
Preface

The Modelica modeling language and technology is being warmly received by the world community in modeling and simulation. It is bringing about a revolution in this area, based on its ease of use, visual design of models with combination of lego-like predefined model building blocks, its ability to define model libraries with reusable components, its support for object-oriented modeling and simulation of complex industrial applications involving parts from several application domains, and many more useful facilities. The Modelica Association is an open non-profit organization that promotes the use and development of the Modelica language, libraries, and tools. In order to increase the distribution and usefulness of Modelica, the Modelica Association has created a conference series especially for the Modelica end-users and developers, to bring together Modelica users, engineers, researchers, language designers, library developers, and tool vendors. This gives people an opportunity to be informed about the latest developments, to influence the future development of Modelica and its libraries, and to get in touch with people solving similar modeling problems.

In October 2000, the first event in this series took place in Lund, Sweden. This was a great success, with more than 80 participants, and many high-quality papers. The next event, the second international Modelica conference at DLR in Oberpfaffenhofen, Germany, March 18-19 2002, was an even greater success with approximately 120 participants and an increased number of submitted and presented papers.

This volume contains the papers presented at the 3rd international Modelica conference at Linköping University, Linköping, Sweden, November 3-4, 2003. A number of high-quality papers were received. The program committee had the difficult task of planning the conference since not all papers could be accommodated during the limited conference time of two days. Thirty-six papers were selected for regular presentations, and six papers were selected for poster presentations.

More information about the Modelica language, the Modelica Association, this and future events can be found at the web page <http://www.modelica.org>, including all papers from this proceedings and earlier proceedings in the Modelica conference series.

The Modelica'2003 conference was arranged by the Modelica Association in cooperation with PELAB - the Programming Environment Laboratory, Department of Computer and Information Science, Linköping University, Sweden.

Linköping, October 10, 2003

Peter Fritzson

Program Committee

- Peter Fritzson, PELAB, Department of Computer and Information Science, Linköping University, Sweden (Chairman of the committee).
- Bernhard Bachmann, Fachhochschule Bielefeld, Bielefeld, Germany.
- Hilding Elmqvist, Dynasim AB, Sweden.
- Martin Otter, Institute of Robotics and Mechatronics at DLR Research Center, Oberpfaffenhofen, Germany.
- Michael Tiller, Ford Motor Company, Dearborn, USA.
- Hubertus Tummescheit, UTRC, Hartford, USA, and PELAB, Department of Computer and Information Science, Linköping University, Sweden.

Local Organization

- Vadim Engelson (Chairman of local organization).
- Bodil Mattsson-Kihlström.
- Peter Fritzson.

Contents

Index of authors	7
Session 2A	9
Automotive Simulation - I	
Johan Andreasson <i>Division of Vehicle Dynamics, Royal Institute of Technology, Sweden</i> : VehicleDynamics library	11
Stefan Heller, Tilman Bünte <i>TU München; DLR Oberpfaffenhofen, Germany</i> : Modelica Vehicle dynamics library: Implementation of driving maneuvers and a controller for active car steering	19
Hilding Elmquist, Sven Erik Mattsson, Hans Olsson, Johan Andreasson, Martin Otter, Christian Schweiger, Dag Brück <i>Dynasim; Royal Institute of Technology; DLR</i> : Real-time Simulation of Detailed Automotive Models	29
Session 2B	39
Thermodynamic Systems - I	
Francesco Casella, Alberto Leva <i>Dipartimento di Elettronica e Informazione, Politecnico di Milano</i> : Modelica open library for power plant simulation: design and experimental validation	41
Tomas Skoglund <i>Tetra Pak Processing Systems, Sweden</i> : Simulation of Liquid Food Process in Modelica	51
Martin Råberg, Jan Tuszyński <i>Carl Bro Energikonsult, Sweden</i> : Thermo hydraulic library for power systems applications	59
Session 3A	73
Automotive Simulation - II	
Michael Tiller, Paul Bowles, Mike Dempsey <i>Ford Motor Company, USA; Claytex, UK</i> : Development of a Vehicle Modeling Architecture in Modelica	75
Leo Laine, Johan Andreasson <i>Chalmers Institute of Technology; Royal Institute of Technology, Sweden</i> : Modelling of Generic Hybrid Electric Vehicles	87
Erik Surewaard, Eckhard Karden, Michael Tiller <i>Energy Management Group, Ford Forschungszentrum Aachen, Germany; Ford Motor Company, USA</i> : Advanced Electric Storage System Modeling in Modelica	95
Session 3B	103
Tools - I	
Per Sahlin and Pavel Grozman <i>Equa Simulation AB, Sweden</i> : IDA Simulation Environment - a tool for Modelica based end-user application deployment	105
Mike Dempsey <i>Claytex Services Limited</i> : Automatic translation of Simulink models into Modelica using Simelica and the AdvancedBlocks library	115

Eva-Lena Lengquist Sandelin, Susanna Monemar, Peter Fritzson, Peter Bunus PELAB , Linköping University: DrModelica - An Interactive Tutoring Environment for Modelica	125
Session 4A	137
Automotive Simulation - III	
John Batteh, Michael Tiller and Charles Newman Ford Motor Company, USA: Simulation of Engine Systems in Modelica	139
Christian Schweiger, Martin Otter Institute of Robotics and Mechatronics, DLR : Modeling 3D Mechanical Effects of 1D Powertrains	149
Session 4B	159
Electrical and Chemical Systems	
Carla Martin, Alfonso Urquia and Sebastian Dormido Department of Computer Science and Automatic Control, UNED, Spain: SPICELib - Modeling and Analysis of Electric Circuits with Modelica	161
Gerald Reichl Department of Automation and Systems Engineering, Technische Universität Ilmenau: WasteWater - a Library for Modeling and Simulation of Wastewater Treatment Plants in Modelica	171
Session 5: Poster session:	177
Jörgen Svensson and Per Karlsson Dept. of Industrial Electrical Engineering and Automation, Lund University: Adaptive signal management	179
Christian Hoffmann and Jens Kahler Department of Automation and System Engineering, Technische Universität Ilmenau, Germany; De Montfort University, UK: Object-oriented simulation of energy supply systems on the basis of renewable energy	189
Torge Pfafferott, Gerhard Schmitz Department of Technical Thermodynamics, Technical University Hamburg-Harburg: Implementation of a Modelica Library for Simulation of Refrigeration Systems	197
Jerzy Mikler and Vadim Engelson PELAB, Linköping University; Royal Institute of Technology, Sweden : Simulation for Operation Management: Object Oriented Approach using Modelica	207
Emma Larsdotter Nilsson and Peter Fritzson PELAB, Linköping University : BioChem - A Biological and Chemical Library for Modelica	215
Dr S.Sumathi, K. Vinod Kumar PSG College of Technology, Coimbatore, India : Simulation and Control of Induction Motor in Dymola	221

Session 7A	229
Mechatronic Systems - I	
Gianni Ferretti, Marco Gritti, Gianantonio Magnani, Paolo Rocco, <i>Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy:</i> A Remote User Interface to Modelica Robot Models	231
Angelika Peer, Naim Bajcinca, Christian Schweiger <i>Institute of Robotics and Mechatronics, DLR:</i> Physical-based Friction Identification of an Electro-Mechanical Actuator with Dymola/Modelica and MOPS	241
Lars Eriksson <i>Vehicular Systems, Linköping University:</i> VehProLib - Vehicle Propulsion Library. Library development issues	249
Session 7B	257
Thermodynamic Systems - II	
Stefan Wischhusen, Bruno Lüdemann, Gerhard Schmitz <i>Department of Technical Thermodynamics, TU Hamburg-Harburg; Imtech Deutschland GmbH, Germany:</i> Economical Analysis of Complex Heating and Cooling Systems with the Simulation Tool HKSim	259
Hilding Elmquist, Hubertus Tummescheit and Martin Otter <i>Dynasim, Sweden; UTRC, USA; DLR, Germany:</i> Object-Oriented Modeling of Thermo-Fluid Systems	269
Rüdiger Franke, Manfred Rode, Klaus Krüger <i>ABB Corporate Research, ABB Utilities GmbH, Germany:</i> On-line Optimization of Drum Boiler Startup	287
Session 8A	297
Mechatronic Systems - II	
Ivan I. Kossenko and Maia S. Stavrovskaya <i>Moscow State University of the Service, Russia:</i> How One Can Simulate Dynamics of Rolling Bodies via Dymola: Approach to Model Multibody System Dynamics Using Modelica	299
Martin Otter, Hilding Elmquist and Sven Erik Mattsson <i>DLR; Dynasim:</i> The New Modelica MultiBody Library	311
Peter Beater and Martin Otter <i>Fachhochschule Südwestfalen in Soest; DLR, Germany:</i> Multi-Domain Simulation: Mechanics and Hydraulics of an Excavator	331
Session 8B	341
Thermodynamic Systems - III	
Francesco Casella and Francesco Schiavo <i>Dipartimento di Elettronica e Informazione, Politecnico di Milano:</i> Modelling and Simulation of Heat Exchangers in Modelica with Finite Element Methods	343
Magnus Holmgren <i>Solvina, Sweden:</i> Process simulation in industrial projects	353
Andreas Idebrant and Lennart Näslund <i>MathCore Engineering AB; Alstom Industrial Turbines AB, Sweden:</i> Gas Turbine Applications using ThermoFluid	359

Session 9A 367

Mechatronic Systems - III

- Peter Beater and Christoph Clauss** *University of Applied Sciences Südwestfalen, Soest; Fraunhofer-Institut für Integrierte Schaltungen, Dresden*: Multidomain Systems: Pneumatic, Electronic and Mechanical Subsystems of a Pneumatic Drive Modelled with Modelica 369

- Johann Bals, Gerhard Hofer, Andreas Pfeiffer, Christian Schallert** *Institute of Robotics and Mechatronics, DLR*: Object-Oriented Inverse Modelling of Multi-Domain Aircraft Equipment Systems and Assessment with Modelica 377

- Mats Beckman and Johan Andreasson** *Division of Vehicle Dynamics, Royal Institute of Technology, Sweden*: Wheel model library for use in vehicle dynamics studies 385

- Niklas Pettersson, Karl Henrik Johansson** *Scania; Royal Institute of Technology, Sweden*: Modelica Library for Simulating Energy Consumption of Auxiliary Units in Heavy Vehicles 393

Session 9B 399

Tools - II

- Wim Lammen, Jos Vankan, Robert Maas and Johan Kos** *National Aerospace Laboratory, The Netherlands*: Approximation of black-box system models in Matlab with direct application in Modelica 401

- Michael Tiller** *Ford Motor Company*: Parsing and Semantic Analysis of Modelica Code for Non-Simulation Applications 411

- Adrian Pop, Peter Fritzson** *PELAB, Linköping University*: ModelicaXML: A Modelica XML Representation with Applications 419

- Peter Aronsson, Peter Fritzson, Levon Saldamli, Peter Bunus and Kaj Nyström** *PELAB, Linköping University*: Meta Programming and Function Overloading in OpenModelica 431

Index of Authors

- Andreasson, Johan 11, 29, 87, 385
Aronsson, Peter 431
Bajcinca, Naim 241
Bals, Johann 377
Batteh, John 139
Beater, Peter 331, 369
Beckman, Mats 385
Bowles, Paul 75
Brück, Dag 29
Bunus, Peter 125, 431
Bünte, Tilman 19
Casella, Francesco 41, 343
Clauss, Christoph 369
Dempsey, Mike 75, 115
Dormido, Sebastian 161
Elmqvist, Hilding 29, 269, 311
Engelson, Vadim 207
Eriksson, Lars 249
Ferretti, Gianni 231
Franke, Rüdiger 287
Fritzson, Peter 125, 215, 419, 431
Gritti, Marco 231
Grozman, Pavel 105
Heller, Stefan 19
Hofer, Gerhard 377
Hoffmann, Christian 189
Holmgren, Magnus 353
Idebrant, Andreas 359
Johansson, Karl Henrik 393
Kahler, Jens 189
Karden, Eckhard 95
Karlsson, Per 179
Kos, Johan 401
Kossenko, Ivan I. 299
Krüger, Klaus 287
Kumar, Vinod 221
Laine, Leo 87
Lammen, Wim 401
Larsdotter Nilsson, Emma 215
Lengquist Sandelin, Eva-Lena 125
Leva, Alberto 41
Lüdemann, Bruno 259
Maas, Robert 401
Magnani, Gianantonio 231
Martin, Carla 161
Mattsson, Sven Erik 29, 311
Mikler, Jerzy 207
Monemar, Susanna 125
Newman, Charles 139
Nyström, Kaj 431
Näs, Lennart 359
Olsson, Hans 29
Otter, Martin 29, 149, 269, 311, 331
Peer, Angelika 241
Pettersson, Niklas 393
Pfafferott, Torge 197
Pfeiffer, Andreas 377
Pop, Adrian 419
Reichl, Gerald 171
Rocco, Paolo 231
Rode, Manfred 287
Råberg, Martin 59
Sahlin, Per 105
Saldamli, Levon 431
Schallert, Christian 377
Schiavo, Francesco 343
Schmitz, Gerhard 197, 259
Schweiger, Christian 29, 149, 241
Skoglund, Tomas 51
Stavrovskaja, Maia S. 299
Sumathi, S 221
Surewaard, Erik 95
Svensson, Jörgen 179
Tiller, Michael 75, 95, 139, 411
Tummescheit, Hubertus 269
Tuszynski, Jan 59
Urquia, Alfonso 161
Vankan, Jos 401
Wischhusen, Stefan 259

Session 2A

Automotive Simulation – I

VehicleDynamics library

Johan Andreasson
 KTH Vehicle Dynamics, Sweden
 johan@fkt.kth.se

Abstract

A Modelica library for vehicle dynamics problems has been developed and a pre-release version is available. The library is based on modular design and contain models of components as well as suspensions, chassis and vehicles. In this paper the modelling structure is discussed and it is illustrated how this simplifies the usage.

1 Introduction

Due to the multidomain qualities of Modelica, it has for long been thought of as a suitable tool for complete vehicle modelling. Detailed models of vehicle power train are available [1] and chassis models have also been presented [2, 3]. This paper presents the `VehicleDynamics` library that provides models for vehicle dynamics studies. A pre-release version is available [4] for download.

The library is divided into sub packages containing models of vehicle chassis and wheels, environments and drivers. The library structure is best understood by considering Figure 1. The chassis, which has been the main focus within this work, contains body, suspensions and wheels. To control the chassis' motion a driver model is used. This could either be open loop from a predefined input or a more advanced driver model to mimic human behaviour.

The chassis have connectors to the wheels to allow the addition of a power train. There is also a MultiBody connector to the body to allow additional models to be attached. This is here illustrated by an aerodynamic model and an additional load, but it is also possible to attach e.g. trailers. Environments representing ground and atmosphere conditions are selected independent of the rest of the vehicle model.

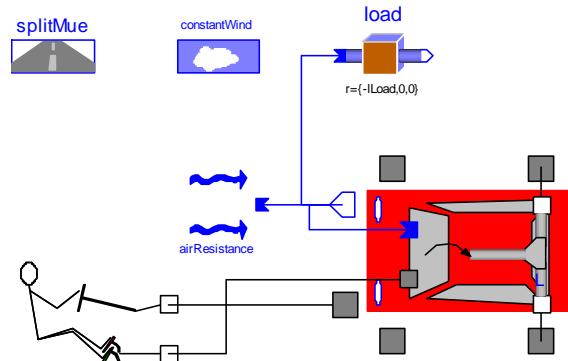


Figure 1: The layout of a vehicle model with a power train and an additional load.

2 Chassis

In vehicle dynamics studies, the chassis is of great importance. Not only the geometry of the suspensions but also bushing and strut characteristics are of great importance and thus, the models often tend to be detailed, containing models representing different fields of expertise. At the same time it is crucial that the models are easily reconfigurable and that it is possible to grasp the contents of a model without needing to understand the details. To allow this, the chassis is defined in a modular and hierarchical way based on four levels. The highest level is the vehicle level and can be seen in Figure 1. The three remaining levels are chassis, suspension and component levels and they are illustrated in Figure 2.

Chassis level Within the chassis level a complete chassis is built up using suspensions, wheels and a body. Here, a four wheel chassis with front wheel steer is shown, but other models can easily be defined, e.g. with four wheel steer or six wheelers. However, there is no need to define a new chassis model for each configuration

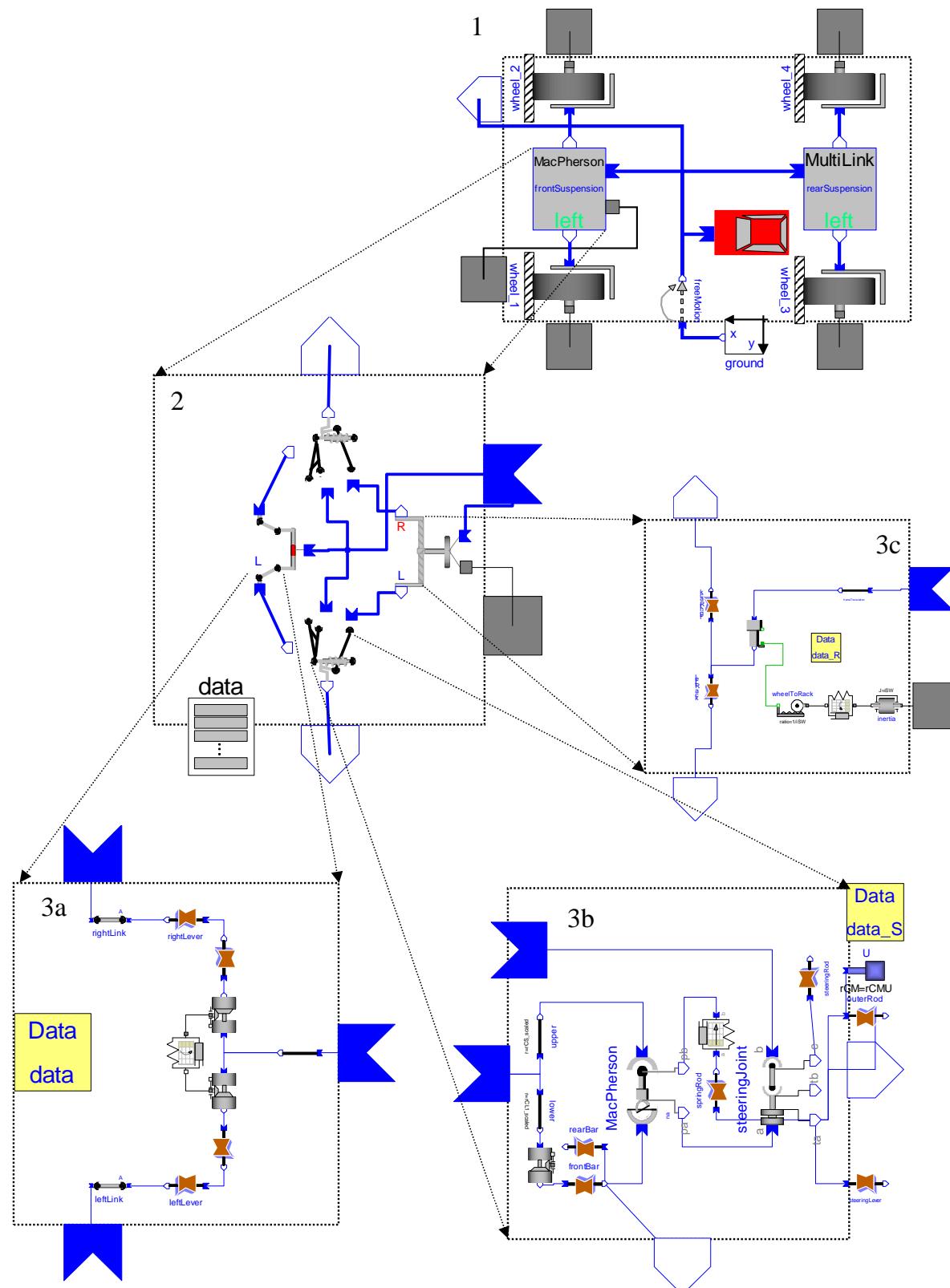


Figure 2: The hierarchical levels of a chassis model. 1: the chassis level, 2: the suspension level, 3: the component level represented by an anti roll bar linkage: 3a, a MacPherson linkage: 3b, and a steering rack: 3c.

of different suspensions or wheels. This is instead handled as described in Section 6, using the `redeclare` constructs in Modelica.

Suspension level Common for all individual suspensions are the linkages that carry the wheels and normally there is some kind of roll-suppressing mechanism between these. If the suspension is steerable there is also a steering rack. Each of these components can be used to build up new suspensions. Thus, the suspension linkage, here a MacPherson, could easily be replaced by another linkage, e.g. a double wishbone or a multi-link. In the same manner, the steering and the anti roll linkages can also be replaced. Furthermore, all parameters are gathered in a data record, making it easy to change a whole suspension setup.

The idea with the suspension level is to make it easy to reconfigure a car by just swapping suspension and therefore, all suspension models should share the same basic interface, i.e. one MBS-cut for the connection to the body. There should also be an MBS cut for each wheel (normally two) that is to be connected to the suspension. Additionally, there may be some extra connectors depending on the suspension. For example, a steerable suspension will also have a connector for a steering wheel.

Component level Within the component level, the foundation for efficient reuse of vehicle models is laid. Components like a-arms, bushings, Mac Pherson struts, trailing arms, multi-links, anti roll linkages, rack steerings etc. are available. In this version, these components are based on the Modelica and ModelicaAdditions libraries. Other basic models that are needed in the component models, such as nonlinear spring-dampers, are described in Section 4.

2.1 Parameterisation

The parameterisation of the chassis is based on a Body Geometric Reference frame (BGR). This frame is oriented according to the DIN standard, the x , y and z axes point forward, left and upward respectively, see Figure 3.

The geometry of the chassis and the suspensions are then defined by a set of points where joints and

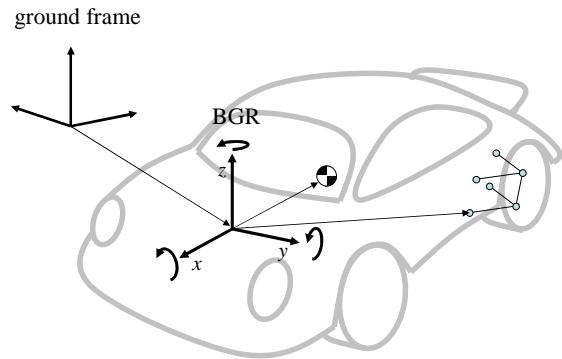


Figure 3: The vehicles motion is specified by how the BGR moves relative to the ground frame. From the BGR, locations of e.g. centre of mass and construction points are defined.

bushings are located. Additionally, the mass and inertia properties of the parts within the linkage can be defined. For a comprehensive parameterisation of these properties, a systematic definition of the parameter names is necessary.

The geometry is mainly defined by the connection joint locations, connection points. Additionally, the direction(s) of a joint's degree(s) of freedom must be given if not defined by the connection joint points. The geometry parameters are defined as:

```
[geometry parameter]
  =[property][connection]_[wheel no]
[connection]
  =[part 1][part 2]..[part n]
```

While the mass and inertia properties are component specific and are thus named according to:

```
[component parameter]
  =[property][part]_[wheel no]
```

Where [property] and [part] are defined according to Table 1.

When there are more than one part of the same type, a number is added to the character. For example, if there are more than one link, as in a double-wishbone, they are numbered L1, L2, etc., starting at the front upper link. The wheels are numbered from front left towards right and rear. Some examples of parameter names are give below:

`rCL1_2` Location of connection joint between chassis and link 1 at front right wheel.

r	location
n	direction of rotation or translation
m	mass
rcm	location of centre of mass
c	stiffness
d	damping
f	force
t	torque
i	inertia element, (gear) ratio
q0	Relative offset
qInit	Initial value
C	chassis
R	steering (rack)
U	upright, part that holds the wheel
P	pivot element
S	strut, 1D force element
L	link or rod
B	body or bushing
A	antiroll
X	undefined/general part
W	wheel

Table 1: Naming of parts and properties.

i22L1_3 Inertia element i22 of link 1 at rear left wheel.

nCU_4 Direction of revolution of the joint that connects the upright 1 to the chassis at the right rear wheel. This could for example be the rotation axis of a swing axle.

rUL1L2_1 Location of connection joint between the upright and link 1 and 2 at the front left wheel. This could for example be the upper spindle joint at a double wishbone suspension.

In many cases it is convenient to mirror components in a car, for example left and right suspensions. To handle this, a three-dimensional scale factor is used. This can rescale and mirror objects, for example `scaleFactor={1, -1, 1}` mirrors the model around the xz-plane.

3 Wheels

Good tyre models are essential for driving simulation of all ground vehicles using pneumatic tyres. However, tyre behaviour is extremely complex, often

requiring different models to cover various aspects. Therefore, these are packaged together with the rim and the hub to form ready-to-use wheel models. The models used in this package are based on a tyre model suggested in [5] and implemented in Modelica in [2]. This model uses steady state force characteristics together with a simple tyre belt deflection model. Additionally, the Magic Formula [6, 7] is also available for the tyre force calculation.

Common for both models are the assumption that the contact patch between the tyre and the road can be approximated by a point. To avoid coupling the wheel and the road models, this contact point is calculated using the inner/outer Modelica language constructs to get information from the Environment model about the current altitude and road condition. As a consequence, the road properties can be defined at the top-level of the model and can also be easily changed.

Due to the contact point assumption, this model has troubles travelling on roads with sharp edges, which often is the case when a real road profile is meshed. To manage this and other issues, a new Wheels library is currently under development [8].

4 Utilities

For vehicle dynamics studies it is essential that the characteristics of flexible elements such as struts and bushings are modelled. To deal with this, a set of basic force elements are available. These are either taking into account the deformation along one degree of freedom, 1D-forces, or six degrees of freedom, 3D-forces.

The 1D-forces apply force depending on the deformation according to the Modelica.Mechanics definitions or depending on the distance between two frames. The force versus deformation and its time derivative are defined as look-up tables.

The 3D-forces calculates the relative rotation between two frames, either as a linearisation around a zero deformation or nonlinear allowing deformations up to π radians. The force can be calculated as a nonlinear spring-damper element, without considering the coupling effects. Linear spring-damper elements with bump stops, taking into account the coupling between the degrees of freedom, are also available. These use two 6×6 matrices for stiffness and damping to calculate the resulting force and torque vectors from the

deformation:

```
[fa+f_bump; ta] =
-C*[r_rela-r_rela0;phi_rela-phi_rela0]
-D*[v_rela;w_rela];
```

The `f_bump` is an additional, stiff, spring-damper force that is active when `r_rela` is outside the edge of the linear region. It is directed perpendicular to the edge that can be defined either as a cylinder, sphere or box. More complex geometries and models, using e.g. fractional derivatives, are currently not implemented.

In addition to the force elements, there are also a set of joints particular relevant for vehicle dynamics studies. Composite joint models (e.g. an aggregation of a revolute, a spherical and a universal joint) are available to reduce the nonlinear algebraic loops that normally occur in suspensions with ideal joints [9].

Also there are joints that applies unphysical constraints to the vehicle. For example, it is in many cases interesting to be able to perform a manoeuvre at constant speed. In other simulation packages like e.g. ADAMS [10] this is solved by adding a power train and applying a cruise control. The drawback is then that the user need to add unnecessary complexity as well as unwanted dynamics to the analysis. Here, it is instead possible to constrain the velocity along the longitudinal axis of the car.

Other cases where it may be interesting to constrain the vehicle in an unphysical way is when studying the effects of flexibility in the suspensions. Typically, there are very high eigen-frequencies due to high stiffness and low mass that are irrelevant for the analysis and thus using joint models that do not consider the acceleration may speed up the simulation without loosing relevant accuracy.

5 Drivers

The driver models used in vehicle dynamics studies are either open loop drivers that apply a predefined motion on the steering wheel or more advanced models that try to mimic the human behaviour, taking into account some states of the body and sometimes also the force-feed-back through pedals and steering wheel.

More advanced studies considering combined cornering and braking/acceleration requires a tight interaction of steering wheel and pedal output. The interface is prepared to be able to handle the aspects

described above, it consists of two rotational flanges for steering and drive. For closed loop driver models, an MBS connector is used to make the model able to sense the vehicle's motion.

6 Usage

The modular design of the vehicle models gives three significant advantages. First, it is easy to reuse already developed models. Secondly, because of the standardised interfaces, much of the test rigs already implemented can be used for new models as well, making it easy to test and verify these. A third aspect that will be illustrated further is the ability to exchange sub models without redesigning the original model which leads to very flexible use.

To illustrate this, it is here described how one model can cover different combinations of suspensions of a front steered four wheeled chassis.

- 1 Double-clicking the chassis in the `StandardCar` example opens the dialog box showed in Figure 4.1. Here it is possible to select the desired models for all the wheels as well as for the front and rear suspension, respectively. As indicated in the figure, a drop down box appears, listing all possible choices.
- 2 Once the desired suspensions and tyre models are chosen, the corresponding parameters can be edited by pressing the triangle at the end of each row. Since all suspension parameters are set in a data record, Figure 4.2, it is easy to select the desired setup from the dropdown box, again only showing the relevant options. The geometry is also indicated in a figure to make it more easy to verify that the right suspension is selected and to understand the parameterisation.
- 3 Even if a specific setup is chosen for the suspension or not, it is still possible to edit each value separately as illustrated in Figure 4.3. Except for the geometry parameters, it is also easy to change mass and inertia properties as well as the characteristics of the force elements.
- 4 To facilitate the modification of force elements, which can be rather complex, it is possible to both edit these as Modelica code, Figure 4.4 and to visualise the characteristics, Figure 4.5.

