary machine models.

In future releases simple thermal models of electric machines will be offered. Additionally, the impact of coupling electric and thermal models will be investigated. Since the time constants of the electric and the thermal part are different, co-simulation could be considered to improve simulation performance.

References

[1] C. Kral, A. Haumer, Modelica libraries for dc machines, three phase and polyphase machines. 4th International Modelica Conference 2005, Hamburg, Germany

[2] C. Kral, A. Haumer, M. Plainer, Simulation of a thermal model of a surface cooled squirrel cage induction machine by means of the SimpleFlow-library. 4th International Modelica Conference 2005, Hamburg, Germany

[3] A. Haumer, C. Kral, J. Gragger, H. Kapeller, Quasi-Stationary Modeling and Simulation of Electrical Circuits using Complex Phasors. 6th International Modelica Conference 2008, Bielefeld, Germany

[4] J. Gragger, H. Giuliani, C. Kral, T. Bäuml, H. Kapeller, F. Pirker, The SmartElectric-Drives Library – Powerful Models for Fast Simulation of Electric Drives. 5th International Modelica Conference 2006, Vienna, Austria

[5] H. Kleinrath, Grundlagen elektrischer Maschinen, Akademische Verlagsgesellschaft, Wiesbaden, 1975

[6] Standard EN 60034-2, Verfahren zur Bestimmung der Verluste und des Wirkungsgrades von drehenden elektrischen Maschinen aus Prüfungen

[7] IEEE Standard 112, IEEE standard test procedure for polyphase induction motors and generators

[8] W. Lang, Über die Bemessung verlustarmer Asynchronmotoren mit Käfigläufer für Pulsumrichterspeisung, Doctoral Thesis, Technical University of Vienna, 1984

[9] M. Aoulkadi, A. Binder, When Loads Stray: Evaluation of Different Methods to Determine Stray Load Losses in Induction Machines, IEEE Industrial Electronics Magazine, 2008, Vol. 2, No. 1, p. 21-30

[10] H. Spaeth, Elektrische Maschinen - Eine Einführung in die Theorie des Betriebsverhaltens, Springer-Verlag, Berlin-Heidelberg-New York, 1973

[11] K.P. Kovac, I. Racz, Transiente Vorgänge in Wechselstrommaschinen, Band I, Verlag der Ungarischen Akademie der Wissenschaften, Budapest 1959

[12] C. Kral, A. Haumer, Consistent Equivalent Circuit Parameters of Induction Motors for the Calculation of Partial Load Efficiencies. IEEE International Symposium on Industrial Electronics, ISIE, Cambridge, United Kingdom, 2008

[13] M. Schelch, Motor-Pre-Calculator, Diploma Thesis, Technikum Wien, Vienna, 2005

[14] D. Lin, P. Zhou, W. Fu, Z. Badics, Z. Cendes, A dynamic core loss model for soft ferromagnetic and power ferrite materials in transient finite element analysis, Conference Proceedings COMPUMAG, 2003

A Modelica Library for High-Voltage AC Circuit-Breaker Modeling

Jörg Lehmann[1] Daniel Ohlsson[2] Hansjürg Wiesmann[1]
[1]ABB Switzerland Ltd., Corporate Research
Segelhofstrasse 1K, CH-5405 Baden-Dättwil, Switzerland
[2]ABB Switzerland Ltd., High-Voltage Products
Fabrikstrasse 13a, CH-5400 Baden, Switzerland
{joerg.lehmann,daniel.ohlsson,hansjuerg.wiesmann}@ch.abb.com

Abstract

The efficient simulation of the current interruption phase in high-voltage circuit breakers is routinely performed based on integral models for the gas dynamics and the arc physics. We present a Modelica framework for such a model. Based on `Modelica.Media`, a thermodynamic library for the required temperature and pressure range has been developed. Making use of the latest `Modelica.Fluid` library and the new streams concept, a library for the gas dynamics and the arc physics in such systems has been implemented.

Keywords: Circuit breakers; Thermodynamics and fluid dynamics of gases and plasmas.

1 Introduction

A high-voltage circuit breaker is a protection device in electrical power systems. It is able to switch between a conducting and an insulating state in the time frame of tens of milliseconds without losing its functionality. As a conductor the circuit breaker carries currents of several kA with minimal electrical losses. When an electrical fault occurs, the breaker can interrupt short-circuit currents of around 100 kA, resulting in an energy input from a burning arc of around 1 MJ. A circuit breaker has a lifetime exceeding 30 years and is operational between ambient temperatures of −50 °C and +40 °C, depending on its rating.

In the breaker chamber of the circuit breaker (see Fig. 1), the current is interrupted by separating two electrical contacts. During this phase, a high-temperature plasma arc is formed. The dissipated energy is radiated from the arc to the surrounding nozzles and leads to the ablation of material from the nozzle walls. This creates a flow of hot gas into an expansion volume with a typical size of some liters. When the pressure in this volume becomes high enough, the flow reverses. The gas flowing back extinguishes the hot plasma at the next zero crossing of the AC current, thus preventing a possible arc re-ignition. This is the so-called self-blast principle where the arc energy is used to extinguish the arc itself instead of achieving the same “blowing” effect by mechanical compression only.

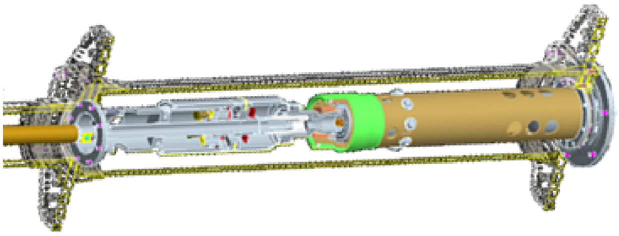


Figure 1: Breaker chamber inside of a high-voltage AC circuit breaker. The arc between the electrical contacts is formed in the central part.

The tuning of this process for optimal current-interruption capabilities is one of the main goals in circuit breaker design. In order to model this process, one needs to combine a broad range of fields such as plasma physics, thermodynamics, fluid dynamics and mechanics. In this context, simulation methods are routinely employed. Since more exact computational-fluid-dynamics methods typically require very long calculation times, integral models based on mass and energy conservation laws are still routinely used in the development process. The core of these models is the description of the arc in the high-current phase [1], i.e., the conversion of the electrical energy input in mass

DOI: 10.3384/ecp09430049

and thermal energy.

There are no public simulation tools for circuit breaker design available. Thus, the alternatives are either to adapt a commercial tool or to create an in-house tool from scratch. Previously, ABB has used such an in-house tool written in C. Although it is based on a well-structured, object-oriented design, incorporating changes requires a very detailed understanding of the C code, in particular in the model for the plasma arc, which forms the the most important and complex part. Moving the focus from software to model development, thereby enabling a faster development cycle, the idea was thus to replace the present tool by an implementation based on the Modelica language.

2 Modelica libraries

With the standard `Modelica.Media` and `Modelica.Fluid` [2, 3] libraries, Modelica provides a solid base for the implementation of integral fluid-dynamics models.

Here, we present a library framework which, on top of these libraries, implements the necessary components for the description of a high-voltage circuit-breaker. The goal is to focus on two aspects which are of general interest: (i) A regularization procedure for flow reversals. (ii) The switching between different states within the arc model, which is necessary due to dynamical changes in the flow topology.

2.1 Thermodynamic data at high temperatures and pressures

The electrical arc which is formed after opening the contacts is a high-temperature plasma consisting of a mixture of SF_6 used as insulation-gas and vaporized PTFE (Teflon) ablated from the nozzle walls. Both thermodynamic and transport data for this mixture are needed for a temperature range of up to several tens of thousands of Kelvins and pressures of up to several tens of bars. Except for temperatures below about 1200 K, the insulation medium is not a single-species gas consisting of neutral SF_6 molecules. Instead, it is composed of various ionization and dissociation products of SF_6 and additionally electrons. The composition of this mixture is strongly dependent on temperature. The same holds true for PTFE and other insulation gases. A rigorous treatment as a many-particle mixture in the spirit of the `Modelica.Media` library is cumbersome in many respects. A more efficient way to handle the complicated mixture is to treat both SF_6 and PTFE as “effective” gases, i.e., formally as one-particle gases and their mixture as a two-particle mixture. This becomes possible if the thermodynamic and transport properties are known as a function of the state variables pressure p and temperature T or equivalent quantities. The functional dependence reflects the complicated composition at higher temperatures which is the consequence of a large number of different scattering processes. This precludes an approach as used in the `Modelica.Media` library, where the thermodynamic quantities like density, enthalpy, etc. are expressed as polynomials in the state variables pressure, temperature and composition [4]. Instead our implementation interpolates these quantities from tabulated data. As an example we depict in Fig. 2 the specific heat of SF_6 and PTFE as a function of temperature. The pronounced peaks around 2000-4000 K reflect the various ionization and dissociation processes.

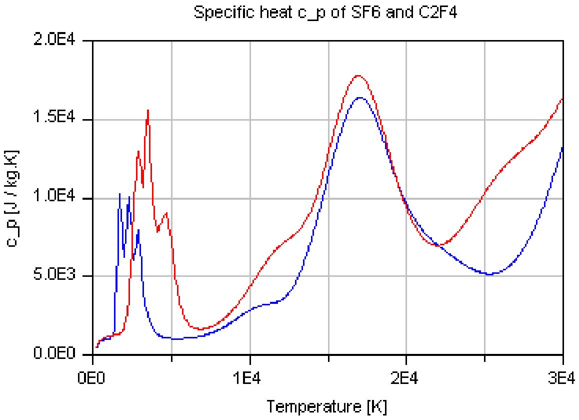


Figure 2: Specific heat of SF_6 (blue line) and PTFE (red line) as a function of temperature for a fixed pressure of 1 bar.

The table data are obtained from a separate, sophisticated program. Thermodynamic properties in local thermodynamic equilibrium are calculated as a function of pressure and temperature by minimization of the Gibbs potential [5]. This involves the calculation of particle densities of all involved species as a function of p, T and composition as well as their chemical potentials. The calculation of transport properties, in particular the electrical and thermal conductivity as well as the viscosity is based on the Chapman-Enskog approximation.

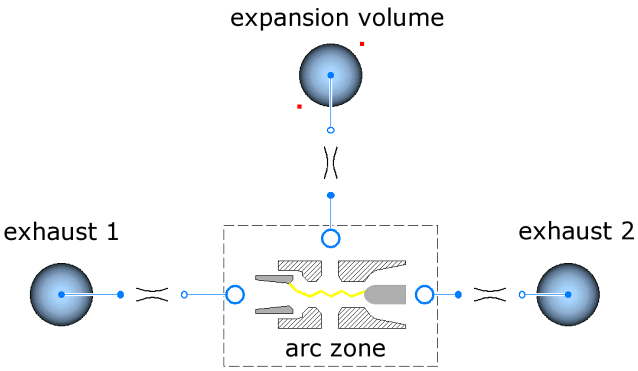


Figure 3: Basic model structure: Arc zone connected via three flow constrictions to the two exhausts and the expansion volume.

2.2 Gas dynamics and arc physics

The model for the gas dynamics, i.e., the fluid mechanics part of the simulation is based on the `Modelica.Fluid` library with the new streams concept of the Modelica 3.1 specification [3]. Figure 3 depicts the basic model structure: The arc zone in the center is connected via three flow constrictions to the two exhausts and the expansion volume.

The description of the exhausts and the expansion volume relies on the standard ideal mixing volume contained in the fluid library. The models for the flow constrictions and the arc zone will be described in the following sections as they are not implemented in the standard library.

2.2.1 Model for flow constrictions

The models contained in the fluid library describe the flow of a fluid through pipes in the presence of various friction mechanisms. We, however, are interested in the flow of gases at high velocities, even up to sonic speeds. Then, friction effects can be neglected to a good approximation and the flow is isentropic. The corresponding physical model will be given below. In order to achieve a robust treatment of flow-reversal situations, the dependence of the mass flow on the pressure drop needs to be regularized. We will describe in detail the approach we have taken, in particular, because it is of general interest and can be applied to other flow problems, as well.

For the model of the flow constriction, we assume a one-dimensional, isentropic flow of a compressible, polytropic gas [6], for which the mass flow $\dot{m}$ (up to a sign) is given by

$$\dot{m} = A\,\rho_{\mathrm{up}}\,c_{\mathrm{up}}\sqrt{\frac{2}{\gamma_{\mathrm{up}}-1}\left[r^{\frac{2}{\gamma_{\mathrm{up}}}} - r^{\frac{\gamma_{\mathrm{up}}+1}{\gamma_{\mathrm{up}}}}\right]} \tag{1a}$$

for $r > [2/(\gamma_{\mathrm{up}}+1)]^{\gamma_{\mathrm{up}}/(\gamma_{\mathrm{up}}-1)}$ and

$$\dot{m} = A\,\rho_{\mathrm{up}}\,c_{\mathrm{up}}\left(\frac{2}{\gamma_{\mathrm{up}}+1}\right)^{\frac{\gamma_{\mathrm{up}}+1}{2(\gamma_{\mathrm{up}}-1)}} \tag{1b}$$

otherwise (choked nozzle). Here, ρ_{up}, c_{up} and γ_{up} denote upstream values of the gas density, speed of sound and isentropic coefficient, respectively, $r = p_{\mathrm{down}}/p_{\mathrm{up}} \leq 1$ is the ratio between down- and upstream pressures, and A is the minimal cross-section of the constriction.

For small pressure differences between up- and downstream, Eq. (1a) behaves asymptotically as

$$\dot{m} = A\,\rho_{\mathrm{up}}\,c_{\mathrm{up}}\sqrt{\frac{2}{\gamma_{\mathrm{up}}}\,\frac{p_{\mathrm{up}}-p_{\mathrm{down}}}{p_{\mathrm{up}}}} \times \left[1 + \mathscr{O}\left(\frac{p_{\mathrm{up}}-p_{\mathrm{down}}}{p_{\mathrm{up}}}\right)\right]. \tag{2}$$

Here, the prefactor of the square-root singularity depends on upstream properties, and hence undergoes a discontinuous change in the case of a flow reversal. When implementing the model (1) in Modelica, this behavior has to be regularized in order to prevent instabilities of the solver near flow reversals.

The `Modelica.Fluid` library contains helper functions for such regularizations based on a polynomial interpolation scheme. Such schemes, however, have to be carefully adapted in order to guarantee monotonicity of the mass flow as a function of the pressure difference. This leads to quite intricate case distinctions for the coefficients of the polynomials. We have instead chosen a regularization scheme that allows one to more easily ensure the monotonicity property. To introduce this scheme, we use the notation $+$ and $-$ for the two ports of the flow constriction and write the mass flow as a function of the pressure difference $\Delta p = p_+ - p_-$ in the form

$$\dot{m}(\Delta p) = \begin{cases} \dot{m}_+(\Delta p) & \text{for } \Delta p \geq 0 \\ -\dot{m}_-(\Delta p) & \text{for } \Delta p < 0\,, \end{cases} \tag{3}$$

where $\dot{m}_\pm$ denotes the mass flow (1) with the respective upstream quantities. In particular, the pressure ratio r is replaced by $r_\pm = 1 \mp \Delta p/p_\pm$, where $\pm$ is the sign of Δp.

Instead of the exact flow function (3), we now consider a regularization of the form

$$\dot{m}(\Delta p) = \begin{cases} \dot{m}_+[\kappa_+(\Delta p)] & \text{for } \Delta p \geq 0 \\ -\dot{m}_-[\kappa_-(\Delta p)] & \text{for } \Delta p < 0\,, \end{cases} \tag{4}$$

where we only require that the functions $\kappa_\pm(\Delta p)$ are (i) monotonic, (ii) vanish at zero, $\kappa_\pm(0) = 0$, and (iii) behave as the identity for large magnitudes of the pressure difference, i.e.,

$$\kappa_\pm(\Delta p) \sim \Delta p \quad \text{for} \quad |\Delta p| \gg \delta p\,. \tag{5}$$

Here, the pressure δp is a parameter determining the size of the regularized region. The behavior for small Δp is used to regularize the flow. Requiring, for instance, continuous differentiability at $\Delta p = 0$, we impose the condition

$$\lim_{\Delta p \to 0-} \left\{ \dot{m}'_-[\kappa_-(\Delta p)]\, \kappa'_-(\Delta p) \right\} = \lim_{\Delta p \to 0+} \left\{ \dot{m}'_+[\kappa_+(\Delta p)]\, \kappa'_+(\Delta p) \right\}, \tag{6}$$

where the prime denotes differentiation with respect to the argument of the function. Note that this relation implies that the functions $\kappa_\pm(\Delta p)$ vanish sufficiently fast for $\Delta p \to 0$ in order to remove the singularity of the derivative of the mass flow.

As can be readily verified, all requirements stated in the previous paragraph are fulfilled by the choice

$$\kappa_\pm(\Delta p) = \frac{|\Delta p| \Delta p}{|\Delta p| + \delta p_\pm} \tag{7}$$

with

$$\delta p_\pm = \frac{\rho_\pm c_\pm p_\mp}{\rho_\mp c_\mp p_\pm} \sqrt{\frac{\gamma_\mp}{\gamma_\pm}}\; \delta p \tag{8}$$

where $\rho_\pm$, etc. denote the corresponding quantities at the respective port.

In Fig. 4, we show a comparison of the regularized flow with the original flow (1), which also shows the substantial change around flow reversal which occurs when the two temperatures T_+ and T_- differ strongly. The inset of this figure shows the regularization around flow reversal on a scale of the order of δp.

2.2.2 Electrical arc model

The most important and at the same time most complex component of the library consists of models for the description of the electrical arc and the surrounding vapor phase, which are formed after separating the electrical contacts in the circuit breaker. The importance of the arc process comes from the fact that it provides the conversion of the dissipated electrical power into mass and energy, which are then injected into the rest of the system.[1]

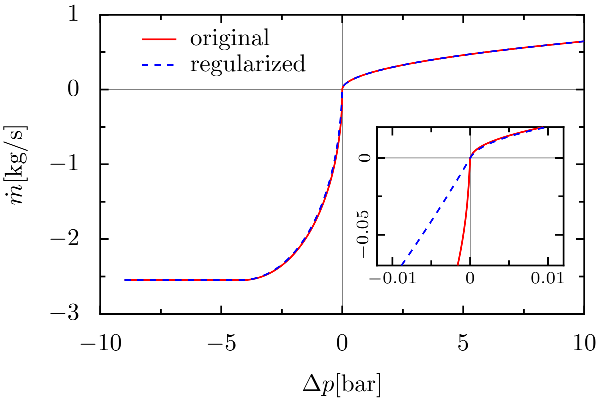


Figure 4: Mass flow of SF_6 across a flow constriction as a function of the pressure difference between the two ports. The flow (1) (solid line) is compared with its regularized version (4) (dashed line). Inset: Magnification of the behavior around flow reversal. The pressure $p_+ = 10$ bar is kept fixed, $\delta p = 0.05$ bar and the temperatures are $T_- = 3000\,°\text{C}$ and $T_+ = 25\,°\text{C}$.

Figure 5 shows sketches of the arc zone with the surrounding nozzles for two different times during the breaking cycle. Comparing with Fig. 3, one can identify the outlets to the expansion volume (in the middle) and to the two exhaust volumes (on the left and right). Furthermore, one can see the two electrical contacts: the fixed tulip on the left-hand side and the moving plug on the right-hand side. Between these contacts, the electrical arc is formed. Schematically we show the plasma arc (in yellow-orange) which carries the electrical current. It roughly has the form of a cylinder of length L_{arc} and radius R_{arc} and consists of a plasma with typical temperatures of the order of $T_{\text{arc}} \approx 20000\,\text{K}$. The plasma arc is surrounded by a zone of colder vapor (blue) with $T_{\text{vapor}} \approx 4000\,\text{K}$. The color gradient indicates the pressure distribution along the arc axis, which exhibits several pressure maxima corresponding to stagnation planes of the gas flow.

The integral description of the gas dynamics we use in our Modelica model, is based on such a two-zone model of a cylindrical arc surrounded by a vapor layer. As main parameters we provide geometric quantities, i.e., the length of the arc region as well as the radius of the surrounding nozzle, and the electrical current

[1] A part of the power is also lost in the form of radiation that leaves the system.

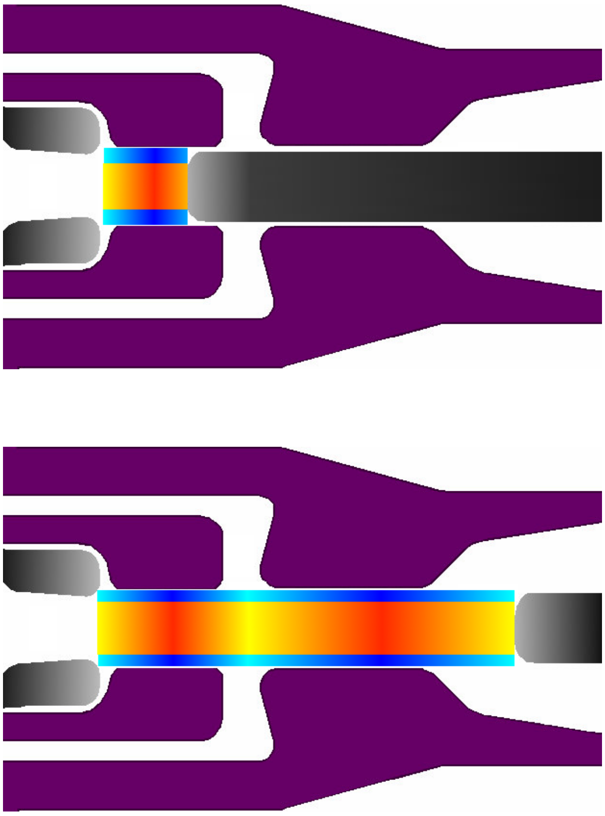


Figure 5: Schematic picture of the arc zone for two positions of the plug near the initial (top panel) and final (bottom panel) configuration in the interruption process.

which is imposed on the arc. Furthermore, the pressure in the volumes connected to the arc zone is needed. Our current model, which relies on a purely steady-state description of the arc, then calculates from these quantities the arc radius, the temperatures in the arc and vapor zones, the stagnation point pressures and the position of the stagnation planes. Our model contains an implicit system of non-linear equations relating these quantities. From the solution of this system, one then obtains the amount of PTFE ablated from the nozzle walls into the arc zone and thus the mass production in the system. Taking into account the position of the stagnation planes, one can calculate the distribution of the convective mass and energy fluxes in the three volumes connected to the arc zone.

More details of the arc model can be found in Ref. [1]. In the next paragraphs, we discuss two of the main issues we encountered when transferring this arc model from a C-based, algorithmic implementation into an equation-based form.

First, looking at Fig. 5, one can observe that the topology of the arc model changes along the breaking cycle. Initially (see top panel), the arc is only burning in a region between the left and middle outlet, while later (bottom panel) also the right outlet is directly connected to the arc. In order to map this situation to a Modelica model, we connect together two separate arc zones, which in the middle form a three-way connection to the expansion volume. At contact separation, when the arc ignites, we first “turn on” the left arc zone. Only later, when the plug has moved sufficiently far enough to the right, we switch on the right arc zone. When an arc zone is “turned off”, an arbitrary amount of gas can still flow across it without any pressure drop. This effectively eliminates any influence of the arc zone on the flow problem.

A second issue concerns the reliable convergence of the solutions of the non-linear system of equations for the arc towards the physically correct solution. This becomes particularly problematic when turning on an arc zone. At such a switch event, the Modelica language unfortunately does not allow specifying suitable initial conditions for the non-linear solver. Consequently, we had to resort to a different route: Even when an arc zone is not active, we still solve the system of arc equations. The various arc parameters are restricted to an appropriate range. In particular, they become continuous as a function of time, guaranteeing convergence of the iterative solution procedure. When the corresponding arc zone is turned on, we couple the arc zone model, i.e., the mass and energy fluxes and the allowed pressure drop, into the rest of the system. Here, we use an appropriate smoothing method.

2.3 Mechanics

As mentioned above, the current interruption process requires the mechanical separation of two electrical contacts. The mechanical motion can be strongly influenced by the pressure buildup in the breaker chamber. In these cases, the motion cannot be described by a predefined travel curve. Instead, the mechanical dynamics has to be modeled and coupled to the fluid-dynamics system. This has been realized by using `Modelica.Mechanics` as a base. An alternative is also to simulate, in a co-simulation setup, the mechanical system in a separate tool. Here, we used a named-pipe based interface to MSC.ADAMS.

3 Conclusions

With the Modelica based implementation of the breaker simulation tool, we have been able to replace the previously used C based tool by a mod-

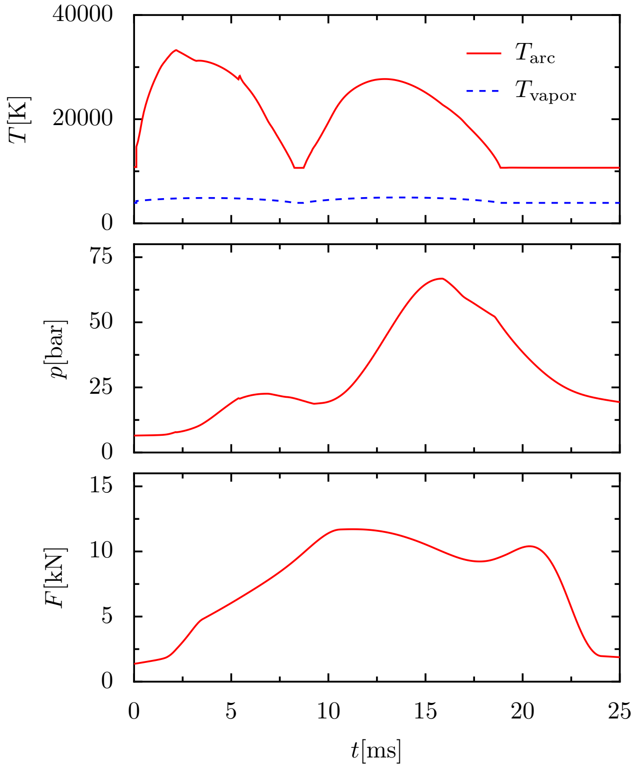


Figure 6: Simulation results from a complete breaker model. Top panel: Temperatures in the arc and the surrounding vapor zones. The two maxima reflect the sinusoidal variation of the imposed current. After arc extinction at $t = 19\,\text{ms}$, the temperatures in the model stay constant. Middle panel: Pressure in the expansion volume. Bottom panel: Reaction force on the mechanical drive. The time is measured from contact separation.

ern, more flexible implementation, which also provides the users with a more powerful simulation environment. A result for a complete breaker model is shown in Fig. 6. Of particular interest for dimensioning the breaker is the pressure buildup in the expansion volume (middle panel) and the reaction force on the drive (bottom panel). In contrast to more elaborate computational-fluid-dynamics simulations, the fast computation times for obtaining these results enable a rapid determination of the essential design parameters for the breaker layout with a reasonably good predictive power.

The transformation of a model written in C into an equation-based formulation was not as straightforward as we expected initially. In particular, achieving robust convergence of the solution of the non-linear system of arc equations, which is crucial for reliable use of the tool, turned out to be rather demanding. Here, it would be advantageous if the Modelica language provided the necessary means for an explicit, direct control of the non-linear solver, particularly the initial conditions for the iterative solution process after state events.

4 Acknowledgments

We thank Frank Kassubek for a useful suggestion regarding the flow-reversal regularization procedure and Fredrik N. Jansson for his work on the co-simulation interface.

References

[1] M. Claessens *et al.*, Progress in circuit-breaker modelling with respect to ablation controlled arcs, pressure build-up and performance limits, CIGRE report 13-112 (1994).

[2] F. Casella *et al.*, The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks, in *Proceedings of the 5th International Modelica Conference* (Modelica Association, Vienna, 2006).

[3] R. Franke *et al.*, Standardization of thermo-fluid modeling in `Modelica_Fluid` 1.0, in *Proceedings of the 7th International Modelica Conference* (Modelica Association, 2009).

[4] B.J. McBride, M.J. Zehe and S. Gordon, NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species, NASA report TP-2002-211556 (2002).

[5] G. Speckhofer *et al.*, A consistent set of thermodynamic properties and transport coefficients for high temperature plasmas, in *Proceedings of the 14th International Symposium On Plasma Chemistry*, edited by M. Hrabosky, M. Konrad and V. Kopecky (Institute of Plasma Physics, Prague, 1999), Vol. 1, p. 269.

[6] L.D. Landau and E.M. Lifshitz, Fluid Mechanics (Butterworth-Heinemann, Oxford, 1987).

Model of a Squirrel Cage Induction Machine with Interbar Conductances

Christian Kral Anton Haumer Bernhard Kubicek Oliver Winter
Austrian Institute of Technology
Giefinggasse 2, 1210 Vienna, Austria

Abstract

Conventional models of squirrel cage induction machines do not consider interbar currents. For the accurate numerical investigation of electrical rotor asymmetries interbar currents have to be modeled. A mathematical model of such a machine with interbar conductances is presented in this paper. An approach for the parametrization of the interbar conductances is introduced and discussed.

Keywords: Squirrel cage induction machines, electrical rotor asymmetries, interbar currents, machine model

1 Introduction

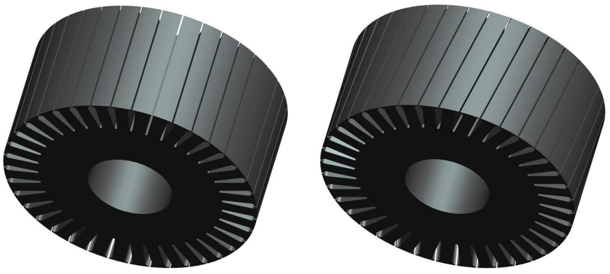
The electric conductors in the squirrel cage of an induction machine – the rotor bars and end rings – are usually either made of copper or aluminum. For die cast rotors the conducting material is not electrically insulated from the sheet iron due to the casting process. Even if copper and aluminum have a much greater electric conductivity than the sheet iron, some fractions of the rotor currents may flow through the rotor teeth, directly from bar to bar.

In mains supplied induction machines the rotor bars are usually *skewed*, i.e., the bars are twisted in tangential direction (Fig. 1 and 2) . Due to the skewing of the rotor bars the following effects occur:

1. Torque saddle points due to harmonic magnetic field waves can be avoided [1–3]

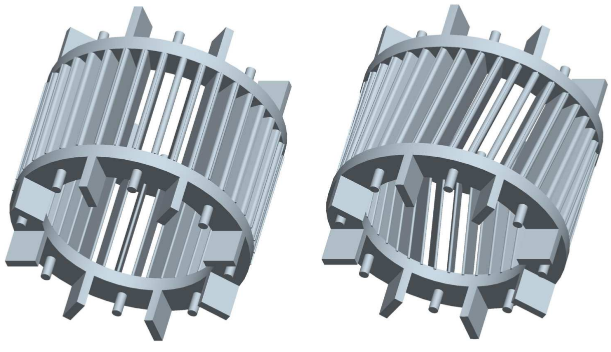
2. The torque speed characteristic and performance of the machine changes [4, 5]

3. Axial pull of the rotor caused by the tangential component of the bar currents



(a) unskewed rotor sheets (b) skewed rotor sheets

Figure 1: Rotor core of an induction machine



(a) unskewed cage (b) skewed cage

Figure 2: Squirrel cage of an induction machine, consisting of rotor core and a skewed or unskewed cage (bars and end rings)

4. Currents are flowing from bar to bar through the sheet iron – the so called *interbar currents*

Interbar currents significantly arise in the case of broken rotor bars or end ring segments. The intermittence of the electric conductance of either a bar or end ring segment causes the current to flow through alternative paths – in the sheet iron from bar to bar.

For the numerical investigation of interbar currents in squirrel cage induction machines finite element methods are usually applied [6–8]. Alternatively, models based on equivalent circuits can be developed to investigate the operational behavior of machines with inter-

DOI: 10.3384/ecp09430003

bar currents. In such models the skewing of rotor bars can be considered by means of skewing factors applied to the calculation of the mutual inductances between stator and rotor [9, 10]. Interbar currents of electrically asymmetrical rotor cages are yet more complex to model since the rotor has to be discretized in axial direction. A Modelica model considering such axial discretization is presented in this paper. To be more precise: the presented model takes interbar conductances into account which in turn conduct the interbar currents.

2 Model Structure

The proposed induction machine model with interbar currents is derived from the *ExtendedMachines* library which was presented in [11]. The *ExtendedMachines* library includes a model of the stator and rotor winding topology, core, friction and stray-load losses, an air gap model and thermal connectors for the coupling with a thermal model or environment. From this library the air gap, cage and winding function models are extended and modified such way that an axial discretization including interbar conductances is modeled.

A diagram of the proposed machine model is depicted in Fig. 3. The stator network considers the stator resistance (`rs`), the stator stray inductance (`lssigma`) and the air gap model (`airgap`), which can be seen as electromechanical power converter. Stray-load losses (`strayLoad`), core losses (`coreStray` and `coreTerminal`, respectively, for alternate use, depending on the model character) and friction losses (`friction`) are also taken into account. On the rotor side a multiphase squirrel cage (`cage`) is modeled. Each leg of the rotor phases has to be grounded (`ground`). Stator and rotor inertia (`intertiaStator`, `inertiaRotor`) are connected with the air gap model and the shaft end (`flange`) and housing (`support`) connector, respectively.

Any symmetric or asymmetric stator winding topology (`windingStator`) can be considered by specifying the location of the begin and end of each turn – for each phase. Details about the modeling of the stator winding topology are explained in [11]. The topology of the rotor cage is fully symmetrical and formed of the periodic structure of bars and end rings as depicted in Fig. 2. Electrical rotor asymmetries are model by modified rotor bar and end ring resistances. The topology model of the squirrel cage is presented in section 3.

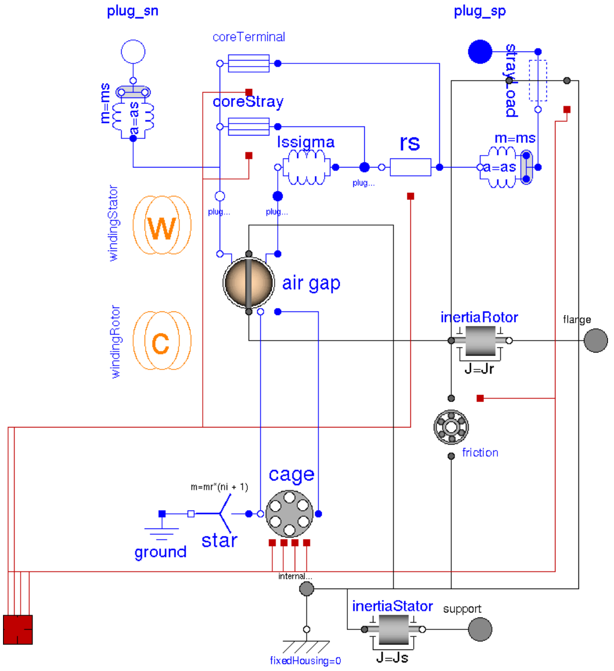


Figure 3: Modelica diagram of the proposed induction machine with interbar conductances

3 Cage Model

The presented cage model (`cage`) takes skewed rotor bars and interbar conductances into account. The interbar conductances are modeled in such a way, that each rotor bar is axially discretized and each intermediate node of two adjacent bars is connected by an an interbar conductance. A sketch of the rotor network model is presented in Fig. 4. This modeling approach does not consider current paths through the rotor yoke [12], which seems to be a reasonable simplification.

The voltages induced in the rotor meshes are calculated in the air gap model which is presented in section 4. These induced voltages are the terminal voltages $v_{[m]}$ of the cage model in Fig. 3. The currents associated with the multiphase ports of the cage model, $i_{[m]}$, are rotor mesh currents.

The cage consists of N_r rotor bars and thus N_r is also the number of end ring segments on both sides – the drive end (index a) and the non drive end (index b). In axial direction each rotor bar is subdivided into $n_i + 1$ segments. The rotor bar segments are tangentially connected by interbar conductances $G_{i[m]}$. In axial direction the rotor is thus modeled by n_i interbar conductors. Each of these conductors represents the conductance of the sheet iron including the contact conductance. Due to this structure the rotor cage consists of $(n_i + 1)N_r + 1$ elementary meshes, which are indicated

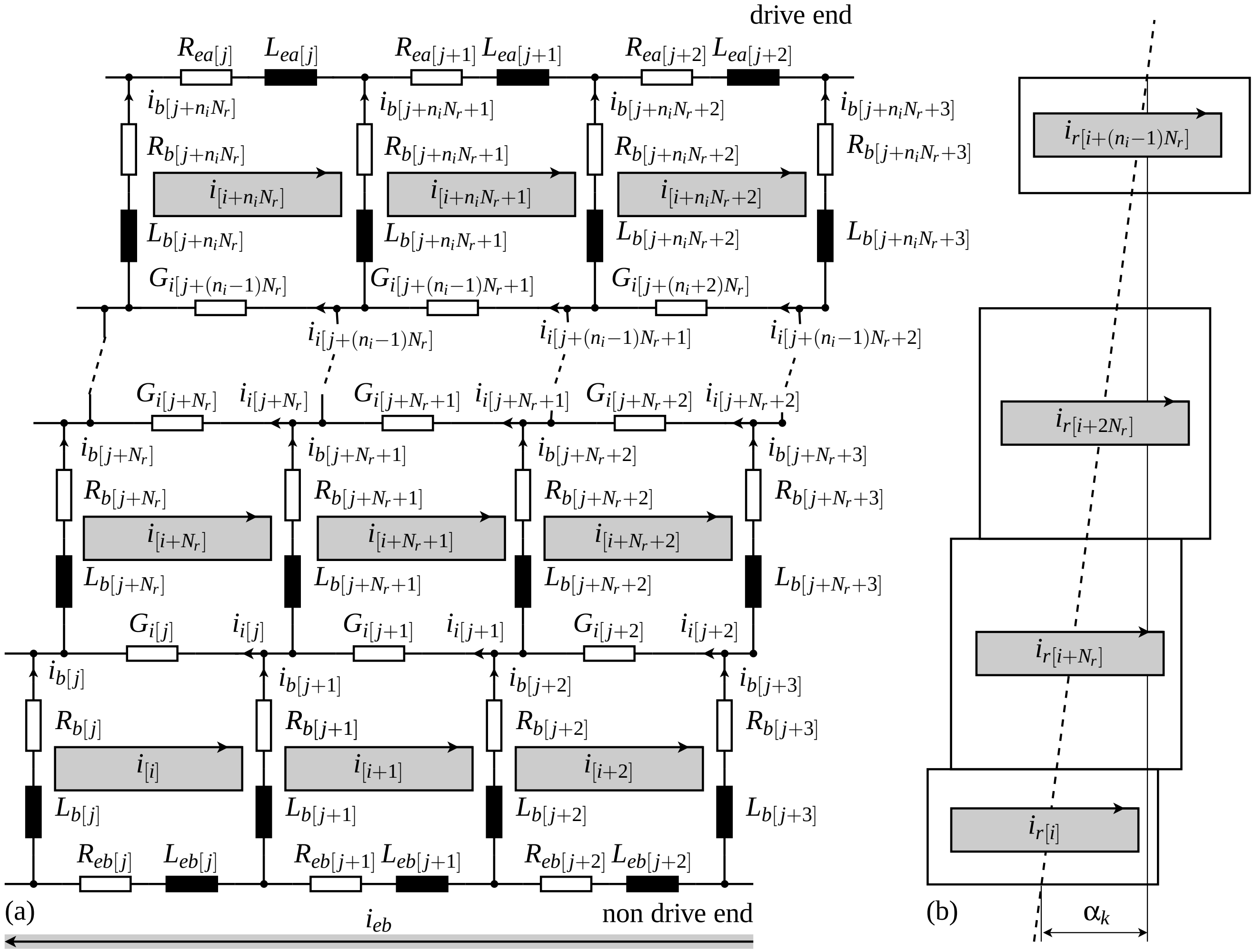


Figure 4: (a) Equivalent circuit of rotor cage with interbar conductances; (b) location of size of skewed rotor loops, skewing angle α_k

in gray in Fig. 4. For these elementary meshes the voltage equations have to be modeled.

The rotor voltage equation of the elementary mesh along the end ring on the non drive end yields

$$0=\sum_{m=1}^{N_r}\left(R_{eb[m]}(i_{[m]}+i_{eb})+L_{eb[m]}\frac{\mathrm{d}(i_{[m]}+i_{eb})}{\mathrm{d}t}\right). \tag{1}$$

From the remaining rotor meshes two different types of meshes have to be distinguished. An *end ring mesh*, is formed by one end ring segment, two bar segments and one interbar conductance. An *intermediate mesh* is formed of two bar segments and two interbar conductances. The indices of the meshes and the currents, respectively, are numbered in ascending order from left to right and from the non drive end to the drive end.