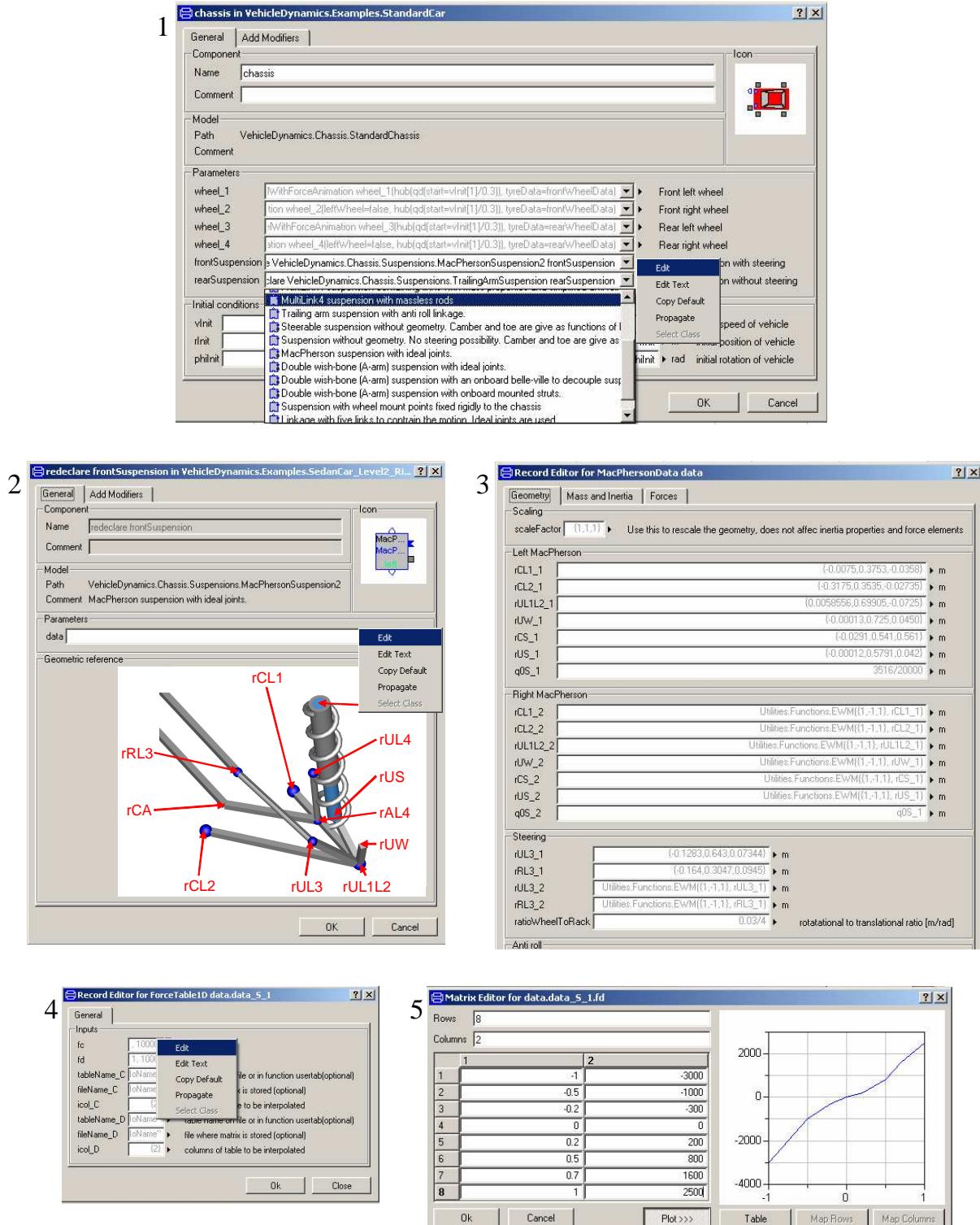


Figure 4: Dialog boxes for modification and parameter settings.

Within the VehicleDynamics library, there is a set of samples available to illustrate the use of the library. Except for the StandardCar, there is a model

of a Formula 3 race car, Figure 5 and a car with a trailer. Furthermore, there are some examples showing how components and suspensions can be tested in-

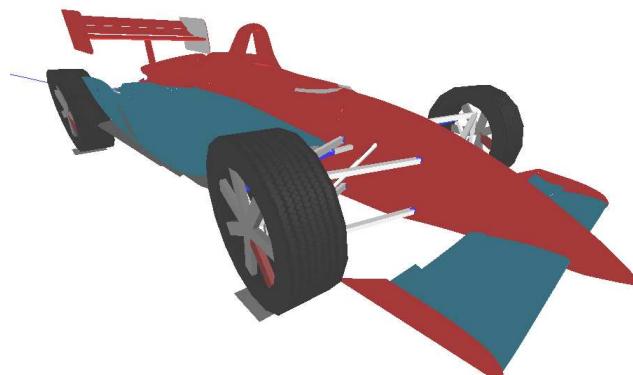


Figure 5: Animation view of the Formula 3 car example.

dividually, see for example Figure 6.

Additionally, there are four variants of the StandardCar corresponding to four different levels of detail of a mid-sized car with a front MacPherson suspension and a rear multi-link suspension. The main idea is to illustrate how Modelica can be used to model vehicles with a wide range of level of detail. The simplest model uses look-up tables to define the deflection of the suspension and an Ackermann function for the steering geometry. The second level uses linkages with ideal joints while level three and four use bushings. A more detailed description of these models can be found in [9]. In Figure 7, two pictures of the level 2 car when performing a double lane change manoeuvre, ISO3888-1:1999, is shown.

7 Conclusions

In this work, a library for modelling of vehicle dynamics related problems is realised. It uses the interfaces from the Modelica and ModelicaAddtions packages to be compatible with other libraries.

VehicleDynamics provides an architecture for vehicle modelling as well as components, suspensions and chassis model to simplify for the user to extend the library according to his/her needs. The modular structure of the model design allows to take advantage of the potential of the Modelica language.

VehicleDynamics is freely available and the source code is completely open. The library can also be used together with the PowerTrain package to model complete vehicles.

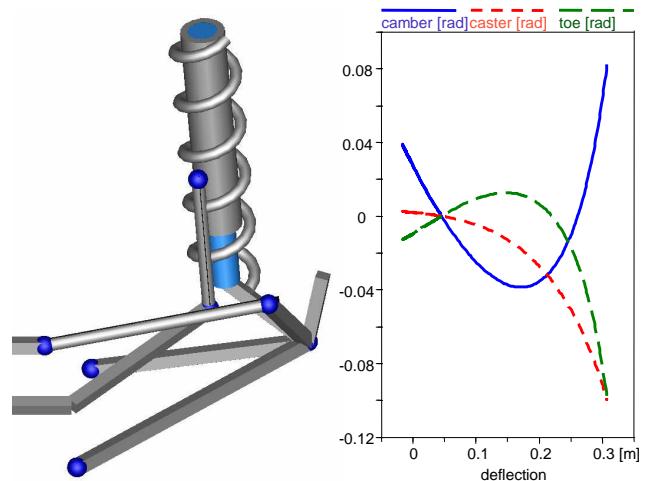


Figure 6: Animation of a MacPherson suspension along with a mapping of the change of camber, caster and toe angles as function of bump motion.

8 Future work

The library is under constant development. Upcoming improvements concern an extension of the flexibility to include swapping between bushings and joints and better ability to add active components such as controllable dampers. To be able to study the gyroscopic effects of the power train and torque oscillations due to Cardan joints, multi-body models of drive shafts and brakes will also be included. The intention is also to convert the VehicleDynamics to the new MBS-library [11] and to improve the documentation.

9 Acknowledgements

This library is based on models developed for the Driving Dynamics project within the Swedish National Research Programme "The Green Vehicle/FCHEV". Some components of this library, such as the Rill tyre model and the aggregation joints for analytically solving kinematic loops, have been developed by Martin Otter, from DLR - Institute of Robotics and Mechatronics, Germany. Part of this library was developed with financial support from Dynasim AB, Sweden and DLR - Institute of Robotics and Mechatronics, Germany.

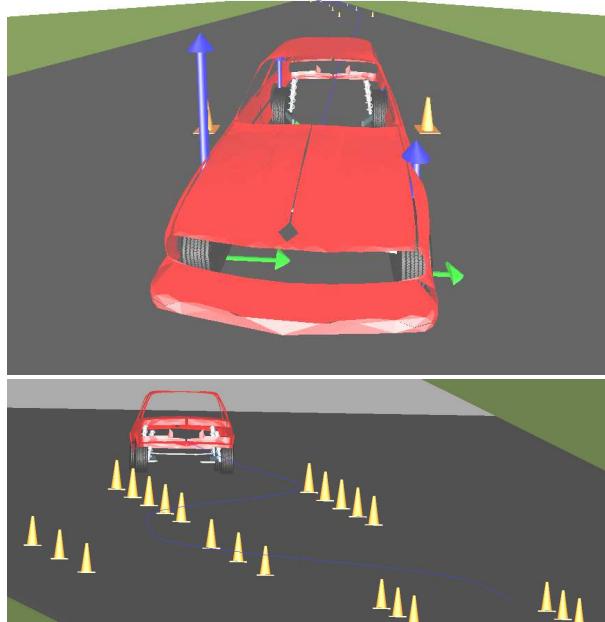


Figure 7: Vehicle performing a double lane change at 20 m/s.

References

- [1] M. Otter, M. Dempsey, and C. Schlegel. Package PowerTrain: A Modelica library for modeling and simulation of vehicle power trains. In Peter Fritzson, editor, *Proceedings of the Modelica'2000 Workshop*, Lund, October 2000. The Modelica Association and Lund University.
- [2] J. Andreasson, A. Möller, and M. Otter. Modeling of a racing car with Modelicas MultiBody library. In Peter Fritzson, editor, *Proceedings of the Modelica'2000 Workshop*, Lund, October 2000. The Modelica Association and Lund University.
- [3] S. Drogies and M. Bauer. Modeling Road Vehicle Dynamics with Modelica. In Peter Fritzson, editor, *Proceedings of the Modelica'2000 Workshop*, Lund, October 2000. The Modelica Association and Lund University.
- [4] Modelica Association. <http://www.modelica.org>.
- [5] G. Rill. *Simulation von Kraftfahrzeugen*. Vieweg, 1994.
- [6] E. Bakker, H.B. Pacejka, and L. Lidner. A new tire model with application in vehicle dynamics studies. *SAE transactions, paper 890087*, pages 83–93, 1989.
- [7] H.B. Pacejka. *Tyre and vehicle dynamics*. Butterworth Heinemann, 2002.
- [8] M. Beckman and J. Andreasson. Wheel model library in Modelica for use in vehicle dynamics studies. In Peter Fritzson, editor, *Proceedings of the 3rd International Modelica Conference*, Linköping, November 2003. The Modelica Association and Linköping University.
- [9] H. Elmquist et al. Realtime simulation of detailed physically based automotive models. In Peter Fritzson, editor, *Proceedings of the 3rd International Modelica Conference*, Linköping, November 2003. The Modelica Association and Linköping University.
- [10] ADAMS, Mechanical Dynamics Inc. <http://www.adams.com/>.
- [11] M. Otter, H. Elmquist, and S.E. Mattson. The new Modelica MultiBody library. In Peter Fritzson, editor, *Proceedings of the 3rd International Modelica Conference*, Linköping, November 2003. The Modelica Association and Linköping University.

MODELICA vehicle dynamics library: Implementation of driving maneuvers and a controller for active car steering

Stefan Heller*

Technische Universität München (TUM),
Institute for Real-Time Computer Systems (RCS),
D-80290 München, Germany

Tilman Bünte†

German Aerospace Center (DLR),
Institute of Robotics and Mechatronics,
Oberpfaffenhofen, D-82230 Wessling,
Germany

Abstract

This paper deals with the assessment and exploitation of the recently released MODELICA-based vehicle dynamics library. A setup of various driving maneuvers is accomplished. These maneuvers will be conducted by providing steering angle and gas/brake position to the car model of the library. The common linearized single track model is derived as an approximative model for the fully detailed vehicle dynamics model. This model is used for synthesis of feedforward control and later also as a nominal model for active car steering control aiming at vehicle dynamics stability improvement. The applied robust steering controller structure is known as the disturbance observer. Simulations are used to demonstrate the effectiveness of the vehicle dynamics enhancement in comparison to the uncontrolled vehicle. Also some experiences with the vehicle dynamics library are pointed out.

real time ability MODELICA comprises some powerful promising features: *hybrid modelling*, *inline integrators* and *symbolic preprocessing*.

The MODELICA vehicle dynamics library [1] basically consists of a detailed mathematical model comprising the governing multibody differential equations. Moreover, there are some rudimental steering schedules for conducting simple maneuvers. This library is also appropriate for the analysis, synthesis and evaluation of control systems concerning vehicle dynamics. All considerations in this paper refer to an unofficial prerelease of the *vehicle dynamics library* [1] and particularly to the *chassis level 2*. The library and some significant features will be outlined in section 2. For the setup of more sophisticated and realistic maneuvers a generic driver module is needed, which represents the action of a real driver. This driver module conducts the maneuvers and is therefore called *maneuver control block*.

The *single track model* is used as an approximative model for the more detailed car model. It is used for the synthesis of a lateral acceleration controller which is contained in the *maneuver control* block. The identification of the parameters of the single track model is explained and the parameters are given in section 3. The *maneuver control* block is introduced in section 4. The lateral acceleration controller provides steering wheel angle suitable for following a predefined lateral acceleration profile. Alternatively, the steering angle can be provided directly to the car model. Analogue is the setting of the position of the gas/brake pedal. This position and hence the speed of the car model are controlled according to a predefined speed profile.

Maneuvers executing full braking need ABS-functionality. Therefore, a wheel slip controller

1 Introduction

As a rather recent field of research the simulation of multiphysical objects gets more and more weight. The behaviour of car models during maneuvers is of interest, e. g. for research and development of cars. The general ability of executing the simulations in real time is important for hardware-in-the-loop investigations. The MODELICA language is able to handle multiphysical objects. Concerning the

*e-mail: Stefan.Heller@mytum.de

†e-mail: Tilman.Buente@dlr.de

is introduced in section 5 which approximates the function of a real ABS-system.

Section 6 deals with the application of the *maneuver control* block. Four maneuvers are conducted which illustrate the action of this block. Also the car model of the vehicle dynamics library is evaluated by means of these maneuvers. Moreover, the maneuvers *braking in a curve* and *μ -split braking* demonstrate the operation of the added wheel slip controller.

In section 7 the active car steering controller for improvement of yaw dynamics is introduced. When the car model is exposed to asymmetric conditions like asymmetric load, side wind or asymmetric road friction while braking critical yaw dynamics can cause instability of the car. This instability can be reduced and the car can be brought back into safe state by active car steering. The controller used in this paper is known as the *disturbance observer* [2]. It determines an additional steering angle which is superimposed mechanically to the steering wheel angle.

The controlled car is evaluated in section 8 by comparing simulations of the maneuvers to simulations with the conventional car.

Finally, section 9 reports on some experiences about working with the vehicle dynamics library.

lateral acceleration a_y , position and orientation to the *maneuver control* block; yaw rate r and speed v_x to the *Vehicle Dynamics Control (VDC)* block; speed v_i and rotational speed ω_i of each wheel to the wheel slip controller.

The gray connections refer to the steering angle signals. During maneuvers with the conventional car, the *VDC* block is inactive. Hence, the steering angle from the *maneuver control* block is equivalent to the input steering angle at the car model. The thin gray connections are for transmission of the reference and the actual additional steering wheel angle between *VDC* block and *mechanical steering angle addition* block.

The value for the gas/brake pedal position in the *maneuver control* block is passed to the wheel slip controller block. For a positive value the acceleration is carried out by equal propelling torques on both wheels of the rear axle. A negative value for the pedal position means braking. Then the deceleration command is distributed on the brakes of the four wheels according to the wheel slip control. The black bondings represent the propelling torques (solid) and braking torques (dashed) of the wheels.

The steering angle is passed to the car model by use of the *position element* of the Mechanics Package of MODELICA. The position element is accordingly used as interface for the gas/brake pedal position. In the latter case the only additional feature is the dependence on the signed value (as described before).

2 MODELICA vehicle dynamics library

The vehicle dynamics library [1] is structured hierarchically using four levels. The uppermost level is called the *vehicle level* and contains the total model of the car. This car model can optionally be completed by a power-train, brakes, a block which has the function of a driver, and environmental conditions, like certain roadtypes (friction) or aerodynamics. On the next level the *chassis* components are modeled explicitly, e.g. with a front and a rear suspension, wheels and body. The *suspension level* allows the reconfiguration of a car with different suspensions. Therefore, the suspensions have the same interface. The lowest level is the *component level* with components like trailing arms, struts, linkages etc. which are based on the standard MODELICA and ModelicaAdditions libraries.

Fig. 1 shows our final setup from the *vehicle level* for simulating the maneuvers with active car steering. The dotted connections indicate the transmission of the signals on the actual state of the car: speed v_x ,

3 Single track model parameters

For controller design the common linearized single track model [2][3] is employed as an approximative model for the fully detailed vehicle dynamics model. For example, the steady state gain $G_V|_{s=0}$ from steering wheel angle δ_L to lateral acceleration a_{ydef} is needed to implement feedforward control for the steering controller in the *maneuver control* block. Hence, first the parameters of the single track model are identified.

The single track model parameters corresponding to the fully detailed vehicle dynamics model are determined by an optimization aiming at best matching of the simulation results for both steady state cornering and dynamic maneuvers. The parameters given in Tab 1 are the single track parameters for the car model in

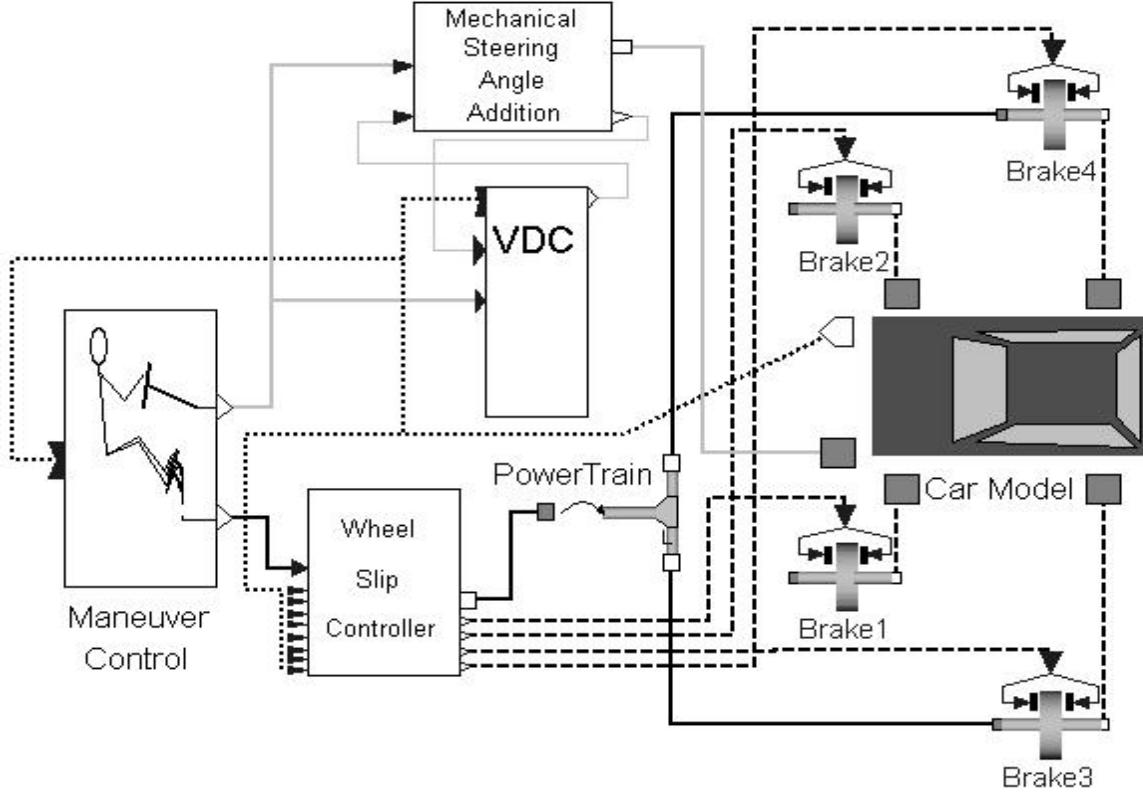


Figure 1: structure of the simulation set up with active car steering controller

the vehicle dynamics library (which is by default parameterized as a BMW 3-series).

4 The *maneuver control* block

As mentioned before, this *maneuver control* block is a model for the real driver's actions which are necessary to perform a certain maneuver. It operates the steering angle and gas/brake pedals of the car model. This block needs information on the actual dynamic state of the car i.e. virtual measurement signals of the actual speed v_x and the lateral acceleration a_y . The *maneuver control* block consists of a lateral dynamics controller (Fig. 2) and a speed controller (Fig. 3).

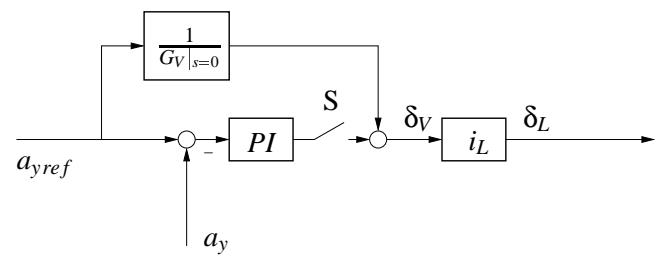


Figure 2: Controller for providing steering wheel angle according to a predefined lateral acceleration profile

The total steering wheel angle output by this block is composed of a feedforward and a feedback part which

may individually be hooked up as adequate for a specific maneuver (Fig. 2). For some maneuvers, a mere feedforward steering is sufficient. For others maintaining a certain lateral acceleration requires feedback control (this means incorporating the PI-controller by closing switch S in Fig. 2). The block i_L is the gear ratio between steering wheel angle δ_L and average steering angle at the two front wheels.

Similarly to the steering angle controller the speed controller (Fig. 3) consists of a feedforward and a feedback part.

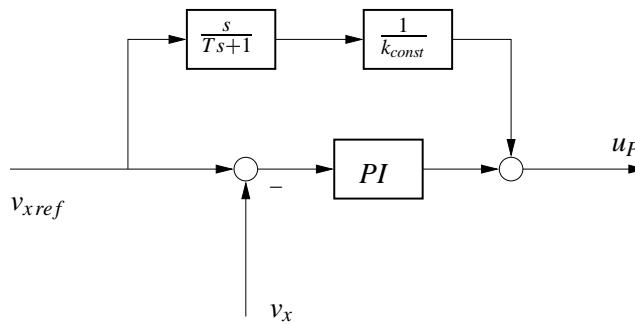


Figure 3: Controller for the position of the gas/brake pedal

The feedforward control is based on the assumption that the actual longitudinal acceleration is proportional to the gas/brake pedal position u_P :

$$a_x = \frac{1}{s} v_x = k_{const} u_P \quad (1)$$

For the model of the library it is $k_{const} = 0.0025 \text{m/s}^2$. This relation has been validated by several simulations. The inverse is used for feedforward control. The low pass filter in Fig. 3 is added for making the included differentiator causal.

5 Wheel slip controller

To be able to execute full braking an ABS-functionality is needed. Therefore, the actual speed v_i and rotational speed ω_i of each wheel must be known from the car model to calculate the actual longitudinal slip at each wheel (with (2)). R is the radius of the wheels.

$$S_i = 1 - \frac{R \omega_i}{v_i} \quad (2)$$

These slips S_i are then used to calculate the average slip S_{avg} of the four wheels: $S_{avg} = \sum_{i=1}^4 \frac{S_i}{4}$. The slip controlled braking force $T_{Bi} =$

$\frac{1}{4} (1 - |S_i|) (1 - S_{avg}) T_P$ is then calculated with the braking force at the pedal T_P for each wheel.

This ensures that the brake torque at the brake pedal is distributed on the brakes of each wheel according to the slip at the wheel. Blocking of a wheel is avoided and the vehicle remains controllable. The wheel slip controller was designed heuristically to copy the basic ABS functionality. In our simulations it turned out that it works satisfactory (see next section).

6 Driving maneuvers

Four different maneuvers have been chosen from [4]. This election is made in regard to expressiveness of the maneuvers to evaluate both the usage of the vehicle dynamics library and the car model and also the performance of a active steering controller for vehicle dynamics. At first, the conventional car (without additional steering) is considered. Therefore, the VDC block is inactive.

Maneuver: steady state cornering . This maneuver is conducted by maintaining a constant lateral acceleration which is adjusted by the steering wheel controller from Fig. 2. Starting from a maximum value, the speed is slowly decreased to cover a certain speed operating domain.

Fig. 4 shows the results of maneuver *steady state cornering*. To maintain a constant lateral acceleration a_y during a constant decline of speed v_x the steering wheel angle δ_L rises.

Maneuver: braking in a curve For this maneuver the steering wheel angle is kept constant. Full braking is applied. Simulation results are shown in Fig. 5.

When the braking is applied the vehicle is in the state of a left turn with high lateral acceleration ($\approx 6 \text{m/s}^2$). Fig. 5e shows the slip at the wheels. The rear left wheel encounters the least vertical load. Therefore, its slip exceeds the other wheels. However, the braking force at this wheel is reduced by the slip controller (Fig. 5f, 5h). Hence the slip remains limited and blocking of the wheels is prevented.

Maneuver: sequence of alternating steering wheel angle steps . A so called *lateral acceleration level*

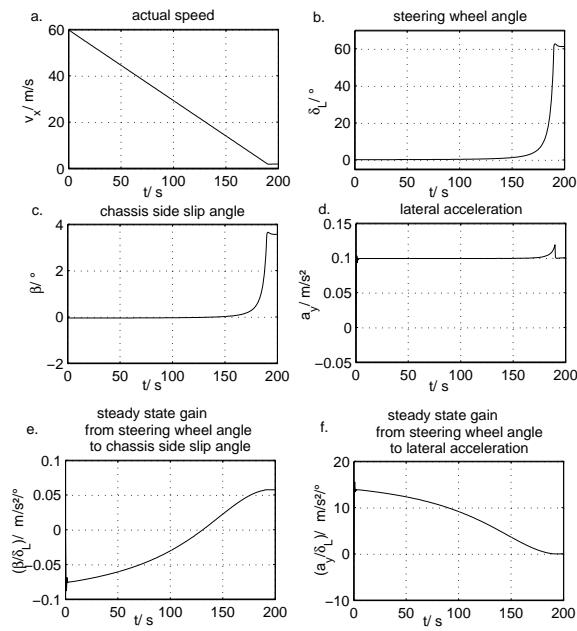


Figure 4: Simulation results of the maneuver *steady state cornering*

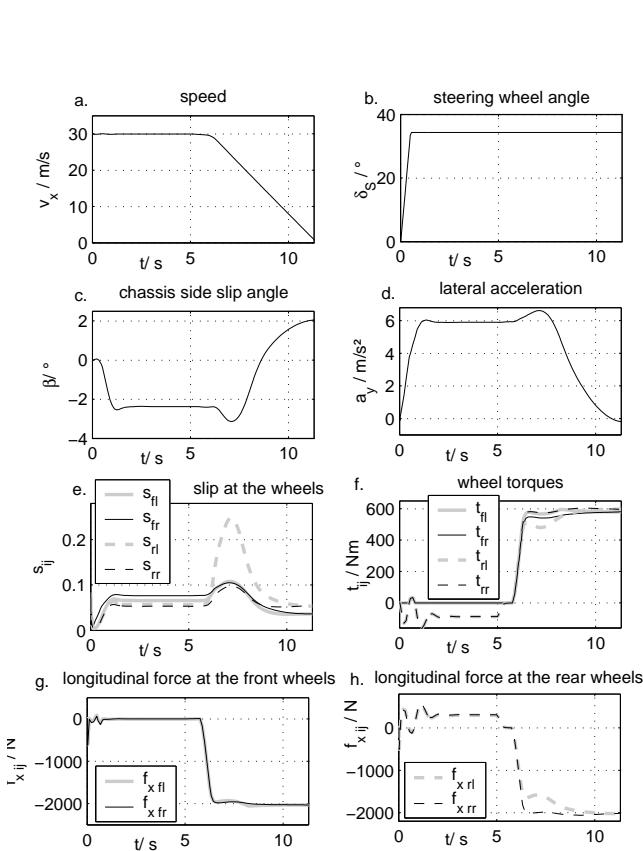


Figure 5: Simulation results of maneuver *braking in a curve* (indices: i= front,rear; j=right,left)

needs to be assigned prior to the simulation. The steering wheel angle is periodically switched between opposite values depending on the actual speed. The step height is computed from the single track model such that it corresponds to a steady state lateral acceleration being equal to the preassigned lateral acceleration level. Again, speed is decreased slowly to cover a certain speed range (Fig. 6). The resulting lateral ac-

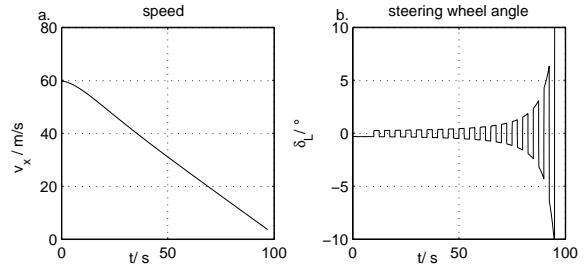


Figure 6: Speed v_x and lateral acceleration a_y of maneuver sequence of alternating steering angle steps

celeration (Fig. 7) gives information on the dynamic steering responses of the car model.

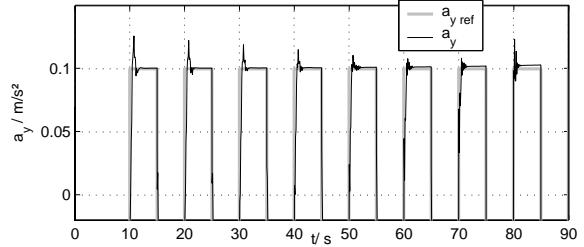


Figure 7: Simulation results of maneuver sequence of alternating steering angle steps

Maneuver: μ -split braking. The steering wheel angle is zero ($\delta_L = 0^\circ$) and not changed during the whole maneuver. Initially the car model is driven at constant speed (initial value: $v_{xo} = 30\text{m/s}$). Then the car model is driven along parallel lanes with different friction. The wheels on the right side of the car drive on the lane with low friction ($\mu = 0.4$). When the braking is applied the asymmetric road friction at the wheels causes a disturbing yaw moment. It is expected that the wheels on the lane with low friction, are less detained and therefore the direction of the car tends towards the lane with higher friction. Fig. 8 shows the results of the simulation. The friction of the road under the right wheels is reduced to $\mu = 0.4$ and as expected rises again to $\mu = 1$ (Fig. 8 b) when the wheels enter the left lane. The stroboscopic diagram in Fig.

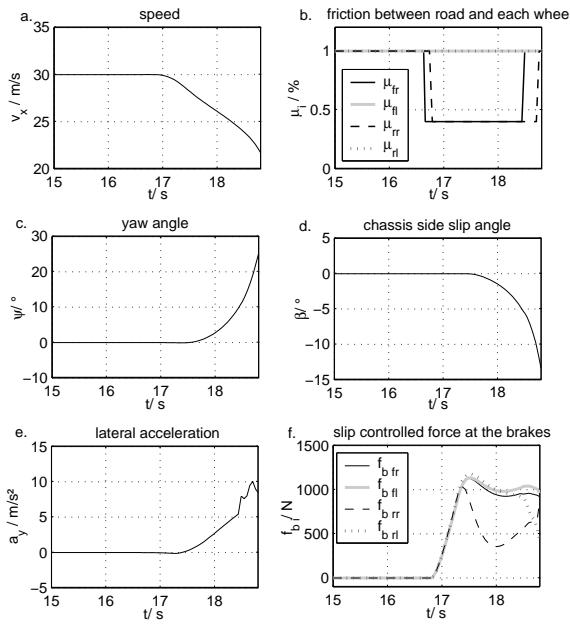


Figure 8: Simulation results of the maneuver μ -split braking

9 shows the course of the vehicle from bird's eye view.

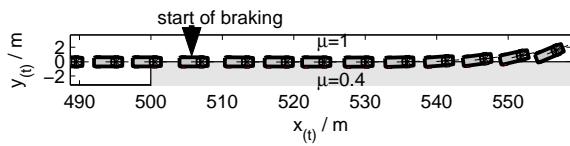


Figure 9: Stroboscopic recorded course of the maneuver μ -split braking

7 Controller for active car steering

The effect of the yaw disturbance torque shall now be reduced by adding a controller for active car steering. To improve the yaw dynamics of the vehicle a robust steering controller known as the disturbance observer [2] is added. This two degree of freedom control architecture is used to improve vehicle handling and to achieve better disturbance rejection.

The controller synthesis is based on the equation (3) which describes the yaw dynamics of the vehicle model [2].

$$r = G\delta_V + d \quad (3)$$

G is the transfer function between steering angle δ_V at the front wheels and the yaw rate r . The external disturbances are d . In equation (4) an adopted nominal model G_N and a multiplicative model uncertainty Δ_M are used for description of G .

$$r = (G_N(1 + \Delta_M))\delta_V + d \quad (4)$$

The aim of the controller is to obtain the transfer function in (5) despite model uncertainty Δ_M and external disturbance d (δ_L is the steering wheel angle).

$$\frac{r}{\delta_L} = G_N \quad (5)$$

External disturbance and model uncertainty are treated as an extended disturbance e (eq. (6) and (7)).

$$r = G_N\delta_V + (G_N\Delta_M\delta_V + d) = G_N\delta_V + e \quad (6)$$

$$e = r - G_N\delta_V \quad (7)$$

The front steering angle δ_V is set according to (8).

$$\delta_V = \delta_L + \delta_C \quad (8)$$

$$\delta_C = -G_A \frac{e}{G_N} = G_A \left(\delta_V - \frac{r}{G_N} \right) \quad (9)$$

The additive steering angle δ_C provided by the VDC block is the output of the steering actuator G_A (9). Eq. (5) is approximated best with an ideal actuator ($G_A \rightarrow 1$). For implementation, the feedback signals r and δ_V are lowpass filtered to limit the controller to low and medium frequency domain. The relative degree of the low pass filter Q is chosen to be at least equal to the relative degree of G_N for causality of Q/G_N . The filter Q is chosen according to (10)

$$Q = \frac{1}{\tau_Q s + 1} \quad (10)$$

The structure of the controller according to equation (11) is shown in Fig. 10.

$$\delta_V = \delta_L + G_A \left(Q \delta_V - \frac{Q}{G_N} r \right) \quad (11)$$

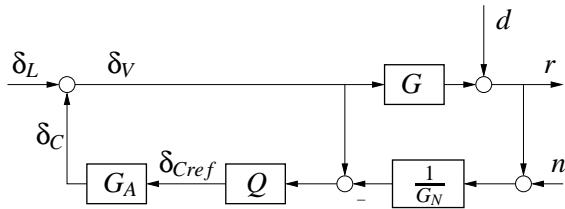


Figure 10: structure of the *Disturbance Observer*

The transfer function is given in eq. (12).

$$\frac{r}{\delta_L} = \frac{G_N G}{G_N(1 - G_A Q) + G_A G Q} \quad (12)$$

Here, as nominal model, the dynamics of the *single track model* is implemented. The virtue of this controller is described in detail in [2]. For physical implementation "additional steering" is assumed, i. e. mechanical superposition of the steering wheel angle δ_L and the output of the actuator. In the model of the actively steered car (1), the controller (11) is implemented in the *VDC* block. A simple actuator model is implemented as part of the *mechanical steering angle addition* block.

8 Comparing maneuvers with active car steering to conventional conducted maneuvers

Finally, the car model with the active steering controller is compared with the conventional car. Therefore, the simulation results of four maneuvers are discussed. Both the conventional and the controlled car are displayed.

Maneuver: braking in a curve. This maneuver is known from section 6. The *VDC* block provides a steering angle δ_C which is added to the steering wheel angle δ_L . As shown in Fig. 11 the additive steering angle first raises because the nominal model is valid for the linear operating range of the tire characteristics, whereas in the simulation the lateral wheel forces are already close to their saturation. Also the yaw rate

r and the lateral acceleration a_y are shown. When the braking causes a disturbing yaw moment the additive steering angle is reduced to compensate for this oversteering. The cause of the temporary oscillations of a_y in Fig. 11 seems to be due to a (yet unclear) imperfection of the car model.

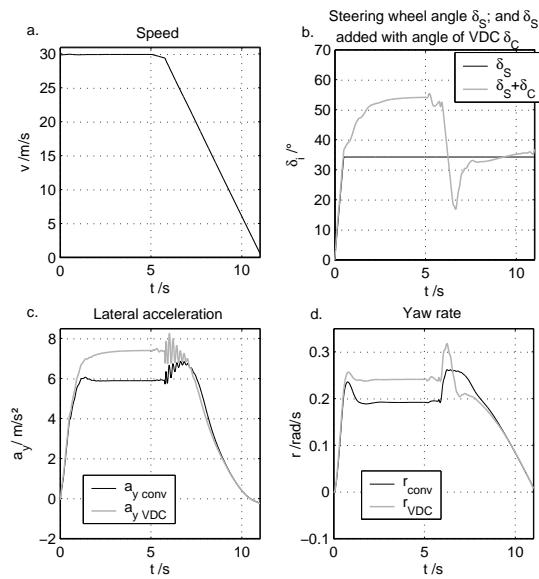
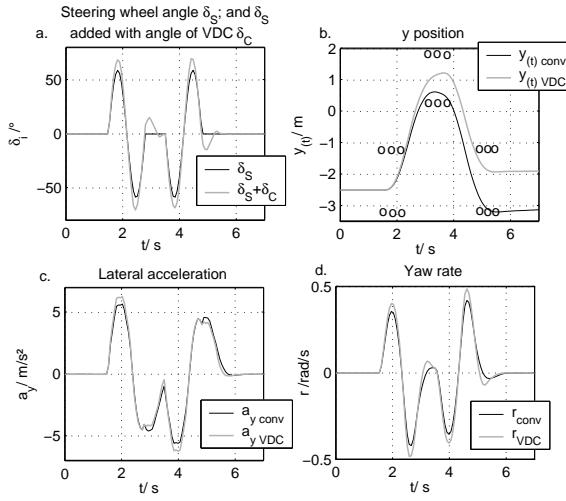
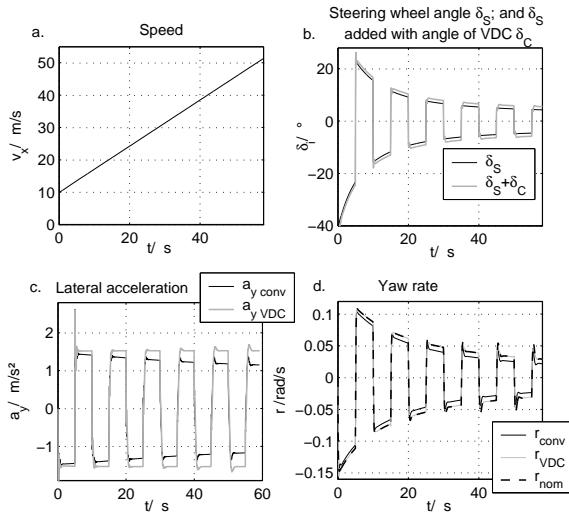


Figure 11: Comparing maneuver *braking in curve*

Maneuver: double lane change. Maneuver *double lane change* is used for assessment of car dynamics in research and development of both vehicles and control systems for vehicle dynamics. The speed of the vehicle is kept constant during the whole maneuver ($v_x = 30\text{m/s}$). Resulting from one period of a sinusoidal steering angle input the vehicle completes a single lane change. The lane change back is caused by a corresponding steering input in the opposite direction. For the assessment of the vehicle model of the MOD-ELICA library the amplitude of the sinusoidal steering angle and the time between the two sinusoidal signals are adapted until the course of the vehicle (y-position in Fig. 12 b.) fulfills the requirements of the standardized *double lane change* (according to ISO 3888; the boundaries of this course are marked in Fig. 12 b.). This maneuver is performed as an *open loop* maneuver, i. e. the drivers steering wheel input is not affected by the course of the vehicle. For a better agreement with reality, the *maneuver control* block needs to be enhanced by a more sophisticated driver model in the future. Nevertheless, in Fig. 12 the stability enhancing effect of the controller can be recognized.

Figure 12: Comparing maneuver *double lane change*

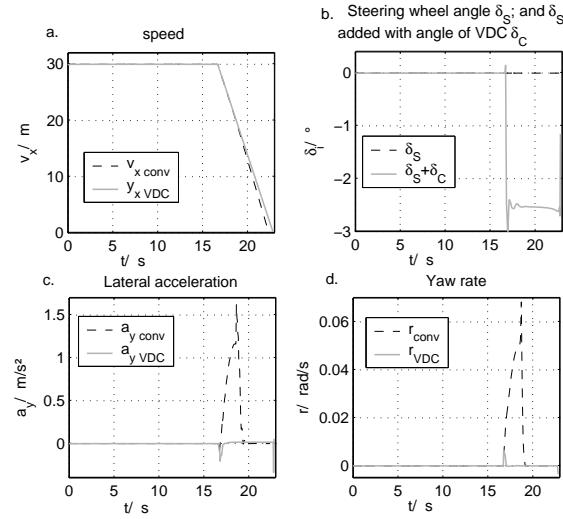
Maneuver: sequence of alternating steering wheel angle steps. This maneuver shows how the controller affects the steering transfer function of the vehicle over the entire speed operating domain. The speed is slowly but continuously increased. Apart from that the simulation is executed similarly as described in section 6. Fig. 13 shows the results of the maneuver

Figure 13: Comparing maneuver *sequence of alternating steering angle steps*

ver simulations. The controller's aim is to make the yaw rate close to the nominal model despite of disturbances.

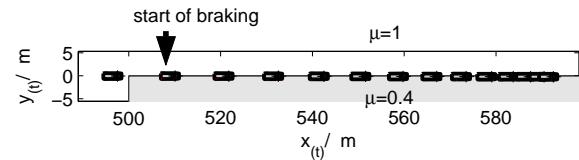
Maneuver: μ -split braking. This maneuver is already known from section 6.

Fig. 14 shows the results of the maneuver simulations.

Figure 14: Comparing maneuver *μ -split braking*

For better clearness the lines of the conventional vehicle are printed dashed.

When braking is applied with the controlled car an additional steering angle δ_C (Fig. 14 b.) compensates for the increasing yaw rate (Fig. 14 c. and d.). This can also be seen in the stroboscopic diagram in Fig. 15. Compared to the conventional μ -split braking (Fig. 9), the distinct stability improvement is obvious. A small

Figure 15: Stroboscopic recorded course of the maneuver *μ -split braking* with active steering controller. (Compare to Fig. 9)

divergence is still present.

9 Experiences with vehicle dynamics library

From a user's point of view, the general advantage of working with a vehicle dynamics model based on MODELICA is its transparency and, as a matter of course, the feasibility of multidisciplinary modelling. Due to the component oriented philosophy, user-specific enhancements to a car model taken from the vehicle dynamics library may easily be accomplished. Our specific comments refer to an unofficial pre-release version of vehicle dynamics library [1],

and particularly to the *chassis level 2*. Therefore, our records may not be applicable to the consecutively released versions. As far as our experiences with the vehicle dynamics library on the basis of the investigated maneuvers are concerned, the simulation results are commensurate with a typical mid-size passenger car. The performance of the simulated vehicle appears to be plausible and realistic but two exceptions which are reported below. Firstly, during maneuvers where the lateral vehicle dynamics is explicitly excited (e.g. braking in a curve and alternating steering wheel steps) poorly damped oscillations at 4Hz of the lateral acceleration occur at all speeds (see Figs. 7,11,13). We act on the assumption that this effect is not realistic and the model should be reviewed in this regard. Secondly, a strange phenomenon appears during the μ -split braking maneuver. In the period between entering the low friction lane (*low μ*) and the start of the full braking a remarkable yaw disturbance torque is generated which at first make the vehicle turn towards the *low- μ* lane. This effect may be explained by the reduction of the lateral force which is due to the toe-in angle of the front tire on the *low- μ* side. However, the effect of this fact seems to be much too excessive compared to reality. We guess that checking the tire model will solve this problem.

Real-Time Computer Systems (RCS) of the Technical University Munich (TUM) for their scientific support.

References

- [1] J. Andreasson, “Vehicle dynamics library.” 3rd International Modelica Conference, Linköping, 2003.
- [2] T. Bünte, D. Odenthal, B. Aksun Güvenc, and L. Güvenc, “Robust vehicle steering control design based on the disturbance observer,” *Annual Reviews in Control*, vol. 26, pp. 139–149, 2002.
- [3] P. Riekert and T. Schunck, “Zur Fahrmechanik des gummibereiften Kraftfahrzeugs,” *Ingenieur Archiv*, vol. 11, pp. 210–224, 1940.
- [4] S. Beiker, “Verbesserungsmöglichkeiten des Fahrverhaltens von Pkw durch zusammenwirkende Regelsysteme.” Fortschritt-Bericht, 2000.

10 Conclusions

The vehicle dynamics library was assessed and exploited in this paper. The single track parameters for the vehicle model of the library were identified. With these parameters feedforward control for the setup of various driving maneuvers was implemented. The maneuvers were conducted by providing steering angle and gas/brake. Also feedback control was implemented for this maneuvers. A robust steering controller for active car steering was introduced and implemented. The stability enhancement concerning the yaw dynamics of the vehicle was shown by execution of significant maneuvers. The model with active car steering controller was compared with the conventional model by means of these maneuvers.

11 Acknowledgement

The authors want to thank Univ.-Prof. Georg Färber and Dipl.-Ing. Philipp Harms from the Institute for

Real-time Simulation of Detailed Automotive Models

Hilding Elmquist¹, Sven Erik Mattsson¹, Hans Olsson¹,
Johan Andreasson², Martin Otter³, Christian Schweiger³, Dag Brück¹

¹Dynasim AB, Research Park Ideon, 223 70 Lund, Sweden, <http://www.dynasim.se/>,
{Hilding.Elmqvist, SvenErik.Mattsson, Hans.Olsson, Dag.Bruck}@Dynasim.se

²KTH Vehicle Dynamics, 100 44 Stockholm, Sweden
<http://www.ave.kth.se/>, Johan@fkt.kth.se

³German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Oberpfaffenhofen,
82234 Weßling, Germany,
<http://www.robotic.dlr.de/control/>, {Martin.Otter, Christian.Schweiger}@dlr.de

Abstract

This paper describes typical modeling and real-time simulation issues that occur in automotive applications. Real-time simulations of detailed Modelica benchmark models for chassis and powertrain are presented. They demonstrate the powerful real-time capabilities of Dymola and the Modelica modeling language. One of the benchmark models for vehicle dynamics is a detailed model with 72 degrees-of-freedom with bushings in both the front and rear wheel suspensions. It was simulated in real-time with a sample rate of 500 Hz on the RT-LAB environment from OPAL-RT using a Pentium 4, 3066 MHz processor. This is made possible by Dymola's unique and elaborate symbolic processing of the model equations.

1 Introduction

Hardware-in-the-loop simulation (HILS) has become common practice in automotive development. In order to cope with the real-time constraints, only rough models are often used. In this paper, we present means to symbolically manipulate models with a high level of detail in such a way that the simulation can be performed in real-time. The effectiveness is demonstrated by several benchmark examples and by corresponding simulation results.

The methods are implemented in the simulation environment Dymola [3, 4] that uses the Modelica [7] modeling language for describing the models. It is described how Dymola solves certain difficult problems in hardware-in-the-loop simulation of automotive systems. Two types of benchmark

models have been chosen to demonstrate the capabilities of Dymola: a transmission model and a set of vehicle dynamics models.

A transmission gearbox is somewhat special because the connection structure changes due to the engagement of clutches and brakes. Furthermore, effective inertias need to be calculated for each of the possible structures. Dymola handles this by appropriate preparation of the equations by symbolic methods before generating the code for the target HILS system.

Vehicle models of different complexities can be used for analysis. Traditionally, idealized models of wheel suspensions have been used, neglecting fast dynamics due to bushings and replacing them with ideal joints or just look-up tables. Dymola has special numeric methods to handle such cases. These methods require elaborate symbolic preprocessing of the equations. One of the benchmark models has 72 degrees-of-freedom with bushings in both the front and rear wheel suspensions. It was simulated in real-time with a sample rate of 500 Hz.

Dymola generates C code which can be used in Simulink and by use of RealTime Workshop downloaded to different HILS targets. Evaluation of the benchmark problems has been made on RT-LAB from OPAL-RT [8], demonstrating real-time performance of complex models.

2 Power train simulation

We will consider modeling and simulation of automatic gearboxes. The figure below shows a typical Modelica model of a gearbox (Lepelletier wheelset, 6-speed, from the commercial Modelica PowerTrain library [10] available from Dynasim; usable, e.g., for the automatic gear box ZF 6 HP 26

from ZF). The model includes planetary and Ravigneaux gear sets, clutches, brakes and inertias.

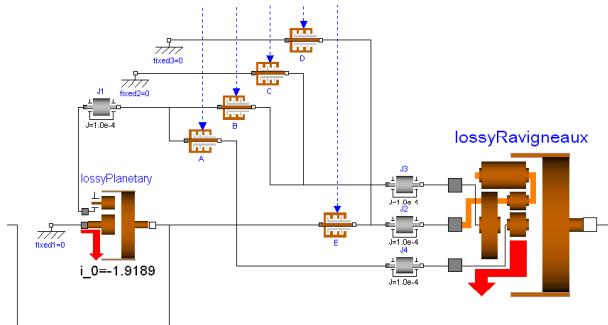


Figure 1: Gearbox model

2.1 Special problems

Simulation of gearbox models in real-time poses special problems. If detailed models of the friction of the clutches and brakes are used, the models become stiff. Typically, ideal friction models are used instead. This means that the number of degrees-of-freedom (DOF) changes if a clutch or brake is stuck or not. This can be handled by constraining the relative acceleration, when in stuck mode, to be zero.

Fast sampling

The differential equations of the gearbox need to be solved at a high speed. The electronic control unit (ECU) for the transmission typically samples its inputs and calculates new control signals every 10 milliseconds. In order to reduce effects of delays due to lack of synchronization, the model variables need to be determined every millisecond.

Accuracy and discontinuities

Special attention is needed to accurately calculate angular velocity. This is important because the angular velocities of the various wheel sets are typically output from the model to the hardware and input to the ECU. The control algorithm of the ECU acts differently when the angular velocity is close to zero. Thus it is important to calculate small velocities accurately. Another reason to achieve high accuracy is that one might otherwise get drift in the angle calculations. The difficulty in achieving high accuracy in the angular velocities close to zero is the highly nonlinear behavior when a clutch sticks. The torque of the clutch in sliding mode is calculated as a function of angular velocity. When the clutch sticks, the constraining torque is instead calculated in such a way that the

relative angular acceleration stays zero. There are thus jumps in the relative angular acceleration.

Event handling

Integration algorithms for non-real-time simulation typically handle discontinuities, such as the one above for friction, by detecting when certain variables cross a boundary. They then calculate the time of the event by iteration and then change the step size to advance the time exactly to the time of the event (crossing). Also for real-time applications, the Dymola run-time system includes handling of calculation of the event time. This is done with little overhead and without iteration. The normal solving of the differential equations is for the real-time case performed with fixed step size. However, at an event the step size is decreased to hit the time of the event. In order to synchronize with real-time again, the size of the next step is increased such as the sum of the two steps around the event is equal to two normal steps. This procedure introduces a small synchronization error during one step, but gives better accuracy in the solution. It has successfully been utilized for gearbox HIL simulations for ECU testing.

Event propagation

After an event, for example if a clutch begins to slide, there might be an immediate event as a consequence. Another clutch might get stuck because its torque decreases below a certain threshold. Before a numerical solution of the differential equations is resumed, event propagation needs to be performed in order that all variables get consistent values. Dymola generates code for iterating the equations, called event iteration, until all Boolean mode variables have converged. This typically takes 1-3 extra evaluations of the equations, i.e., the calculation time to handle such an event might exceed the available time for the step. This is typically handled by configuring the HILS system to allow a certain number of overruns.

Effective inertia calculation

The effective inertias depend on the selected gear. Calculation of effective inertias shows up as systems of equations that need to be solved simultaneously.

Dymola symbolically converts the differential and algebraic equations (DAE) to an algorithm for calculating the derivatives. The integration algorithm uses the derivatives to update the state variables. Many times, the derivative algorithm is

just a sequence of assignment statements for algebraic variables and derivatives. However, the conditional constraint equations for torques and accelerations in the clutch and brake models implies that, in order to solve for the accelerations, a system of simultaneous equations needs to be solved. Dymola automatically calculates the coefficients of the linear system of equations and invokes a numerical solver for larger systems of equations. Small systems of equations are solved by producing symbolic code. The effective inertia typically shows up as the determinant of such a coefficient matrix. It should be noted that this is not a domain-specific procedure, but Dymola does it automatically by solving the systems of equations.

Underdetermined models

In certain cases, several clutches are engaged, giving parallel paths for the power. In such cases, the torque at each clutch cannot be determined individually; only the sum can be determined. Mathematically, this shows up as a singular system of equations. However, it is possible to find consistent solutions. Dymola determines one such consistent solution.

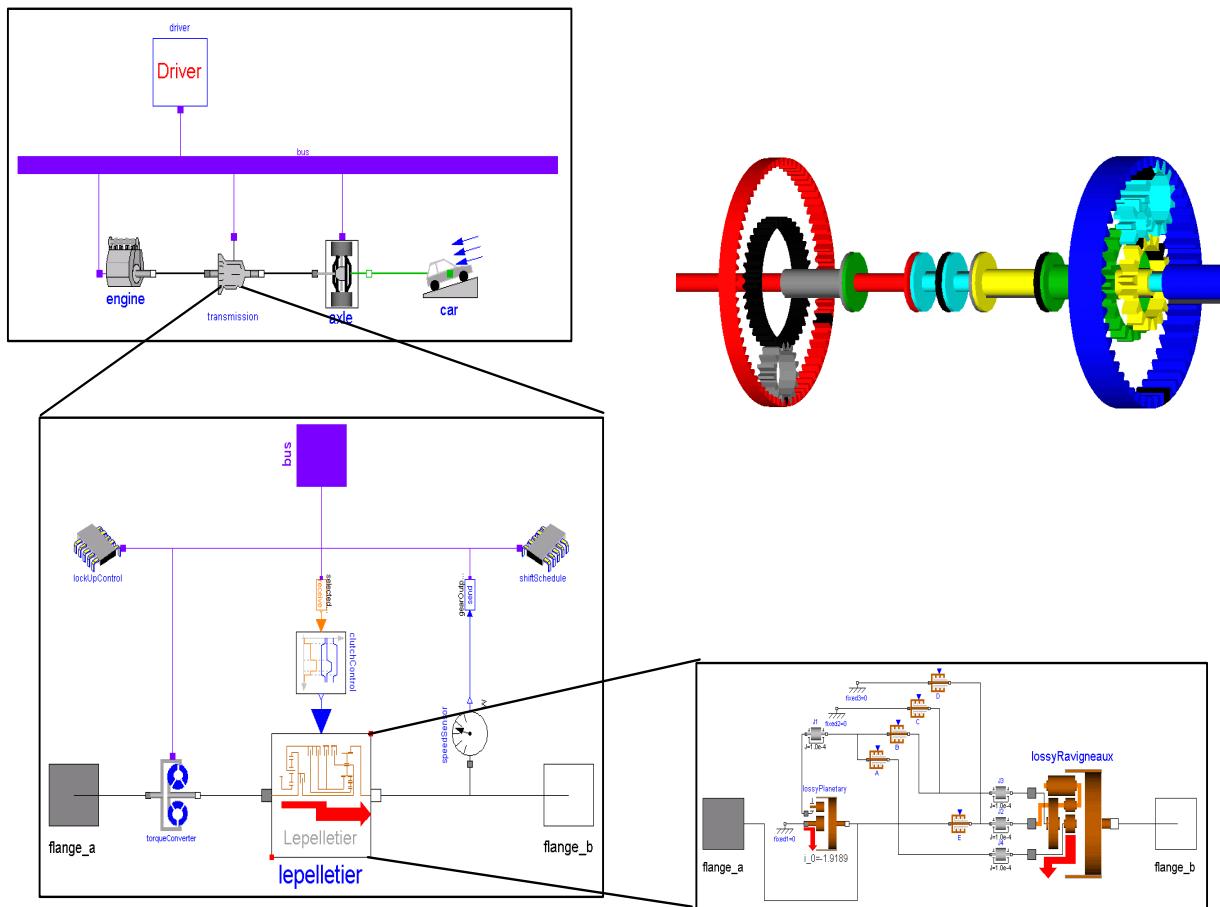


Figure 2: The transmission example with the gearbox model and its animation

2.2 Transmission example

As a benchmark example, we will consider modeling of a 6 speed gearbox (Lepelletier wheelset, e.g. ZF 6 HP 26) together with a simple vehicle and driver model. This model is suitable for carrying out driving cycle shift strategy analysis and is available in the Powertrain library. The hierarchical structure of the model and the 3D representation used for animation is shown in the picture below.

The engine model is based on steady-state engine maps. The ECU function included in this model controls idle and maximum speed, both constant limits, by a proportional controller. The transmission is a detailed model of an automatic transmission and incorporates a torque converter with a lock-up clutch. The gearbox itself is of Lepelletier type, which provides six different gear ratios. It is modeled using basic gearbox elements, inertia elements and different clutches and brakes. The different gear ratios are a result of applying different pressures to the clutches and the brakes in order to engage or disengage them.

The driveline model is essentially a rigid model with no compliance in the drive shafts and no tire-slip modeling. The vehicle is in this example modeled as a lumped mass and the resistance forces associated with the vehicle are modeled as different physical effects. The control system determines the shift point based on throttle position and vehicle speed when compared to the defined shift map. The driver model is based on a PI controller.

The model has 689 nontrivial equations and 15 state variables. There is a linear system of 77 simultaneous equations corresponding to the mass matrix inversion. After evaluating all parameter values and simplifying, the system reduces to 50 simultaneous equations. Symbolic manipulation reduces the size of the linear system of equations that has to be solved numerically to 7. The model was simulated with the explicit Euler method with a step size of 1 ms. As shown, the car follows the desired velocity very well.