The rotor voltage equations for the end ring meshes of the non drive end are

$$\begin{aligned}0 &= v_{[m]}+R_{b[m]}i_{b[m]}+L_{b[m]}\frac{\mathrm{d}i_{b[m]}}{\mathrm{d}t}\\ &-R_{b[m+1]}i_{b[m+1]}-L_{b[m+1]}\frac{\mathrm{d}i_{b[m+1]}}{\mathrm{d}t}\\ &+R_{eb[m]}(i_{[m]}+i_{eb})+L_{eb[m]}\frac{\mathrm{d}(i_{r[m]}+i_{eb})}{\mathrm{d}t}-\frac{i_{i[m]}}{G_{i[m]}},\end{aligned} \tag{2}$$

for $m\in[1,2,\ldots,(n_i+1)N_r]$.

For the intermediate meshes the voltage equations are

$$\begin{aligned}0 &= v_{[m]}+R_{b[m]}i_{b[m]}+L_{b[m]}\frac{\mathrm{d}i_{b[m]}}{\mathrm{d}t}\\ &-R_{b[m+1]}i_{b[m+1]}-L_{b[m+1]}\frac{\mathrm{d}i_{b[m+1]}}{\mathrm{d}t}\\ &-\frac{i_{i[m]}}{G_{i[m]}}+\frac{i_{i[m-N_r]}}{G_{i[m-N_r]}},\end{aligned} \tag{3}$$

for $m\in[N_r+1,N_r+2,\ldots,n_iN_r]$.

The remaining end ring meshes on the drive end are

$$\begin{aligned} 0 &= v_{[m]} + R_{b[m]} i_{b[m]} + L_{b[m]} \frac{\mathrm{d} i_{b[m]}}{\mathrm{d}t} \\ &- R_{b[m+1]} i_{b[m+1]} - L_{b[m+1]} \frac{\mathrm{d} i_{b[m+1]}}{\mathrm{d}t} \\ &+ R_{ea[m-n_iN_r]} i_{[m]} + L_{ea[m-n_iN_r]} \frac{\mathrm{d} i_{[m]}}{\mathrm{d}t} + \frac{i_{i[m-N_r]}}{G_{i[m-N_r]}}, \quad (4) \\ &\text{for } m \in [n_iN_r+1, n_iN_r+2, \ldots, (n_i+1)N_r]. \end{aligned}$$

For the interbar currents the following equations

$$i_{i[j]} = i_{[j+N_r]} - i_{[j]} \tag{5}$$

apply for $j \in [1, 2, \ldots, n_iN_r]$. Additionally, the bar currents and the mesh currents are related by

$$i_{b[j]} = i_{[j+1]} - i_{[i+N_r]} \tag{6}$$

$$i_{b[i+k]} = i_{[i+k]} - i_{[i+k-1]} \tag{7}$$

and $j = N_r(n-1)$ with $n \in [1, 2, \ldots, n_i+1]$, and $k \in [2, 3, \ldots, N_r]$.

4 Air Gap Model

In the air gap model (`airgap`) the magnetic coupling of the m_s phase stator winding and the N_r phase rotor cage (or winding) is modeled. Due to the axial segmentation of the rotor bars, the air gap model and the cage model are coupled through a multiphase connector with $(n_i+1)N_r$ phases. The magnetic coupling between the windings of the stator and rotor is determined by the respective number of turns and the actual location of the turns with respect to each other. The relationship between the induced stator voltages, $v_{s[l]}$, and the induced rotor voltages, $v_{r[m]}$, and the respective currents $i_{s[l]}$ and $i_{r[m]}$ is expressed by

$$v_{s[j]} = \sum_{l=1}^{3} L_{ss[j,l]} \frac{\mathrm{d} i_{s[l]}}{\mathrm{d}t} + \sum_{m=1}^{(n_i+1)N_{ir}} \frac{\mathrm{d}}{\mathrm{d}t} \left(L_{sr[j,m]} i_{r[m]} \right), \tag{8}$$

$$v_{r[m]} = \sum_{j=1}^{3} \frac{\mathrm{d}}{\mathrm{d}t} \left(L_{rs[m,j]} i_{s[j]} \right) + \sum_{n=1}^{(n_i+1)N_r} L_{rr[m,n]} \frac{\mathrm{d} i_{r[n]}}{\mathrm{d}t}, \tag{9}$$

where $L_{ss[j,l]}$ are the stator inductances,

$$L_{sr[j,m]} = L_{rs[m,j]} \tag{10}$$

are the mutual stator and rotor inductances, and $L_{rr[m,n]}$ are the rotor inductances of the machine.

For the stator it is assumed that the complex winding factor, $\xi_{s[j]}$, and the number of turns, $w_{s[j]}$, can be determined for each phase based on the exact topology of each phase winding [13]. For the rotor cage the number of turns is equal to one,

$$w_{r[m]} = 1 \tag{11}$$

and the complex winding factor,

$$\xi_{r[m]} = \sin\left(\frac{\pi p}{N_r}\right) e^{-\mathrm{j}2\pi pm/N_r}, \tag{12}$$

is formed of the chording factor of two adjacent rotor bars, and the tangential location of the respective rotor mesh – assuming it were unskewed. In order to consider the skewing of the rotor bars accordingly, the tangential displacement of the rotor meshes belonging to two particular rotor bars, is modeled by a complex displacement factor, $\delta_{r[m]}$. The inductances in (8) and (9) can thus be expressed as

$$L_{ss[j,l]} = L w_{s[j]} w_{s[l]} \mathrm{Re}(\xi_{s[j]} \xi^*_{s[l]}), \tag{13}$$

$$L_{sr[j,m]} = L w_{s[j]} w_{r[m]} \zeta_{r[m]} \mathrm{Re}(\xi_{s[j]} \xi^*_{r[m]} \delta_{r[m]} e^{\mathrm{j}\gamma_m}), \tag{14}$$

$$L_{rr[m,n]} = L w_{r[m]} w_{r[n]} \zeta_{r[m]} \zeta_{r[n]} \mathrm{Re}(\xi_{r[m]} \xi^*_{r[j]} \delta_{r[m]} \delta^*_{r[n]}), \tag{15}$$

where L is the base inductance of one turn with a width equal to one pole pair. In this equations the expression

$$\zeta_{r[m]} = \begin{cases} \frac{1}{2n_i} & \text{if } m \text{ refers to an end ring mesh} \\ \frac{1}{n_i} & \text{if } m \text{ refers to an intermediate mesh} \end{cases} \tag{16}$$

applies, which determines the axial length of each end ring or intermediate mesh. The axial length of each mesh is modeled such way, that two bar segments of equal length e contribute to one interbar conductor (Fig. 5). Expression (16) is also used to determine the resistances and stray inductances of the bar segments with respect to the total bar resistance and stray inductance, respectively.

In the induction machine model of Fig. 3 the winding factors of the stator winding and the rotor cage are determined by the models `windingStator` and `windingRotor`, respectively. These models compute the winding factors, number of turns, displacement factors and (16) and propagate them to the air gap model. The input parameters are the skewing angle α_k (Fig. 4), the number of phases and the remaining topology data of the stator winding.

The electromagnetic torque of the induction machine is

$$T_e = \sum_{j=1}^{3} \sum_{m=1}^{(n_i+1)N_r} \frac{\partial L_{sr[j,m]}}{\partial \gamma_m} i_{s[j]} i_{r[m]}, \tag{17}$$

and applies to the stator and rotor inertia, with inverse signs, however. In this equation γ_m indicates the electrical angle of the rotor with respect to the stator.

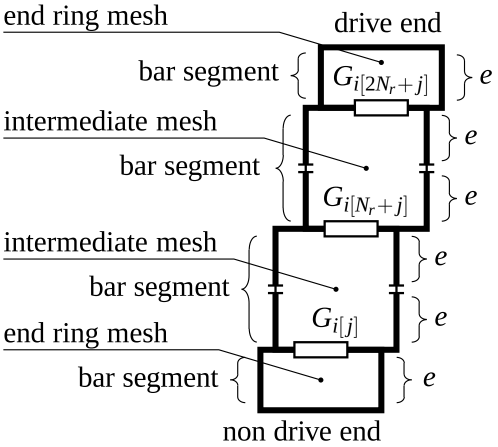


Figure 5: Sketch of two skewed rotor bars with $n_i = 3$ interbar conductances: two equal bar segments (length e) contribute to one interbar segment; this results in different bar segment lengths for end ring meshes and intermediate meshes

5 Practical Application

5.1 Model Development and Compilation

The model was developed with Dymola 6.0b and the C code was compiled in parallel on a 64 bit Linux cluster. The compilation and execution times of the models very much rely on the number of interbar segments, n_i. Two different simulations were carried out for the cases $n_i = 4$ and $n_i = 9$. For the second case a newer version of the Gnu C Compiler (GCC) had to be used in order to keep the compilation time in a reasonable range. The compilation of one model required approximately 8 GB memory and took one hour. Hence, a 64 bit GCC version was use to compile the 32 bit files. The simulation model was used in a project where 25 different machine configurations were investigated. Since the different configurations could be investigated independently the compilations and simulations were performed in parallel on different nodes on the Linux cluster.

For each model a total (real time) simulation time span of 12.5 s was performed. The cases $n_i = 4$ and $n_i = 9$ required 10 and 72 hours of CPU time, respectively. For the computation of each simulation in the project, a total of approximately 7000 hours of CPU time accumulated. During the comparison of these two cases it turned out, that $n_i = 9$ shows only a slight increase of accuracy and is not necessary for most investigations.

Figure 6: Squirrel cage rotor with respect to the configuration with one broken rotor bar (segment)

5.2 Parametrization of the Model

For the investigated project, the regular induction machine parameters were determined from a electromagnetic design tool. This way the resistances and stray inductances of the bars and end ring were calculated. The design tool does not calculate the interbar conductances, however, since these quantities very much rely on the manufacturing process and the contact conductances between the conductor and the sheet iron. In practice, the interbar resistances between two bars can usually not be measured without destroying the rotor cage [14–17].

Significant interbar currents arise in the rotor if one bar or end ring of the machine is broken. Therefore the configuration of one broken bar (Fig. 6) and one broken end ring segment (Fig. 7) are investigated. In order to compare measurement and simulations a method has to be applied which allows assessment of the effects caused by a broken rotor bar or end ring segment. Such a method is the Vienna Monitoring Method (VMM), originally introduced in 1997 [18–20]. The VMM calculates a fault indicator which is related with the fault extent. Interbar currents deteriorate the fault indicator and thus the comparison of measurement and simulation results can be used to parametrize the interbar conductances. In this context it is assumed that the interbar conductances of the entire rotor topology are equal

$$G_{i[m]} = \frac{G_{i\mathrm{ref}}}{n_i} \tag{18}$$

for $m \in [1, 2, \ldots, n_i N_r]$.

Measurement and simulations results for the two investigated configurations and $n_i = 4$ are depicted in Fig. 8:

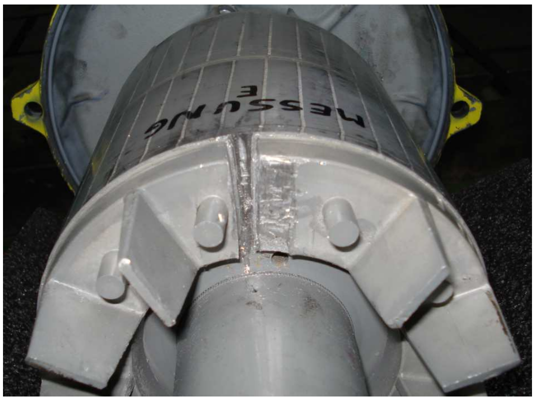


Figure 7: Squirrel cage rotor with respect to the configuration with one broken (removed) end ring segment

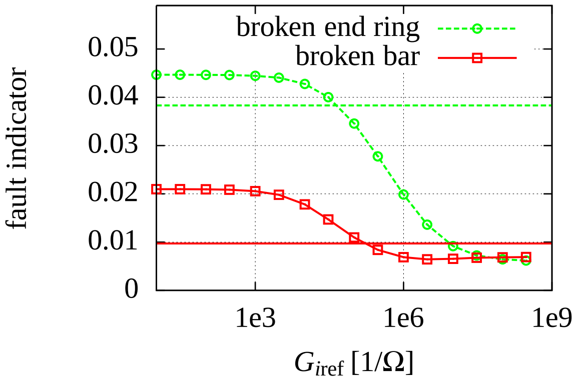


Figure 8: Fault indicator of the VMM versus interbar conductance G_{iref}

- *Broken end ring segment*: The measured fault indicator is shown as horizontal dashed line. The simulation result to match decreases with increasing total interbar conductance G_{iref}. The intersection of the simulation and measurement results indicates well parametrized simulation model.

- *Broken rotor bar*: The fault indicator obtained by measurements is depicted as horizontal solid line. The intersection of this line with the simulation result leads to a solution close to the one obtained for one broken end ring segment.

From the two intersection points of Fig. 8 an average value of $G_{\mathrm{iref}} = 2.3 \times 10^5 \, \frac{1}{\Omega}$ is obtained, which satisfies the required accuracy of both fault configurations. If the two intersection points were much further apart, this would indicate that the simulation model is not reflecting the real machine behavior.

6 Conclusions

This paper presents a mathematical model of a squirrel cage induction machine with interbar conductances and skewed rotor bars. For this purpose each rotor bar is axially subdivided into bar sections which are tangentially connected by interbar conductors. This way a discretized rotor topology model is developed. The Modelica models including equations of the air gap and cage are presented in detail.

The simulation model was used in a project where the impact of interbar currents, in combination with electrical rotor asymmetries are studied. In order to parametrize the interbar conductances two different fault configurations are investigated by measurements and simulations. The obtained parameter of the total interbar conductance confirms that the proposed simulation model leads to reasonable results and accurately reflects real machine behavior under rotor fault conditions.

References

[1] R. Weppler, “Ein Beitrag zur Berechnung von Asynchronmotoren mit nichtisoliertem Läuferkäfig,” *Archiv für Elektrotechnik*, vol. 50, pp. 238–252, 1966.

[2] J. Stèpina, “Oberwelleneinflüsse, Querströme und unsymmetrische Sättigung in der programmierten Berechnung von Einphasen-Asynchronmotoren,” *Siemens-Zeitschrift*, vol. 46, pp. 819–824, 1972.

[3] J. Stepina, “Querstroeme in Kaefiglaeufern,” *e&i*, vol. 92, pp. 8–14, 1973.

[4] W. Neuhaus and R. Weppler, “Einfluß der Querströme auf die Drehmomentkennlinie polumschaltbarer Käfigläufermotoren,” *ETZ-A*, vol. 88,3, pp. 80–84, 1967.

[5] Y. Kawase, T. Yamaguchi, Zhipeng Tu, N. Toida, N. Minoshima, and K. Hashimoto, “Effects of skew angle of rotor in squirrel-cage induction motor on torque and loss characteristics,” *IEEE Transactions on Magnetics*, vol. 45, no. 3, pp. 1700–1703, March 2009.

[6] G. H. Müller and C. F. Landy, “Finite element analysis of field distribution of squirrel cage induction motors having broken rotor bars and interbar currents,” *Proceedings of the International*

Conference on Electrical Machines, ICEM, pp. 577–581, 1994.

[7] S. L. Ho, H. L. Li, and W. N. Fu, "Inclusion of interbar currents in a network- field coupled time-stepping finite-element model of skewed-rotor induction motors," *IEEE Transactions On Magnetics*, vol. 35, no. 5, pp. 4218–4225, September 1999.

[8] Katsumi Yamazaki and Yuta Watanabe, "Interbar current analysis of induction motors using 3-d finite-element method considering lamination of rotor core," *IEEE Transactions On Magnetics*, vol. 42, no. 4, pp. 1287–1290, April 2006.

[9] G. Müller, *Elektrische Maschinen - Grundlagen, Aufbau und Wirkungsweise*, VEB Verlag Technik, Berlin, 4 edition, 1977.

[10] S.E. Zouzou, A. Ghoggal, A. Aboubou, M. Sahraoui, and H. Razik, "Modeling of induction machines with skewed rotor slots dedicated to rotor faults," *SDEMPED*, 2005.

[11] C. Kral and A. Haumer, "Simulation of electrical rotor asymmetries in squirrel cage induction machines with the extendedmachines library," *International Modelica Conference, 6th, Bielefeld, Germany*, , no. ID140, pp. 351–359, 2008.

[12] David G. Dorrell, Piotr J. Holik, Patrick Lombard, Hans-Jørgen Thougaard, and Finn Jensen, "A multisliced finite-element model for induction machines incorporating interbar current," *IEEE Transactions on Industry Applications*, vol. 45, no. 1, pp. 131–141, 2009.

[13] C. Kral and A. Haumer, "Modelica libraries for DC machines, three phase and polyphase machines," *International Modelica Conference, 4th, Hamburg, Germany*, pp. 549–558, 2005.

[14] S. Williamson, C. Y. Poh, and A. C. Smith, "Estimation of the inter-bar resistance of a cast cage rotor," *IEEE International Electric Machines and Drives Conference, IEMDC, Wisconsin, USA*, vol. 2, pp. 1286–1291, 2003.

[15] D. G. Dorrell, T. J. E. Miller, and C. B. Rasmussen, "Inter-bar currents in induction machines," *IEEE Transactions on Industry Applications*, vol. 39, no. 3, pp. 677–684, May-June 2003.

[16] D. Gersh, A. C. Smith, and A. Samuelson, "Measurement of inter-bar resistance in cage rotors," *Conference Proceedings of the Eighth International IEE Conference on Electrical Machines and Drives, EMD*, , no. 444, pp. 253–257, 1997.

[17] R. F. Walliser and C. F. Landy, "Assessment of interbar currents in double cage induction motors with broken rotor bars," *IEEE Transactions on Energy Conversion*, pp. 159–164, 1994.

[18] R. Wieser, C. Kral, F. Pirker, and M. Schagginger, "On-line rotor cage monitoring of inverter fed induction machines, experimental results," *Conference Proceedings of the First International IEEE Symposium on Diagnostics of Electrical Machines, Power Electronics and Drives, SDEMPED*, pp. 15–22, 1997.

[19] R. Wieser, C. Kral, F. Pirker, and M. Schagginger, "Rotor fault detection of inverter fed induction machines including experimental results," *Seventh European Conference on Power Electronics and Applications, EPE*, vol. 2, pp. 2532–2538, 1997.

[20] R. Wieser, C. Kral, F. Pirker, and M. Schagginger, "On-line rotor cage monitoring of inverter-fed induction machines by means of an improved method," *IEEE Transactions on Power Electronics*, vol. 14, no. 5, pp. 858–865, September 1999.

Enforcing model composability in Modelica

Sébastien Furic[1]
[1]LMS Imagine, France
sebastien.furic@lmsintl.com

Abstract

Modelica provides intuitive constructs to create and group *model definitions*. However, models themselves do not *compose*. In other words, the connection of type-compatible and locally balanced submodels does not generally yield a valid (e.g., balanced, structurally non-singular) model. Starting from simple examples of such invalid models (resulting from commonly encountered situations when using Modelica), this paper explains how those problems could be avoided by introducing a safer notion of physical connector, similar in some aspects to the VHDL-AMS notion of *terminal*. An extension of the notion of *connection* is also presented, providing new opportunities to make efficient use of ideal models in Modelica.

Keywords: model composition; high-level physical connector; effort variable; flow variable; connection graph; effort graph; flow graph

1 Introduction

A commonly encountered situation in Modelica when defining models by connection of submodels representing well-identified part of the whole design is that, even if each submodel has been checked for the absence of structural inconsistency, there is no guarantee that the result is itself structurally consistent. One may argue that this situation is normal since some combinations of models are “not physical”. However, in the acausal modeling world, we know that one has to be careful about the “not physical” argument: for instance, “high-index” problems also are “not physical” from a certain point of view. But every experienced Modelica user knows that models yielding systems having non-minimal state can be given a meaning, and the result of accepting those models does not make Modelica an “everything runs anyway” simulation language: models still have well-defined semantics and Modelica can be used to express hard problems directly, without need for user assistance.

We claim that, given a proper notion of *model composition* (by generalizing the semantics of connections), we can, as in the case of models having non-minimal state mentioned above, give a sound meaning to a larger class of assemblies of type-compatible Modelica submodels. And fortunately the result is still not an “everything runs anyway” simulation language, but a more powerful one, where new kinds of models can be easily defined, offering Modelica new valuable possibilities.

2 Modelica modeling annoyances

Reading the documentation of the Modelica Standard Library reveals implicit assumptions made here and there, leading to difficulties when one has to use models in a given discipline. Let's consider for instance the Electrical library[1]: in the documentation we can read this definition of Modelica.Electrical.Analog.Basic.Ground:

“Ground of an electrical circuit. The potential at the ground node[2] is zero. Every electrical circuit has to contain at least one ground object.”