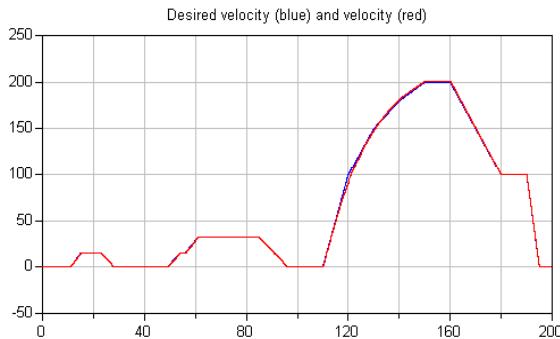


Figure 3: Desired velocity (blue) velocity (red)

The results are shown with a comparison to offline simulation using DASSL with a required relative tolerance of 10^{-6} . The difference is as shown below very small.

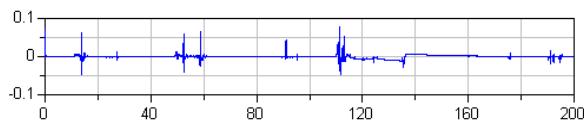


Figure 4: Velocity error (Explicit Euler – DASSL)

The gearshift is identical for explicit Euler and DASSL.

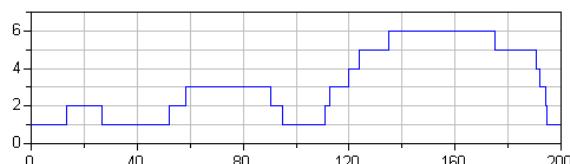


Figure 5: Gear shift

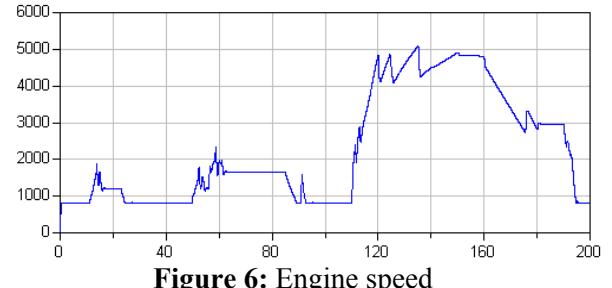


Figure 6: Engine speed

Also for engine speed, the agreement with the DASSL result is good.

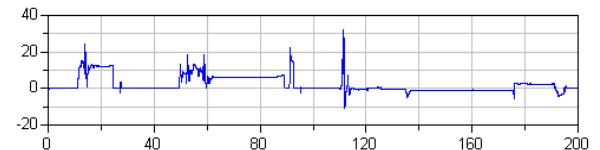


Figure 7: Engine speed error (explicit Euler – DASSL)

Real-time simulation

The benchmark model was run in the RT-LAB environment from OPAL-RT using a Pentium 4, 3066 MHz processor. The plot below shows the actual CPU time needed per step.

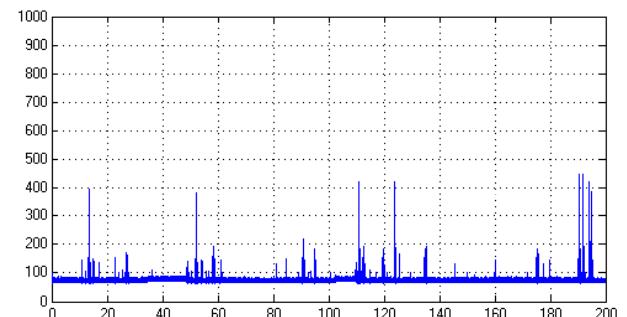


Figure 8: CPU time/step (microseconds)

The plot shows that the simulation runs in real time, because the time needed for each step is well below 1 ms. The CPU time needed per step is not constant, because of event handling due to locking or unlocking of clutches or brakes during gear shifting. Moreover, the linear system of size 7 being solved numerically has a coefficient matrix or a Jacobian, which does not depend on any continuous time variables, it changes only when there are discrete events. Its elements are in fact weighted sums of terms of the type

```
if axle.Break.locked then 1 else 0;
if transmission.wheelset_E.locked
then 0 else 1;
```

Dymola exploits the fact that the Jacobian does not change during continuous time simulation. It

generates simulation code that only calculates the Jacobian and its LU-factorization during event iterations. This saves CPU time because the QR factorization is a major effort compared to the back substitution. The number of operations to factorize is proportional to the cube of the number of unknowns, i.e., $O(n^3)$, where n is the number of unknowns, which in this case is seven. Back substitution to calculate the solution when having the factorized Jacobian is much less computationally demanding.

To illustrate the importance of symbolic manipulation, a test was done where Dymola did not reduce the original system of 77 equations, but utilized that the Jacobian of the system only changed at discrete events. The plot below shows the actual CPU time needed per step.

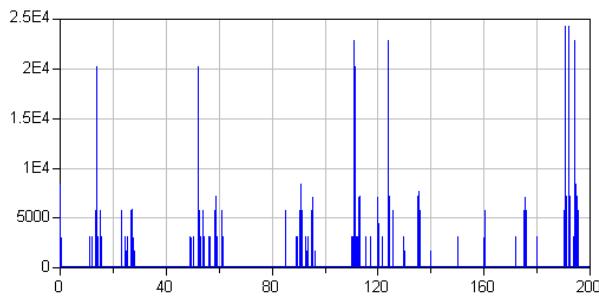


Figure 9: CPU time/step (microseconds) for the non-reduced case.

The plot shows that the CPU time needed per step varies a lot. This simulation does not run in real time. At certain steps the CPU time is nearly 25 ms. Much CPU time is needed, when there are discrete events and the Jacobian of the linear system with 77 unknowns needs to be calculated and LU-factorized. During continuous time simulation, the linear system is solved using the factorized Jacobian for back substitution, which is as shown a fast calculation.

3 Vehicle Dynamics Simulation

The free Modelica library VehicleDynamics [1] is used as basis for the evaluations in this report. This library is based on the multibody systems library ModelicaAdditions.MultiBody. The library is flexible since it is easy to replace wheel suspensions, tire models, etc. In particular, wheel suspensions are available with different levels of detail.

3.1 Special problems

Symbolic simplifications

Symbolic simplifications are very important for handling of multibody systems models. The model equations are written in the most general form. However, a motion could, for example, be constrained to be a rotation around a certain axis (e.g. {1,0,0}) in a local coordinate system. Parameters that are exactly zero are important to utilize symbolically; certain terms in the general model equations are cancelled and thus better efficiency can be achieved. The number of operations in the generated code is typically reduced by a factor of 3 to 10.

Mass matrix inversion

The differential-algebraic equations for a multibody system have a special structure. For a tree-structured mechanism, a large system of simultaneous equations involving accelerations, forces and torques will be present. It is important that such systems can be identified and reduced in size. It can typically be reduced in size to the number of degrees-of-freedom. This corresponds to finding the mass matrix of the mechanism.

Closed kinematic loops

Closed kinematic loops typically occur in suspensions with ideal joints. In such cases, the equations contain a nonlinear system of equations for each loop involving positions and orientations of the parts belonging to the loop. A linear system of equations involving velocities also appears. On acceleration level, equations from each loop appear in one large system of equations (corresponding to the mass matrix for tree-structured mechanisms accompanied with the constraint equations on acceleration level).

The non-linear system of equations is special in the sense that it involves trigonometric relations. It turns out that analytical solutions can be found [9]. The multibody library has been extended with composite joint models, for which the equations have been rewritten to give the analytical solution for a large class of kinematic loops occurring in vehicles and mechanisms.

Stiff models – Bushings

High fidelity models use bushing models instead of ideal joints. Such bushings are very stiff. This means that the differential equations are also stiff, i.e., that the corresponding linearized model has

eigenvalues in a large range. The explicit Euler method is not feasible for these models since a very small step size needs to be utilized (typically less than 50 microseconds). Implicit Euler allows a larger step size, but the accuracy is often not good enough. If neither the explicit nor the implicit Euler method is satisfactory, Dymola can utilize methods with higher order or mixed explicit/implicit methods for such models.

Tire models

The VehicleDynamics library [1, 2] contains two types of tire models: the standard tire model of Pacejka and the tire model of Rill. The Rill tire model is about 1 to 2 orders of magnitudes faster than the Pacejka tire model and should therefore be used when speed is important, such as for real-time simulation. The Rill tire model is based on the steady-state force/torque characteristics of a tire together with a simple transient tire deflection model.

3.2 Realtime Simulation Benchmarks

A mid-sized sedan with a front MacPherson suspension and a rear MultiLink suspension has been chosen as a benchmark model for vehicle dynamics simulations.

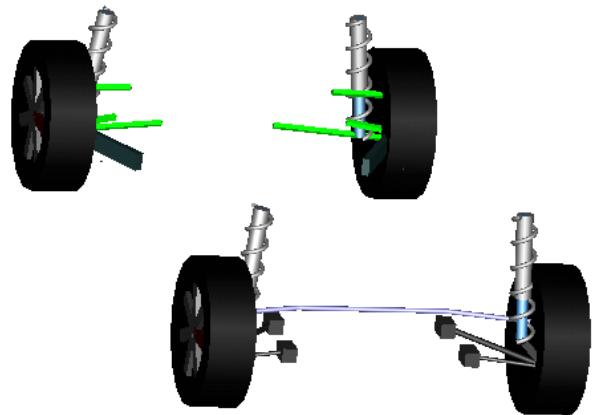


Figure 10: Front MacPherson suspension and rear MultiLink suspension.

The hierarchical structure of the vehicle models is shown in Figure 11.

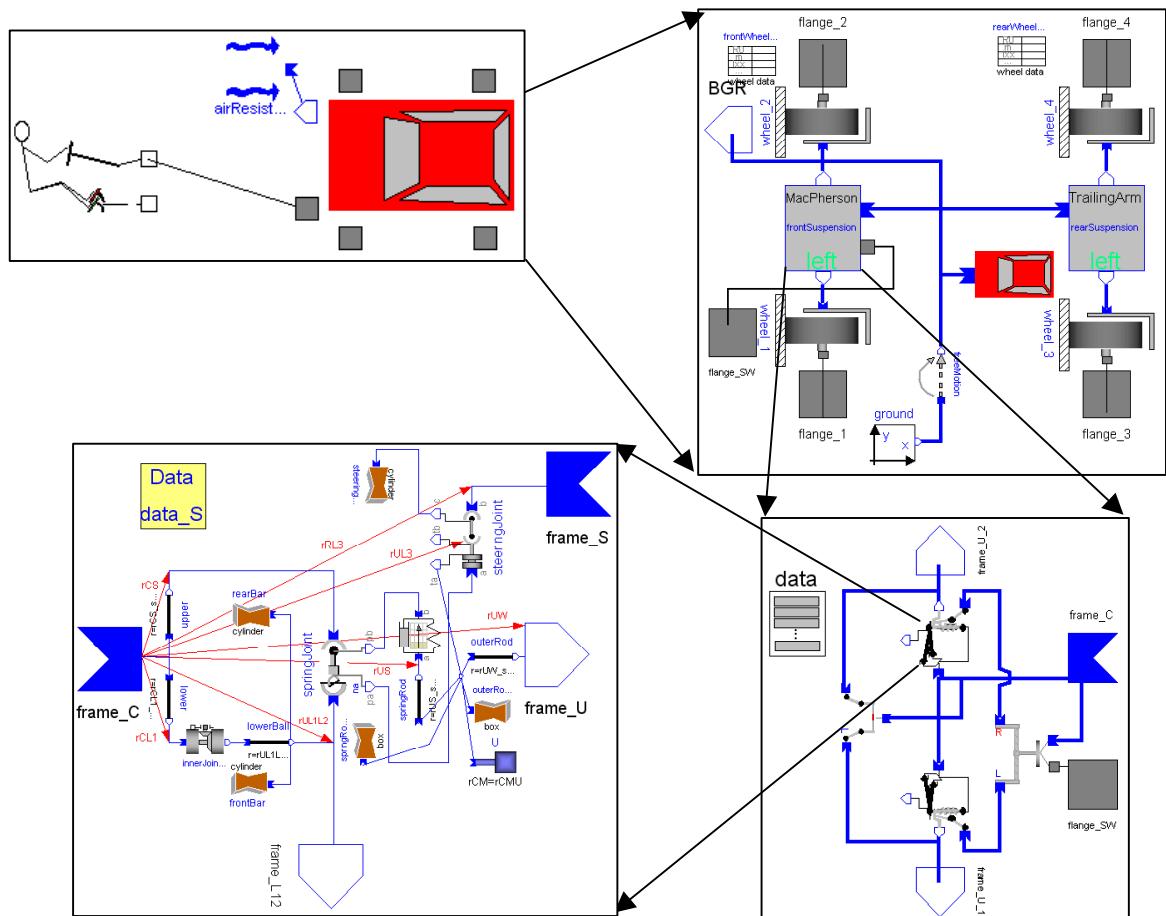


Figure 11: The hierarchical structure of the vehicle models

We have investigated models with different levels of detail.

1. Suspension is modeled by tables defining polynomials for Camber and toe-in angles. Steering is defined by an Ackermann function.
2. Suspension is modeled by linkages with ideal joints.
3. Suspension is modeled by linkages joined by bushings. The mass and inertia of the bar connecting two bushings are neglected.
4. Suspension is modeled by linkages joined by bushings where the small mass and inertia of the bar connecting two bushings are taken into account.

Level 1 – Linkage tables

The wheel suspensions are described by tables defining Camber and toe-in angles as functions of wheel bounce, i.e., a vertical motion of the wheel with constrained changes of the Camber and toe-in angles. This could easily be extended to handle also Camber and toe-in as functions of side force, which would make it possible to mimic the behaviour of suspensions with bushings and other flexible elements. This has been the common way to model vehicle dynamics in order to keep model complexity low for realtime simulation. Note, this method requires that the characteristics must either be measured, meaning that the suspension has to be built, or that the suspension characteristics have to be calculated from a more detailed model. This approach is justified if the simulation model is utilized, e.g., for improving controllers and ECUs for an existing vehicle. It is not useful, if the suspension and steering system shall be improved, e.g., based on optimization or parameter studies of a simulation model.

Steering is defined by an Ackermann function. The tables for Camber and toe-in angles are implemented as scaled polynomials. Dymola's symbolic engine differentiates these polynomials twice to handle the reduction of degrees-of-freedom.

The chassis has 6 degrees-of-freedom (DOFs), each wheel has 2 DOFs (bounce and rotation each) and the steering 1 DOF. The total DOF is 15. The tires each have 2 state variables for the deflection in x and y directions, i.e., $4*2 = 8$ states. The total number of states for the vehicle dynamics itself is thus $2*15 + 8 = 38$.

The steering in the benchmark model is a parameterized, given function which is filtered by a second order low pass filter to model driving behavior. The driver model of the benchmark

model contains two additional state variables for the accelerator behavior. This is not used in this model since the vehicle maneuver is made with idle gear. The total number of state variables is thus $38 + 2 = 40$.

Level 2 – Linkage with ideal joints

The table description used in level 1 is limited to only Camber and toe-in angles. It would of course be possible to extend to Castor angle trail as well as track width and wheel base translations. However, in many cases, in particular when trying new designs, it's easier to describe the suspension in terms of the linkage that is used.

The suspensions in level 2 consist of rigid mechanical components, i.e., all flexible elements, except for the struts, are replaced by ideal joints. Instead of a multi-link suspension, a trailing arm with similar geometry is used. The advantage over level 1 is that the suspension can be modelled with physical data and no precalculations or measurements are therefore needed.

The level 2 model uses a MacPherson type front wheel suspension, with the wishbone attached to the chassis via an ideal revolute joint (1 DOF). A strut is placed between the chassis and the wishbone via two spherical joints. The eigenrotation of the strut around its axis (1 DOF) is constrained by the distance constraint of an additional rod with two spherical joints on each end (1 constraint). One of the spherical joints of this rod is attached to the steering. In total, the suspension has therefore one degree of freedom, if the steering angle is given. The anti-roll bar is approximated by a spring/damper combination where the vertical force acting at its mount point on the lower part of the MacPherson strut is proportional to the relative vertical distance of the left and the right mount points. The rear suspension is a type of trailing arm with one DOF, the anti-roll bar is modeled like in the front suspension.

When using base elements of the MultiBody library to build up the MacPherson suspension, several non-linear algebraic loops appear. By using composite joint models (e.g., an aggregation of a revolute, a spherical and a universal joint) that contain analytic solutions of the non-linear kinematic relationships within the aggregation, the non-linear algebraic loops no longer occur in the generated code [9]. Note that this simplification is transparent to the end user.

The total DOF is 15 as for the level 1 model; The wheel bounce DOFs are replaced by the DOFs of the two trailing arm rotations and the two

wishbone rotations. The model has also 40 states. Note, that the elasticity of the tires in vertical direction has been modified slightly (both for the level 1 and the level 2 cars) in order to approximately compensate for the neglected bushings.

Level 3 – Linkage with bushings and massless bars

Using ideal joint models for the linkage is not accurate enough for severe driving conditions since bushings with certain flexibility are used in the real vehicle. Flexible elements are introduced in the suspensions of the level 3 model. The front suspension has bushings in the A-arm mounts. The rear multilink suspension has no ideal joints and the links are modelled as mass-less bars. If the mass and inertia of the rod connecting two bushings were not neglected 6 DOF would be added for every such pushrod. However, the mass and inertia are usually very small compared to the wheel and carrier masses, and therefore it is a good approximation to neglect the pushrod masses and inertias.

If the bushings were described solely by springs, then no states would be added, since springs in series connection lead to algebraic equations to solve for the spring deflections. Since bushings have a damping part, there are the states of the dampers ($= 2*6$). Once the states of one damper are known, the states of the other damper can be computed by relative kinematics. To summarise, a pushrod has 6 states, if the mass and inertia of the rod connecting the two bushings is neglected. There are 3 such bushing pairs at each rear wheel, i.e. the number of states is $2*3*6 = 36$ states.

Additionally, the elasticity in the steering is taken into account by having a spring/damper system in the rack steering adding one additional DOF.

The total DOF is 36 and the model has 118 states.

Level 4 – Linkage with bushings and non-massless bars

A slightly more detailed model is obtained by not neglecting the masses of the push rods. The total DOF is 72 and the model has $2*72 + 8 + 2 = 154$ states.

3.3 Simulation results

The benchmark models have been studied under a double lane change maneuver. The steering wheel has been operated as shown in Figure 12.

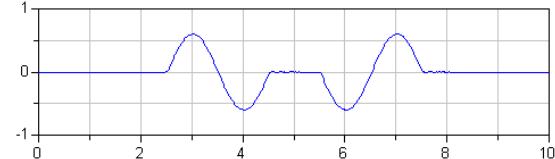


Figure 12: Steering wheel angle (rad)

We first show a comparison of the behavior of the four models. Below are shown plots of the side accelerations for the four cases.

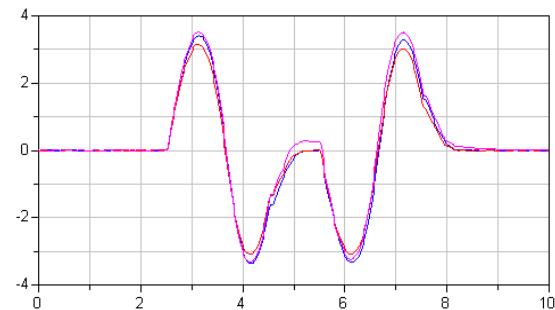


Figure 13: Side accelerations for level 1-4 models.

The level 3 and 4 models show a different behaviour than level 1 and 2. The differences can be spotted especially in the section between the lane changes: While the level 1 and 2 cars reach zero yaw and lateral acceleration, level 3 and 4 are too slow to get back to zero before the second lane change is started. This is essentially because of the elasticity in the suspensions. The level 1 and 2 models behave very similar. The tables used in level 1 were generated from suspensions close to those used in level 2. The behaviour of the level 3 and 4 models is practically identical. The oscillations of the links with small masses have very little effect on the deformation of the bushings that carry the wheel.

Real-time simulation

Let us discuss the problems of using these four models for real-time simulation.

It is possible to use explicit Euler with a step-size of 1 ms for the models of level 1 and 2. Comparisons with results from offline simulation with DASSL (relative tolerance= 10^{-6}) show that the error in side acceleration is less than 0.25%. The major task when using the explicit Euler method is the calculation of the derivatives. Each of the level 1 model and the level 2 model has a

linear system of simultaneous equations corresponding to the mass matrix inversion. Dymola's symbolic processing reduces this system of equations to a system of about 10 equations. There are no nonlinear systems of equations, because the equations for the closed kinematics loops of level 2 have been solved analytically in the model library. The RT-LAB environment from OPAL-RT using a Pentium 4, 3066 MHz processor runs these two models easily in real-time, because it needs only 0.1 ms for an Euler step for the level 1 model and 0.3 ms for the level 2 model.

It is not possible to use explicit Euler to simulate the level 3 model or the level 4 model, because these models use bushing models instead of ideal joints. The bushings introduce very fast modes. Explicit Euler requires the step size to be smaller than the shortest time constant utilized (typically less than 50 microseconds). Typically, the fastest modes are not excited to a degree that it is necessary to resolve them for the intended purpose. In such cases the problem is referred as stiff. The *implicit* Euler method solves the numerical *stability* problem and allows larger step sizes to be used. It is the *accuracy* required that restricts how large step sizes can be used. Using the implicit Euler method, on the other hand, implies that a nonlinear system of equations needs to be solved at every step. The size of this system is at least as large as the size of the state vector, n. Solving large nonlinear systems of equations in real-time is somewhat problematic because the number of operations is $O(n^3)$ and the number of iterations might vary for different steps. Reducing the size of the nonlinear problem is advantageous. The method of inline integration [5, 6] was introduced to handle such cases. The discretization formulas of the integration method are combined with the model equations and structural analysis and computer algebra methods are applied on the augmented system of equations. Implicit Euler allows larger step size, but the accuracy is often not good enough. If neither the explicit nor the implicit Euler method is satisfactory, Dymola utilizes methods with higher order or mixed explicit/implicit methods for such models.

Each of the level 3 model and the level 4 model has a linear system of simultaneous equations corresponding to the mass matrix inversion. Dymola's symbolic processing reduces this system of equations to a system of about 20 equations.

The level 3 model and the level 4 model have been simulated with a special inline mixed explicit/implicit method, developed by Dynasim. This results in a nonlinear system of equations. For

the level 3 model the size is about 130 and for the level 4 model the size is about 80. The level 4 model has 154 state variables. The large possible reduction of the size of the implicit non-linear system of equations from 154 to about 80 is due to the fact that certain subsystems are linear even after amendment of the corresponding discretization formulas. Dymola automatically detects such structures during the structural analysis of the equations. The remaining nonlinear system of equations has to be solved by a Newton method; 2-3 iterations are typically needed, i.e. 3-4 residual calculations need to be performed. The step size was chosen to 2 ms. Comparisons with results from offline simulation with DASSL (relative tolerance=10⁻⁶) show that the error in side acceleration is less than 0.5%.

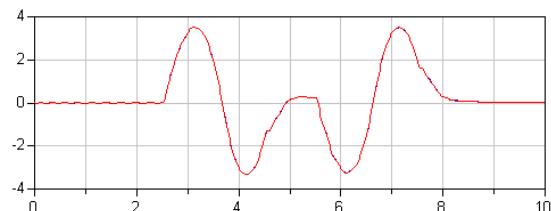


Figure 14: Side accelerations for the level 4 model

The difference between the results of the implicit method and DASSL is less than 0.5%

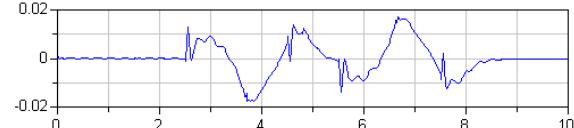


Figure 15: Side acceleration errors for the level 4 model (Euler – DASSL)

The realtime benchmarks were run on a computer equipped with a Pentium 4 processor running at 3066 MHz and a 333 MHz single-channel memory architecture.

As shown in Figure 16, the execution time is shorter for some time intervals, because of slower dynamics there requiring a smaller number of Newton iterations.

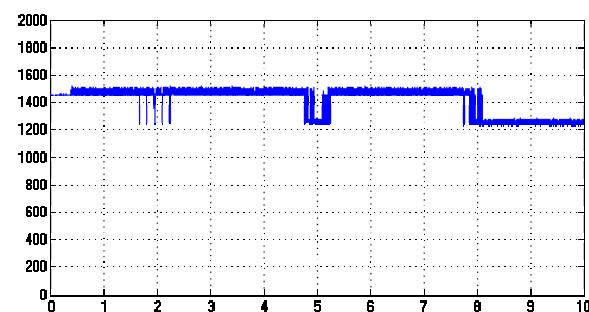


Figure 16: CPU time/step, when simulating level 4

It is worth noting that the level 4 model runs faster than the level 3 model, for which 1.7 ms per step is needed, although the level 4 model is more detailed. Obviously, the neglection of the push rod masses is not useful when Dymola's inline integration method together with its symbolic transformation capabilities are used. For offline simulations it is the opposite: the level 3 model runs faster as the level 4 model when using DASSL.

4 Conclusions

The paper has described typical efficiency issues in automotive real-time and HIL simulations. The examples given demonstrate the powerful real-time capabilities of Dymola and the Modelica modeling language. The models presented may indeed serve as benchmark examples as they are in the front-line of what can be simulated in real-time today. One of the benchmark models for vehicle dynamic simulation has 72 degrees-of-freedom with bushings in both the front and rear wheel suspensions. It was simulated in real-time with a sample rate of 500 Hz. The presented examples show that it is possible to simulate high-fidelity models in real-time for power trains and vehicle dynamics simulations. This is made possible by Dymola's unique and elaborate symbolic processing of the equations.

Acknowledgements

This work was in parts supported by *Bayerisches Staatsministerium für Wirtschaft, Verkehr und Technologie* under contract AZ300-3245.2-3/01 for the project *Test und Optimierung elektronischer Fahrzeug-Steuergeräte mit Hardware-in-the-Loop-Simulation*.

5 References

- [1] Andreasson, J.: *VehicleDynamics library*, Proceedings of the 3rd International Modelica Conference, Modelica 2003, Modelica homepage: <http://www.Modelica.org>.
- [2] Beckman, M. and J. Andreasson: *Wheel model library in Modelica for use in vehicle dynamics studies*, Proceedings of the 3rd International Modelica Conference, Modelica 2003, Modelica homepage: <http://www.Modelica.org>
- [3] Brück, D., H. Elmquist, S.E. Mattsson, H. Olsson: *Dymola for Multi-Engineering Modeling and Simulation*, Proceedings of Modelica 2002. Modelica homepage: <http://www.Modelica.org>.
- [4] Dymola. *Dynamic Modeling Laboratory*, Dynasim AB, Lund, Sweden, <http://www.Dynasim.se>
- [5] Elmquist, H., F. Cellier, M. Otter: *Inline Integration: A new mixed symbolic/numeric approach for solving differential-algebraic equation systems*. Proceedings: European Simulation Multiconference. June 1995 Prague, pp: XXIII-XXXIV.
- [6] H. Elmquist, S.E. Mattsson, H. Olsson. *New Methods for Harware-in-the-loop Simulation of Stiff Models*. Proceedings of Modelica 2002. Modelica homepage: <http://www.Modelica.org>.
- [7] Modelica, <http://www.Modelica.org>.
- [8] OPAL-RT, <http://www.opal-rt.com>.
- [9] Otter, M., H. Elmquist, S.E. Mattsson: *The New MultiBody Library*. Proceedings of the 3rd International Modelica Conference, Modelica 2003. <http://www.Modelica.org>.
- [10] PowerTrain Library 1.0 - Tutorial, German Aerospace Center (DLR), Oberpfaffenhofen, 2002, <http://www.dynasim.se/www/PowerTrainTutorial.pdf/>

Session 2B

Thermodynamic Systems - I

Modelica open library for power plant simulation: design and experimental validation

Francesco Casella, Alberto Leva*

Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio, 34/5 - 20133 Milano (Italy)

Abstract

The open Modelica library ThermoPower for the simulation of thermal power plants is presented, by illustrating the modelling principles and the main features of the developed models. The library has been validated against experimental data coming from a laboratory drum boiler, and the main results are shown in the paper. The library, plant model, and validation data are publicly available through the Web.

1 Introduction

Dynamic simulation plays a key role in the design of the control system of thermal power generation plant, in particular when innovative design efforts are undertaken. There is a long track of research and engineering effort in this field, dating from the pioneering paper [8] through [12, 16, 1, 15] and numerous other works. Also, many software packages have been developed in the academic as well as commercial field, see e.g. [4, 7, 21, 3, 2, 22, 19, 13, 17] and, in particular, [18]. Commercial modelling tools often suffer from the drawback of being opaque: it is not clear to the user which equations are actually been used to describe a certain component, and it is hard, if not impossible, to incorporate the user's specific know-how in the model library [6]. Conversely, in university laboratories many tools have been developed, in which the user has full control over the model equations; however, due to the intricacies of modelling thermo-hydraulic systems and to the difficulty of integrating the corresponding equations [16], ad-hoc modelling paradigms and software packages are employed, which are neither interchangeable nor interoperable with each other, not to mention their actual availability.

Moreover, when it comes to validating the models, it is very difficult to obtain complete and accurate ex-

perimental data sets from real plants [14]. Therefore, there is a strong need for shared and agreed-on models, which can be actually run by currently available simulation tools, as well as of benchmark data for model validation. The adoption of the Modelica language is a great opportunity in this direction.

The goals of the research work presented in this paper can be summarised as follows.

1. Develop an open Modelica library for the modelling of thermal power plants based on first principle models, which is highly readable and extensible, and where models of the same physical component with different level of detail may co-exist.
2. Demonstrate that models of real-life complexity can be dealt with by current Modelica tools.
3. Validate the library against experimental data from a laboratory plant.
4. Make the library code and the experimental data available to the scientific community.