The last sentence is rather confusing for the newcomer: why should every electrical model contain a “ground node”? After all, we all remember having in the past built paper-and-pencil electrical circuits without ground but for which it was possible to attribute unambiguously a meaning. Worse, the sentence seems to imply that sometimes *more than one* ground submodel has to be used: how many exactly for a given problem? And where to place them on the circuit? Let's make some experiments...

[1] In the present paper, we will focus on the electrical domain, because models are often simple yet general enough to illustrate our thesis.

[2] Notice the use of “node”.

DOI: 10.3384/ecp09430023

2.1 A first example

Sometimes using Modelica results in frustrating experiences for the newcomer: even a simple R circuit may not simulate! Figure 1 below shows the diagram of such a circuit built under Scicos[3]:

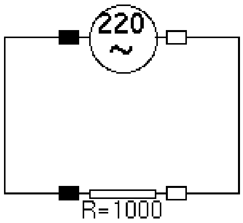


Figure 1: A naive R circuit

The system seems to be well defined: a voltage source imposes a voltage difference between the pins of a resistor and the resistor imposes the current flowing into the circuit. We can verify that by inspecting the equivalent "flat" model that, according to Modelica semantics, is:

```
model RCircuit
  Real src.p.v;
  Real src.p.i;
  Real src.n.v;
  Real src.n.i;
  Real src.v;
  Real src.i;
  Real res.p.v;
  Real res.p.i;
  Real res.n.v;
  Real res.n.i;
  Real res.v;
  Real res.i;
equation // generated by Source
  src.v = src.p.v - src.n.v;
  src.p.i + src.n.i = 0;
  src.i = src.p.i;
  src.v = 220 * sin(314.15 * time);
equation // generated by Resistor
  res.v = res.p.v - res.n.v;
  res.p.i + res.n.i = 0;
  res.i = res.p.i;
  res.v = 1000 * res.i;
equation // connection equations
  src.n.v = res.n.v;
  res.p.v = src.p.v;
  src.n.i + res.n.i = 0;
  res.p.i + src.p.i = 0;
end Rcircuit;
```

That system captures the required constraints. Indeed from:

```
src.v = src.p.v - src.n.v;
res.v = res.p.v - res.n.v;
src.n.v = res.n.v;
res.p.v = src.p.v;
```

we deduce, as expected, that:

```
src.v = res.v;
```

And then, from:

```
res.v = 1000 * res.i;
src.v = 220 * sin(314.15 * time);
```

we found that:

```
res.i = 0.220 * sin(314.15 * time);
```

Then by exploiting:

```
res.i = res.p.i;
res.p.i + src.p.i = 0;
src.i = src.p.i;
```

we deduce that:

```
src.i = -res.i;
```

Everything seems fine... However the simulation of that simple model fails miserably. Why? The careful reader may have noticed that, in order to resolve the above system for the "variables of interest" (`src.v`, `res.v`, `src.i` and `res.i`), we did not use all the constraints. If we would have done so, we would have found that the unused constraints:

```
src.n.i + res.n.i = 0;
src.p.i + src.n.i = 0;
res.p.i + res.n.i = 0;
```

coupled with the fact that:

```
res.i = res.p.i;
src.i = src.p.i;
```

would have lead to an over-constrained (but consistent) system of equations having five equations and only four unknowns. Also, we did not solve the system for all the variables: `src.p.v`, `src.n.v`, `res.p.v` and `res.n.v` remain undetermined. Trying to determine their value leads to the opposite problem we encountered while solving for the flow variables: the subsystem is under-constrained.

From the mathematical analysis point of view, our system has a *singular jacobian matrix*. However, that singularity is not the result of unfortunate values taken by the coefficients of the matrix: even the *incidence matrix*[4] of the system is singular.

[3] Freely available modeling tool with Modelica capabilities (http://www.scicos.org).

[4] A matrix having the same size as the jacobian matrix, and whose coefficients indicate eventual contributions of each variable to the corresponding jacobian matrix

Like many other newcomers the user that built the model on Figure 1 would probably be said that she/he has overlooked the advice given in the documentation of Modelica.Electrical.Analog.Basic.-Ground: indeed, adding a ground submodel to the circuit would have saved the situation by introducing the missing voltage equations and one degree of freedom for the current needed for the calculation of all the absolute voltages (that we don't need to know) and “outgoing currents” (that we don't need to know too).

Things learned from that error are:

- using Modelica libraries implies learning (sometimes implicit) rules *that are not enforced at the language level*
- Modelica forces users to give equations to compute unneeded quantities

2.2 Ideal models

Another look at the documentation that comes with the Electrical library reveals that the ground submodel is not the only one that carries out structural modeling assumptions. For instance, the description of Modelica.Electrical.Analog.Ideal.IdealOpeningSwitch includes the following warning:

“In order to prevent singularities during switching, the opened switch has a (very low) conductance Goff and the closed switch has a (very low) resistance Ron. The limiting case is also allowed, i.e., the resistance Ron of the closed switch could be exactly zero and the conductance Goff of the open switch could be also exactly zero. Note, there are circuits, where a description with zero Ron or zero Goff is not possible.”

It is legitimate to ask oneself what is a *low conductance* (resp. *low resistance*). Also, which are those circuits that disallow zero Ron and/or zero Roff?

The difficulty when one has to determine the appropriate conductance (resp. resistance) of a nearly-ideal opening switch is that the answer depends both on the circuit itself and on the compiler/solver pair.

Consider a model having several switches in parallel (for instance, a model of a fault-tolerant circuit in a nuclear plant): it is well-known that the two-pin model equivalent to those switches put together is a variable resistor whose resistance is given by the product of the variable resistances of the individual switches divided by the sum of those same resistances. It follows that small (resp. big) values of the individual resistances may lead to tiny (resp. huge) values for the resistance of the equivalent two-pin model. This implies that for large electrical models, where one cannot predict the sequences of openings/ closings of switches, it is nearly impossible to determine with a high degree of confidence a suitable value for the resistances of the switches, even if one got the complete source code describing the circuit model.

Concerning circuits where a description with zero Ron or zero Goff is not possible, experimenting a little with the language rapidly convinces us that the *connection* of the graph of linked model instances is a key property, and it brings us back to the problem of effort reference depicted in section 2.1: each *connected component* of a circuit has to contain one `Ground` instance. Consider the following circuit:

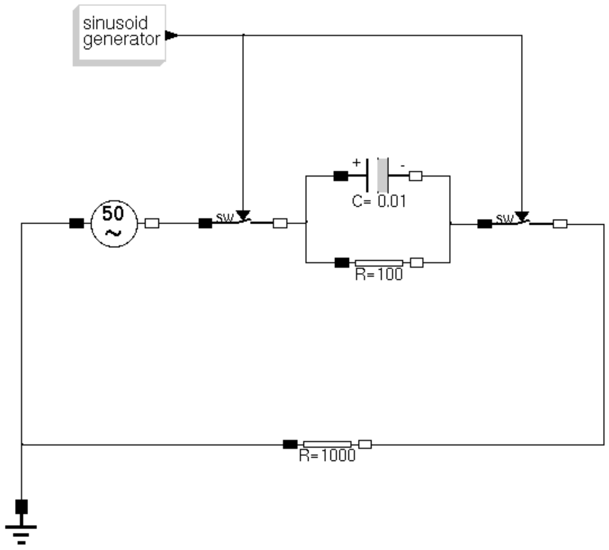


Figure 2: A circuit with switches

In this circuit, the two coupled opening switches control the connection of two subcircuits: when the switches are closed the capacitor is charged, and when the switches are opened, the capacitor discharges itself through the resistor connected in parallel with it.

What happens if we use ideal switches? If both switches are opened, we get two subcircuits, one of them having no `Ground` instance attached. We are then in the same situation as depicted in Figure 1: the solver fails due to the structurally singular jacobian matrix. But if we add a `Ground` instance as in Figure 3 below, the simulation can be run... until both switches are closed, in which case we have a connected graph of model instances with two effort ref-

coefficient: zero indicates no contribution and one indicates a (possibly null) contribution.

erences. Removing the added `Ground` instance and forcing the switches to remain closed allows the model to be simulated. So we are in a situation where, in order to be able to simulate our model, we have either to put or to remove a `Ground` instance from the circuit, depending on the state of the switches.

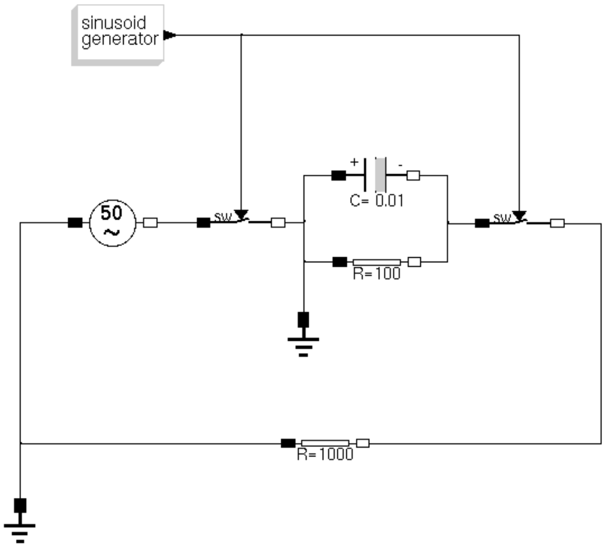


Figure 3: A modified version of the circuit with switches

The conclusion from experiments with ideal models could be that one should avoid using them because they lead to under- and over-constrained subsystems of equations. This is not the correct conclusion in our opinion: we think that ideal models are extremely useful in some situations (e.g., where one needs to save computation effort for instance) and that the modeling language should be able to properly handle them by yielding simulation code that conforms to our expectations.

2.3 Connection variables

Note: Without loss of generality, we deliberately consider here the simple pattern of a main model built out of submodels connected together. Submodels themselves are supposed to contain only simple equations (i.e., equalities). Indeed, the purpose of this paper is not to discuss multiple ways to build Modelica models but to illustrate problems with the current approach, based on use of first-class connection variables.

Information exchange between models in Modelica is usually performed by means of *connectors* and *connections*. Connectors are aggregations of *connection variables*, divided into *flow* and *potential* variables. The connection of several compatible connectors expresses constraints between connection variables that match:

- values of potential variables have to be equal
- values of flow variables have to sum to zero

A special case exists for unconnected connectors:

- values of potential variables are not constrained
- values of flow variables have to be zero

From the outside of submodels point of view, Modelica provides a rather high-level construct (connection equations) to express connections of models without having to manipulate connection variables directly. That approach offers several advantages:

- the code reflects the topology of the model, so the code is easy to understand
- it is possible to add a new branch to a model without having to worry about the consequences on the new connection constraints, so the code is easy to extend
- it is impossible to break fundamental invariants attached to both kinds of connection variables, so the code is robust

In contrast, from the inside of submodels point of view, things are not as simple and clear: users have to manipulate connection variables directly, and, as a consequence, nothing prevents them to violate fundamental invariants such as flow preservation. Worse: the compiler has no way to prove that models preserve those invariants since, inside submodels, everything is ultimately an ordinary equation.

That situation is a bit odd: after all, why not having the counterpart of connection equations inside submodels? Having a construct that would handle connection variables automatically the way **`connect`** does outside submodels not only would help users to write correct (and machine-checkable) models, but also would lead to more general models. Indeed, the fundamental problem with the connection variable approach is that it depends on the assumption that connection variables always exist in the sense that there is always enough constraints in the model to define all of them. That assumption is false, as demonstrated in the previous sections, and the goal of ground-like models (to be provided by users!) is precisely to compensate the lack of potential constraints and the excess of flow constraints that arise naturally in connection variable-based semantics.

At this point, it is legitimate to ask ourselves why not simply make connection variables second-class citizens of the language and let higher-level constructs directly deal with them. Indeed, this would be a solution: if for instance the compiler would introduce connection variables on demand (i.e., just as many connection variables as necessary to solve the problem for variables of interest) we would avoid the problems encountered so far. But let's think a bit more about those connection variables: what kind of information do they carry under the assumption that you can't directly access them? The answer is rather simple: nothing meaningful for the user. In consequence, the thesis we develop in this paper goes even further: we simply advocate a language with no connection variables at all in the physical domain.

3 Proposal for an enhanced Modelica approach

3.1 The example of VHDL-AMS

Several modeling languages already adopted a theoretical model with no connection variables[5]*. Among them, VHDL-AMS[2] is probably the most famous one in the Modelica community. In this section, we present an quick overview of the way VHDL-AMS handles connection of analog submodels.*

VHDL-AMS defines *terminals* to represent physical connection points (the equivalent of Modelica connectors, in the analog domain). A terminal is declared to be of a given *nature*, i.e., energy domain. Here is an example of declaration of an electrical nature:

```
subtype voltage is real;
subtype current is real;
nature electrical is
  voltage across
  current through
  ground reference;
```

The first two lines are just for convenience: they define two subtypes of `real` (the equivalent of Modelica's `Real` type) used to represent voltages and currents, respectively. More interesting is the declaration of `electrical`:

- `voltage` **`across`** declares the *across type* associated with `electrical`
- `current` **`through`** declares the *through type* associated with `electrical`
- `ground` **`reference`** names the *reference terminal*, a special terminal that holds the reference potential associated with `electrical`

We can observe that no name is introduced for neither the across type nor the through type. Only `ground`, the name of the reference terminal, is introduced by the definition of `electrical`: in VHDL-AMS there is no need to introduce any ground submodel since the ground is a connection point, not a submodel[6].

Natures can be used to define terminals using the following syntax:

```
terminal p, n: electrical;
```

The above line declares two terminals `p` and `n` having `electrical` nature. How do we access across and through values held by the terminals in VHDL-AMS? Here is the trick: we cannot. As written above, terminals only represent physical connection points and, as such, they do not stand for placeholders for any kind of connection variables. The only way offered by VHDL-AMS to access across and through quantities inside the equivalent of submodels is by means of *branch quantity* declarations:

```
quantity
  v across
  i through
  p to n;
```

The above declaration reads as follow:

- `v` **`across`** declares a variable having the across type of `p` and `n`, that holds the value of the voltage difference between `p` and `n`
- `i` **`through`** declares a variable having the through type of `p` and `n`, that holds the value of the current that flows from `p` to `n`

We can notice in the declaration above that nothing requires the existence of any kind of connection variable, as expected. Indeed, the declaration really just states that:

- the *across quantity between* `p` *and* `n` can be refered to as “`v`”
- the *through quantity flowing from* `p` *to* `n` can be refered to as “`i`”

Branch quantity declarations coupled with *port maps* (the equivalent of Modelica's connection equations in VHDL-AMS) define a connection graph that is used to elaborate the missing constraints in VHDL-AMS models. Which advantages do they provide over the Modelica approach? First, inside submodels,

[5] Simscape, from The MathWorks™, offers an intermediate level of abstraction (connection variables are visible as port attributes) but users are strongly encouraged not to use them directly.

[6] Remember footnote 2.

we get the conciseness and the clarity already offered outside submodels in Modelica by means of connection equations: the code is easy to read, easy to maintain and more robust (compare the branch quantity declaration given above in VHDL-AMS with the definition of partial "two-pin" submodels in Modelica libraries). Second (and the most important regarding the subject of this paper), thanks to those high-level constructs, **a VHDL-AMS compiler has a global view of the whole connection graph**:

- concerning potentials: where a Modelica compiler only knows locally that absolute potentials in a connection set should be equal, a VHDL-AMS compiler also knows which loop(s) of the connection graph a given potential contributes to (this allows circuits as in Figure 1 to be successfully compiled)
- concerning flows: where a Modelica compiler only knows what happens locally in a connection set, a VHDL-AMS compiler also knows how flows traverse submodels (since, inside submodels, flows are explicitly declared as such) and, consequently, how flows traverse the whole connection graph.

The strength of VHDL-AMS regarding analog modeling comes precisely from that global view of the connection graph: **a VHDL-AMS compiler can fully apply both *Kirchhoff laws*[7] to the whole system** whereas a Modelica compiler can only apply the first one, and moreover, only partially (actually, to each connection set). It follows that a VHDL-AMS compiler can provide the exact number of missing constraints to solve the whole system without introducing any connection variable and without resorting to the "manual adjustments" (e.g., positioning ground-like submodels) required by Modelica.

Undeniably, VHDL-AMS outperforms Modelica here. Alas, there are still some limitations: models such as depicted in Figure 3 still remain impossible to define... The reason is that VHDL-AMS requirements for solvability (see [2]) impose one equation to be present in ideal switches, so users have to state explicitly that the current flowing through the submodel is zero when in "open" mode[8]. Global flow analysis finds that our VHDL-AMS model with two switches is well-structured whereas two constraints may dynamically impose the value of the flow on the same branch of the circuit. We are in the same situation encountered in Modelica while solving the simple R circuit: under some switching conditions, the incidence matrix of the system is singular.

We conclude from those observations that having high-level constructs such as VHDL-AMS branch quantity declarations in an acausal modeling language greatly enhances expressiveness. However, that approach, even if better than what Modelica currently provides, does not suffice to resolve all the issues. In the next sections, starting from a variant of branch quantity declarations seen above, we propose an enhancement over VHDL-AMS, that solves the issues encountered in models involving ideal submodels.

3.2 High-level physical connectors

In order to benefit from the full power of global connection graph analysis in Modelica, we need to be able to define the equivalent of VHDL-AMS terminals and branch quantities. In the case of electrical modeling, it would give something like:

```
type Voltage = Real(unit="V");
type Current = Real(unit="A");
connector Pin
  across Voltage;
  through Current;
end Pin;
```

Several comments need to be made at this point. First, unlike in VHDL-AMS, we do not begin by defining a nature to be subsequently used in terminal or branch quantity declaration. Instead, we define directly a *high-level physical connector* for the physical domain of interest. This avoids introducing too many new language constructs (and keywords) while still remaining general enough, as we will see below. Second, we impose that any high-level physical connector contains exactly one across type definition and its associated through type definition. It is however possible to add traditional connection variables, parameters, etc., in the same connector definition, in the pure Modelica spirit. Also, high-level physical connectors can be aggregated into enclosing connec-

[7] The first law, also called *Kirchhoff's junction rule*, states that the directed sum of the flows at every node of a circuit is zero. The second law, also called *Kirchhoff's loop (or mesh) rule*, states that the directed sum of the efforts around any closed circuit is zero.

[8] VHDL-AMS features *generate statements* to generate constraints conditionally, so one might hope to use them here... Alas, the switching condition is dynamic in our case and generate statements are limited to static cases. Also, we don't know, at submodel creation time, under which condition constraints have to be present (or not).

tors, like any other kind of Modelica connector. Third, the careful reader may have noticed that we did not define any *reference connector* when defining `Pin` above. The main reason is that we don't want to pollute name spaces with new names, especially when context always permits to disambiguate the code. Indeed, references to the reference connector always occur in a connection statement. For obvious reasons, we impose that at least one of the connected entities has to be an instance of a compatible connector type[9] other than the reference connector. In consequence, we propose that the keyword **`reference`**, used in any connection statement in place of a connector reference, represents the *ad-hoc* type-compatible reference connector.

3.3 Connections and effort/flow variables

In order to conveniently express connections in a natural Modelica style, we propose the usage of the keyword **`connect`** to be extended in two ways:

- applied to high-level physical connectors[10], it is used to express that its arguments have to be considered as the same physical connection point, or *node*
- applied to more than two connector references, it enables users to enhance readability of connection statements by allowing local connection sets to be expressed directly[11]

We also propose to declare the equivalent of branch quantities by means of two separate constructs:

```
across(v, p, n);
through(i, p, n);
```

The first line reads "`v` denotes the potential difference between `p` and `n`": it is used to declare an *effort variable* (the first argument) holding the effort difference measured between the "plus" and "minus" connectors denoted respectively by its second and third arguments. Notice that we do not declare the type of `v`: instead we impose that at least one of the two connector arguments to **`across`** has to reference a user-defined instance of a high-level physical connector, so the type can be deduced from context (it is of course the across type of the corresponding high-level physical connector type). Similarly to **`across`**, **`through`** introduces a new variable, but in that case the introduced *flow variable* holds the flow quantity flowing from the "plus" connector to the "minus" connector[12].

3.4 Sources and sensors

Our theoretical model is not yet expressive enough: pure sources and pure sensors are missing. The *raison d'être* of those entities is to enforce conceptual decoupling of the effort/flow world from the signal world by enabling users to express input/output constraints between both worlds. This eases documentation, model analysis and debugging.

Sources and sensors are defined by means of *tees*. The proposed syntax for sources, or *input tees*, is:

```
across(input v_in, p, n);
through(input i_in, p, n);
```

The two lines above introduce respectively and effort source and a flow source. The corresponding effort/flow is constrained from the outside world by means of the type-compatible input connector signal whose name follows **`input`**. Similarly, the syntax for sensors, or *output tees*, is:

```
across(output v_out, p, n);
through(output i_out, p, n);
```

Unsurprisingly, the two lines above introduce respectively an effort sensor and a flow sensor. The type-compatible output connector signal is constrained by the corresponding effort/flow.