The paper is organised as follows: Section 2 summarises the principles by which the entire library has been structured; Section 3 discusses the modelling assumptions and the main features of the developed models, while Section 4 is devoted to a brief description of the laboratory plant and of the experimental data set; in Section 5, the main results obtained with the plant simulator are shown. Conclusions and perspectives for future work are given in Section 6.

2 The library principles

This section outlines the principles of the presented library, motivating the adoption of Modelica as the host environment. A more detailed discussion is reported in [6], to which the interested reader is referred, while a

*Corresponding author, e-mail leva@elet.polimi.it

longer explanation of the modelling principles adopted in the library can be found in [15], the corresponding methodological foundations being discussed e.g. in [11].

Use of first-principle equations. The equations of the library models are derived from mass, energy and momentum balances, and (when necessary) from well established empirical correlations. Therefore, all the quantities involved in the models can be given a physical meaning.

Openness and transparency. The features of Modelica are exploited to obtain a code that tightly matches the way describing equation are written on paper. This greatly facilitates documenting and maintaining the library, and allows the users to understand exactly what they are simulating. Also, Modelica's powerful syntax can be exploited to investigate different modelling options quickly, and the inherently open nature of the environment permits modifications and improvements with a limited effort.

Readability-reusability trade-off. The inheritance mechanism is used sparingly, and with great care. Even though inheritance appears very attractive when structuring a component library, it is very difficult to define sufficiently general basic models in the application domain addressed here. Moreover, in a complex hierarchy of models, modifying the equations of some ancestor could have unexpected effects on the siblings, potentially impairing readability, not to say correctness. Since even fairly complex models can be described with a few dozen lines of code, it is advisable that the behaviour of a single component be described in a single place, rather than scattered through many different classes. Inheritance should be limited to the definition of 'prototype' components, i.e. partial classes containing connector declarations and auxiliary equations such as $\Delta p = p_{in} - p_{out}$. In the library there is one notable exception to this design principle, see section 3.4.1.

Partial Differential Equations. For the purposes of this work, models based on 1-dimensional partial differential equations are needed, which are not supported by Modelica in their native form. Therefore, such equations are reduced to ordinary differential ones by appropriate methods (e.g. finite volumes, finite elements) prior to their insertion in a Modelica model.

Standard interfaces. In the library, connectors are designed so as to be totally independent of the modelling assumptions adopted for the component. The same terminals are used no matter whether the fluid is assumed

to be one- or two-phase, the model is lumped- or distributed-parameter, the momentum balance is static or dynamic, the cross-sectional fluid velocity distribution is uniform or not, the phases in two-phase flows are assumed to have the same velocity or not, and so forth. To clarify with an example, we report the definition of the `waterFlangeA` and `waterFlangeB` connectors, which describe the flanges of the components that carry a water/steam flow:

```
connector WaterFlangeA
  Pressure      p;
  flow MassFlowRate   w;
  input SpecificEnthalpy hBA;
  output SpecificEnthalpy hAB;
end WaterFlangeA

connector WaterFlangeB
  Pressure      p;
  flow MassFlowRate   w;
  input SpecificEnthalpy hAB;
  output SpecificEnthalpy hBA;
end WaterFlangeB
```

In the code `p` is the fluid pressure, `w` is the mass flowrate entering the component, `hAB` and `hBA` are the specific enthalpies of the fluid in case its direction is from an A-type flange to a B-type flange and vice-versa. Correct models are obtained by always connecting two flanges of complementary type. These connectors support flow reversal.

The paradigm of connectors is exploited to standardise also the interfaces involving 1-dimensional distributed quantities used in modelling components like heat exchangers. Such connectors are characterised by a number of uniformly spaced nodes, and contain the nodal values of the quantities under question, no matter how the spatial discretisation is dealt with inside the component. An example is the `DHT` connector, whose definition is

```
connector DHT;
  parameter Integer N=2 "Number of nodes";
  Temperature T[N];
  flow HeatFlux phi[N];
end DHT;
```

Flexible level of detail. Encapsulation is exploited to allow for models with different degrees of detail, and fully interchangeable. This means that, in different situations, the same component or part of the plant can be modelled with different detail levels, with a small

effort on the part of the analyst.

Substance property calculation. Medium models for water, steam, and ideal gas mixtures are already provided by the free Thermofluid library [23]. Simulation efficiency could possibly be increased by using third-party property calculation packages written in C or FORTRAN. The library is open to such extensions.

Models for different fluids. It would be possible to make the equations of a component highly independent of the fluid contained, thus reducing the total number of library components. This is not very convenient for the presented library, however. In thermal power plants there are essentially two fluids (water/steam and ideal gas mixture), and the thermo-hydraulic phenomena involving these fluids are described by equations that can be very different also from the structural standpoint. Therefore, attempting to write equations in a ‘general’ form involves a significant complication of the equations themselves. It is preferable to write specialised models for the two fluids, and this is the approach adopted. The same specialisation applies to connectors, of course.

A great number of modelling environments and libraries for power plant simulation are available in the literature, see e.g. [4, 7, 21, 2, 22, 13, 17, 5], and in the last years several were developed within the Modelica environment (a remarkable example is [23]). There is not the space to give an exhaustive review here. However, two peculiarities of the proposed modelling approach, and therefore of the library, are worth pointing out. The first, as already mentioned, is the ‘flat’ structure of the model hierarchy, aimed at maximising the readability. The second is that, by writing the models as is done here, one can (but is not obliged to) reach the maximum level of detail that is advisable for simulations aimed at system-oriented studies, i.e., for example, at the synthesis and validation of the control system.

3 Developed models

3.1 Boundary conditions

Ideal pressure sources and sinks have been defined (`SourceP`, `SinkP`), as well as mass flowrate sources and sinks (`SourceW`, `SinkW`); note that the difference between source and sink is purely conventional, as both of them can handle flow in either direction. Hydraulic and thermal variables can be either constant, or determined by external signals.

3.2 Branching components

Flange terminals only support connection of *two* components; therefore, the `FlowJoin` and `FlowSplit` components are provided to model flow branching. The model are based on static mass and energy balances equations, supporting all the feasible flow directions and avoiding numerical singularities.

3.3 Elementary physical components

3.3.1 Valves

The `ValveLiq` and `ValveVap` models are based on the standard IEC 535 sizing equations for valves with liquid and vapour flow, respectively [10]; critical flow can be modelled in both cases, as well as check valve behaviour. Flow reversal is supported, avoiding numerical singularities for small or zero pressure drop. The opening characteristic can be customised.

3.3.2 Mixers, collectors, tanks

The `Mixer` and `Collector` models are based on standard mass and energy balances, assuming uniform pressure and temperature in the control volume; they differ only by the number of connecting flanges. Heat exchange with the metal wall can be also accounted for. `Tank` models a gas-pressurised tank, with gas charge and discharge valves.

3.3.3 Pumps

Since storage of mass and energy are negligible, the `PumpMech` model is expressed by algebraic characteristic equations derived from the manufacturer’s design data, that relate the pump head and the resistant hydraulic torque applied by the fluid to the shaft to the rotation speed and the volumetric flow rate. A boolean parameter allows to account for the total rotor inertia, when required. It is also possible to use the simpler model `Pump`, where the rotation speed is an input signal, the hydraulic torque is not computed, and a constant efficiency is assumed to determine the enthalpy difference between the inlet and the outlet.

3.3.4 Drum

The `Drum` model is the core of drum boilers models [9, 16]. In order to describe correctly the dynamics of fast transient, the model does not assume that the liquid and vapour phase are in thermodynamic equilibrium, i.e. at saturation state. Referring to figure 1, the basic

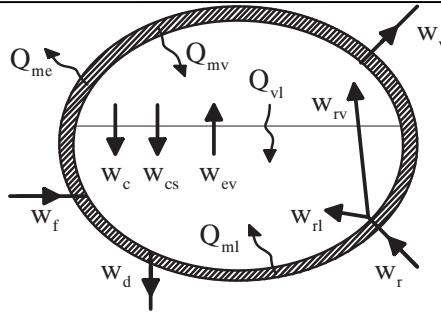


Figure 1: Steam drum.

equations are mass and energy balances for the liquid and vapour phases:

$$\frac{dM_v}{dt} = w_{rv} + w_{ev} - w_v - w_c - w_{cs} \quad (1)$$

$$\frac{dM_l}{dt} = w_f + w_{rl} + w_c + w_{cs} - w_d - w_b - w_{ev} \quad (2)$$

$$\begin{aligned} \frac{dE_v}{dt} = & w_{rv}h_{rv} + w_{ev}h_{vs} - w_vh_v - w_ch_{ls} - w_{cs}h_{vs} + \\ & + Q_{mv} - Q_{vl} - p\frac{dV_v}{dt} \end{aligned} \quad (3)$$

$$\begin{aligned} \frac{dE_l}{dt} = & w_fh_f + w_{rl}h_{rl} + w_ch_{ls} + w_{cs}h_{vs} - w_dh_d + \\ & - w_bh_l - w_{ev}h_{vs} + Q_{ml} + Q_{vl} - p\frac{dV_l}{dt} \end{aligned} \quad (4)$$

$$\frac{dE_m}{dt} = -Q_{ml} - Q_{mv} - Q_{me}, \quad (5)$$

where M_v , M_l , E_v , E_l , V_v , V_l are the mass, internal energy, and volume of the vapour and liquid phase holdups, E_m is the thermal energy of the metal wall, p is the drum pressure, w is a mass flowrate, h is a specific enthalpy, Q is a heat flow. The meaning of the subscripts is: rv : risers (vapour fraction), rl : risers (liquid fraction), l : liquid phase, v : vapour phase, c : condensation, cs superficial condensation, ev : evaporation, f : feed, d : downcomer, b : blowdown, vs : saturated vapour, ls : saturated liquid.

The bulk and superficial condensation flowrates, evaporation flowrate and convective heat exchange between the two phases are computed according to

$$w_c = \frac{(1 - x_v)\rho_v V_v}{\tau_c} \quad (6)$$

$$w_{cs} = K_{cs}A_{sup}(T_s(P) - T_l)) \quad (7)$$

$$w_{ev} = \frac{x_l\rho_l V_l}{\tau_{ev}} \quad (8)$$

$$Q_{vl} = K_{sup}A_{sup}(T_v - T_l) \quad (9)$$

where ρ_l , ρ_v , T_l , T_v are the liquid and vapour densities and temperatures, x_l , x_v are the steam qualities in the

liquid and vapour phases, τ_c , τ_{ev} are suitable time constants, A_{sup} is the area of the liquid surface, and K_{cs} , K_{sup} are suitable coefficients. The (non ideal) phase separation at the risers outlet is modelled as follows: h_{rl} is the saturated liquid enthalpy at the drum pressure, while h_{vl} is such that the corresponding steam quality is $1 - (\rho_v/\rho_l)^\alpha$.

The model is implemented in order to have the following state variables: drum pressure, liquid and vapour entropy, level, and metal wall temperature.

3.4 Building blocks for complex components

3.4.1 1-dimensional fluid flow

The Flow1D model describes the 1-dimensional flow of single-phase water in a tube of constant cross-section. The basic equations are the distributed-parameter mass, momentum, and energy balances:

$$A\frac{\partial p}{\partial t} + \frac{\partial w}{\partial x} = 0 \quad (10)$$

$$\frac{\partial w}{\partial t} + A\frac{\partial P}{\partial x} + \rho g A \frac{dz}{dx} + \frac{C_f \omega}{2\rho A^2} w|w| = 0 \quad (11)$$

$$\rho A \frac{\partial h}{\partial t} + \rho A u \frac{\partial h}{\partial x} - A \frac{\partial p}{\partial t} = \omega \varphi \quad (12)$$

where ρ is the fluid density, w is the mass flowrate, p is the pressure, A is the tube cross-section, g is the acceleration of gravity, z is the elevation, C_f is the Fanning friction coefficient, ω is the tube perimeter, u is the fluid velocity, h is the fluid enthalpy and φ is the heat flux entering the tube across the lateral surface. Equations (10)–(11) describe the fast pressure and flowrate wave dynamics, while Eq. (12) describes the slower dynamics of heat transport with the fluid velocity; the equations are then discretised with the finite volume method, considering a single volume for the former two (which need a coarser approximation in the frequency range of interest for power generation models), and many volumes for the latter.

Among the relevant features of this model, the following ones are worth mentioning: flow reversal is fully supported; the dynamic momentum term $\partial w/\partial t$ can be switched off to avoid fast pressure oscillations; the C_f coefficient can be either constant or computed by the Colebrook equation; the compressibility effect resulting from the finite volume approximation of (10) can be associated to either the upstream or downstream pressure; a bank of identical tubes in parallel can also be modelled.

The Flow1D2ph model can also deal accurately with two-phase flow; although being based on the same

equations (10)–(12), the significant differences with respect to Flow1D suggest writing two completely independent models.

The Flow1D2phDB model extends (in Modelica's terms) Flow1D2ph by also computing the heat transfer coefficient γ via Dittus-Böltz equation; correspondingly, the DHT connector (which is replaceable) is substituted by the extended DHThtc connector, which makes the values of γ visible to the outside.

3.4.2 Pressure drop

The PressDrop model provides the model for a generic pressure drop proportional to the kinetic pressure. The equation is modified by adding a small linear term, to avoid singularities with small or zero flowrates, thus reading:

$$p_{in} - p_{out} = \frac{K_f(|w| + Kl)w}{\rho} \quad (13)$$

The same modification also applies to the models described in section 3.4.1.

3.4.3 Metal wall

The MetalWall model describes a generic cylindrical metal wall, accounting for the thermal resistance due to heat conduction and for the heat storage due to thermal capacity; uniform temperature is assumed in the radial direction. More sophisticated models could be derived to better reproduce the actual radial temperature dynamics, e.g. in thermal stress studies.

3.4.4 Heat exchange modules

The heat flux exchanged between two (or more) objects, such as a fluid flow and a metal wall, is in general a function of the corresponding surface temperatures; therefore, it can be computed by a model interfaced via DHT connectors. The ConvHTe and ConvHTc models provide simple examples for co-current and counter-current 1D configurations, with given heat exchange coefficient γ . ConvHTe_gamma extends the former by using a variable value of γ , provided by the connected object through its DHT_gamma connector. More complex configurations can be easily described with a few lines of code.

3.5 Complex physical components

A whole range of heat exchanger models can be assembled using the components described in Section

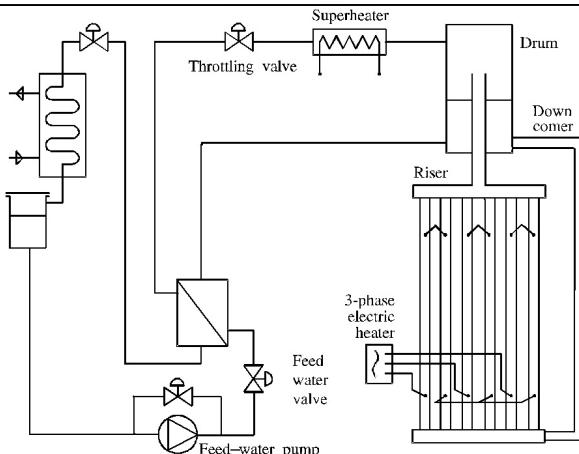


Figure 2: The laboratory plant.

3.4, depending on physical configuration, operating conditions and desired degree of detail. None of these models probably deserves to be included in the library as such; if a specific aggregate model is to be used many times, the user can easily define it as a new model inside his plant model. Some of them may nevertheless be included in the library to serve as examples.

4 The laboratory plant and data

4.1 Overview

The laboratory plant employed to validate the presented library is a physical model of the evaporating section of a heat-recovery boiler, with a power scaling factor of 1:600. The laboratory plant layout is shown in fig. 2.

To be precise, only the circulating loop of the laboratory plant exactly reproduces the thermo-hydraulic conditions of the real boiler. The other components (preheater, valves, pumps, etc.) provide the correct boundary conditions for the evaporator. In particular, the superheater supplies the necessary (limited) steam superheating to allow a reliable measurement of the steam flow upstream of the throttling valve.

The steam generation takes place at a nominal operating pressure of 60 bar, as in the real plant. The evaporator is made of six electrically heated parallel tubes, one downcomer and a vertical-axis drum, plus the necessary headers and connection tubes. A feed-water valve may be used for drum-level control, and the throttling valve to control the drum pressure. The heat rate to the evaporator is modulated by a power regulator.

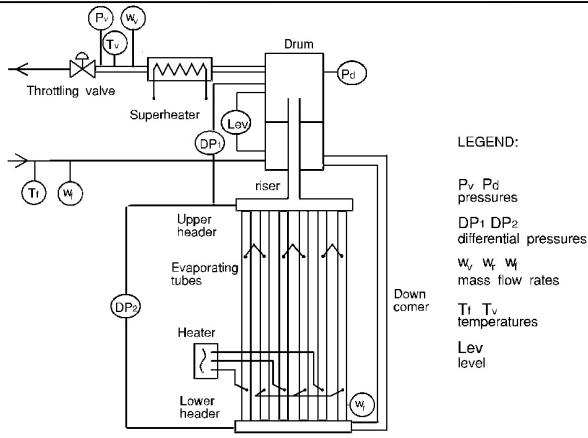


Figure 3: The available plant measurements.

The relevant process measurements are summarised in fig. 3; the measurement of the electric power released to the evaporator is also available.

4.2 Plant tests

Many static and dynamic tests were performed on the plant. These tests are plant responses obtained by imposing step variations to the evaporator electric power, the throttling valve position, and the feed-water control valve position. During these tests, the plant was in an open loop. Step variations were given, starting from two different sets of steady-state conditions: the former at high load (around 100% of the maximum load), and the latter at about half load. The boiler pressure was kept nearly proportional to the load: full-load tests were done at about 60 bar, and half-load tests at about 30 bar. Step variations were always imposed both upwards and downwards, their amplitude being in the range 10-15%. Altogether, seventeen step-response tests were executed and logged.

4.3 Data reconciliation

Experimental data coming from the tests were analysed, in order to build a consistent database. The main problem evidenced was a discrepancy between the feed-water and the superheated steam flow rate measurement. Those flow rates must balance at any steady state, and even a small imbalance between causes a significant modification of the drum-level transients. Hence, it is very important that the corresponding measurement errors be corrected. In the case at hand, it is assumed that the feed-water flow is error-free (it is in fact much more accurate than the steam flow mea-

surement).

On the basis of steady-state measurements, the calibration constant of the instrument was recomputed. Moreover, to compensate for unpredictable measurement errors, the record of steam flow rate relative to every step response was biased, so as to impose perfect balance at the initial steady state.

A further problem is that the heat rate to the superheater (supplied by an electrical resistor) is not measured. At any steady state, the heat rate Q may be estimated by means of the thermal balance

$$Q = w_v (h_v(p_v, T_v) - h_d), \quad (14)$$

where w_v is the superheated steam mass flow rate, h_d the fluid enthalpy at the drum outlet, and $h_v(p_v, T_v)$ the steam enthalpy at the superheater outlet, evaluated at the local steam temperature T_v and pressure p_v . Unfortunately, h_d is not easy to evaluate because the fluid at the drum outlet is generally wet steam, whose quality x_d is close to one, but unknown. It has been assumed that the steam quality is 1, and $h_d = h_{vs}(p_d)$, where $h_{vs}(p_d)$ is the vapour saturation enthalpy at the drum pressure p_d . Note that a (realistic) wetness of 3%, at 60 bar, yields $h_{vs}(p_d) - h_d \approx 47 \text{ kJ/kg}$, i.e. a temperature difference of about 14°C at the superheater outlet. In addition, the analysis of experimental data shows that x_d is not constant when the operating condition is changed, but the information available is not sufficient for deriving an empirical correlation for x_d . This is the most important uncertainty in the experimental data, that could not be removed. Fortunately, this uncertainty is relevant only for the superheated steam temperature, while it is almost negligible for the evaluation of the other process variables. The heat rate to the superheater was generally kept constant during any dynamic test. Therefore, its value was computed from the initial steady state, using (14) and the approximation $h_d \approx h_{vs}(p_d)$.

The experimental data records, completed with the corrected steam flow rate and the superheater heat rate, were assumed as the validation database.

5 Experimental validation

5.1 The simulation model

The Modelica diagram of the simulation model is shown in figure 4. This model proves that cases of realistic complexity (i.e., hundreds of differential equations) can be treated effectively. There is not the space to give details. For further information, the reader is referred to the library and model code.

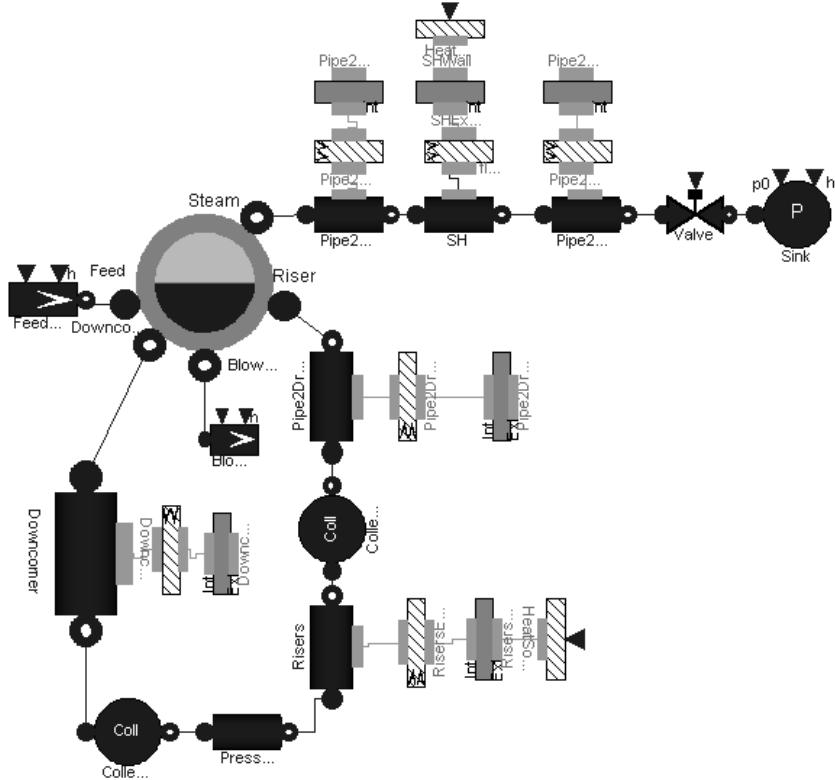


Figure 4: Modelica diagram of the simulation model.

5.2 Model calibration

Steady-state measurements were used to estimate the process parameters affected by an intrinsic uncertainty, i.e. the friction coefficients for the different components in the circulation loop, the friction correlation in the superheater, and the heat losses of the evaporator. For the evaporator, it was assumed that friction obeys to Colebrook's law, and a concentrated pressure drop was introduced upstream of the evaporating tubes, to account for the flow measurement orifices and other flow discontinuities. A multiplicative corrective coefficient was introduced in the second flow equation, and was calibrated with steady-state data to match the circulation flowrate.

The calibration of the friction correlation for the superheater was done selecting the tube roughness so that the relation between the Reynolds number and the friction coefficient matched the points computed from experimental data. As for the evaporator heat losses, considering the evaporator thermal balance at different steady states, it was found that the experimental data fit the formula

$$Q_{lost} = k(T_{wd} - T_{amb}), \quad (15)$$

where Q_{lost} is the lost heat rate, T_{amb} the ambient temperature, and T_{wd} the drum metal wall temperature. Note that Q_{lost} is typically around 10% of the input electrical power, and varies significantly with the drum pressure.

5.3 Individual validation of components

Individual validation of a component can be carried out for components when the available measurements supply complete boundary conditions for that component. In the case presented, only the model of the choked-flow valve could be validated individually, since all its boundary conditions (inlet steam pressure, flow rate and temperature) were measured.

5.4 Global validation of the plant model

The global validation of the whole plant model is aimed specifically at the analysis of relevant alternatives in terms of component modelling and overall model structuring. In the following the validation tests are listed, together with the results achieved from the point of view of modelling. It is important to notice that the tests were made in open loop and applying step

stimuli: this leads to very informative results on the model correctness, as no control system can conceal discrepancies between the model outputs and experimental data, and the stimuli cover a frequency range wide enough to evidence the model behaviour with respect to phenomena that are ‘fast’ with respect to the dominant plant dynamics. It is also worth stressing that the model was calibrated only once (at high load), and no modification to the model parameters was made to perform the various simulations presented.

5.4.1 Heat rate steps

Negative step variations were applied at high load to the electrical power fed to the heating system. Feed-water was not regulated, so the pressure variation due to the heat rate perturbation caused also a variation of the feed-water flow rate. To reproduce the actual conditions, the simulator was fed with the measured feed-water flow rate as an input.

The main result is that the process behaviour is reproduced very accurately, except for the superheated steam temperature. Its measurement is very noisy, however, and its variations are comparable with the errors due to uncertainty on the steam quality at the drum outlet. Recall also that the heat rate released to the superheater is not measured. These facts confirm that the uncertainty exists, is relevant, cannot be eliminated with the available measurements, but is confined to the outlet steam temperature.

An example of these tests is shown in figures 5 and 6, depicting the drum pressure and level transients, respectively. Notice that the pressure dynamics are reproduced correctly over the frequency range that is interesting for control (corresponding to a typical time scale of some tenth or a few hundreds of seconds). This is true thanks to the non-equilibrium model of the drum.

5.4.2 Throttling valve steps

Responses to positive and negative throttling valve steps, both at high and low load, showed good agreement between the model output and data. For the reason above, in these tests the feed-water flow rate (that acts as a disturbance) was an input for the simulator. In particular, the non-equilibrium phenomena represented in the drum model allow to reproduce both low- and mid-frequency dynamics in the pressure responses correctly, and are necessary for this purpose, as witnessed by the effects of the involved parameters (e.g., τ_{ev}) on the responses. Also the effects of thermal ex-

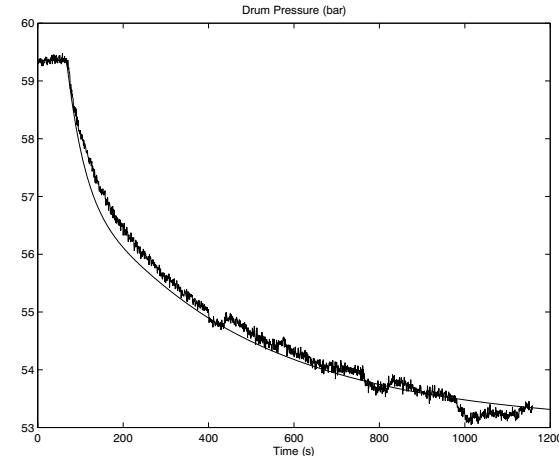


Figure 5: Drum pressure transient for a -10% heat rate step at high load (simulated vs. experimental data).

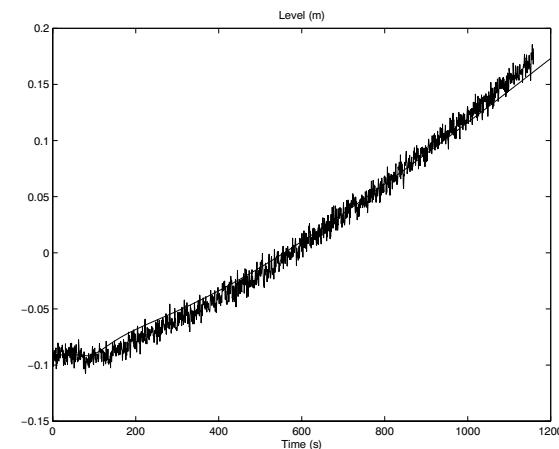


Figure 6: Drum level transient for a -10% heat rate step at high load (simulated vs. experimental data).

changes between the fluid in the drum and the drum metal wall were investigated, showing that the corresponding heat transfer coefficient has a significant influence on the superheated steam temperature. This phenomenon is often neglected in the simulation models proposed in the literature.

Figures 7 and 8 reports the drum pressure and the level transients in one of these tests, namely a negative valve step at low load, and confirm the considerations made in the previous section. Notice that in this particular transient bulk boiling actually takes place within the liquid drum subvolume.

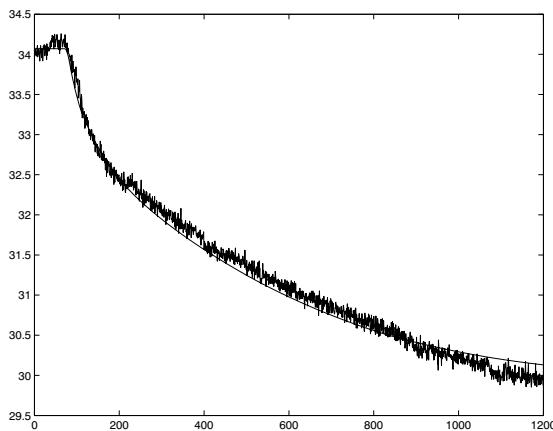


Figure 7: Drum pressure transient for a throttling valve step leading to a 13% pressure reduction at low load (simulated vs. experimental data).

5.4.3 Feed-water valve steps

Positive and negative feed-water valve steps were applied. Figure 9 shows the drum level transient in one of these tests, demonstrating good accordance between model and data.

6 Conclusions and work in progress

An open Modelica library for the simulation of thermal power plants has been presented. The library has been used to build a high-fidelity model of a laboratory drum boiler, which has been successfully validated against available data.

The library has been conceived in order to emphasise model readability and extensibility; it contains a limited number of components which nevertheless allow modelling a wide range of different physical components. It should be stressed that the Modelica language allowed translating sophisticated modelling concepts into working code with remarkable ease.