The theoretical model introduced here differs from VHDL-AMS's one: in particular, we allow the declaration of pure effort sources, whereas VHDL-AMS forbids them[13]. At this point, we have just defined a variant (with minor enhancements) of what VHDL-AMS already proposes, but we still have to solve the issues encountered with the use of ideal submodels.

3.5 Conditional connections

We have seen in previous sections that submodels such as ideal switches introduce dynamic *structural singularities (i.e., the incidence matrix of the system*

[9] Remember that Modelica features a *structural* type system.

[10] Of course having compatible types, which actually means "same types" here due to type invariance imposed by acausal semantics.

[11] This should not preclude connection sets to be split over several connection statements, however.

[12] Notice that the decoupling of across and through variable declarations enables decoupled sign conventions.

[13] That limitation of VHDL-AMS probably has its roots in the way *structural sets of equations* (i.e., VHDL-AMS's equivalent of connection equations) are required to be elaborated: by means of an application of the *Modified Nodal Analysis* (see [3]).

becomes singular under some circumstances that happen at unpredictable times during simulation). This suggests that a possible way to circumvent those problems could be to dynamically adjust the constraints to be solved instead of trying to hide numerical problems into equations that, as a result, introduce stiffnesses and numerical singularities into systems of equations. We then propose here a generalization of connection equations called conditional connections *and explain why this kind of construct can be used to solve issues encountered so far.*

Conditional connections are just *guarded* connect statements in Modelica programs: syntactically, we simply allow connect statements to appear in a branch of a conditional statement[14]. The semantics are naturally generalized to allow dynamic changes in the connection graph of a model: whenever a conditional construct activates a connect statement during simulation, the connection constraints get updated in accordance. Here is the code of an ideal switch using conditional connections:

```
model Switch
  Pin p, n;
  input Boolean on;
equation
  if on then
    connect(p, n);
  end if;
end Switch;
```

It is interesting to notice that the "if" clause above is not "balanced". Indeed, its only purpose is to specify conditions under which the topology of the physical connection graph of models containing instances of `Switch` changes[15].

It is straightforward to see why conditional connections solve issues encountered with the use of ideal switches. Indeed, models such as `Switch` above do introduce equations only on demand[16], to satisfy both Kirchhoff's laws: we have just defined a dynamic version of an elaboration method *à la* VHDL-AMS. It follows that, for instance, having several closed switches in parallel or several open switches on the same branch of a circuit is no longer a problem, provided we know how to cope with more general connection constraints. An elaboration algorithm that satisfies those requirements is presented in the next section.

4 Generation of connection equations

4.1 Physical connection graphs, across graphs and flow graphs

Across declarations and through declarations not only introduce a new identifiers, they also introduce new edges in a structure called a *physical connection graph* of the model. That directed graph is built as follow:

- vertices represent *connection sets* of high-level physical connectors[17] of the original model
- edges represent either across declarations or through declarations of the original model. Direction information is preserved: edges are directed from the connection set which the positive high-level physical connector belongs to to the connection set which the negative high-level physical belongs to. Also, causality is preserved, i.e., input/output information associated with tees is represented in the graph

It is important to notice that physical connection graphs are not explicit in the original model structure. However, they can be seen as a kind of dual of the original model structure considered as a graph, for a given configuration of conditional connections: vertices in the original model (i.e., submodels containing only simple equations) are used to build edges in the physical connection graph, and, conversely, edges in the original model (introduced by means of **`connect`***) are used to build vertices in the physical connection graph. Notice also that we associate exactly one physical connection graph with a model for each configuration of the conditional connections. This implies that physical connection graphs are generally disconnected: indeed, the associated model may eventually contain transformers and gyrators, and make use of several physical domains, for example.*

A physical connection graph can be seen as the superimposition of two simpler graphs having the same

[14] *When* and *if* clauses should be equally considered here. But due to the lack of a rigorous hybrid theoretical model in Modelica currently, we will focus on continuous-time equations and if clauses only in the focus of this paper.

[15] After all, since our aim is to equip Modelica with composable submodels, it is not so surprising that composition statements themselves do not require balancing constraints!

[16] In particular, a model composed of ideal switches only does not contains any variable nor equation.

[17] The reference connector is view as an ordinary connector here.

vertices: the *effort graph* and the *flow graph*. Edges of the effort (resp., flow) graph represent across (resp., through) declarations of the original model. Figure 4 below shows a representation of the connection graph corresponding to the model in Figure 1.

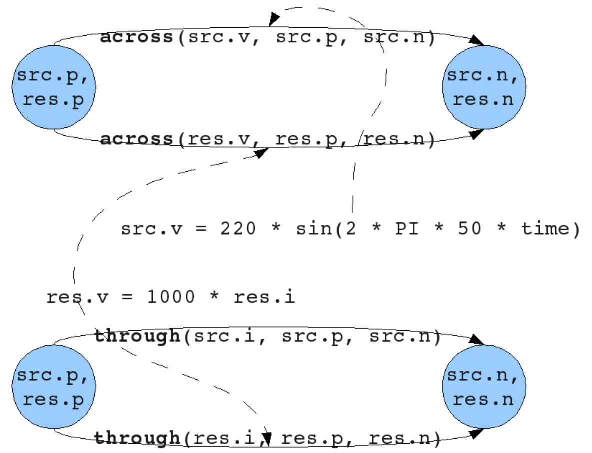

across(src.v, src.p, src.n)
src.p,
res.p
src.n,
res.n
across(res.v, res.p, res.n)
src.v = 220 * sin(2 * PI * 50 * time)
res.v = 1000 * res.i
through(src.i, src.p, src.n)
src.p,
res.p
src.n,
res.n
through(res.i, res.p, res.n)

Figure 4: a simple connection graph

The effort graph is represented at the top of the figure and the flow graph at the bottom. Blue vertices represent connection sets (shared between both the effort graph and the flow graph). Directed edges between two blue vertices represent across or flow definitions, depending on the nature of the subgraph they belong to. For information, relations between coupled effort and flow variables are represented by dashed arrow-ended curves labelled with the corresponding explicit equations.

4.2 Elaboration algorithm

The elaboration algorithm we propose in this paper operates by determining effort constraints and flow constraints independently. Notice that since we want to enable dynamic changes in the topology of the connection graph associated with a model, the algorithm should be applied to each configuration required by the simulation[18].

Effort constraints are determined this way:

- for each connected component of the effort graph, perform a depth-first traversal with marking
    - for each detected loop, generate the sum-to-zero of effort variables along the loop by following this sign convention: effort variables associated with edges oriented in the direction of the loop traversal are counted as positive and the other ones as negative.

Flow constraints are determined this way:

- for each connected component of the flow graph, perform a depth-first traversal with marking
    - for each detected loop with at least two vertices[19], generate, if not already done[20], the sum-to-zero of flow variables at each vertex along the loop until all the variables of the loop have been used at least once. The following sign convention is used for summation: flow variables associated with incoming edges are counted as positive and the other ones as negative
- generate equations constraining flow variables associated with edges that do not belong to any loop to have a null value.

The proof of the algorithm is omited but we give here the general idea behind it:

- a loop of length *n* where each pair of high-level physical connectors is connected exactly by one across definition and one through definition in the physical connection graph yields *1* effort constraint (the directed sum of effort variables equals zero) and *n - 1* flow constraints (the directed sum of flow variables is generated at each vertex but one[21])
- it follows that the final system of equations has *2n* unknowns (*n* effort variables and *n* flow variables) and *2n* equations (*n* equations explicitly given by the user, as required by balancing constraints, and *n* equations automatically introduced by the elaboration algorithm).

5 Example

In order to illustrate the power of high-level physical connectors, let's try to define the model depicted in Figure 2. We will reuse the definitions of `Pin` and

[18] Typically, each time the branch of a conditional equation becomes active. But a simulation environment may optimize simulation time by anticipating configurations, by caching old configurations, etc. Those optimization strategies are beyond the scope of this paper.

[19] Loops with only one node should not yield flow constraints.

[20] Since the same vertex may belong to several loops.

[21] Flow variables associated with the ignored vertex have been already used in equations associated with its neighbors, so no equation is generated for that vertex.

Switch previously introduced in 3.2 and 3.5, respectively. The model also requires the definition of a voltage source, a resistor and a capacitor. They are given here:

```
model VoltageSource
  constant Real PI = acos(-1);
  parameter Real V0;
  Pin p, n;
  across(v, p, n);
  through(i, p, n);
equation
  v = V0 * sin(2 * PI * 50 * time);
end VoltageSource;

model Resistor
  parameter Real R;
  Pin p, n;
  across(v, p, n);
  through(i, p, n);
equation
  v = R * i;
end Resistor;

model Capacitor
  parameter Real C;
  Pin p, n;
  across(v, p, n);
  through(i, p, n);
equation
  C * der(v) = i;
end Capacitor;
```

Notice the conciseness of those definitions, thanks to the use of high-level physical connectors: no inheritance from an abstract TwoPin class is needed. Also, the respective roles of v and i are explicit in the code: this greatly helps understanding the physics behind submodels. The VoltageSource submodel deserves a special comment: despite the absence of i in the equation of the submodel, we have to define it because otherwise it would make the submodel a pure voltage source and, as a consequence, the current would not traverse it. Since we want a two-pin-like submodel, we have to define the effort/flow pair of variables.

Just for fun, we can also define a (useless) ground submodel:

```
model Ground
  Pin p;
equation
  connect(p, reference);
end Ground;
```

Notice that, contrary to Modelica's traditional ground submodel, our ground submodel does not introduce any explicit equation (nor any variable at all) in models into which it is used.

The circuit (with logic of switches omited) can then be defined as:

```
model Circuit
  Ground gnd;
  VoltageSource src(V0=50);
  Resistor res1(R=1000);
  Resistor res2(R=100);
  Capacitor cap(C=0.01);
  Switch sw1(on=...), sw2(on=...);
  ...
equation
  connect(gnd.p, src.p, res1.p);
  connect(src.n, sw1.p);
  connect(sw1.n, cap.p, res2.p);
  connect(cap.n, res2.n, sw2.p);
  connect(sw2.n, res1.n);
  ...
end Circuit;
```

We will study the two possible configurations where, respectively, both switches are open and both switches are closed.

The first configuration yields the physical connection graph depicted in Figure 5 below. Both effort and flow graph have the same topology because we only used two-pin-like submodels to hold user-defined equations.

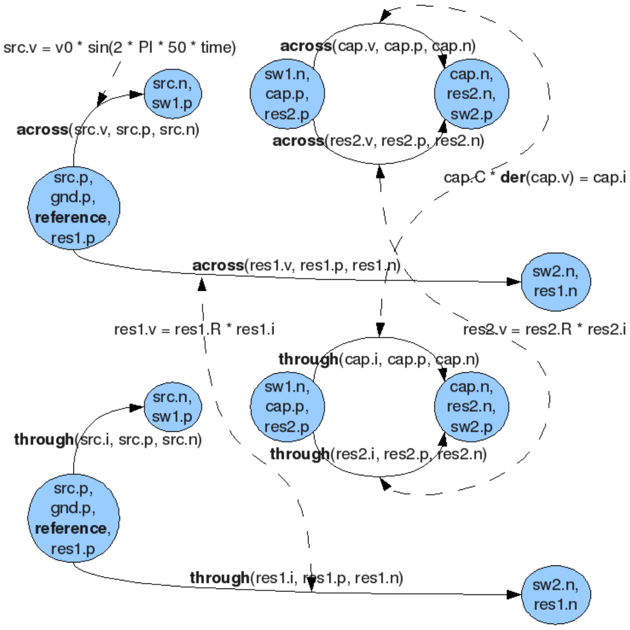

src.v = v0 * sin(2 * PI * 50 * time)
across(cap.v, cap.p, cap.n)
sw1.n, cap.p, res2.p
cap.n, res2.n, sw2.p
src.n, sw1.p
across(src.v, src.p, src.n)
across(res2.v, res2.p, res2.n)
src.p, gnd.p, reference, res1.p
cap.C * der(cap.v) = cap.i
across(res1.v, res1.p, res1.n)
sw2.n, res1.n
res1.v = res1.R * res1.i
res2.v = res2.R * res2.i
through(cap.i, cap.p, cap.n)
sw1.n, cap.p, res2.p
cap.n, res2.n, sw2.p
src.n, sw1.p
through(src.i, src.p, src.n)
through(res2.i, res2.p, res2.n)
src.p, gnd.p, reference, res1.p
through(res1.i, res1.p, res1.n)
sw2.n, res1.n

Figure 5: physical connection graph corresponding to model in Figure 2 in "open" mode

Applying the elaboration algorithm to the physical connection graph gives for instance (depending on the order into which vertices are examined):

```
cap.v - res2.v = 0;
src.i = 0;
res1.i = 0;
cap.i + res2.i = 0;
```

Only one effort constraint has been created, since the effort graph only has one loop. Three flow constraints have been created: two for the the leftmost subgraph (which contains no loop) and one for the rightmost subgraph (which is a loop with two vertices). We finally get a system with eight unknowns and eight equations once the four explicit equations are merged with the automatically generated ones.

The second configuration yields the physical connection graph depicted in Figure 6 below.

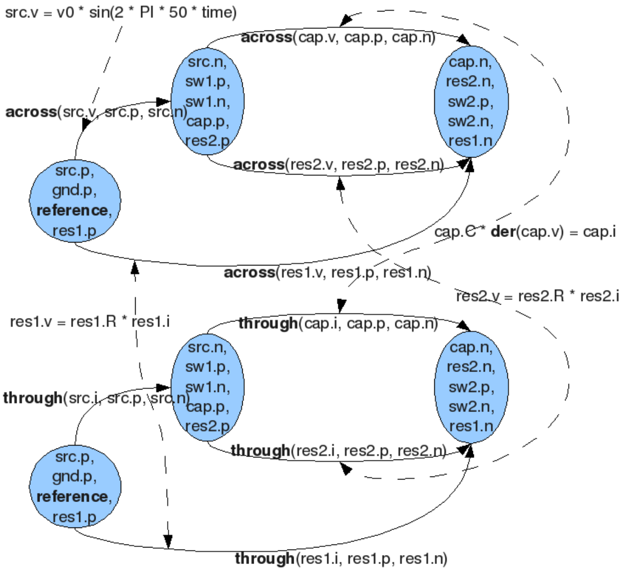

src.v = v0 * sin(2 * PI * 50 * time)
across(cap.v, cap.p, cap.n)
src.n,
sw1.p,
sw1.n,
cap.p,
res2.p
cap.n,
res2.n,
sw2.p,
sw2.n,
res1.n
across(src.v, src.p, src.n)
across(res2.v, res2.p, res2.n)
src.p,
gnd.p,
reference,
res1.p
cap.C * der(cap.v) = cap.i
across(res1.v, res1.p, res1.n)
res2.v = res2.R * res2.i
res1.v = res1.R * res1.i
through(cap.i, cap.p, cap.n)
src.n,
sw1.p,
sw1.n,
cap.p,
res2.p
cap.n,
res2.n,
sw2.p,
sw2.n,
res1.n
through(src.i, src.p, src.n)
through(res2.i, res2.p, res2.n)
src.p,
gnd.p,
reference,
res1.p
through(res1.i, res1.p, res1.n)

Figure 6: physical connection graph corresponding to model in Figure 2 in "closed" mode

Applied to that graph, our elaboration algorithm gives for instance:

```
res1.v - res2.v - src.v = 0;
res2.v - cap.v = 0;
-src.i - res1.i = 0;
cap.i + res2.i + res1.i = 0;
```

Again, a system with eight unknowns and eight equations is finally produced. Two effort constraints and two flow constraints have been generated, one for each loop in both the effort graph and the flow graph.

It is interesting to notice that in both cases only eight equations are produced where the equivalent traditional Modelica program would have produced 38 equations! (each two-pin-like submodel introduces six variables and the ground submodel, two). Also, equations generated by our algorithm always correspond to physical properties of the model: instead of obfuscating the final system of equations like traditional connection equations do, equations generated by our algorithm may be helpful to users to analyse their models, and, eventually, to debug them.

6 Conclusion

In this paper, we have proposed a solution to enforce composability of Modelica submodels: any composition of (possibly ideal) submodels that is physically sound now yields a non-singular system of equations. Now it would be interesting to exploit the additional syntactic information offered by our proposal to better statically typecheck models: it should help detecting errors more accurately than a solution based on traditional Modelica connectors, especially if coupled with a method that would take incidence information into account[22]. Also, and more importantly, we hope that, coupled with a rigorous hybrid theoretical model, our proposal would serve as a basis to get rid of several special-purpose or composition-unfriendly features of Modelica designed to cope with limitations in expressiveness, especially the recent Stream proposal but also the Overconstrained Connection-Based Equation Systems proposal, among others[23]. Another interesting application would be a type system that would be powerful enough to ensure true "plug compatibility": separate compilation of submodels would become possible and intellectual property protection would directly benefit from that.

[22] Notice that explicit incidence information does not violate intellectual property!

[23] As the first sentence of the Revised^5 Report on the Algorithmic Language Scheme nicely says: *"Programming languages should be designed not by piling feature on top of feature, but by removing the weaknesses and restrictions that make additional features appear necessary."*

7 Acknowledgments

This work is part of the contribution of LMS to the ITEA2 European project OpenProd[24]. Many thanks to my colleagues from LMS Imagine for their help and suggestions, especially El Djillali Talbi, Antoine Viel, Loig Allain, Yohan Broussaud, Olivier Broca and Denis Fargeton. Also, many thanks to Michael Tiller from Emmeskay, Wilfrid Marquis-Favre from INSA-Lyon and David Broman from Linköping University: discussions with them are always extremely valuable.

References

[1] The Modelica Association, ***Modelica Standard Library 3.0***, PDF Documentation, available at http://www.modelica.org/libraries/Modelica/releases/3.0.1/ModelicaStandardLibrary_3_0_Documentation.zip

[2] P. J. Ashenden, G. D. Peterson, D. A. Teegarden, ***The System Designer's Guide to VHDL-AMS — Analog, Mixed-Signal, and Mixed-Technology Modeling***, Systems On Silicon, Morgan Kaufmann, 2003, ISBN 1-55860-749-8

[3] C. Tischendorf, ***Topological Index Calculation of DAEs in Circuit Simulation***, 1997

[4] G. Dauphin-Tanguy, ***Les bond graphs***, HERMES Science Europe Ltd, 2000, ISBN 2-7462-0158-5

[5] P.J.L. Cuijpers, J.F. Broenink, P.J. Mosterman, ***Constitutive Hybrid Processes: a Process-Algebraic Semantics for Hybrid Bond Graphs***, in SIMULATION, Vol. 84, Issue 7, pp 339-358, 2008

[24] See http://www.ida.liu.se/~pelab/OpenProd/

Module-Preserving Compilation of Modelica Models

Dirk Zimmer
Department of Computer Science, ETH Zurich
CH-8092 Zurich, Switzerland
dzimmer@inf.ethz.ch

Abstract

Large Modelica models pose serious problems for compilation and simulation. The standard process for the compilation of Modelica models is insufficient since it requires the flattening of the system and generates thereby overly large executables. In this paper we elaborate the concept of module-preserving compilation. This technique aims to generate more compact executables and thereby shall enable the simulation of very large systems in the future. To this end, we introduce an appropriate terminology and design a set of data structures and algorithms that enable the embedment of module preservation into the translation of Modelica models. This paper represents theoretical work only and aims to open up a fruitful discussion on this topic. *Keywords: Flattening; Translation; Causalization.*

1 Motivation

The object-oriented modeling paradigm of Modelica promotes a modular design of systems. Simple Modelica models are thereby composed in order to form a complex, hierarchically structured top-model. The individual submodels are mostly stated in declarative non-causal form. This is a prerequisite for their general applicability. Whereas the declarative form benefits the usability, it prevents the models from being directly "executed." Hence, the models must be translated into a computationally feasible form (e.g. an executable program), mostly for the purpose of time integration.

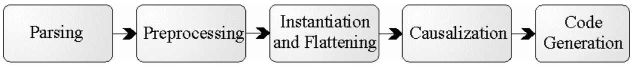


Figure 1: Compilation stages of Modelica code

Figure 1 represents a common compilation scheme that is shared by typical Modelica translators like Dymola [3] or OpenModelica [4]. We see that Modelica models are getting instantiated in the middle stage of the compilation process. The instantiation is carried out in a flattened form. This means that the hierarchic structure is destroyed and that the resulting system represents one large system of equations.