The library is being released to the public, and is open to contribution from other research groups (see URL: <http://www.elet.polimi.it/upload/casella/thermopower/>).

The benchmark boiler model together with the experimental data is being released as well.

The development of component models using gases as working fluid (compressor, turbine, combustion chamber, basic components for heat exchangers etc.) and of finite element models for the 1-dimensional fluid flow model is planned for the near future. It could also be

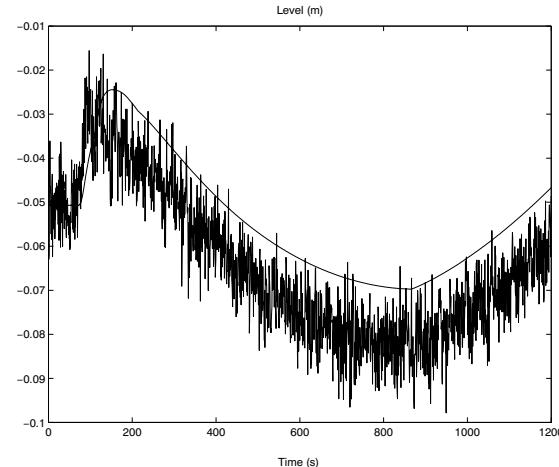


Figure 8: Drum level transient for a throttling valve step leading to a 13% pressure reduction at low load (simulated vs. experimental data).

interesting to investigate the combined use of the ThermoPower library with control libraries and electro-mechanic libraries to build complete models of power generation equipment.

7 Acknowledgements

The authors are grateful to W. Prandoni and D. Laudato, who realised the physical experiments, and to the CESI research centre (particularly to G. Benelli), who made the relative data available. Many thanks are also due to prof. C. Maffezzoni, for inspiring the presented research and contributing to it with numerous ideas, hints, discussions, and constructive criticisms.

References

- [1] Åström, K.J. and Bell, R.D. (1993), “A nonlinear model for steam generation process”, Prepr. 12th IFAC World Congress, Sydney, 3, 395–398.
- [2] Bartolini, A., Leva, A. and Maffezzoni, C. (1995), “Power plant simulator embedded in a visual programming environment”, Proc. IFAC Conf. SIPOWER 95, Cancún, 119–124.
- [3] Barton, P.I. and Pantelides, C.C. (1994), “Modeling of combined discrete/continuous processes”, AiChE Journal, 6, 966–979.
- [4] Breitenecker, F. and Solar, D. (1986), “Models, methods, experiments - modern aspects of simu-

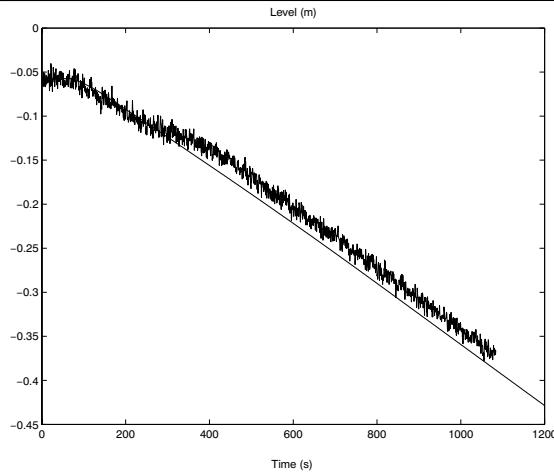


Figure 9: Drum level transient for a feed-water valve step leading to a 40% feed-water mass flow rate reduction at low load (simulated vs. experimental data).

lation languages”, Proc. 2nd European Simulation Conference, Antwerpen, San Diego, 195–199.

- [5] Carpanzano, E., Ferrarini, L. and Maffezzoni, C. (1999), “Simulation environments for industrial process control”, Proc. ESS ’99, Erlangen-Nuremberg, 443–450.
- [6] Casella, F. and Leva, A. (2003), “Modelling of distributed thermo-hydraulic processes using Modelica”, Proc. MathMod’03, Wien.
- [7] Casti, J.L. (1992), “Reality rules - picturing the world in mathematics: I, II”, Wiley, New York.
- [8] Chien, K.L., Ergin, E.I., Ling, C. and Lee, A. (1958), “Dynamic analysis of a boiler”, Trans. ASME, 80, 1809–1819.
- [9] Collier, J.G. (1981). “Convective boiling and condensation (2nd ed.)”, McGraw-Hill, New York.
- [10] ISA (1986). “Flow equations for sizing control valves”, Instrument Society of America, Research Triangle Park.
- [11] Incropera, F.P. and De Witt, D.P. (1981), “Fundamentals of heat and mass transfer”, Wiley, New York.
- [12] Lausterer, G.K., Franke, J. and Eitelberg, E. (1983), “Modular modelling applied to a Benson

boiler”, Proc. 1st IFAC workshop on Modelling and Control of Electric Power Plants, Como.

- [13] Leva, A., Bartolini, A. and Maffezzoni, C. (1998), “A process simulation environment based on visual programming and dynamic decoupling”, Simulation, 71(3), 183–193.
- [14] Leva, A., Maffezzoni, C. and Benelli, G. (1999), “Validation of drum boiler models through complete dynamic tests”, Control Engineering Practice, 7, 11–26.
- [15] Leva, A. and Maffezzoni, C. (2003), “Modelling of power plants”, in D. Flynn (Ed.), “Thermal power plant simulation and control”, IEE, London, 17–60.
- [16] Maffezzoni, C. (1992), “Issues in modelling and simulation of power plants”, Proc. IFAC Symposium. on Control of Power Plants and Power Systems, Munich, 1, 19–27.
- [17] Maffezzoni, C. and Girelli, R. (1998), “MOSES: modular modelling of physical systems in an object-oriented database”, Mathematical Modelling of Systems, 4(2), 121–147.
- [18] Mattsson, S.E., Elmquist, H. and Otter, M. (1998), “Physical system modeling with Modelica”, Control Engineering Practice, 6 (1998), 501–510.
- [19] Oh, M. and Pantelides, C.C. (1996), ”Modelling and simulation language for combined lumped and distributed parameter systems”, Computers & Chemical Engineering, 6-7, 611–633.
- [20] Perry, R.H. (1984), “Perry’s Chemical engineer’s handbook”, McGraw-Hill, New York.
- [21] Sage, A.P. (1992), “Dynamic systems”, in Atherton, D.P. and Borne, P. (Eds.), “Concise Encyclopedia of Modelling and Simulation”, Pergamon, Oxford, 91–92.
- [22] Troch, I. (1995), “Modelling for optimal control of systems” Surveys on Mathematics for Industry, 5 , 203–292.
- [23] Tummescheit, H., Eborn, J. and Wagner F.J. (2000), “Development of a Modelica base library for modeling of thermo-hydraulic systems”. Proc. the Modelica Workshop 2000, Lund, 41–50. Library URL: <http://sourceforge.net/projects/thermofluid>.

Simulation of Liquid Food Processes in Modelica

Tomas Skoglund

Tetra Pak Processing Systems, Ruben Rausings gata, S-22186 Lund, Sweden,
tomas.skoglund@tetrapak.com, www.tetrapak.com

Abstract

Traditionally, liquid food processing equipment has been designed and engineered from a static perspective, where it has been taken for granted that dynamic behaviour easily could be handled by “add on” of control equipment such as sensors and computers with control programs including control loops. However, as production demands, e.g. mixing accuracy, are escalated, this approach fails, and the importance of simulating the dynamics of the system becomes crucial. A tool that makes it possible to minimise the cost and time for building prototypes and making experiments would be of considerable value, particularly if the tool enables reuse of earlier work. Equally important is the possibility to test various design ideas to improve the equipment performance to an extent that otherwise would not be conceivable.

This article describes how the Modelica based tool Dymola¹ has been used to build up a library (“FoodProcessing”) primarily aiming at simulating certain dynamic behaviour in liquid food processing plants, particularly characterised by incompressible fluids with complex rheologic behaviour, transport delays and dynamically changing concentrations.

1. Introduction

When starting a project aiming at building a model library for simulation of liquid food processes, an analysis should be performed to define:

1. Which processes and phenomena are involved?
2. Which physical properties are involved?
3. Which product (fluid) properties are relevant?
4. Which components shall be included?

Another important aspect to consider is to whom the library is directed, i.e.:

1. Who is the user?
2. Which symbol standards are relevant?
3. How shall model variations be handled?

In this work the above premises were evaluated as a base for the creation of a food processing library.

¹ Dymola by Dynasim AB in Lund, Sweden

2. Basic library structure

To meet the demands from the analysis of above mentioned questions, two major library design decisions were taken:

1. To facilitate the usage of the “FoodProcessing” library for process and automation engineers, the library should:
 - separate models “ready to use”, from models used for building other models (Fig. 2.1).
 - use relevant symbol standards as much as possible (see paragraph 6.3).

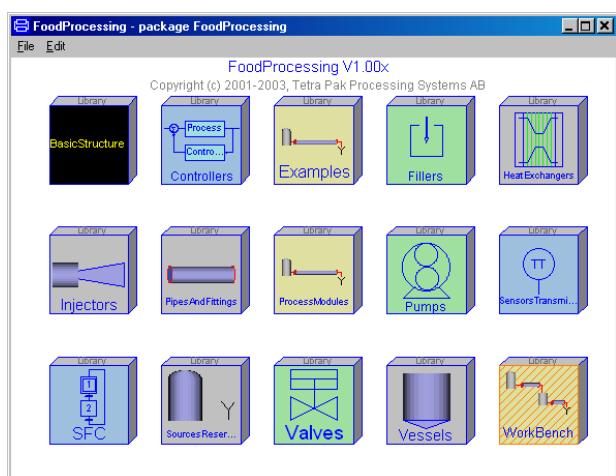


Fig. 2.1 The ‘top view’ of the library where the coloured (grey) boxes contain models ready-to-use and the black box contains models for model builders only.

2. New connectors must be created to enable fluids with rheologic complex characteristics and dynamically changing concentrations. The connectors contain information about:
 - Flow rate
 - Pressure
 - Thermal energy
 - Fluid concentrations
 - Fluid properties

There is more than one way to represent these, but to facilitate the understanding from the user group point of view, the most commonly used physical properties have been chosen. The Modelica code for the connector ProductIn is:

```

connector ProductIn
  flow SIunits.VolumeFlowRate Q;
  SIunits.VolumeFlowRate Qs;
  SIunits.Pressure p;
  FoodProcessing.BasicStructure.Phys
  Data.ProductData PrData;
end ProductIn;

```

where ProductData is:

```

record ProductData
  SIunits.Density rho;
  SIunits.ThermalConductivity
  lambda;
  SIunits.SpecificHeatCapacity cp;
  SIunits.CelsiusTemperature TempC;
  Real n "Flow behaviour index,
    dynamic viscosity power law
    n-value [-]";
  Real K "Consistency, dynamic
    viscosity power law K-value,
    [Pa.s^n]";
  Real Conc[5] "Concentration
    [weight %] of component 1-5";
end ProductData;

```

The *across variable* Q_s is used as a copy of the *through (flow) variable* Q to be able to easily “pick up” the flow rate with flow sensors, something that cannot be done directly with *through variables*. (For sensor aspects, see paragraph 7.) The copying of Q to Q_s is done in the component models with the simple equation:

```
ProductIn1.Qs = ProductIn1.Q;
```

3. Physical equations

The fundamental physical equations governing a fluid system are partial differential equations. By limiting the main scope to one-phase incompressible fluids (even though some gas phases also have to be dealt with), the room discretization need only consider dynamically change of fluid concentrations and temperature. In other words, to obtain ordinary time differential equations, the control volumes often can be quite large. Furthermore, since this library is aiming at bulk properties, only one-dimensional discretization is required along the flow channels, such as pipes and heat exchangers.

For the model description of the components (with one ore more control volumes) groups of relationships are included

- Conservation equations:
 - mass conservation
 - energy conservation (thermal)
 - volume conservation (incompressibility)
 - momentum conservation (dynamically from Newton’s 2nd law). In a pipe with the length

L and the same cross section area throughout the whole pipe we have:

$$\rho L \frac{dv}{dt} = p_1 - p_2 + \Delta p_w + \rho g \Delta h$$

where:

v = flow velocity [m/s]

ρ = density [kg/m^3]

p_1 = pressure at pipe inlet [Pa]

p_2 = pressure at pipe outlet [Pa]

Δp_w = pressure drop due to wall friction [Pa]

g = gravity constant of acceleration [$9,81 \text{ m/s}^2$]

Δh = difference in level between pipe inlet and outlet [m]

This whole set of conservation equations is a result of approximations (simplifications) due to certain limitations in the aim of the simulation objectives, i.e. neither kinetic energy nor compressibility is included. So far, in this scope, also effects of chemical reactions can be ignored.

- Constitutive equations:

– pressure drop

– heat flow

– component characteristics

– etc

These equations are typically unique for individual components and express relations between the above variables and component parameters/variables. Many times algebraic equations are enough, but sometimes dynamic effects need to be addressed, i.e. differential equations are required.

The pressure drop model in pipes handles the flow regime from laminar to turbulent for smooth pipes.

- Transport delay:

As concentration and temperature may vary when a fluid flows through a system, the transport time from one point to another becomes an important effect that needs to be included in models of pipes etc. Including true transport delay in the models reduces the need for very high degree of discretization, which is an approximation that converges as the discretization goes to infinity:

In case of constant flow; let the transfer function $G(s)$ represent the concentration in a volume V through which there is a constant flow rate Q , and in which there is a perfect mixing. Then with $\tau = V/Q$ we have

$$G(s) = 1/(1+s\tau)$$

Suppose now that a pipe is seen as this volume, but sliced into n pieces of volumes. Then we get:

$$G_n(s) = [1/(1+s\tau/n)]^n \rightarrow e^{-\tau s} \text{ as } n \rightarrow \infty$$

Which proves the statement.

4. Media models

Many liquid food-stuffs behave strongly non-Newtonian where only one viscosity parameter is not enough, and the main concern is to choose relevant rheologic model. A model that covers many liquid foods is the Ostwald de Waele “power law” model [7]:

$$\sigma = K\dot{\gamma}^n \quad \text{and} \quad \mu = \frac{\sigma}{\dot{\gamma}} = K\dot{\gamma}^{n-1}$$

where:

σ = shear stress [Pa]

$\dot{\gamma}$ = shear rate [s^{-1}]

n = flow behaviour index [-]

K = consistency [Pasⁿ]

At this stage this is the chosen model, but in the future probably it has to be extended to a more complex model such as “Herschel-Bulkley”. This needs to be considered in the library structure to facilitate a future “upgrade”.

In typical food processes the food is heated, cooled or mixed. To be able to handle these changes in temperature and concentration, models are required for how relevant fluid properties depend on these. In other words the relationships:

Fluid property = f(Temperature, Concentration) is required for:

- Rheologic properties such as viscosity or, for the Ostwald de Waele (power law) model, consistency and flow behaviour index. More complex fluids require more parameters.
- Thermal properties. (Specific heat capacity and thermal conductivity. Since the specific heat capacity is well approximated with a straight line dependency of the temperature for relevant food stuffs, the thermal energy can be handled by using just the specific heat capacity and the temperature.)
- Density

Approximate models for these have been included in the library.

5. Approximations and simplifications

Generally speaking, the physical relationships and media models have to be approximated/simplified with the target in mind to get a library with components and media that, when used within the simulation scope, meet relevant demands concerning the following aspects:

- accuracy
- speed
- robustness

In this library, models with more or less approximations are built for conservation equations, constitutive equations and media models.

6. Component models

A library structure can be built in many different ways. As mentioned above, this library structure is built to facilitate simulations from a user perspective. Therefore the components are divided into component groups on the top level (Fig. 2.1). In each group, models with different complexity (more or less approximations) can be chosen. Fig. 6.1 shows the content of a sub library “PipesAndFittings” containing various components such as pipes and bends etc.

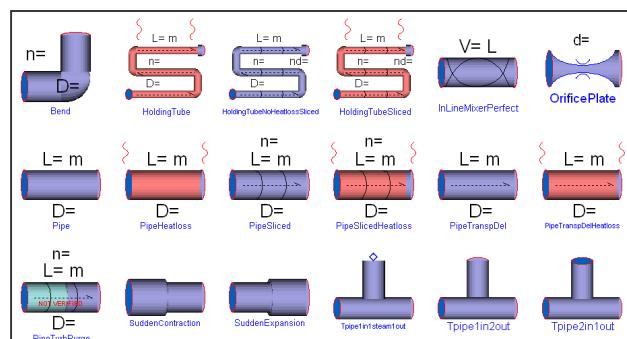


Fig. 6.1 Component sub library “PipesAndFittings”.

6.1 Variations in models

Sometimes there is a wish to easily run simulations with different model types (e.g. more or less approximations) without having to swap component. Modelica has various features for that. However, using such a feature would require that the users write the Modelica code for it, e.g. “replaceable....” and “redeclare....”. Because of this, some alternative model types are included in one model and handled via parameters to change

the type with just a simple change of a (Boolean) parameter. For example a PID-controller is developed that handles both analogue and sampled control depending on just a Boolean parameter. (Fig 6.2 and 6.3)

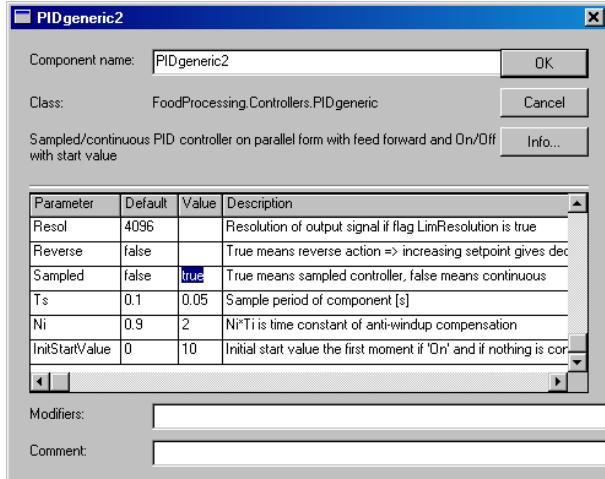


Fig. 6.2 Parameter list where the parameter “Sampled” is set to “true”

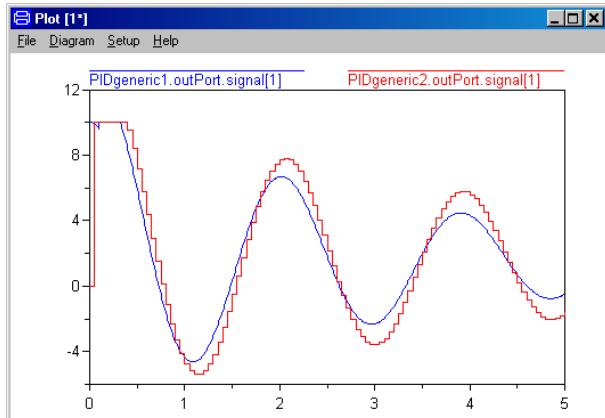


Fig. 6.3 Simulation results with plotted output from a PID-controller in a certain scenario with parameter “Sampled” set to “false” (smooth curve) and “true” (stepwise curve). In the sampled case, the simulation is slower due to a heavier computation task than in the continuous (not sampled) approximation.

6.2 Parameter settings

To facilitate the work for the user, some of the characteristics for the commercially available and used flow components are stored in data files referred to by a string parameter (the component type name). In this way the user can easily choose and change the type and size of the component, e.g. valve type and size. (Fig 6.4)

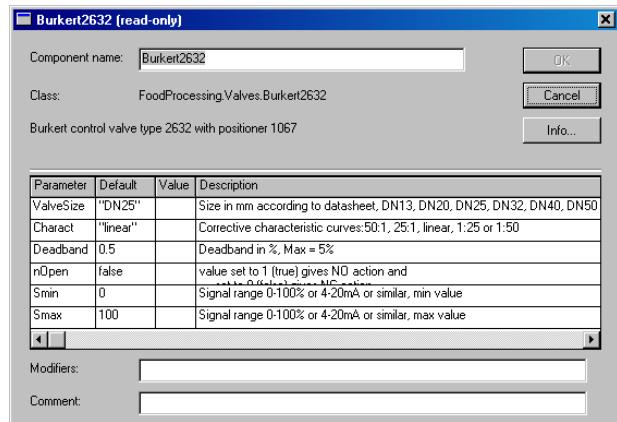


Fig. 6.4 The single string parameter “ValveSize” points on several valve parameters in a data file.

6.3 Component icons

Within the industry there are different standards for symbols (e.g. ISO 3511, “Process measurement control functions and instrumentation – Symbolic representation”). Further more, within Tetra Pak, these standards have been adapted to a branch and company standard. To increase the intuitive understanding the library icons follow these as much as possible (Fig 6.5).

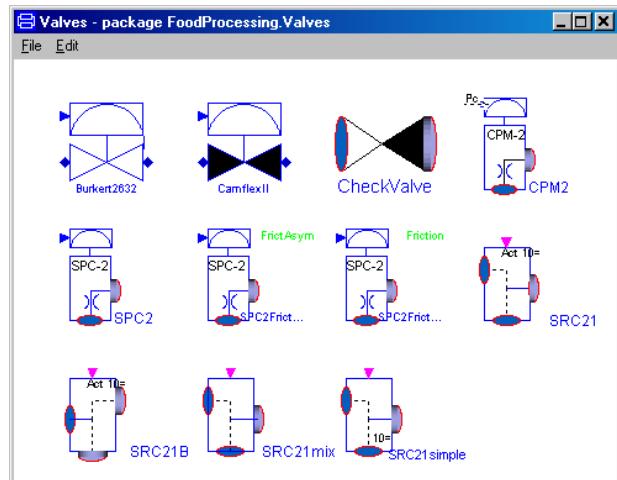


Fig. 6.5 Component sub library “Valves” with Tetra Pak standard symbols built on ISO, branch and company standards.

Also sequential function control charts (SFC) (=Petri nets) have its industry standard symbols (IEC 848, “Preparation of function charts for control systems”). Fig 6.6 shows the limited sub library SFC, e.g. parallel and alternative handling are missing.

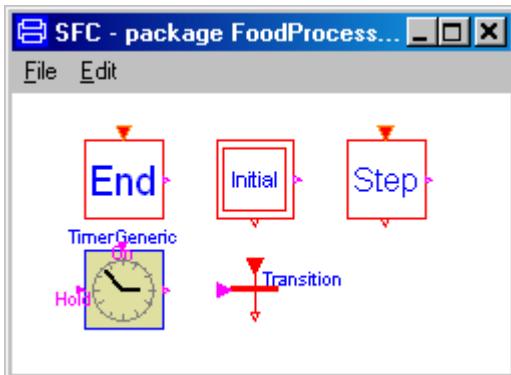


Fig. 6.6 Component sub library “SFC” for sequence control.

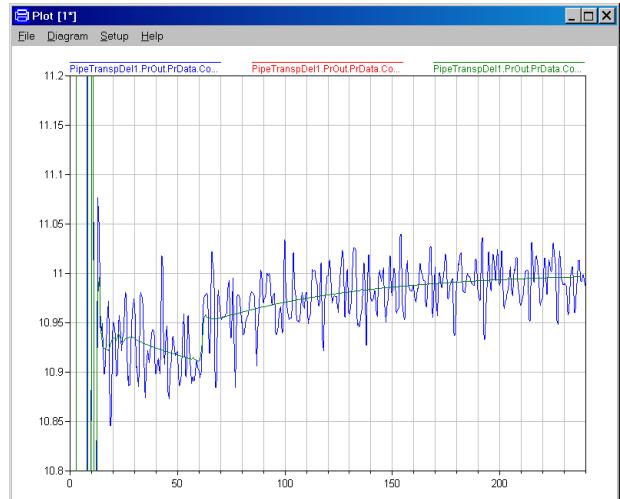


Fig. 7.2 Concentration in a pipe when the concentration transmitter in the control loop is “perfect” or noisy.

7. Sensor and transmitter models

Sensors with transmitters are also important to model since they are a part of closed loop systems. They are also not perfectly describing the property they are aimed for. Two “distortion” factors are involved:

- dynamic behaviour
- inaccuracy

Another user aspect is that they should be able to connect as standard symbols on a drawing, i.e. like “pick-ups” on the measured point (Fig 7.1).

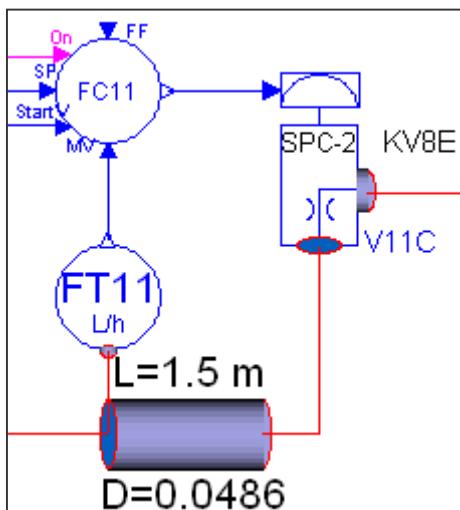


Fig. 7.1 Flow sensor with transmitter (FT11) connected as a “pick-up” on a pipe in a flow rate control loop.

The possibility to simulate inaccuracy is valuable for high performance control when the transmitter accuracy or noise is in the same range as the target of the control accuracy. Fig 7.2 shows a simulation of start-up of a blending system with and without noisy information from a concentration transmitter.

8. Interfacing other libraries

Liquid food processing involves heating with steam and an existing library handling that is ThermoFluid [8]. Therefore, instead of developing new models for steam systems, this model domain is interfaced with the FoodProcessing domain by certain components, such as steam injectors (Fig. 8.1), which are used to inject steam directly into the food stream.

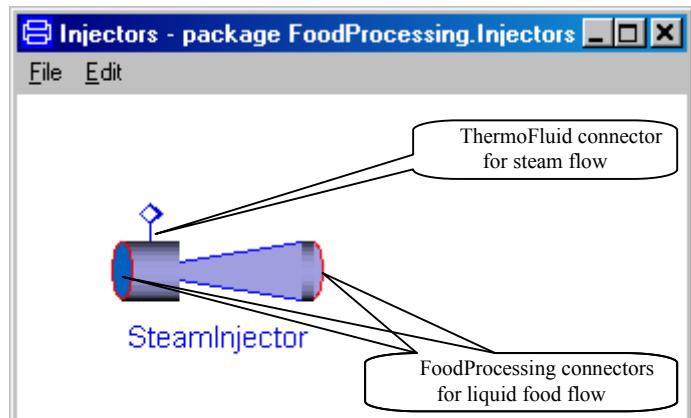


Fig. 8.1 Component “SteamInjector” with connectors to interface FoodProcessing with ThermoFluid [8].

9. Simulation example “in-line blending”

In-line blending is commonly used as an efficient way to produce standardised food such as standard milk with a predefined content of fat. Modern systems are designed in different ways depending on flexibility requirements etc, but are typically accurate and responsive to disturbances. To reach the high control performance, the control system sometimes becomes quite complex, as well as the process systems. Fig. 9.1 shows a “top view” of a simpler type of such a system. Fig. 9.2 shows the process part of it and fig 9.3 and 9.4 show a 5-minute simulation result of the same system.

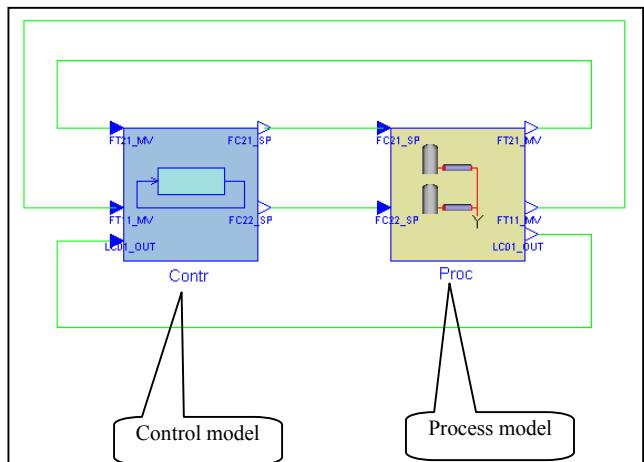


Fig. 9.1 “Top view” with “process” and “control” of a system model for milk blending.