The process of flattening benefits further tasks of the compilation process. First of all, it enables the removal of alias variables (that mostly result from the objects' interfaces) and thereby reduces the system size. The process of causalization is able to handle algebraic loops that extend over many different submodels. State selection and index reduction reduce the dimension for the numerical ODE solver. By these and other means the overall system can be significantly simplified and the resulting code is competitive to the best manually coded simulations.

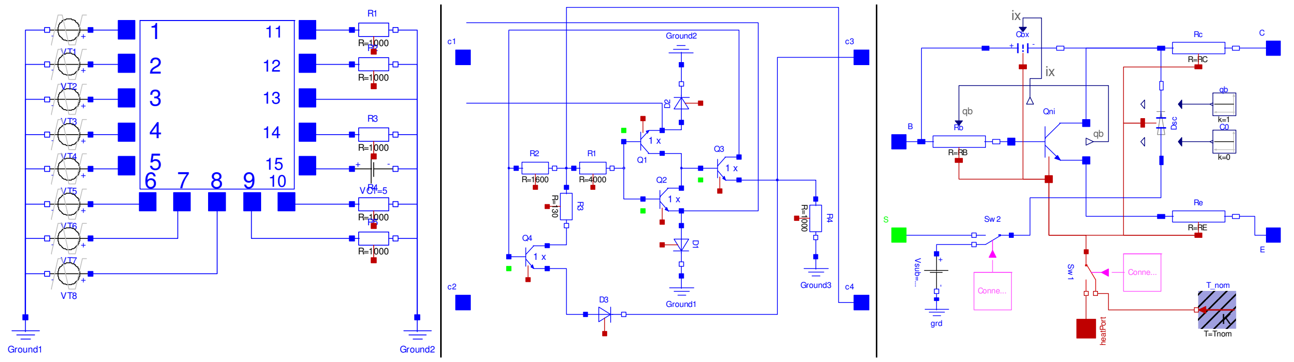
Unfortunately, the process of flattening also has its deficiencies. It gets problematic for very large systems. Since the model is always processed as a whole, it does hardly scale and gets increasingly inefficient for large models. Also the generated code starts to lack in quality. It gets overly large and contains many redundant parts.

To get a better understanding of the problem, let us look at an example. The *Verification Package for Modelica Spice 2.1* [2] includes the model of a four bit adder (c.f. figure 2). Because it is modeled down to the layer of single bipolar junction transistors, the model is very detailed and indeed very large: it contains in total 481'915 scalar equations. Dymola (v7.2) fails in the attempt to simulate this model. The translation needed almost 1 GB of RAM and finally generated an 88MB executable. Its simulation in Dymola failed in the simulation-environment.

Whereas the final reason of breakdown is most probably a minor bug that could be corrected, we shall not overlook that there is a serious issue with very large models. This is by no means specific to Dymola, it is a principal problem concerning the general processing of languages like Modelica.

Large models contain often a high degree of redundancy, and the larger they get, the more redundant they typically are. This rule of thumb also applies to our example model. The four-bit adder contains 2 two-bit adders. The two-bit adder contains 2 one-bit adders and each one-bit adder circuit contains 9

DOI: 10.3384/ecp09430028



(a) four-bit adder (b) NAND (c) BJT

Figure 2: Model diagram of the four bit adder and two of its components

NANDs. The NAND gate itself consists in 4 bipolar junction transistors (BJTs) plus 3 diodes. Consequently, the four-bit adder contains 36 NANDs, i.e., 144 BJTs plus 108 diodes.

On the modeling layer, this redundancy is not a problem, since the number of individual modules is far lower than the number of their corresponding instances. On the computational level, all the hundreds of instances get flattened and corresponding code is generated for each of them. In direct consequence, the code gets large, bulky, and redundant.

It is an evident question to ask: If hundreds of sub-model instances share the same equations, can they not share (at least partly) the same code? And can this code be modularized in form of a function? It is the aim of this paper to examine how and when a given modularization on the modeling layer can be preserved and mapped to code modules in the final executable. This shall provide future benefits for both the speed of the translation and the size of the executable.

2 Modules and their Representation

Most readers will be very familiar with the typical process of module creation. It is mostly applied to a structure that occurs several times in a system. It can be described by 5 steps:

1. Extract all the elements you want to put into your module from one occurrence of your structure.
2. Determine all the variables that are part of your module, and separate this set of variables into two distinct sets: The set of local variables that occur in your module only and the set of interface variables that are also being used elsewhere.
3. Form an interface for your model given the corresponding set of interface variables.
4. Replace all occurrences of your structure by instances of your module (e.g. sub-model declarations or function calls, respectively).
5. Connect the interface of your module with the corresponding variables.

This way of modularization can be applied to transform code segments into functions but also to group clusters of equations into a Modelica model. Hence modules are a common concept for both the source and target of a Modelica compiler.

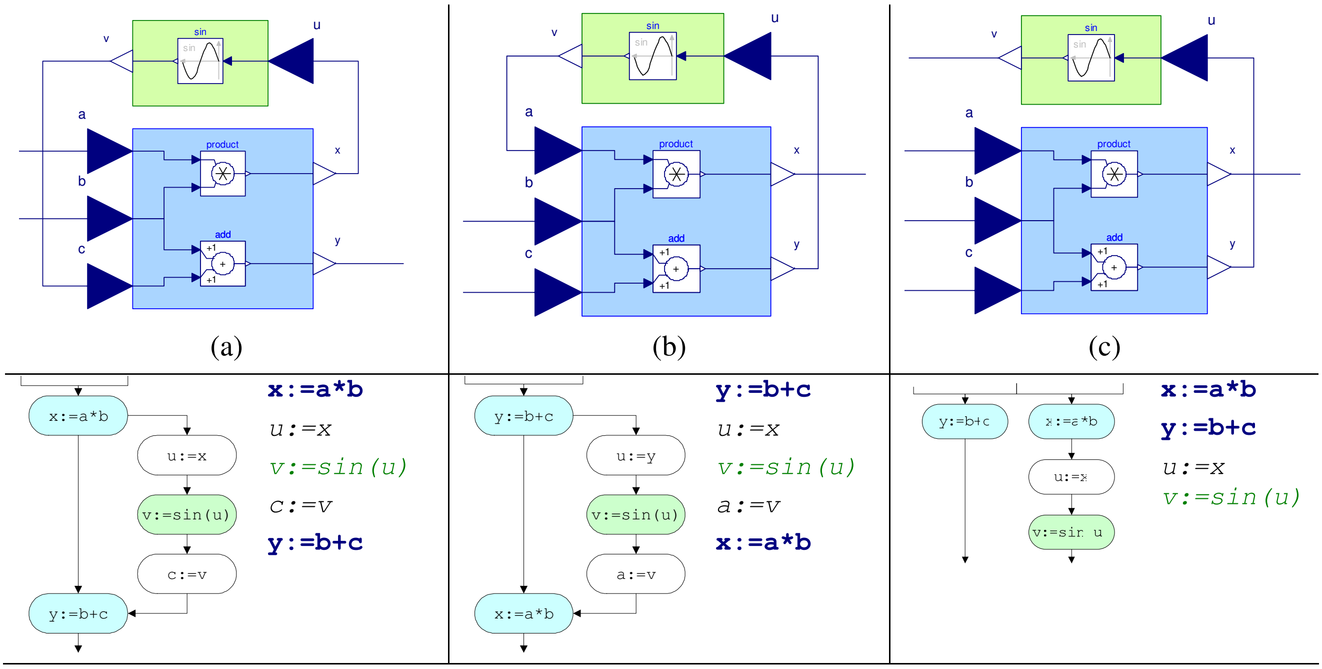
On the modeling layer, the modular design is given by the modeler. A module is represented by a Modelica model and it consists essentially in an unordered set of equations. In order to form meaningful modules, the modeler aspires to create sub-models that form a semantic entity and offer a preferably small interface that wraps a more complex inner part.

The target of the compilation is program code. That is an (ordered) list of statements. These statements are mostly computations of operators and value assignments. If several pieces of code share equivalent sub-lists of statements (again, disregarding the naming of variables), these statements can be modularized. Such a code module is typically represented as a function.

Module-preserving compilation aspires a mapping between the modules on the model level and the potential modules on the code level. In concrete terms: how and when can a Modelica model or a part of it be translated into a function of an imperative programming language?

3 Entities of Modularization

In principle, any arbitrary code segment can be modularized, but to gain any advantage, the subpart needs to occur frequently in the main code and it needs a feasible interface. One might attempt to find such suitable subparts in the flattened code by pattern-finding algorithms, but this approach is hardly



```
x:=a*b
u:=x
v:=sin(u)
c:=v
y:=b+c
```

```
y:=b+c
u:=x
v:=sin(u)
a:=v
x:=a*b
```

```
x:=a*b
y:=b+c
u:=x
v:=sin(u)
```

Figure 3: A causal block in different contexts with the corresponding code.

promising since this is a computationally demanding task. It can be expected to fail for very large systems when modularization is needed the most.

The hierarchic structure of the equation-based model gives us a priori information about those patterns that may occur frequently in the resulting code. In order to use this information, we have to know about the requirements for the translation of a sub-model into a function.

A typical model in the Modelica library has a non-causal interface with non-causal equations. Hence, many models (like the model of a mechanical rod) can be causalized in many different forms that all require a different code for the computation. Thus, code can only be shared for model instances of the same causality.

However, thinking that causal models can be directly transformed into code, is misleading. Figure 3 (a) and (b) present a simple counterexample. The corresponding code for the presented models is placed underneath the modeling diagrams. We see that the same causal block (blue) not only yields different code, also its code separates into two parts. Hence it cannot be expressed by a single function.

The problem is that causality only gives rise to a partial order, but the transformation into code requires an absolute order. The fact that the variable x is determined by a and b and that y is determined by b and c does not say anything about the order between x and y. This might be stipulated by the remaining system as in (a) and (b), but it might be left for choosing as well as in (c).

The causal relations between assignments are best expressed by a causality graph $G(E,V)$. This is a directed acyclic graph where the vertices V_G correspond to the assignments. The edges E_G are formed by those pairs of assignments (s_1, s_2) where v is a variable of s_2 and determined by s_1. Examples of such causality graphs are placed beside the code segment in figure 3.

Those assignments belonging to a certain model M induce a sub-graph G_M of G. We are interested in a very specific form of sub-graphs:

Definition 1:

- A vertex-induced sub-graph G' of G is called *path-complete* iff all paths in G between any vertex pair (s_1', s_2') in G' are also included in G'.

Path-complete sub-graphs are of high interest to us because each one of them can be translated into a separate cohesive program segment and thereby can be modularized into a function.

Any vertex-induced sub-graph G_M can be decomposed into a set of path-complete sub-graphs $\{G_{E1}, G_{E2}, ...\}$, but since there are many such decompositions, we need to specify further restrictions to derive a unique decomposition. First of all, we demand the decomposition to be minimal in the sense that the decomposition contains no pair (G_{Ea}, G_{Eb}) that can be merged to another path-complete sub-graph.

Since G is a directed acyclic graph, any *minimal* decomposition into path-complete sub-graphs is given an *absolute* order by G. There may now be vertices of G_M that cannot be uniquely assigned to one of the decomposition's sub-graphs. If we define these vertices to be assigned to the sub-graph of the lowest order, we get a unique decomposition. We denote this as the busy, minimal decomposition into path-complete sub-graphs. Fortunately, this decomposition can be derived incrementally, as will be described in section 5.2.

We recognize that the code for any model M may be split into several entities E_1, E_2, ... that represent cohesive code segments and that such a decomposition into program segments can be uniquely determined. Thus, we define:

Definitions 2 and 3:

- A *causal entity* E of a model M represents a list of vertices of a sub-graph G_E that results out of the busy, minimal decomposition of G_M into pat-complete sub-graphs. The order of the list E is partially determined by the underlying directed graph G_E.
- A *causal interface* I_E of a causal entity E represents a pair of variable sets. The first set contains the input variables that are formed by the ingoing edges from G to the sub-graph G_E. Correspondingly, the second set is formed by the outgoing edges and represents the output variables.

The causalization of a model can now be precisely defined by the causal signature:

Definition 4:

- The *causal signature* S_M is a complete list of causal interfaces I_E for all causal entities E belonging to a given model M. The list determines the order of the corresponding causal entities.

For illustration, let us look at the causal signatures from Figure 3. Each model has a different one:

(a) [({a, b},{x}) ,({c},{y})]

(b) [({b, c},{y}) ,({a},{x})]

(c) [({a, b, c}, {x, y})]

We have seen that (a) and (b) require different code. They also have different causal signatures with different causal entities. We further recognize that each pair corresponds to a block of code, hence to a potential function. Code that is generated for (c) shall not be used for (a) and (b), but not necessarily vice versa. Code for (a) and code for (b) would be usable also for (c). Thus, we define the terms sub- and super-signature:

Defintion 5:

- A causal signature S_M is *sub-signature* of another causal signature S_M' over the same model if S_M can be transformed into S_M' by merging[1] subsequent pairs of the list. S_M' is then defined as a *super-signature* of S_M.

Example: The signature [({a, b, c}, {x, y})] from example (c) is a super-signature for both (a) and (b).

Sub-models that share the same causal entities can share the same code. The number of causal entities corresponds thereby to the number of separate code blocks that could be turned into functions. In order to reuse code efficiently, one may decide to replace the code of one causal entity by the code of one or several entities that originate from a model instance that has a causal sub-signature.

Example: If the causal signature of 2(a) occurs frequently, the compiler may decide to create two code modules in form of the functions f_1 and f_2:

```
function f1(a,b)
       x := a*b;
       return (x);
end
```

```
function f2(b,c)
       y := b+c;
       return (y);
end
```

If the causal signature of 2(c) occurs only once, the corresponding code may now be formulated using these modules by the segment:

```
x := f1(a,b)
y := f2(b,c)
```

4 Which entities shall be preserved?

Let us look at the academic example of figure 4. It contains a lot of addition blocks and hence a compiler might be tempted to create a code module (function) for the block. However, this is obviously not a good idea. Modularization of the correspondent causal entity will decrease the performance and quality of the code in this case.

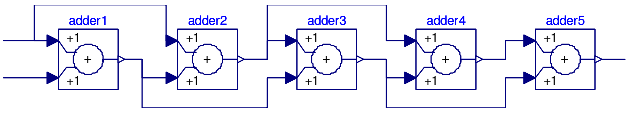


Figure 4: A bad example for remodularization

The reason for this is that the modularization of a causal entity needs to provide its interface (in concrete terms: function parameters and return value). Once implemented, it is hardly possible to optimize across the interface, and hence the systems cannot be simplified. Auxiliary or alias variables cannot be

[1] Please regard: The merging of two causal entities corresponds to the merging of its causal entities, and hence variables can get removed from the interfaces.

removed out of the system. Furthermore, the simple additions are replaced by more costly function calls. We see that preserving modules per se does not improve the code. It is a tool that demands proper application.

Modularization is not for free, it incurs additional cost. Memory is needed to define the interface of the function. Computational time is needed for the assignment of the interface values and the corresponding function call.

Thus, the modularization of causal entities is only meaningful, if the additional computational cost is marginal to the cost of the function and if the memory cost of the interface is compensated for by the memory savings that are attained by replacement of multiple instances through function calls. Let us therefore make a distinction between the inner and outer complexity of a model.

4.1 Inner complexity of a module

Definitions 6 and 7:

- The *inner computational complexity* $C_{i,E}$ of a causal entity E is the total amount of all memory assignments and basic computations from code that corresponds to *E*.
- The *inner data complexity* $D_{i,E}$ is the total amount of local data that is required for those computation.

Since both definitions for the inner complexity refer to the actual code, their estimates are dependent on the simplification mechanisms of the preceding compilation stages.

Attaining a fair estimate for $D_{i,E}$ is actually unproblematic. However, its complexity may depend on the modularization of potential sub-entities. Estimates for $C_{i,E}$, can be difficult to obtain when the number of computations is unsure, for instance, an iterative solver has to be applied in order to solve a non-linear equation system. Fortunately, it turns out that a determination of $C_{i,E}$ is not necessary.

4.2 Outer complexity of a module

Definitions 8 and 9:

- The *outer computational complexity* $C_{o,E}$ of a module is the amount of all memory assignments and basic computations that refer to data of its interface and to data outside the module.
- The *outer data complexity* $D_{o,E}$ of a module is the total amount of data in its interface, defined by I_E.

Knowing the interface of a potential code module means knowing about its outer complexity. However, the interface may contain more than intuition suggests. The interface variables of the corresponding equation-based model M are not the only members of the interface. If the causal entity E represents only a part of the model M, auxiliary variables will be added to the interface. Furthermore, if the causal entity defines integrators and hence possesses state variables, these state variables have to be part of the interface as well, since they are determined by the global algorithm for synchronous time integration. The same is true for variables that trigger events. They are also part of the interface, since their values must be accessible to the event finding algorithms.

Variables that form the simulation output are not necessarily part of the interface. The tracking of the correspondent data can be done within a code module.

The distinction between inner and outer complexity is however dependent on the computational framework that will embed the resulting code. Here, we assumed a typical environment for synchronous time integration, but in a different computational framework like QSS [5], the integrators and event triggers are local and the corresponding variables do not belong to the outer complexity.

4.3 Frequency of entities

There is no incentive to turn any causal entity E into a code module, if there is only one instance of it. The number of occurrences N_E is therefore a crucial criterion. It influences cost and benefit of the modularization.

The **cost** of modularization is:

- Additional computational cost: $N_E\ C_{o,\ E}$
- Additional data complexity required: $D_{o,\ E}$

The **benefit** of modularization is:

- Saved computational cost: 0
- Saved data complexity: $(N_E\ \text{-}1)\ D_{i,E}$

Let μ be a coefficient that translates the computational complexity into data complexity. It needs to be determined by experience, but mostly it will be chosen in such a way that $\mu{\cdot}C_{o,E}$ is close to $D_{o,E}$. Now we can compare the overall cost with the total benefit:

$$N_E\ D_{i,E} > N_E\ \mu\ C_{o,E} + D_{o,E} + D_{i,E}$$

Consequently, let R be the fraction:

$$R = (\mu{\cdot}C_{o,M} + D_E / N_E)\ /\ D_{i,E}$$

A modularization becomes profitable if $R < 1$. This implies that $\mu \cdot C_{o,E}$ must be smaller than $D_{i,E}$. Furthermore we see that the inner computational complexity is irrelevant. If we presume that $\mu \cdot C_{o,E}$ equals $D_{o,E}$, the computational complexity can be neglected entirely.

One of the difficulties of modularization is that a causal entity E for a model M may contain a sub-entity E' from a sub-model M' of M. The inner data complexity $D_{i,E}$ is then dependent on the potential modularization of E'. If E' forms a module of its own, $D_{i,E}$ obtains a lower value, and E itself is less likely to be modularized. Hence the module preservation of sub-models influences the modularization of its super-models. Fortunately, Modelica enforces a strict model hierarchy, implying that models cannot be sub-models of themselves. This will simplify the further analysis in section 5, but first let us look at an example.

4.4 Example

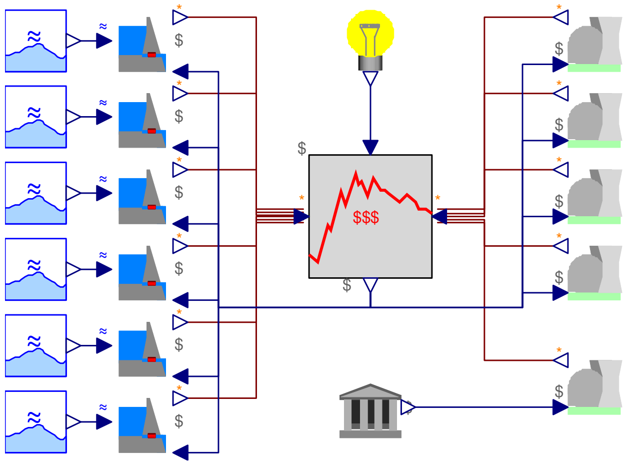


Figure 5: Simple model of an electric energy market with producer models.

Figure 5 presents a very simple model of an electric energy market. The electric power originates from 5 nuclear power plants and 6 hydropower plants. Each of the power plants has its own parameters, and the hydropower plants are dependent on the waterflow from its rivers. Furthermore, one of the nuclear power plants is state owned and works for a fixed-price scenario whereas all other power plants compete on the free market. The actual model of the market is placed in the center and determines the price from the current balance of supply and demand.

The diagram represents the top model. Its implementation is based on the system dynamics library [1]. The overall model contains about 1100 variables, whereby 800 of them represent alias variables. The multiple power plants are an obvious target for modularization. Let us therefore examine their causal signatures.

A hydropower plant produces a certain amount of power given its current state. The price is determined by the market. The current price influences the monetary profit or loss of the plant and hence drives a controller that aims at maximizing the profit. Each hydropower plant possesses two state variables: the desired outflow f and the current water level w. Its behavior is controlled by the inputs the inflow i and the current price $\$$. The power p forms the output. Without knowing much of the interior we can determine the causal signature that is shared by all six hydropower plants:

$$[(\{f, w, i\},\{p, \mathrm{d}w/\mathrm{d}t\}),(\{\$, w\},\{\mathrm{d}f/\mathrm{d}t\})]$$

We see that state and derivatives have become part of the causal interfaces. Furthermore we need two code modules to compute the model of the power plant. Also the nuclear power plants demand two code modules although their causal signature is simpler since it has only one state: the current production level l:

$$[(\{l\},\{p\}),(\{\$, l\},\{\mathrm{d}l/\mathrm{d}t\})]$$

The state owned nuclear power plant is an exception though. Its signature is a super-signature of the other plants:

$$[(\{l, \$\},\{p, \mathrm{d}l/\mathrm{d}t\})]$$

Table 1 presents the ratio R for all four causal entities. 3 of the 5 entities are suited for modularization and so the overall data complexity can be reduced to roughly 50%.

Table 1: Analysis of causal entities

Entity	$D_{o,E}$	$D_{i,E}$	N_E	R
$(\{f, w, i\},\{p, \mathrm{d}w/\mathrm{d}t\})$	5	10	6	0.75
$(\{\$, w\},\{\mathrm{d}f/\mathrm{d}t\})$	3	12	6	0.46
$(\{l\},\{p\})$	2	1	4	2.75
$(\{\$, l\},\{\mathrm{d}l/\mathrm{d}t\})$	3	14	4	0.51
$(\{l, \$\},\{p, \mathrm{d}l/\mathrm{d}t\})$	4	15	1	1.53

5 Revised compilation process

The proposed methods so far are feasible to apply an analysis to an already flattened model and to optimize the resulting code by modularization. But the flattening alone can represent an unaffordable task, and hence the modularization shall be integrated in all of the important stages of the compilation process.

5.1 Preparation

In a first preparatory stage, we attempt to estimate R for any causal entity E of a model M from the non-causalized Modelica model itself. The idea is to get rid of all the small models that contain just a few equations. Therefore, this analysis does not need to be pursued for large models. The inner and outer data complexity of the corresponding Modelica model enables an estimation value $\check{R} = \check{D}_{o,M} / \check{D}_{i,M}$ that mostly is a lower bound for the effective R of its causal entities. This is because causalization reduces the inner complexity mostly more than the outer complexity, and the split into causal entities mostly increases the overall interface. Furthermore N_E is assumed to be infinite.

Sub-models with $\check{R} < 1$ are not expected to contain modules that are valuable to preserve. The same is true for sub-models that occur only once. All other models are put into the set Ω, and their causal entities may form modules of the program code. Please regard that equations of models that are not in Ω can still become part of a code module if any of their super-models is in Ω. Hence the selection criterion for Ω can be chosen even stricter than suggested.

5.2 Instantiation and Causalization

In the classic scheme, all models get instantiated first and then causalized. For very large systems this procedure is not feasible anymore. Ideally, the process of module preservation shall be implemented in such a way that the full flattening of the model can be avoided. Thus we propose to instantiate and causalize in several alternating iterations.

To this end, the models are being instantiated into a buffer of fixed size. When the capacity limit is reached, the equations in the buffer are causalized as much as possible. Those equations that could be causalized are transformed into assignments and added to the causality graph G. Last, the buffer is cleared and the non-causalized equations are put aside for a latter iteration.

In order to causalize the whole system, many sweeps over the buffer may be required. During the whole process, the causality graph G is constantly growing. When an assignment s of a model M in Ω is added, G and the induced sub-graph G_M grow by one vertex. A decomposition $\{G_{E1}, G_{E2}, ..., G_{En}\}$ into path-complete sub-graphs will be affected in two possible ways:

- a causal entity is enlarged $G'_{Ek} = G_{Ek} + s$ (the entity that is of lowest order in G, in case there are several options).
- else, the new vertex forms a new causal entity $G_{E(n+1)} = s$.

This procedure will lead to a busy, minimal decomposition into path-complete sub-graphs. We further recognize that existing causal entities can just grow, but they will not be cut or merged. This is very important because this means that we can track all entities: When a common causal entity in the graph G_E exceeds a certain threshold size, we can decide to modularize G_E in the graph by storing it separately. In this way, we can avoid to store the complete causality graph in plain form. This does not mean that the corresponding causal entity will form a code module. This modularization within the causality graph is a separate mechanism that is suggested in order to save a potentially substantial amount of memory.

5.3 Model hierarchy

At the end of the causalization, we have a complete causality graph where larger common parts share the memory. The graph contains a potentially large number of causal entities that all have to be analyzed for a potential modularization. This analysis has to be executed in a certain order: A model may have instances with different causal signatures. Some of these signatures may be super-signatures of others. This will influence the modularization of the sub-signatures, and thus all causal entities belonging to a model have to be analyzed at once.

Furthermore, the modularization of an entity of a model M may influence the modularization of an entity in any of M's super-models (remember section 4.3). Therefore we have to execute the analysis according to the model hierarchy starting with the lowest models first.

The term model hierarchy might be misleading since it suggests a tree-like structure. In fact, a Modelica model hierarchy can also be represented by a directed acyclic graph that gives rise to a partial order on its models representing the vertices of the graph. The bottom-up procedure can therefore be implemented as a (breadth first) graph traversal.

5.4 Modularization

For each model M in Ω, we built up a prefix tree of its causal signatures, where each node owns a counter, and the branches denote the corresponding causal entity E. A path from the root to a leaf then represents a causal signature S_M of a model instance. Figure 6 presents an exemplary prefix tree for the block model of figure 3.

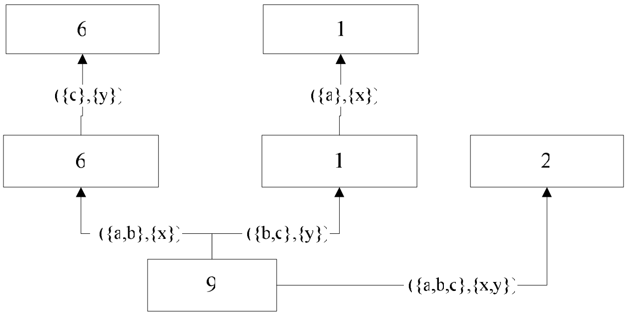


Figure 6: Prefix tree of causal entities

We want to find out, which causal entities are profitable to modularize. To this end, we have to consider the super-signatures first (right branch in figure 6). We can either create an extra causal entity for a super-signature or re-use existing ones. In general, this is a hard optimization problem. For our purposes a simple heuristic procedure shall be sufficient:

- Let E_0 be the first causal entity of the super signature S_{M0} for the signatures $S_1 \ldots S_n$ and let $E_1 \ldots E_n$ be their corresponding first causal entity.
- We find the best R: $R_{min} = \min \{ R(E_1) \ldots R(E_n) \}$
- If $R(E_0) > R_{min}$ we decide to split the super-signature and integrate it into the path that belongs to R_{min}.
- At last, we mask out the root and repeat this process recursively for all sub-trees.

If we assume the values out of column 1 from table 2, the prefix tree of figure 6 will transform into the one depicted in figure 7. Assuming the values of column 2 will cause no changes in the original tree.

	Assumption 1	Assumption 2
$R(\{a, b, c\}, \{x, y\})$	1.2	0.8
$R(\{a, b\},\{x\})$	0.6	0.85
$R(\{\mathrm{b, c}\},\{x\})$	1.8	2.0

Table 2: Scenarios for super-signatures

Finally, we can compute R for all causal entities of the prefix tree and either chose to modularize the code or not, given the criteria from section 4.3. Please remember that in order to compute R, symbolic simplifications should take place beforehand.

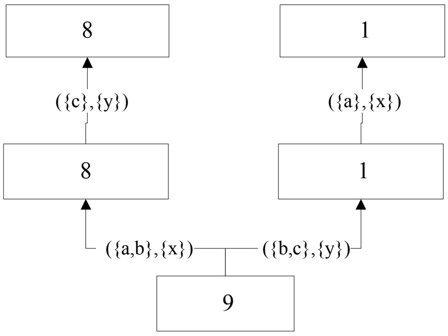


Figure 7: Prefix tree of causal entities

5.5 Summary and run-time efficiency

```
Step 1: Preparation
Estimate $\check{R}$ for all models and enter selected models into the set $\Omega$.

Step 2: Instantiation and causalization
Install a buffer of limited capacity.
While there are non-causalized equations do
    Fill buffer with non-causalized equations.
    Attempt to causalize them.
    Reject non-causalized equations for future iterations.
    Track causal entities for the models in $\Omega$.
    Store larger entities separately.
end

Step 3: Modularization
For all models $M$ in $\Omega$.
in order of the model hierarchy do
    Built up the prefix tree of causal signatures for $M$.
    Simplify the code of the corresponding causal entities.
    Manage occurring super-signatures.
    Compute R for all remaining entities and decide to modularize the entity if R < 1.
end
```

It is important to note that none of these processes has to solve an NP-hard optimization task. The precise algorithmic efficiency depends on the concrete implementation. However, let us look at the causality graph. If we (realistically) suppose a maximum number of variables in an equation, the memory demand is linear to the size of the system and even less than linear if modularization can be applied. The fact that the code modules can be created ad-hoc helps to keep the memory demand small. The most expensive algorithm that works on this graph is the stepwise causalization. In worst case, it will lead to a quadratic run-time.

6 Further issues

The implementation of a mechanism for remodularization has implications for other processes in the compilation. In the following, we investigate the most important points that need to be concerned:

6.1 Algebraic loops

A proper implementation of module-preserving compilation requires that the process of model instantiation and causalization is conducted in several iterations. As long as the model contains no algebraic loop and/or requires index reduction, this is a non-issue. For instance, the complete domain of system dynamics is mostly non-critical.

However, many systems cannot be represented in lower triangular form and thus a block lower triangular (BLT) form is typically aspired. A standard algorithms for this purpose, the Dulmage-Mendelsohn permutation, [7,8] cannot always be applied since it assumes that the whole system is readily available. This is not naturally the case for very large systems.

Other algorithms for a BLT transformation are therefore required that are able to cope with local information only. It is possible in doing so by applying a tearing method directly on the whole system that identifies the corresponding blocks of equations (denoted as algebraic loops) later on. Such mechanism have already been developed (although for another purpose) in the SOL framework [9,10].

In general, the tearing will be needed for the efficient solution of the algebraic loops. A tearing method selects (using certain heuristics) a sufficient number of tearing variables and assumes them to be known. Now the algebraic loop can be causalized and an equal number of residual equations results. In order to solve the system, an iterative numerical solver is typically applied. We need to investigate how modularization can be applied for the torn system of equations.

For causal entities there are two cases that need to be considered with respect to algebraic loops:

- *Causal entities that contain a complete algebraic loop.* This is in principal unproblematic. The code can be wrapped like any other code. However, depending on the heuristics, it is not guaranteed that equivalent models will be torn in an equivalent way. This is still a serious issue.
- *Causal entities that are only part of a loop.* The modularization of such entities is in general not very meaningful. They may contain residuals or tearing variables that would enlarge the interface of these entities. Furthermore the entities may contain additional computations that are not necessary to compute the residuals. These increase the computational effort and (what is worse) may not be fail-safe with respect to the numerical solver.

6.2 Symbolic Differentiation

The mechanisms for index reduction (see [7]), but also the application of iterative solvers may require the differentiation of subparts of the equation system. In the case of index-reduction the differentiation often generates algebraic loops.

The differentiation adds new equations to the system. Whereas there are given models for the original set of equations there are no models for the differentiated equations and hence no modularization can take place on differentiated subparts of the system.

We therefore propose that differentiated equations become part of their original model can therefore also be part of causal entities. However, this topic also needs further investigation.

6.3 Pre-compilation and re-modularization

Causal entities map to an enclosed code segment (i.e. a function) that of course can be provided also in pre-compiled form. Hence an M&S-environment may decide to maintain a library of precompiled code from the most frequently used causal entities. The underlying motivation is to decrease the compile time.

There remains doubt that pre-compilation will represent an effective means. It could as well be that the reading from the disc is slower than the actual compilation process. Only for very large code segments pre-compilation will be profitable for sure. Such code segments would correspond to sub-models that are not only large but also hardly decomposable into further sub-models. Otherwise the smaller submodels will be modularized and the large model shrinks in its inner data complexity. Ideal vehicle models in a traffic simulation could be one such example.

6.4 Modeling requirements

Module-preserving compilation requires that the provided model owns a suitable hierarchic structure and this needs to be provided by the modeler. Since the compilation by itself is not able to detect any patterns or to form feasible substructures the most principle rule of information processing applies: garbage in – garbage out. Badly structured or flat models cannot be handled efficiently. An implementation of a finite-element mesh within Modelica would represent one such example.

7 Conclusions

The concept of module-preserving compilation bases on the observation that a causalized model can be decomposed into a list of causal entities whose interfaces form the causal signature. Each causal entity corresponds thereby to a potential code module. Regarding the outer and inner complexity of a potential code module we could derive a criterion for the selection of appropriate code modules.

The integration of module preservation in the translation process is clearly a non-trivial task that involves a whole bunch of issues. The model hierarchy needs to be taken into account. Furthermore we suggest managing the different causal entities in form of prefix trees. Methods for tearing and state selection need to be provided that do not require the complete system to be readily available.

Module-preserving compilation represents not more than one tool to optimize code and hence it cannot solve all problems. It will fail for flat or unstructured models and it may be difficult to apply if there are huge algebraic loops.

Nevertheless, there are many suitable examples like the large electric circuits of figure 2. We also presented a model based on system dynamics (figure 5). Although this example is still quite small, module preservation is expected to decrease the code complexity already substantially. Evidently, much larger models of an energy market can be envisioned with much more elaborated models. A model of the complete European grid could be one example. Such a model would contain several thousands of power plants and many different market places. For such large models, module preservation becomes a vital tool in order to enable a simulation of the system at all.

Our proposal for module-preserving compilation aims to be a general approach for a wide range of models. It is however possible to simplify the outlined procedure significantly if we can make certain assumptions about the model structure or require additional hints or help from the modeler himself. Such an approach will lose generality but might be better achievable in a practical implementation.

References

[1] Cellier, F. E.: World3 in Modelica: Creating System Dynamics Models in the Modelica Framework. In: *Proc. 6th Interna-tional Modelica Conference*, Bielefeld, Germany (2008) Vol.2 393-400.

[2] Cellier, F.E., C. Clauß, A. Urquía: Electronic Circuit Modeling and Simulation in Modelica. In: *Proc. 6th Eurosim Congress on Modelling and Simulation*, Ljubljana, Slovenia (2007) Vol.2, 1-10.

[3] Dynasim AB, *Dymola Users' Manual*, Version 6.0, Lund, Sweden, 2006.

[4] Fritzson, P., P. Aronsson, H. Lundvall, K. Nyström, A. Pop, L. Saldamli D. Broman: The OpenModelica Modeling, Simulation, and Software Development Environment. In: *Simulation News Europe* (2005) 44/45.

[5] Kofman, E., S. Junco: Quantised State Systems: A DEVS Approach for Continuous Systems Simulation. In: *Transactions of SCS*, (2001) 18(3), pp.123-132.

[6] Pantelides, C.: The Consistent Initialization of Differential-Algebraic Systems. In: *SIAM J. Sci. and Stat. Comput.* (1988) Vol 9, No. 2, 213-231.

[7] Pothen, A., Chin-Ju Fan: Computing the Block Triangular Form of a Sparse Matrix. In: *ACM Transactions on Mathematical Software* (1990) Vol 16, No. 4 303-324.

[8] Tarjan, R.: Depth-first search and linear graph algorithms. In: *SIAM Journal on Computing*. (1972) Bd. 1, No. 2, 146-160.

[9] Zimmer, D.: Introducing Sol: A General Methodology for Equation-Based Modeling of Variable-Structure Systems In: *Proc. 6th International Modelica Conference*, Bielefeld, Germany, (2008) Vol.1, 47-56

[10] Zimmer, D.: Enhancing Modelica towards variable structure systems. In: *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)*, Berlin, Germany (2007) 61-70

Biography

Dirk Zimmer received his MS degree in computer science from the Swiss Federal Institute of Technology (ETH) Zurich in 2006. He gained additional experience in Modelica and in the field of modeling mechanical systems during an internship at the German Aerospace Center DLR 2005. Dirk Zimmer is currently pursuing a PhD degree with a dissertation related to computer simulation and modeling under the guidance of Profs. François E. Cellier and Walter Gander. His current research interests focus on the simulation and modeling of physical systems with a dynamically changing structure. To this end, the Sol-project was founded in 2007 together with F.E. Cellier and the support of the Swiss National Science Foundation.

Extendable Physical Unit Checking with Understandable Error Reporting

Peter Aronsson[1] David Broman[2]

[1]MathCore Engineering AB, Teknikringen 1F
SE-58330 Linköping, Sweden, peter.aronsson@mathcore.com
[2]Dept. of Computer & Information Science, Linköping University, davbr@ida.liu.se

Abstract

Dimensional analysis and physical unit checking are important tools for helping users to detect and correct mistakes in dynamic mathematical models. To make tools useful in a broad range of domains, it is important to also support other units than the SI standard. For instance, such units are common in biochemical or financial modeling. Furthermore, if two or more units turn out be in conflict after checking, it is vital that the reported unit information is given in an understandable format for the user, e.g., "N.m" should preferably be shown instead of "m2.kg.s-2", even if they represent the same unit. Presently, there is no standardized solution to handle these problems for Modelica models. The contribution presented in this paper is twofold. Firstly, we propose an extension to the Modelica language that makes it possible for a library designer to define both new base units and derived units within Modelica models and packets. Today this information is implicitly defined in the specification. Secondly, we describe and analyze a solution to the problem of presenting units to users in a more convenient way, based on an algorithm using Mixed Integer Programming (MIP). Both solutions are implemented, tested, and illustrated with several examples.

Keywords: dimensional analysis, unit checking, dimensions, error reporting, language design

1 Introduction

Modelica is a full fledged object-oriented equation-based modeling language. However, its expressiveness can sometimes lead to models containing errors that are hard to detect and isolate[3].

One important area where modeling errors can give devastating consequences is inconsistency of physical units and dimensions within equations and component connections. We have earlier proposed a design and made a prototype implementation for dimensional inference and unit consistency checking[1] in the MathModelica[7] and OpenModelica[9] tools. Such checking will help the users by reporting at compile time if they have made a unit inconsistency error in their model. However, this becomes less useful when modeling something that cannot be expressed in the dimensions as defined by the SI standard. This is a common scenario for biochemical modeling based on the SBML standard[12]. Such models frequently use the non-standard dimension "Item" for counting, e.g., molecules. MathModelica has a translator tool[2] for translating SBML models into Modelica (and vice versa). To make the translation tool more robust, user defined units should be considered in the dimensional analysis too. Another example is in financial applications, where it is required to use the dimension "money".

It is also a problem that the system of units (and potential extensions) is not described in the Modelica language standard, i.e., the language specification only specifies how to parse unit expressions, not what the units mean, or how the checking should be performed. This may result in that tools from different vendors are not compatible, where some tools accept certain Modelica libraries, while others reject them due to unit inconsistency.

Another problem is how to present the resulting units (e.g. from unit inference) to the user, when one or more units are inconsistent. For instance, presenting the unit "m2.kg.s-2A-1" to the user is not very understandable. Instead, the tool should translate this into a more appropriate derived unit (or combination of derived units and base units), like "Wb" or perhaps "V.s". The preferred choice of these two might be different depending on domain and context. For instance, if this unit is reported in a domain of Magnetic models, "Wb" might be preferred, but if it is reported in a context where only units "V", "A" and

DOI: 10.3384/ecp09430027

"Ohm" are used it is probably more appropriate to use "V.s". The presentation should not only contain standard SI units, but also extended units defined by the user and these could also be selected by regarding domain and context information.

In this paper we propose a solution to these problems by specifying both base units and derived units in a generic way, so that new dimensions easily can be added. We propose this as an extension to the Modelica language so that different Modelica tools can behave alike. At the same time, the library developer is also given a more powerful mechanism for specifying nonstandard units in a uniform way. Section 2 presents the proposed Modelica language extension that enables the model user to describe both base units and derived units. In Section 3 we show a new method of how a tool can interpret the dimensional units inferred by the type checker and presents unit errors to the user in a more readable form. This is done by formulating a mixed integer programming (MIP) problem that will select more appropriate units depending on both context and potentially also user preferences. We have made an implementation and an evaluation in the MathModelica and OpenModelica tools, described in Section 4. Finally, Section 5 contains related work and Section 6 concludes the paper.