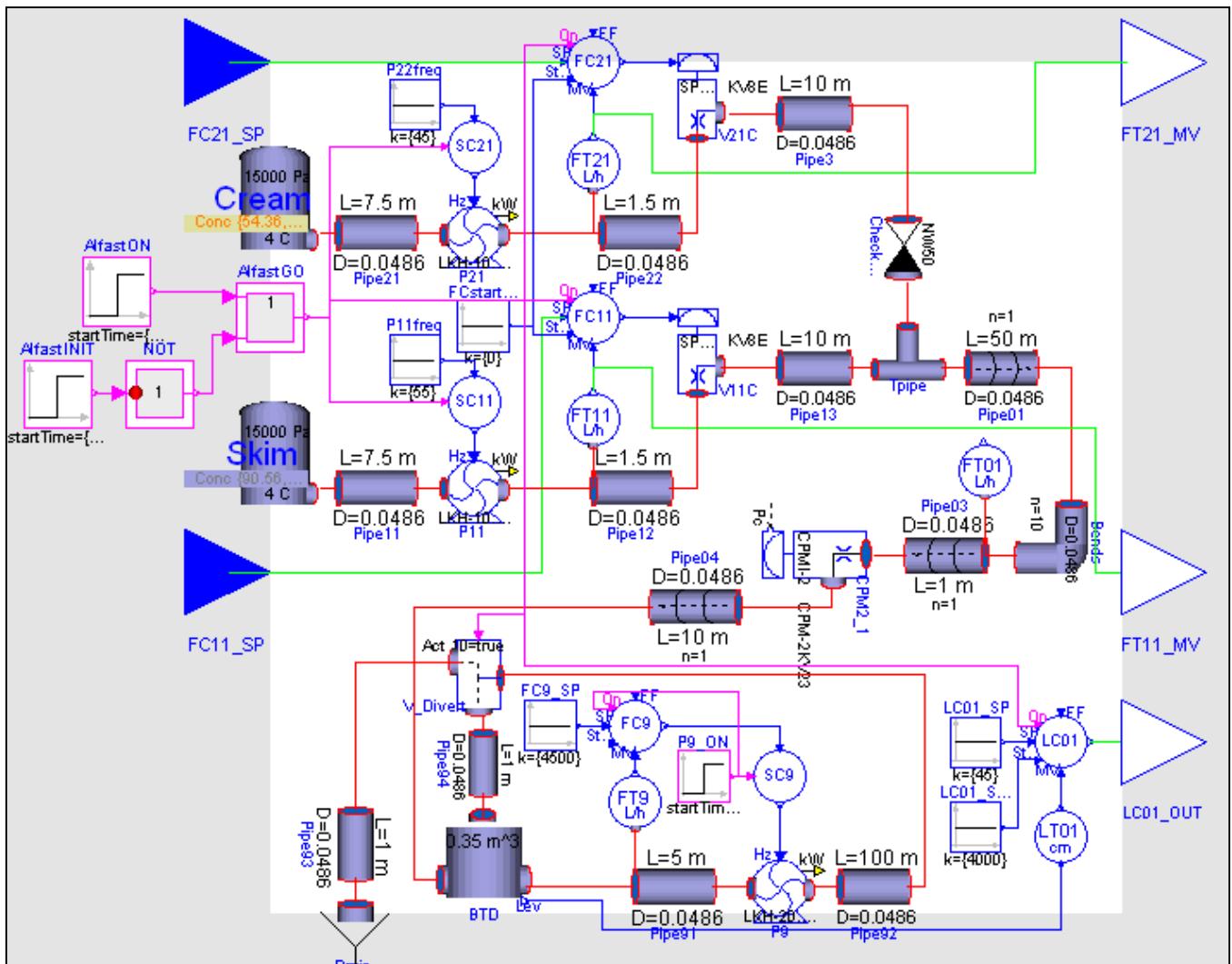


Fig. 9.2 View of the “process system” model for milk blending.

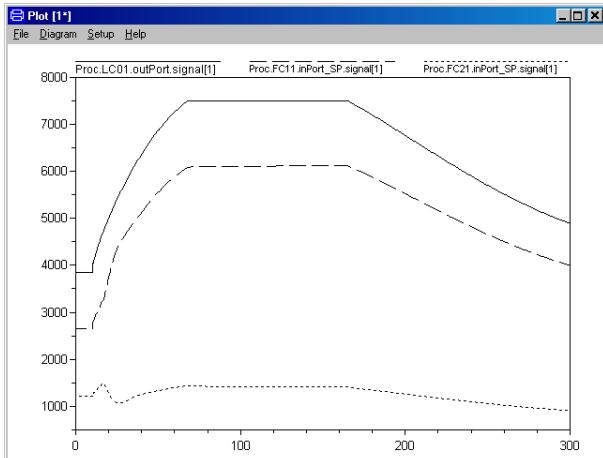


Fig. 9.3 Simulation result of the system model for milk blending. Flow rates: *solid line* = set point of total flow, *broken line* = set point of skim milk flow and *dotted line* = set point of cream flow.

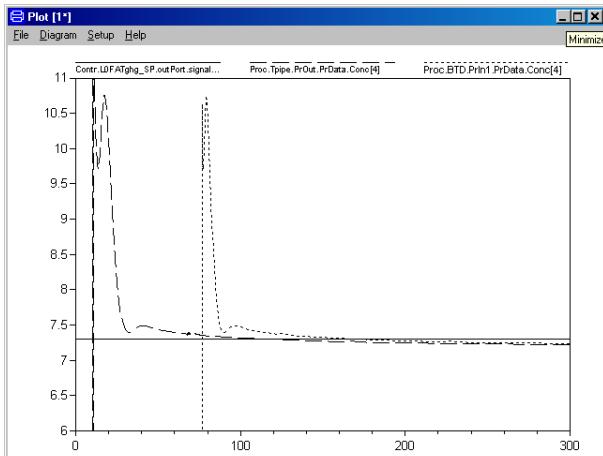


Fig. 9.4 Simulation result of the system model for milk blending. Fat concentration: *solid line* = set point, *broken line* = process value at mixing point and *dotted line* = process value 11 m downstream before a buffer tank.

10. Conclusions

This article has described how simulation has a great potential to contribute significantly to the development of liquid food processing equipment such as:

- pasteurizers for milk and juice
- sterilizers for milk and juice
- milk standardisation systems
- juice blending systems
- aseptic tank systems
- complete lines (evaluation of performance, e.g. product loss)

Modelica/Dymola has shown many advantageous possibilities within the area of liquid food process

simulation. This goes for model/library builders as well as model/library users.

The described “FoodProcessing” library is handling non-Newtonian fluids with characteristics depending on concentration and temperature. It also handles transport delays in fluid channels. Today the library contains about 250 models totally with approximately 2000 equations.

Beside simulation for development of food processing equipment, further potential spin offs have been identified, useful for manufacturers of food equipment:

- training of operators
- education of process and control engineers
- demonstrations and sales
- testing of control systems (hardware-in-the-loop)
- trouble shooting

The development of the “FoodProcessing” library will proceed whereas the question concerning how the potential spin offs are going to be explored, will be answered by the future.

11. Acknowledgements

For discussions, ideas and help; Thank you Carl Cöster, Jonas Eborn, Ivar Gustavsson and Hubertus Tummescheit.

12. REFERENCES

- [1] J. Eborn, *On Model Libraries for Thermo-hydraulic Applications*, PhD thesis ISRN LUTFD2/TFRT - - 1061 - - SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, (2001).
- [2] H. Tummescheit, *Design and Implementation of Object-Oriented Model Libraries using Modelica*, PhD thesis ISRN LUTFD2/TFRT - - 1063 - - SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, (2002).
- [3] M. Tiller, *Introduction to Physical Modeling with Modelica*, Kluwer Academic Publishers, Massachusetts, USA, ISBN 0-7923-7367-7, (2001).
- [4] J. Eborn and K. J. Åström, *Modelling of boiler pipe with two-phase flow instabilities*, In Fritzon, Ed., Modelica 2000 Workshop Proceedings, pp. 79-88, Modelica Association, Lund University, Lund, Sweden, (2000).
- [5] S.M.O. Fabricius and E. Badreddin, *Modelica Library for Hybrid Simulation of Mass Flow Transfer in Process Plants*, In Otter, Ed., Proceedings of the 2nd International Modelica Conference, pp. 225-234, Modelica Association and DLR, Oberpfaffenhofen, Germany, (2002).

- [6] Coulson, J. M. and Richardson, J. F., *Coulson & Richardson's CHEMICAL ENGINEERING Volume 1*, Sixth edition, Fluid Flow, Heat Transfer and Mass Transfer (Butterworth Heinemann, 1999).
- [7] Bolmstedt U., *Viscosity & Rheology – Theoretical and practical considerations in liquid food processing*, New Food, Volume 3 Issue 2, pages 15-20, Russel Publishing Ltd.
- [8] J. Eborn and H. Tummescheit, Modelica library *ThermoFluid* available via the *Modelica* home page www.modelica.org.

Thermo hydraulic library for power systems applications

Martin Råberg

Jan Tuszynski

Carl Bro Energikonsult AB

SE 205 09 Malmö, Sweden

martin.raberg@carlbro.se jan.tuszynski@carlbro.se

Abstract

The thermo hydraulic library presented here has a long history starting in the 70's with dynamic simulations of servo systems and power plants at ASEA (ABB), then with parallel efforts in the 80's at Sydkraft, to finally in the 90's move into the ENERGY library of the Sydkraft group. The library was initially implemented in the Dymola language of Dynasim, and in recent years transformed gradually into Modelica. This paper presents the basic rules and structures of the library, and provides examples of the dynamic modeling ordered by the power industry from Carl Bro Energikonsult AB¹ in Sweden. The examples show both the suitability of the rules of the ENERGY library, and give important feedback of 'lessons learned' for further library development and for identification of missing features of Modelica and generally of dynamic simulation capabilities today.

1 Introduction

The history of modeling energy systems at Carl Bro Energikonsult AB traces back to the application of MMS² by Sydkraft and development of the Dymola-based ENERGY library in the 90's. The library was originally developed to model the complex thermo hydraulic processes of thermal power plants, but it proved applicable to energy systems in general where various fluid media transport energy throughout processes. Such a general "non-intended" application of the library is modeling of the ventilation system of complex buildings. Various rules to model media transportation were developed, and cover today different cases of heat transfer, mixing media, chemical reactions etc.

The structure and rules of the library establish a base for easy use and consistent applications. The rules were defined at the initial establishment of the library and developed further based on practical experience of the library use. We also found out that when people used the library they found it difficult and wanted to take short cuts, e.g. "I can do it simply for this application only", "I have no time to study handbooks...", etc. We are now convinced that this individual approach is the way to trouble – missed quality, reuse not possible, poor documentation, etc.

This paper will firstly present structures, rules and components of the library, and then go through a number of typical models delivered to Carl Bro Energikonsult AB's customers. The examples cover model descriptions, results and 'lessons learned'. Conclusions of our applications address missing features of the Modelica as experienced by us, and general needs for complementary tools required for efficient and cost effective modeling of the energy systems.

2 Energy Lib

2.1 Model structure

The Energy library is a component archive for the basic simulation tool Dymola / Modelica. The foundation of the library is the classic concept of a network of interconnected nodes, or finite thermo dynamical control volumes.

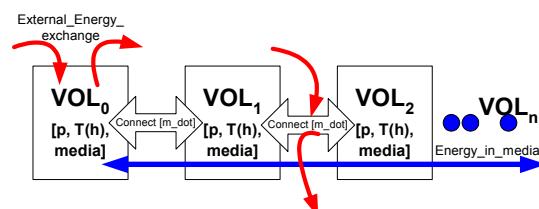


Figure 1 Basic network

¹ Carl Bro Energikonsult was formerly Sycon Energikonsult AB - technical consultants of Sydkraft utility.

² Modular Modeling System, EPRI, Babcock

The state of the media transported through the network, is calculated mainly in nodes, while node-connecting elements calculate mass and energy exchanged between the nodes. The main objective of the modeling is then to simulate energy flows carried in the media and energy flows passed between the media containments and the environment (energy sources and sinks)

The model structure builds then on a number of basic rules / assumptions, where those most important are the following:

- The state of the media (liquid, gas or both) is presented in a state vector of dynamically calculated primary elements: pressure [p], enthalpy/temperature [h/T], and media composition [X].
- Media properties are derived from media ‘tables’ identified by X and [p, T] / [p, h] states. The media property vector and state vector will accordingly provide complete description of the node behavior.
- Each node is identified by the node pointer (node identifier) available through node ports for any component in the network. In the other words, any component of the model can read both node state and node media properties by knowing node identifier only.
- Connecting elements transfer basically media mass flow [w (m_{dot})], and media energy content [h] on the outlet.
- Outlet energy content depends naturally on the inlet energy and on the energy transfer between the connecting element and the environment, and can follow one of the basic “iso-transformations”. Note that all energy content of the media is expressed in the static enthalpy [h]; it is assumed that the media transform all their kinetic energy ($v^2/2$) into ‘h’.
- Connecting elements will normally not change media composition, and accordingly outlet media is assumed the same as on the inlet. This assumption has implications for the simulation of reversible flows.
- Each node (VOL) can change its media through mixing of incoming media and through the chemical reactions between the same
- Simplified nodes are allowed by inheriting selected components of the node state vector of the other nodes. E.g. Pressure calculated dynamically in VOL₀ (figure 1) could be inherited by VOL₁ and VOL₂
- In the same way the connecting element can inherit mass flow from other element, reducing calculations to energy content only

2.2 Structure of the Energy library

The library is composed basically of four library levels.

Level 0: ModelComponent

Level 1: SubUnit

Level 2: Unit

Level 3: System,

Shown in figure 2

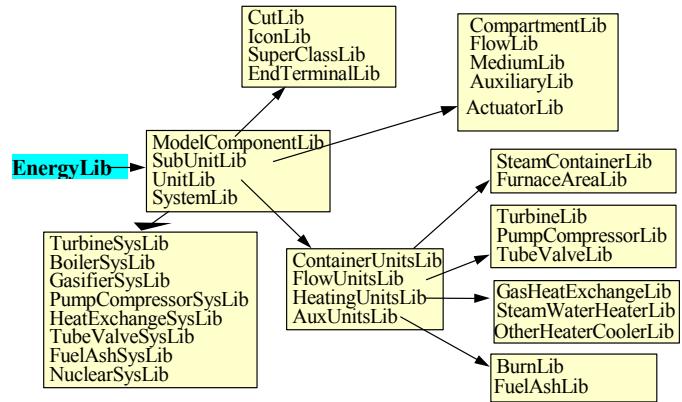


Figure 2 Structure of the Energy Lib

The components level 0 includes various basic sub-components specific for energy models. The original formulation, which builds on the object inheriting features, is now redone to Modelica formulations.

The sub-unit level 1 includes all basic thermodynamical concepts of the basic structure introduced above. The library is divided into four groups: CompartmentLib, FlowLib, MediumLib, ActuatorLib and AuxiliaryLib. Some details concerning compartments (i.e. VOL of figure 1) and flows (connecting elements) are discussed below.

MediumLib covers ‘tables’, or modules describing media properties. Initially the tables could be read directly or indirectly. The direct method means high resolution read-up by direct use of the media properties tables of the external programs. Indirect methods build on the polynomial or splines matching of the selected working area of the media table. The purpose of using polynomials instead of table interpolation is to speed up calculations, especially in calculations of derivatives, as C_p (dh/dT) or the coefficients α_h (dp/dh) and α_p (dp/dp). As media calculations recently generally have improved and the modern algorithms address derivability efficiently, we are going to reformulate our original concepts accordingly.

ActuatorLib, and *AuxiliaryLib* cover various types of valve actuators and e.g. auxiliary calculations of heat transfer between different media and materials. Here the heat transfer dynamics of the walls is represented. Other modules of this group represent chemical calculation (e.g. balance coefficients for different groups of chemical reactions) and calculations of special phenomena as (e.g. gas/steam moisture removal, fast particle separators etc.)

The unit level 2 includes models of machinery and equipment used at power plants (energy processes). The library is structured basically in four groups: ContainerUnitsLib, FlowUnitsLib, HeatingUnitsLib and AuxUnitsLib.

The system level 3 covers mainly complex machinery or whole plants. The library is filled up gradually with models of the actual simulations and only to a lesser extent as a result of library development effort. It should be noted that the specific solutions taken in plant simulation cases are usually the supplier's properties and general availability of those for the Energy library must be negotiated.

Levels 2 and 3 are introduced below through the presentation of the actual simulation cases

2.3 Selected features of the basic components

Basically all models of the Energy library are derived of the local conservation equations (mass, energy and momentum) converted to ordinary differential equations valid for the distinct, separable control volumes of the library modules. This approach can be exemplified on the basic components of VOL and the connecting element.

Node /Volume/

The basic structure of the VOL module is the following:

1. Calculate media property [MP] vector according to the node state vector [p, h, X]. This is basically a call to media 'tables' of the media identified by X. The MP-vector is composed of the normally required property data as e.g. density, entropy, viscosity, and saturation data for steam (x – steam content in water, p_s, T_s, etc). Our tables calculate as well a number of derivate properties, e.g. $C_p = dh/dT$. The derivate used for pressure and enthalpy

calculations are elasticity coefficients dp/dp and dp/dT (ρ – density)

2. Two basic calculations characterizing the particular node can now be expressed in,
 - The sum of all mass flows (Σw_i) connected to the node
 - The sum of all energy flows (Σe_i) passing through the node³
3. As the media in the node is assumed to be in rest (which is actually not necessarily true) mass and energy conservation equations are used here, but in an extensive form valid for the whole volume. Those equations describing dM/dt (M-total media mass in the node), and dU/dt (total internal energy of the node), are converted to state equations of, dp/dt and dh/dt , functions of (Σw_i , Σe_i , X_properties)⁴

Using Σw_i and Σe_i as the inputs to the state calculating equations allows easy adaptation of the basic node model to the particular kind of the sought after module.

$$\sum w_i = \sum_{i=1}^n w_i + \rho \cdot \frac{dV}{dt}$$

and

$$\sum e_i = \sum_{i=1}^n w_i \cdot h_i + Q - W - (h \cdot \rho - p) \cdot \frac{dV}{dt}$$

where:

n	number of ports connected
w_i	mass flow from (-) / to (+) the port
V	node volume
Q	heat energy flow in (+), out (-) of the node
W	work energy flow in (-), out (+) of the node

Please note now that for simple, constant volume nodes $dV/dt = 0$, and no additional heat transfer is expected, $= 0$. On the other hand nodes with moving pistons (as in compressors) can be modeled by adding the term dV/dt , and Q can be given by simple heat transfer through the walls ($A \cdot \alpha \cdot \Delta T$), or by the heat of the chemical reactions (combustion).

Adapting node dynamics to model frequency

It is quite well known that the models should be adapted to the frequency range actual for the

³ Both Σw_i and Σe_i should be treated as 'auxiliary variables' and not strict physical meaning implied by 'mass' and 'energy'

⁴ For single phase media we use states of [p, T]; derivative of dh/dt is then replaced then by dT/dt

particular application. The approach used in the Energy library is through switching off dynamics of nodes of frequencies out of the range simulated. That switching off was done originally by replacing derivatives by residua, e.g. $\text{residue}(p) = \sum w_i$; and $\text{residue}(h) = \sum e_i$. In the modern Modelica version the same effect will be reached by simple zero setting of both $\sum w_i$ and $\sum e_i$.

Elementary ‘Connecting Module’

Connecting element in its elementary form transports media from the inlet to the outlet and behaves according to the equation of the momentum conservation,

$$\frac{d(M \cdot v)}{dt} = w_{in} \cdot v_{in} - w_{out} \cdot v_{out} + (A_{in} p_{in} - A_{out} p_{out} - F_f)$$

For normal frequency ranges $d(Mv)/dt$ can be assumed = 0, and all pressure drop accounted to F_f ; loss on friction. Assuming $F_f = K_{\text{loss}} * w^2$, the basic form for calculation of pipes and valves will get into the form of $w = K * \sqrt{\Delta p}$. Calculation of K is based on the common knowledge of pipe and valve characteristics.

In case media inertia should be considered, the basic momentum equation can be rewritten into a differential equation of dw/dt ,

$$\frac{dw}{dt} = \frac{1}{L} \cdot (A_{in} \cdot p_{in} - A_{out} \cdot p_{out} - F_f)$$

where L is the length of the pipe.

Note that having ‘w’ as a state variable of the connection will actually simplify calculation of F_f , which requires knowledge of the Reynolds number and depends accordingly on the mass flow in the first place.

Special cases of the connecting module

Pretty straight forward calculations of connecting elements get complicated if,

- Compressible media transported at the over-critical pressure drops over the element. This case is solved by introducing in w -form factor Φ allowing similar structure to the one given above; $w = K * \Phi * \sqrt{p_{in}}$. Note that for p -ratios higher than critical the Φ -factor will be constant and ‘w’ will depend on p_{in} only. The form for ‘w’ is not reversible, as the known ‘w’ will not allow calculation of p_{out} . Furthermore the form is strongly non-linear close to pressure ratios 1.

- Junctions, or direct coupling of pipes and valves.

The junction problem can be described as forcing calculations into non-relevant stiff nodes where several pipes meet. Introducing a non-dynamical node described above can solve the problem, which means that we solve algebraic equations instead of integrating state vector derivatives. The library approaches junctions through simple methods of finding resultant C coefficient of the above forms, or by special handling of pipe-valve-pipe group approximating pressure drop over the valve

- Changing energy content of the media along the connection.

For simple connectors we assume that no heat exchange is taking place and accordingly $h_{out} = h_{in}$. This is of course not true in case of a change of energy content in the media. The special modules are provided to calculate outlet energy content at the isentropic (turbine exhaust), isenthalpic or isothermal transitions. The module is strongly coupled to the media table modules

- A heat exchanger is a case of connector where heat of the media is exchanged with the environment. The basic heat flow is simple to calculate as $Q = C * (T_{inside} - T_{outside})$, the problem is anyhow serious as both temperatures are varying along the connector, and lumped parameter approach is not longer valid. Two solutions are applied;

1. By assuming logarithmic temperature profile along the connector
2. By dividing the whole length of the connector in segments, each segment composed of a node and single connector. The nodes of this solution will calculate dh/dt only inheriting average pressure of the boundary nodes. In a similar way, connectors will inherit common ‘w’ and transport changing energy along all segments.

- Examples of our models presented below show the second solution most often applied. The first method takes no consideration of time aspects of stabilizing the logarithmic temperature profile, and can therefore not model the rapid transients we have simulated.
- Chemistry is actually a case of changing media composition when media components are reacting with each other in the node. Typical examples are in burner chambers of gas turbines, or in gasifiers. The problem is addressed through the following:

1. The (dominating) chemical reactions are identified
2. Reaction equilibrium form is defined, with equilibrium coefficient expressed as an empirical function of media state (normally [p,T])
3. Mass balance equation is now expressed in mole form, ΣN_i .

3 Experience and Lessons Learned

3.1 Short overview

All modeling examples introduced here originate from our assignments from conventional and nuclear power plants, from local utilities, or from using simulation models as a validation tool during research of the new concepts of energy systems.

All modeling was done on commercial basis, where costs of the modeling were critically evaluated against potential advantages. The following were the main reasons cited by our customers:

- Tool for designing control systems
- As above, for the control system evaluation including formal validation of concepts proposed
- Preparation of commissioning. Evaluation of tests proposed, selection of controller parameters, etc.
- Training and education

The examples below address those purposes and give the experience feedback of the lessons learned.

3.2 Controller Design

Customer: Barsebäck Kraft AB.

The customer required a model of the process for design and testing of the reactor water level controller for the auxiliary feed-water system. There was no access to the real process during controller development. Controller design through predefined load cases on models using pre-validated equations. The controller parameters were then used on the real process with good result.

At the start of the project it did not include a modeling phase. Parameters from Oskarshamn Nuclear Power Plant should be used with slight adjustments.

The controller strategy is fairly simple, it contains a reactor level controller connected in cascade with a flow controller that acts on a valve. The flow controller can be tested on a cool reactor with a good result. The dynamics of the level control loop changes with the reactor temperature and pressure. This could not be tested on a cool reactor. A heated reactor is expensive and should be in operation.

A model is built to tune the level controller. The controller is tuned to be able to handle predefined load cases in particular ways. To achieve this the model is changed several times as the load cases get more and more complicated. In figure 3 the final model is shown.

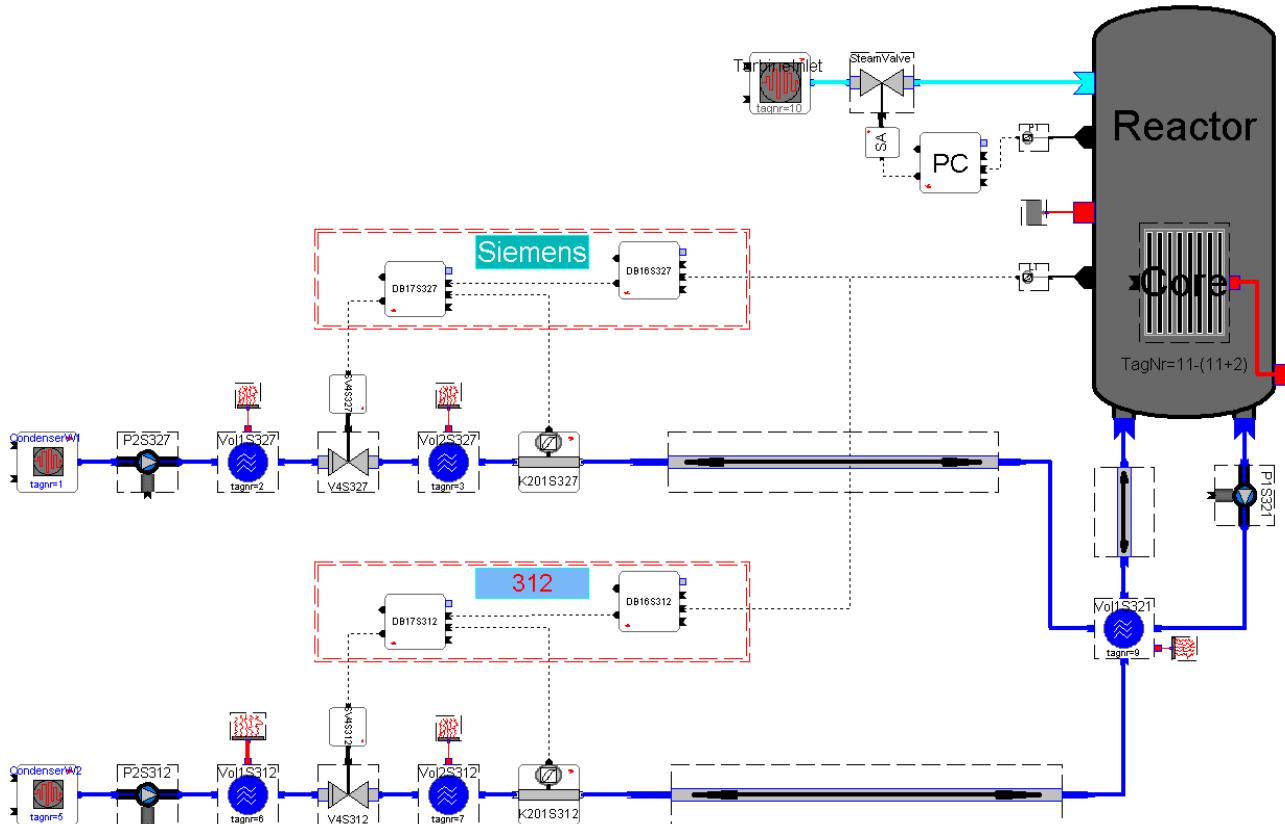


Figure 3. The model of the reactor and the main- and auxiliary feed water systems, (312) and (327).

The reactor model started as a model of an expansion vessel. The model was then upgraded in several stages to accommodate the increased demands on the result.

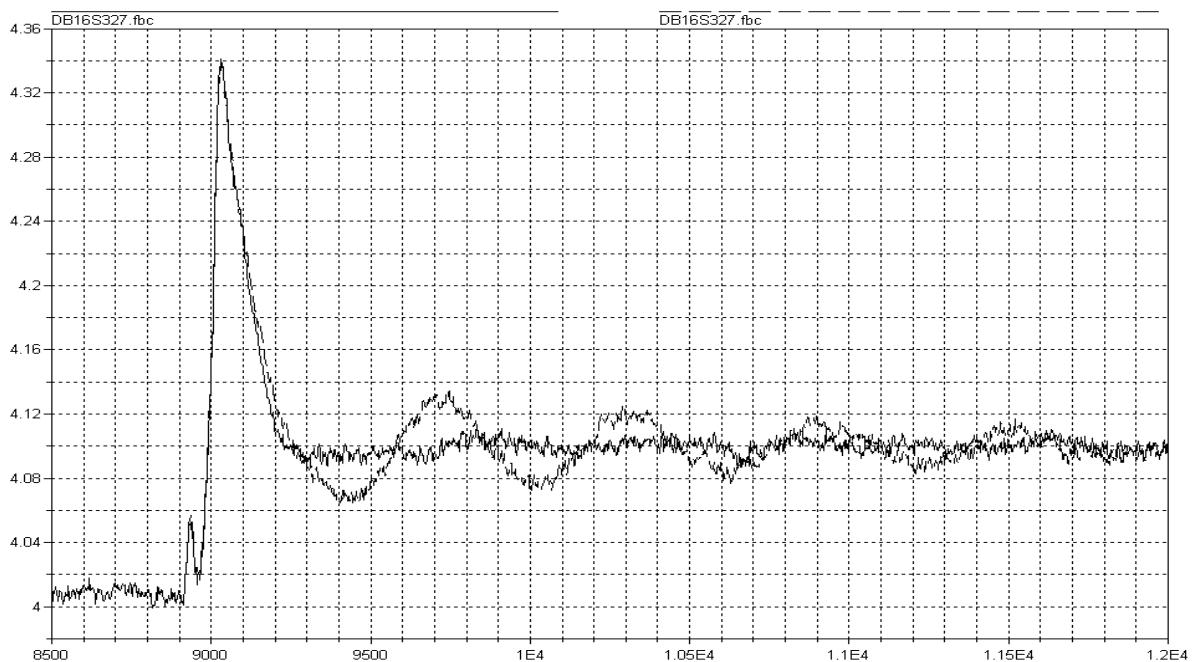


Figure 4. The plot shows the simulated reactor level with two sets of controller parameters. The transient originates from the start of the auxiliary feed-water pumps.

The solid line is the filtered reactor level from the simulated controller. The line is from a simulation with the controller parameters designed through simulation.

The dashed line is the reactor level from a simulation with the implemented controller parameters. The derivative part was decreased in the implemented controller since it was thought to be too aggressive.

The controller implemented today is faster and more robust than the controller used before the start of the project.

Lessons Learned: Pre-validated models can be used in other, not directly related, projects with good result.