2 Extendable Unit Definitions

The Modelica specification [10] includes a section describing the syntax of unit expressions, i.e., how for example an expression such as `"kg.m/s2"` should be parsed. However, besides a reference to ISO standard 31/0-1992, no information is given regarding the *semantics* of how to perform the actual unit checking. This general openness of the specification makes it possible for different tool vendors to implement their own way of handling unit checking, giving implementation freedom, but also limits the possibility for models to be exchanged and treated in the same way by different tools.

Instead of letting a reference to an ISO standard define the meaning of base units (e.g. "V" and "s") and derived units (e.g. "N.m"), we propose in this section that the definition should be stated directly in the source code of Modelica classes. Possible benefits with this approach are:

- Tools from different tool vendors use and interpret exactly the same set of unit definitions.
- Besides the standard SI units, it is easy for users and library developers to add both new base and derived units for a particular library.

Our goal is that both this work with extendable unit definition and our previous work on general unit checking should form a foundation for a new semantic description of units in the Modelica specification. Even though we today have a running test implementation, the work is still at an early stage, and more work on formalizing the semantics is required for inclusion in the specification. Moreover, our intension is not that unit checking should be a core part of the specification. Instead, we propose that such a language feature should be defined as an *optional module* in the specification, enabling tool vendors to explicitly choose and officially state if the functionality of such a module is supported.

2.1 Requirements

We have during the design work of extendable unit definitions for Modelica considered the following requirements:

- *Backwards compatibility.* Models designed with the earlier definitions where the meaning of units was implicit, should also work in a new environment where the units are defined by the library developer.
- *Only library definitions.* Both base units and derived units should be able to be added by library developers, i.e., the tools should not have any prior knowledge about defined units.
- *Declarative and easy to use.* The new extension for defining new units must be declarative in the sense that the order of definitions should not matter. It must also be easy to use, e.g., defined units should be stated in a user friendly format such as "N.m"; not using its unit vector format.
- *Weights for different domains.* It should be possible to prioritize certain units for a specific domain, to enable better error reporting.
- *Prefixes are pre-defined.* Prefixes, such as "m" for milli and "k" for kilo are pre-defined in the specification, i.e., these are not extendable.

Following these requirements, an overview of our design proposal is outlined in the following three subsections.

2.2 Informal Syntax

Adding new syntax to a language is the least interesting and challenging issue from a language design point of view, but results nevertheless often in large debates at design meetings. Hence, the following proposed syntax is only for presentation purpose and can most likely be changed in a version that is accepted for inclusion in the Modelica specification.

We introduce a new keyword **`defineunit`**, which is used both for defining new base units and derived units. For example, to define the three first base units of the SI-system, the following lines can be added to an arbitrary Modelica class.

```
defineunit m;
defineunit kg;
defineunit s;
```

Derived units are defined by combining base units or other derived units. For example, to define the derived unit Newton, the following line is added.

```
defineunit N(exp="m.kg.s-2");
```

The expression consists here only of base units. The syntax of the unit expression is the same as the syntax specified in the current Modelica standard. However, it would be very inconvenient if the derived units always must be defined using base units. Hence, we allow expressions also to include other derived units. For example, this line would define the derived unit Pascal:

```
defineunit Pa(exp="N/m2");
```

Note that both a derived unit (N) and a base unit (m) are used in the unit expression.

There is also an optional parameter weight that can be used for specifying how important an unit is in the domain. This is used by the algorithm presented in Section 3 for better error reporting. If no weight argument is specified, a default value of 1 is used. The weight can also be specified explicitly by using a named parameter. For example

```
defineunit Pa(exp="N/m2", weight=2);
```

states that Pascal is a unit that is more important in this library and will therefore have higher priority when used in error reporting.

2.3 Formal Syntax

The **`defineunit`** extension can be defined in the EBNF grammar of the Modelica specification, by adding the following production:

```
unit_clause :
  defineunit IDENT
      [ "(" named_arguments ")" ]
```

The `unit_clause` is then used inside the element production as follows:

```
element :
  unit_clause |
  ...
```

Where **`...`** mean the rest of the right side of the original `element` production.

2.4 Informal Semantics Overview

The semantics of the extendable unit definition is not intended to be described in detail here. Instead, the intent is to give a brief overview of how a compiler can treat the unit definitions. A more complete and formalized description is postponed as future work in conjunction with a language extension proposal for the Modelica language design group.

From the syntax description, it is clear that unit definitions can be placed anywhere in the element section of a class. Hence, units can be defined within any restricted class, e.g., packages, models, and functions. When checking equations and/or statements within a model, two passes are performed. In the first pass, all components and sub-components of the model are traversed and unit definitions collected. This includes searching both the components' scope and their parents' scope. In the second pass, the ordinary instantiation/elaboration takes place. During this elaboration, equations and statements are checked for unit consistency using the unit definitions collected in the first pass.

The order of how the unit definitions are collected in the first pass is not important. If the set of unit definitions contains several elements with the same unit name, it is an error if their respective unit expressions are different. For example, if Newton (N) is defined more than once, each definition must have the same expression, i.e., `"m.kg.s-2"`. After elimination of identical unit definitions, the resulting set of unit definitions is used to generate an internal normalized representation of units. Following the approach outlined in our previous work [1], each unit is then represented in a vector format. To be able to generate this normalized form, it is required that all definitions and dependences between derived units and base units form a directed acyclic graph (DAG). Hence, derived units are not allowed to be defined so that they form cyclic structures. If such a cyclic structure is detected, an error should be reported. For example, the following definitions should be rejected:

```
defineunit U1(exp="m.U2");
defineunit U2(exp="U3/s");
defineunit U3(exp="U1.kg");
```

If several unit definitions exist with the same name and expression, but with different weights, these weights are used later in pass two for better error reporting. The weights for unit definitions with the same unit name are multiplied together, forming the new weight. For example, if the following definitions of Pascal exist:

```
defineunit Pa(exp="N/m2", weight=1.5);
defineunit Pa(exp="N/m2", weight=2);
```

The resulting unit definition is:

```
defineunit Pa(exp="N/m2", weight=3);
```

In the current implementation a library must redefine all types that should be treated with a different weight factor. For example, if a library would like to have higher weights on Pascal, types that are using `Pa`, such as `Pressure`, must be redefined in the library. The main rationale for this design choice is better performance of the instantiation/elaboration process of the compiler.

3 Reporting Units

The unit checker described in previous work[1] uses a vector of seven rational numbers; one for each dimension. The reason for using rational numbers is to be able to handle a `sqrt` function or exponents of arbitrary rational numbers, e.g., x^(2/3), which is very commonly used in engineering equations. In this work, the length of this vector is determined by the number of dimensions added to the system. The library developer adds all definitions of base units and derived units to the standard library, including the standard SI units (see Section 2). Every unit is thus described by a vector of at least 7 elements. For instance, the unit Watt (“W”) corresponding to the base units “m2.kg.s-3” is described by the vector (2,1,-3,0,0,0,0). The problem is, given a sought unit with dimension vector $\dim_t$ (the target unit), to find a linear combination of units (both derived and base) that matches the dimension vector $\dim_t$. But, in order to select more appropriate units we should prefer units that are close to the target unit. Also, we should prefer to use derived units instead of base units, as this will probably be closer to what an engineer expects.

As a first attempt, we can formulate the problem as:

For a target unit, t, that has dimensional vector $\dim_t$

minimize

$$\sum_{j=1}^{NU} p_j x_j \quad \text{where } p_j = \frac{1}{w_j}(1+\left|\dim(j)-\dim_t\right|)$$

$$\text{subject to } \sum_{j=1}^{NU} \dim(j) x_j = \dim_t$$

Where

- NU is the number of units (base and derived)
- w_j is a real number > 0
- dim(j) is the dimensional vector for the j:th unit
- x_j is the sought exponent for each unit
- |v| is the L2 norm of vector v

This formulation works fine as long as x_j is a positive integer value. If negative values were allowed those would contribute negatively to the objective, and thus favor negative exponents over positive ones. So, to allow negative exponents in units we must handle them separately. This can be done by instead setting up the problem as:

$$\text{minimize} \sum_{j=1}^{2NU} p_j x_j$$

where

$$p_j = \frac{1}{w_j}(1+\left|\dim(j)-\dim_t\right|)$$

$$w_{j+NU} = w_j \qquad ,1 < j < NU$$

$$\dim(j+NU) =- \dim(j) \qquad ,1 < j < NU$$

$$\text{subject to } \sum_{j=1}^{2NU} \dim(j) x_j = \dim_t$$

With the formulation above we double the problem size and represent negative exponents with a set of separate variables. The weights for the newly introduced variables are identical to its positive correspondent exponent, and the dimensional vector is negated.

If the dimensional units only were described with Integers (e.g. as done in Dymola v.7 [5]), this formulation would be sufficient. However, because we allow Rational numbers as exponents and because it is most likely that derived units should be expressed only by integers, we need to reformulate the problem. We let the variables of base units be of type Real (or preferably rational) and the derived units be of type integer, thus leading to a mixed integer programming problem.

3.1 Example

Let us consider an example. For simplicity we limit the example to use three base units (m,s,kg) and define four derived units according to Table 1 below.

Unit	*Vector representation*
m	(1,0,0)
kg	(0,1,0)
s	(0,0,1)
N	(1,1,-2)
Pa	(-1,1,-2)
J	(2,1,-2)
W	(2,1,-3)

Table 1. A subset of the SI units.

Suppose that a unit of a certain term is inferred to "m.kg2.s-3", corresponding to the vector representation (1,2,-3).. If we use (1,1,1,1,1,1,1) as weight vector the problem becomes:

$$\text{minimize } \mathrm{p\,x}$$

$$\text{subject to } m\,\mathrm{x} = \dim_t$$

$$m =$$

$$\begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 2 & 2 & -1 & 0 & 0 & -1 & 1 & -2 & -2 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & -1 & 0 & -1 & -1 & -1 & -1 \\ 0 & 0 & 1 & -2 & -2 & -2 & -3 & 0 & 0 & -1 & 2 & 2 & 2 & 3 \end{pmatrix}$$

$$p = 1 +$$

$$\begin{pmatrix} \sqrt{13} & \sqrt{11} & \sqrt{21} & \sqrt{1} & \sqrt{6} & \sqrt{3} & \sqrt{2} & \sqrt{17} & \sqrt{19} & 3 & \sqrt{38} & \sqrt{34} & \sqrt{43} & 3\sqrt{6} \end{pmatrix}$$

$$\dim_t = (1,2,-3)$$

The m matrix sets up the constraints for the dimensions, the first seven columns corresponds to the values in Table 1 above, and the seven last columns are their negated values. The criteria vector *p* gives the weight for each variable as the distance of the dimension vector of that dimension to the target dimension plus one. The reason for adding one to the distance is to be able to control that even selecting a perfect match can be avoided by using weights. For instance, the first element has a value of $1+\sqrt{13}$ since the distance (norm) from (1,0,0) to (1,2,-3) plus one is

$$1+\sqrt{13} \ \ (1+\sqrt{(1-1)^2+(0-2)^2+(0--3)^2}\ ).$$

When solving this problem it will give the values: (0,1,0,1,0,0,0,0,0,1,0,0,0,0)

which correspond to the unit "kg.s-1.N".

By adjusting the weight vector different results are obtained. For instance, if we increase the weight only for "Pa" the results instead become:

(0,0,1,0,1,1,0,0,0,0,0,0,0,0)

which correspond to unit "s.Pa.J", i.e. it prefers to use unit "Pa" in the result.

3.2 Use of Rational Numbers

So far we have not used any rational numbers in our examples. So how does rational numbers affect the proposed solution?

Since we formulated the problem as a MIP (Mixed Integer Programming), it can allow both integer variables and real variables. The idea is to limit the derived units to integer values, so that units like "W-(1/3)" are not produced. Otherwise it will be hard for the user to find out what is missing to correct the error, since the user himself has to translate the derived units into base units and then apply the exponent.

As an example we will take the unit "W(1/2)", which corresponds to the unit vector (1,1/2,-3/2,0,0,0,0) The solution when derived units are integers and base units are reals becomes: "kg-(1/2).s(1/2).N". If the problem is solved with all variables as real values[1] the solution is instead: "N(1/2).Gy(1/4)" which is much harder for a user to interpret.

An alternative formulation could be to instead formulate the linear programming problem using only integers, by multiplying the base unit vector by the greatest common divisor among the rational numbers, and then solve the corresponding integer linear programming (ILP) problem. The solution must then be divided by the greatest common divisor. The problem with this formulation is that it can not guarantee that derived units are only expressed in integer exponents. For example, given the unit vector (5/2,3/2,-9/2,0,0,0,0), the corresponding MIP solution becomes "m-1/2.kg-1/2.s-1/2.N.J". However, transforming to ILP gives "Pa.J.W.Gy", which results in "Pa(1/2).J(1/2).W(1/2).Gy(1/2)", which is hard for a user to understand.

3.3 Adjusting Weights According to Context

As illustrated by the example above, the weights for each unit can be modified to control the solution. This can be used to guide the solver into selecting units that are preferred for a given context. For instance, let us consider a simple equation for calculating the power:

$$P = RI^2$$

[1] The real values are "Rationalized" before presentation by approximation to rational numbers with small integers.

Suppose the variables P and I have defined units of “W” and “A” respectively. The resistance is inferred to “m2.kg.s-2.A-2” elsewhere (i.e. missing a s-1 to be “Ohm”). If this problem is solved it will regardless of weights result in “Wb”, since that will result in a perfect match, giving the lowest cost (since the distance is zero, the cost will be1/w_j). However, a user might be more familiar if units closer to “W” and “A” is used. By adjusting the weights (increasing “W” and “A”, and decreasing the weight of the rest of the derived units so they are smaller than weights of base units), the result instead becomes: “s-1.A-2.W”. Of course, if it instead is evident from the context that Ohm is the preferred unit, we could decrease its weight and increase the rest of the derived units, resulting in: “s-1.Ohm”.

In conclusion, the resulting unit can be controlled by modifying the weights of derived units. To find out the weights one could look at the current context the unit is defined in. For instance, in an electrical component that does not have any units from the magnetic domain declared, the weights of the units “Wb”,”T” and “H” could be decreased.

The possibility we have chosen is to let the library developers themselves define the weights according to their preferred units. This is the suggested approach described in Section 2.

3.4 Minimizing the Number of Used Derived Units

One problem with the proposed solution is that the same minimal value can be obtained by either selecting a mixture of several derived units or by selecting multiples of only one derived unit. For example, let us consider the unit vector for “Ohm3”, which corresponds to the vector (6,3,-9,-6,0,0,0). With ones as weight, the result becomes “F-1.Ohm.H”; this is not preferable. If weights of units are adjusted according to previous section this might be avoided, but it is not always the case that a context of units may help (the context may be empty).

An alternative is to make an automated adjustment on the units to try to minimize the use of derived units. This can be expressed by the following algorithm:

1. Run the MIP problem with standard weights (or user preferred weights).
2. If several derived units are reported, increase weight on one of them and rerun MIP problem. If less derived units are reported, keep the adjusted weight and repeat 2, otherwise try next derived unit. Repeat until all derived units reported has been tried.

Let us try this idea on our example. As stated above the first run of the problem gave “F-1.Ohm.H” as result. We first increase the weight of “F” and rerun. This gives “Ohm3” as we expect. Same result is also given if we increase weight for “H”. However, if we increase the weight of “Ohm”, the result becomes “F-1.S-1.H”, which is clearly not a good choice.

4 Implementation and Evaluation

A prototype for reporting units has been implemented in Mathematica and a full implementation is now completed in the MathModelica/OpenModelica frontend.

4.1 Testing the Modelica Standard Library

The unit checking and error reporting functionality have been tested in MathModelica on the Modelica Standard library v 2.2.1, which is the latest version where unit checking corrections (based on Dymola version 7.0 unit checking functionality) of the library have not been performed. The unit checker reported the same problems as Dymola did on version 2.2.1 and after applying the corrections made in version 2.2.2, the affected models passed the unit check. This gives an indication of that both tools behave correctly, or at least they behave in the same manner.

However, there are some cases in the standard library where Dymola does not report unit errors, but MathModelica and OpenModelica does. One such case is the use of the built-in `exp` function, which is used in e.g. Semiconductor models in the Electrical package. The problem can be illustrated with this simple model:

```
model UnitProblem
 Real i(unit="A");
 Real v(unit="V") = 2.4;
equation
 i = exp(v);
end UnitProblem;
```

Dymola does not report any errors for this model, even though the exp function should have a dimensionless argument and give a dimensionless return value. Thus, since the MSL is primarily developed with Dymola, the unit conversion corrections that are done for other models are not done for models containing `exp`, `log`, and other dimensionless built-in

functions. This is also reflected in the `Modelica.Math` library where these functions are declared as unspecified unit with e.g., `input Real x;` instead of dimensionless, using input
`Real x(unit=”1”);`

To correct these defects we propose to make the exponent function, logarithm function, and others, that are dimensionless to be declared with unit “1” in the MSL, and that the usage of these functions in the library are corrected so a dimensionless unit is passed and returned from these functions.

4.2 Unit Extendibility

The unit extendibility has been tested and evaluated by adding unit definitions (`defineunit`) for all SI base units and derived units according to [5]. These definitions have been added to `SIunits.mo` in the Modelica standard library. Preliminary tests show that this approach is backwards compatible compared to having these definitions built-in, i.e., unit checking works as expected even if the SI units are defined in the standard library. Models have also been tested, where additional base units (e.g. a currency base unit of “USD”) were added.

4.3 Usability of Error Reporting

Preliminary tests have been conducted for evaluating the usability and readability of errors when different weights are used in different libraries. However, further more comprehensive tests must be performed in the future to verify that the reported units are indeed understandable.

5 Related Work

Unit checking exists in several Modelica tools, such as Dymola[9] and Simulation X[6]. There are also unit checking and dimensional analysis in other non Modelica related tools and languages. See the “Future work” section in previous paper [1] for these references.

To our knowledge, no earlier work has been published on how to select units for presentation. Tools with unit checking have for certain some way of selecting which units to present to the user but the method of how this is done is not clearly stated, and the user can not affect the outcome as is suggested in this paper.

6 Conclusions

We have showed a new method of solving the problem of presenting inferred and inconsistent units by the unit checker in a format that is more understandable for the user. The method is based on forming a mixed integer programming (MIP) problem to decide which base units and derived units to use in the communication with the user. We have also proposed an extension to the Modelica language, where unit definitions can be stated within any restricted class, making it possible to define new user defined units that are not part of the standard SI units.

A prototype has been implemented in Mathematica, followed by a complete implementation in MathModelica and OpenModelica. The same unit errors on the Modelica standard library that Dymola have detected were also reported by our tool, but we also detected more inconsistent units, and proposed further corrections of the standard library.

7 Acknowledgements

This research was funded by CUGS (National Graduate School in Computer Science), MathCore Engineering, the Swedish Research Council (VR), and the Biobridge project supported by the European Commission in the sixth framework programme.

References

[1] D. Broman, P. Aronsson, P. Fritzson, “Design Considerations for Dimensional Inference and Unit Consistency Checking in Modelica”, 6Th International Modelica Conference, March 3-4, 2008, Bielefeld, Germany.

[2] J. Brugård et. Al, “Creating a Bridge between Modelica and the Systems Biology Community”, 7th International Modelica Conference, Como, Italy, 2009.

[3] Peter Bunus, “Debugging Techniques for Equation-Based Languages”. Ph.D. Thesis. Department of Computer and Information Science. Linköping University. 2004.

[4] Bureau international des poids et mesures (BIPM). Le Système international d’unités, The International System of Units. Organisation intergouvermentale de la Convention du Mètre, 8th Edition.

[5] Dynasim. Dymola version 7.0 http://www.dynasim.com [Last access: August 23, 2009].

[6] ITI. SimulationX. http://www.iti.de/ [Last access: August 20, 2009].

[7] MathCore. MathModelica http://www.mathcore.com [Last access: August 23, 2009].

[8] Mathematica. Wolfram Research Inc. http://www.wolfram.com. [Last access: August 23, 2009]

[9] S.-E. Mattson, H. Elmqvist, “Unit Checking and Quantity Conservation”, 6Th International Modelica Conference, March 3-4, 2008, Bielefeld, Germany.

[10] Modelica Association. “Modelica - A Unified Object-Oriented Language for Physical Systems Modeling Language Specification Version 3.1” 2009. Available from http://www.modelica.org.

[11] The OpenModelica Project. Available from: http://www.openmodelica.org

[12] Systems Biology Markup Language (SBML), Available from: http://www.sbml.org