3.3 Validation of the new concept

Customer: Elforsk AB and Sydkraft AB, Miljö och Utveckling.

Development and validation of models used to comprise an Evaporative Gas Turbine process (EvGT) model. The plant is a research plant, with extensive instrumentation, located at Lund Institute of Technology. This model includes non-linear processes, e.g. evaporation and condensation into a gas mixture with a fully dynamic gas composition.

The model was developed over a period of several years and started within a licentiate thesis. The plant model is composed of several, separately validated, component models, which consists of several sub models.

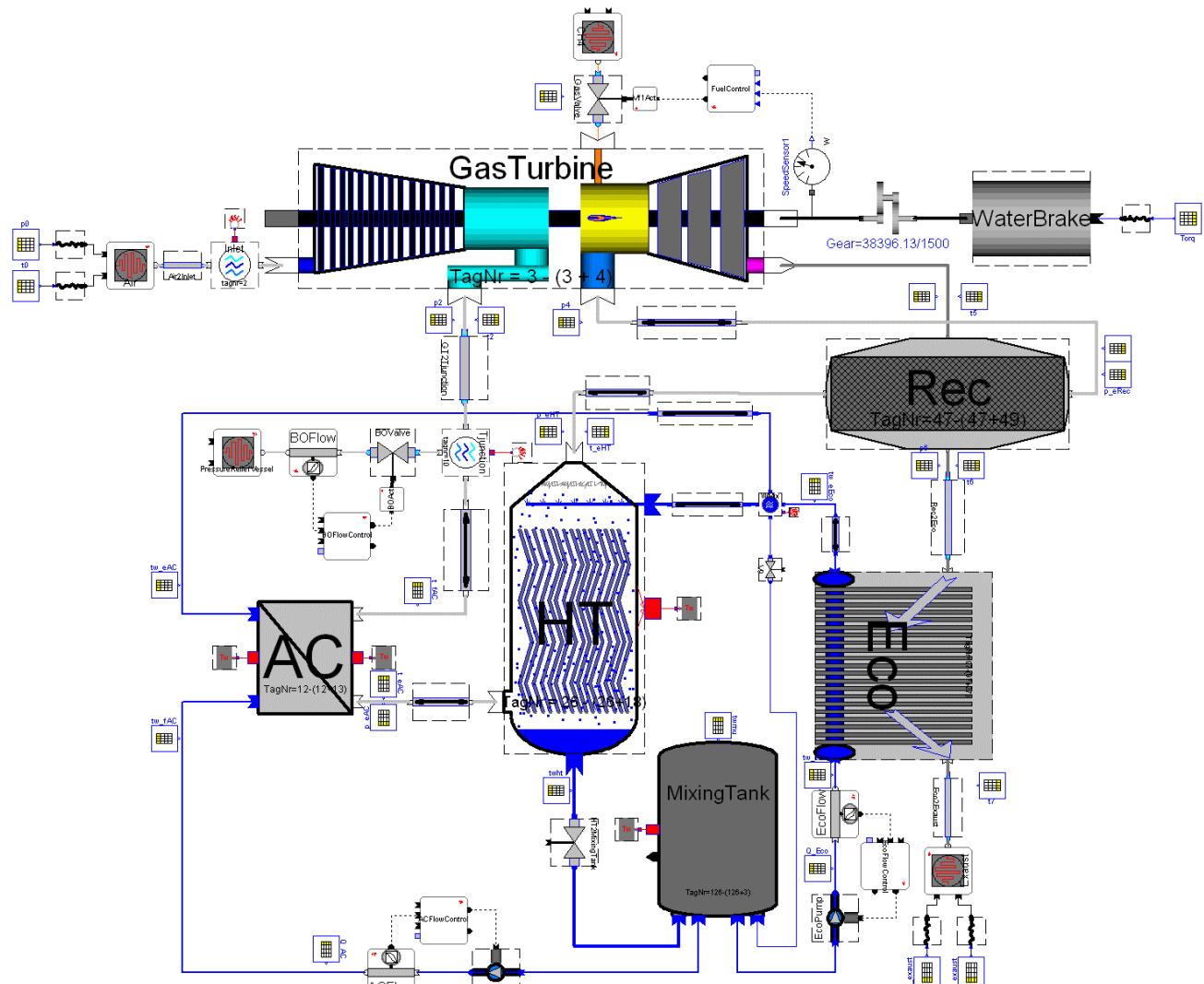


Figure 5 The model of the pilot plant at LTH.

The validation of component models was carried out through test benches. These test benches were fed with series of measurements for flow, pressure, temperature, composition and so on. The result was then compared with the measurements.

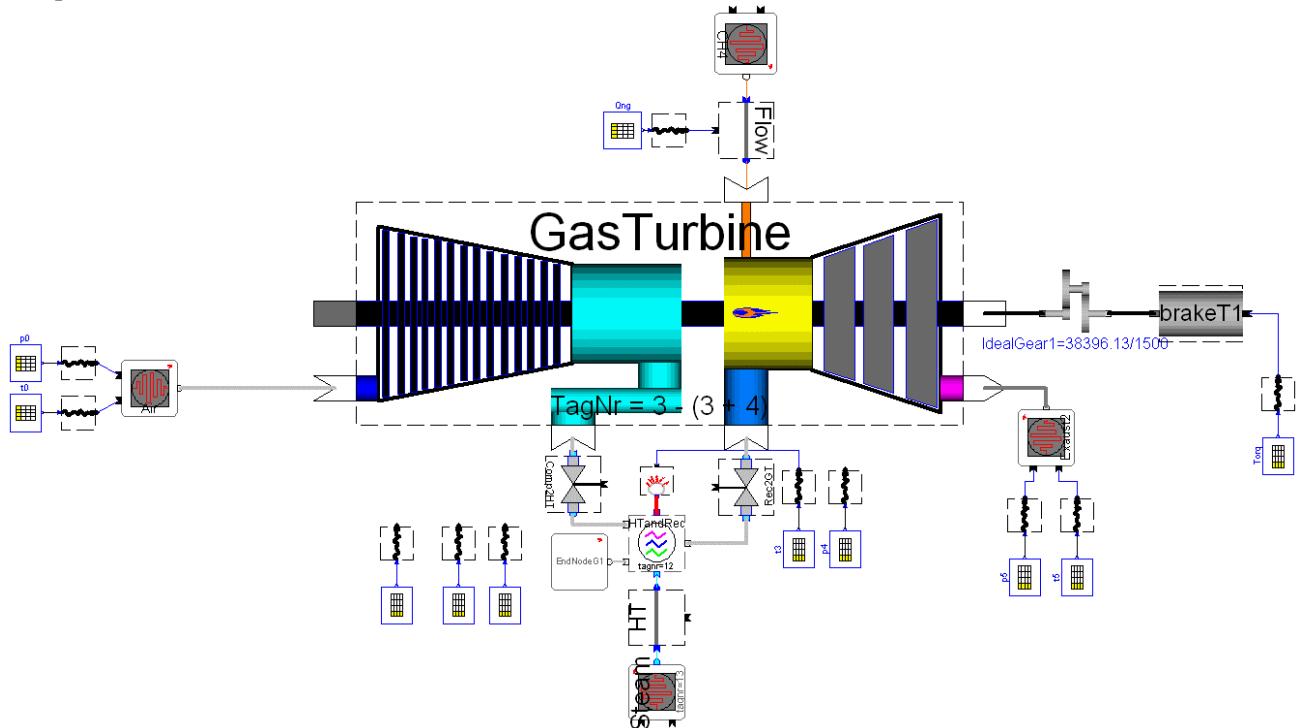


Figure 6 The test bench for the gas turbine.

In the test bench for the gas turbine several simplified component models had to be used to generate good boundary conditions. These simplified component models used measurements during the simulation to get the right boundary conditions. Please notice that the model is fed with measurements of the mass flow of fuel and torque and that the shaft speed is free.

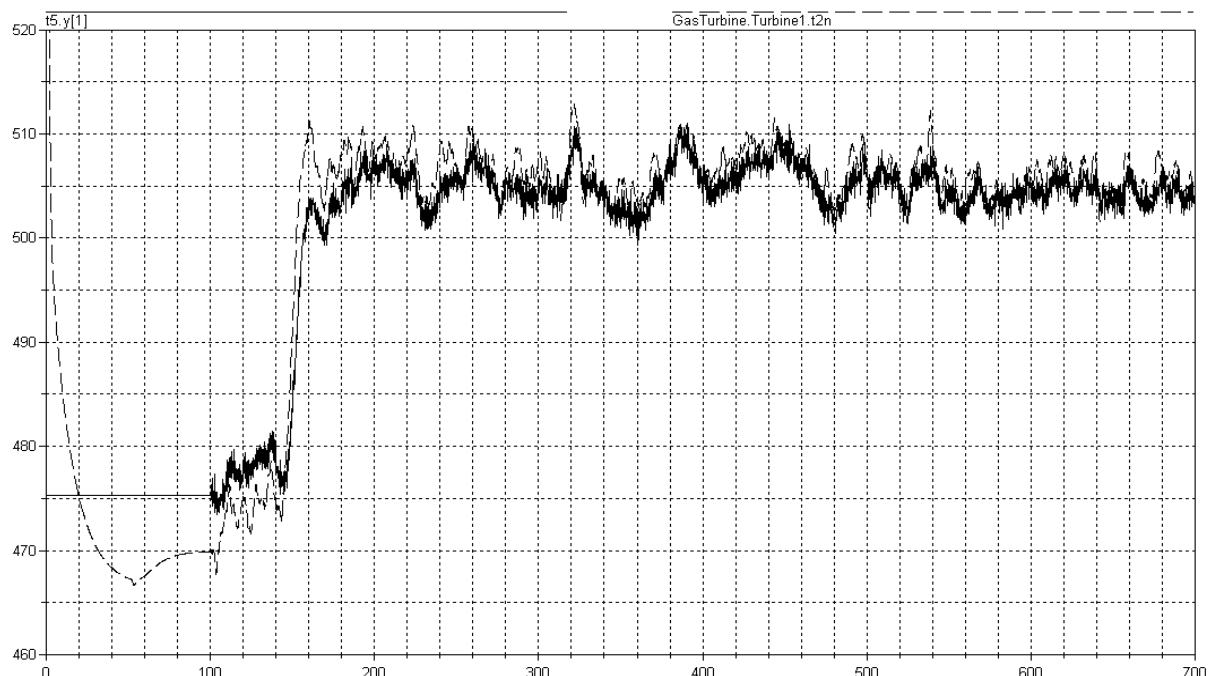


Figure 7 The exhaust gas temperature from the gas turbine in un-validated load case. The solid black line is the measurement and the dashed line is the simulated values.

The reason that there is a mismatch in the beginning is that the initial condition does not correspond with the load case. The load case is a load change from 50 to 60% shaft power. The faster responses that can be observed in the model are thought to depend on the transmitter, which is not included in the model.

The model shall be used to predict test runs on the pilot plant, stability tests and design tests on future plants.

Lessons Learned: The model delivers results with an error within 5% in load cases that the model was not validated against. The dynamic model of the evaporation tower delivers better results than the static design methods used.

3.4 Check of a complex pre-validated model

Customer: Värmeforsk AB (Växjö Energi)

Dynamic modeling of a direct condenser at Växjö Energi. A direct condenser is used to condense steam during a turbine trip instead of letting it out to the atmosphere. This specific direct condenser heats the district heating system, this means that even the heat are used. The direct condenser is exposed to powerful transients almost without any preceding sign. Still it is supposed to keep a stable steam pressure and a steady temperature on the district heating water leaving the condenser.

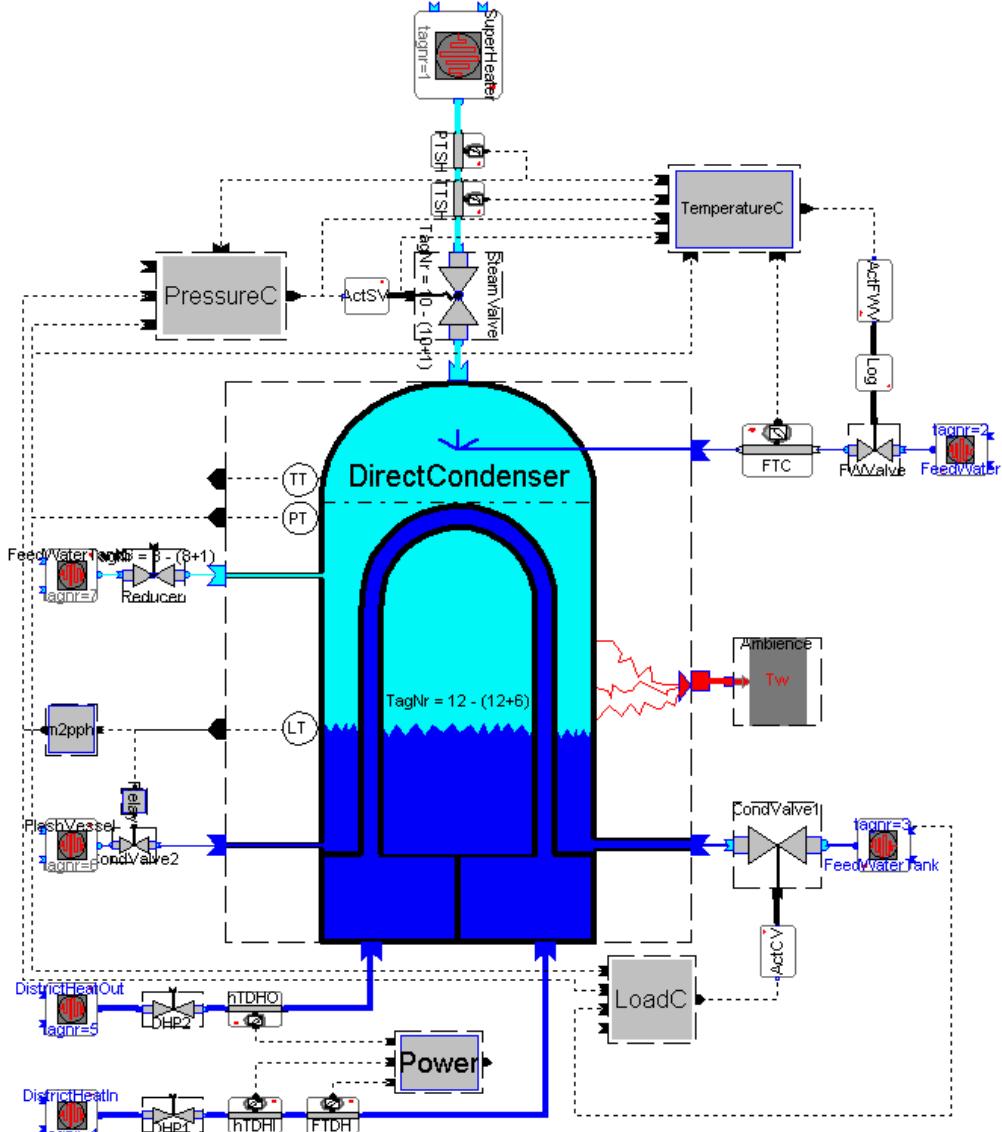


Figure 8 The direct condenser test bench. The control system is modelled as islands according to their function.

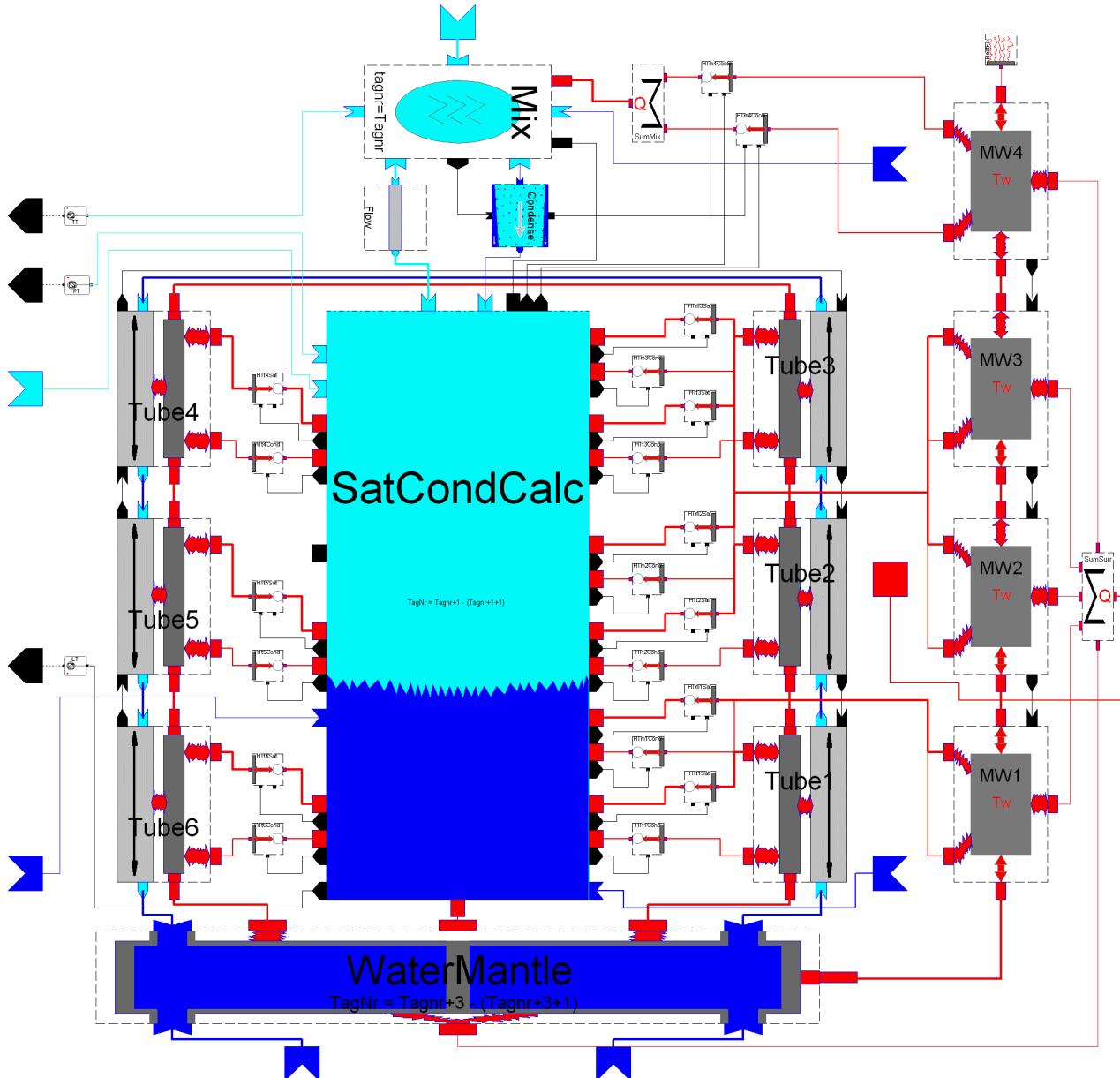


Figure 9 Inside the direct condenser.

The tube model used here handles several parallel identical tubes. It is divided in to six segments to get a temperature profile in the flow direction to use in the heat transfer calculations.

The condenser and the involved parts of the process and control system where modelled using only documentation available before commissioning. When Carl Bro Energikonsult AB was ready the model where sent to Värmeforsk and Växjö Energi delivered measurements from a turbine trip, to be used in the model, to Carl Bro Energikonsult AB.

There are some assumptions, e.g. regarding the heat transfer during condensation on vertical tubes, which were not tuned to this particular case. Normally the uncertainty of a heat transfer calculation is ± 10 to 20% .

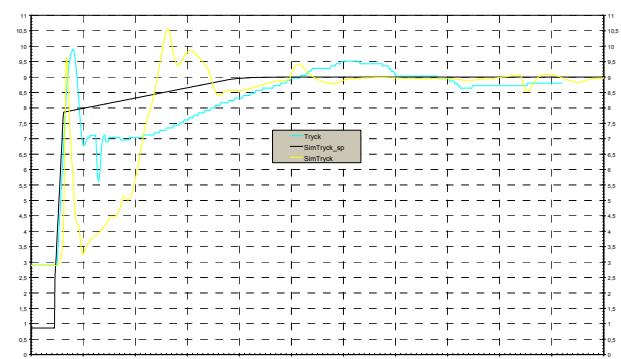


Figure 10 The pressure in the direct condenser in bar.

In this case dynamic factors of such complex processes as the build up of the condensate film on the tubes have to be considered.

Lessons Learned: Although not a perfect fit the model delivers a result good enough to allow tuning of control parameters and preventing design and commissioning problems.

3.5 Modeling of a small project (pressed for time)

Customer: Sydkraft Värme Syd AB:

Testing of the control scheme for solar collector system with a total area of 1 200 m² with demands on high availability. The problem was to interconnect five separate solar panels. The panels are an integrated part of the walls on a recreation facility named Kockum Fritid.

This modeling was part-task in a project stage pressed for time and crucial for the final design of the system. As a result of the wall integration collectors faced east, south and west.

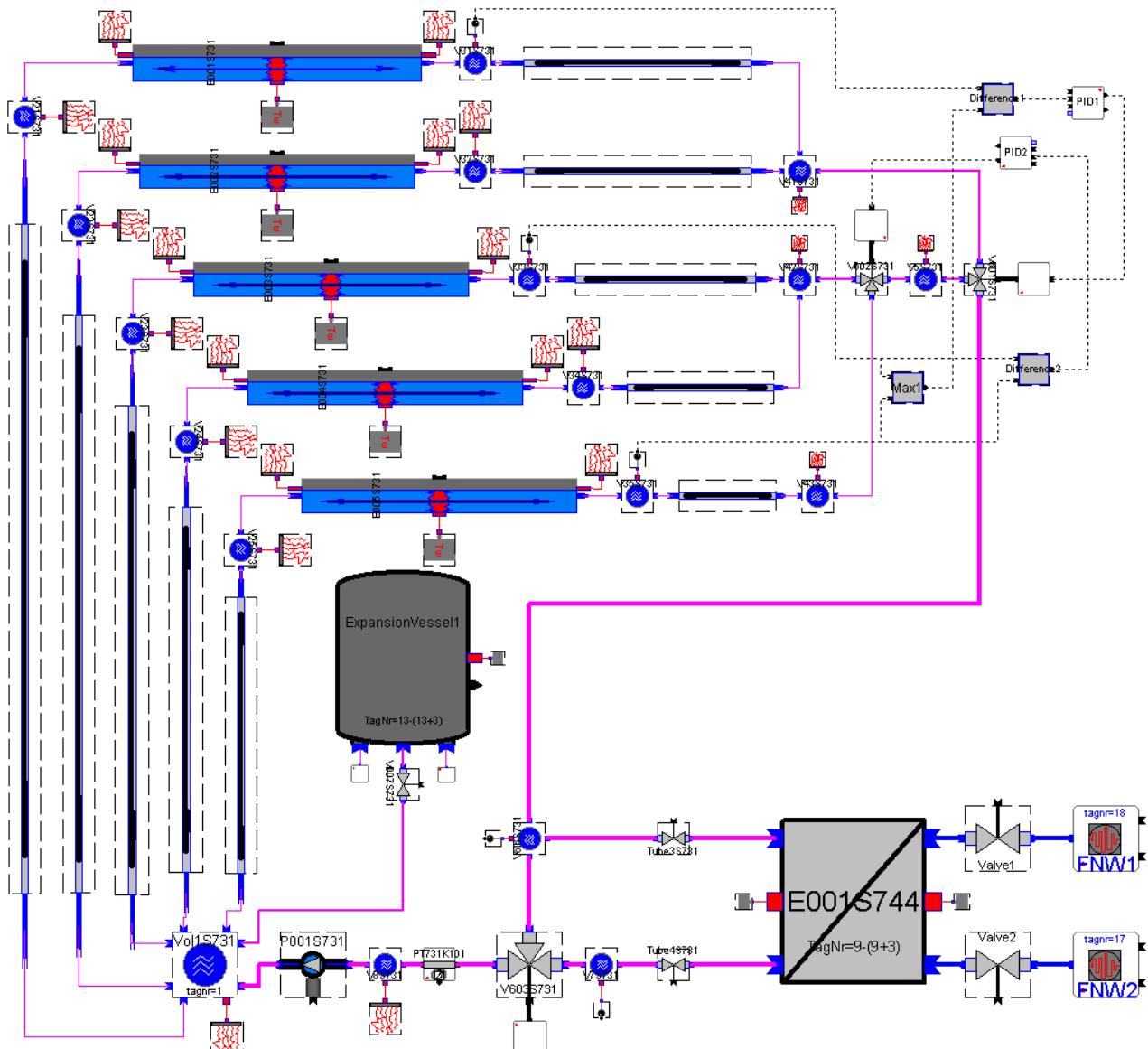


Figure 11 The model of the solar collector side of the system.

This first model was too complex to handle in this project. The decision to go right to the core of the problem was taken. This meant that the design work should carry on as in a normal project but the question if the flow

from all solar collectors could be mixed should be answered through simulation. The model used to answer the core question is shown below.

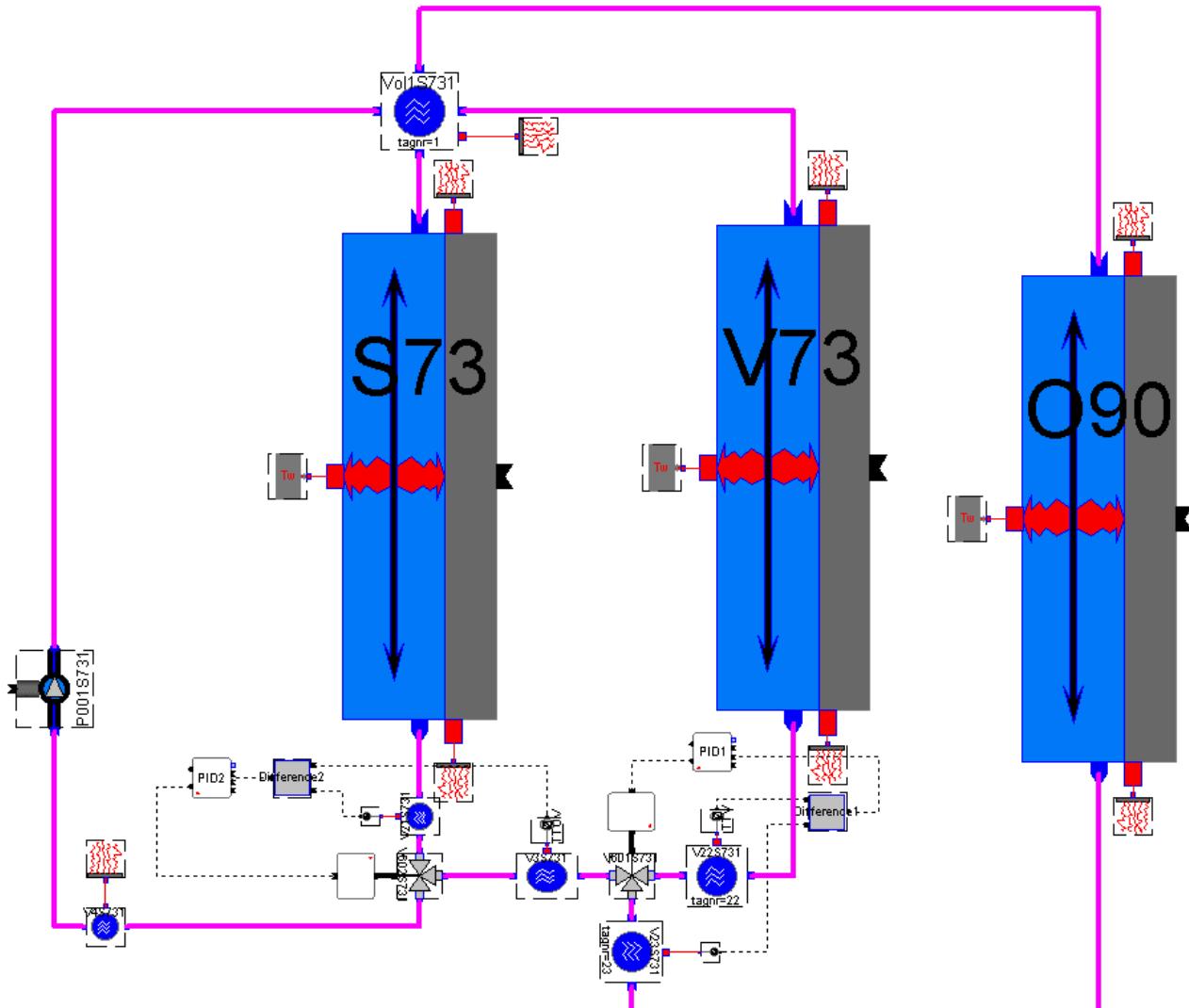


Figure 12 The basic model of the solar collector side of the system.

From this model the conclusion that the solar collectors could be connected to one system was taken. While the solar collector experts recommended a solution with five completely separate systems, the selected solution validated in the model, showed to be more efficient and cheaper, more robust and easier to maintain. The final system has a documented availability well above 99%.

Lessons Learned: The use of simulation can have a profound influence on the outcome when used in the early design phase of a project. Simulation can be used as a design tool even in small projects pressed for time and money.

3.6 Design through simulation.

Customer: Sydkraft Värme Syd, Kungsbacka

Simulation of a typical district heating system with several production units and an atmospheric heat accumulator, allowing evaluation of the complete process architecture, including design data and control system. The main idea behind the simulation was to study the interaction between the atmospheric heat accumulator, the boilers and the rest of the district heating system. The atmospheric heat accumulator has two functions; to store and distribute heat and maintain a constant pressure in the system.

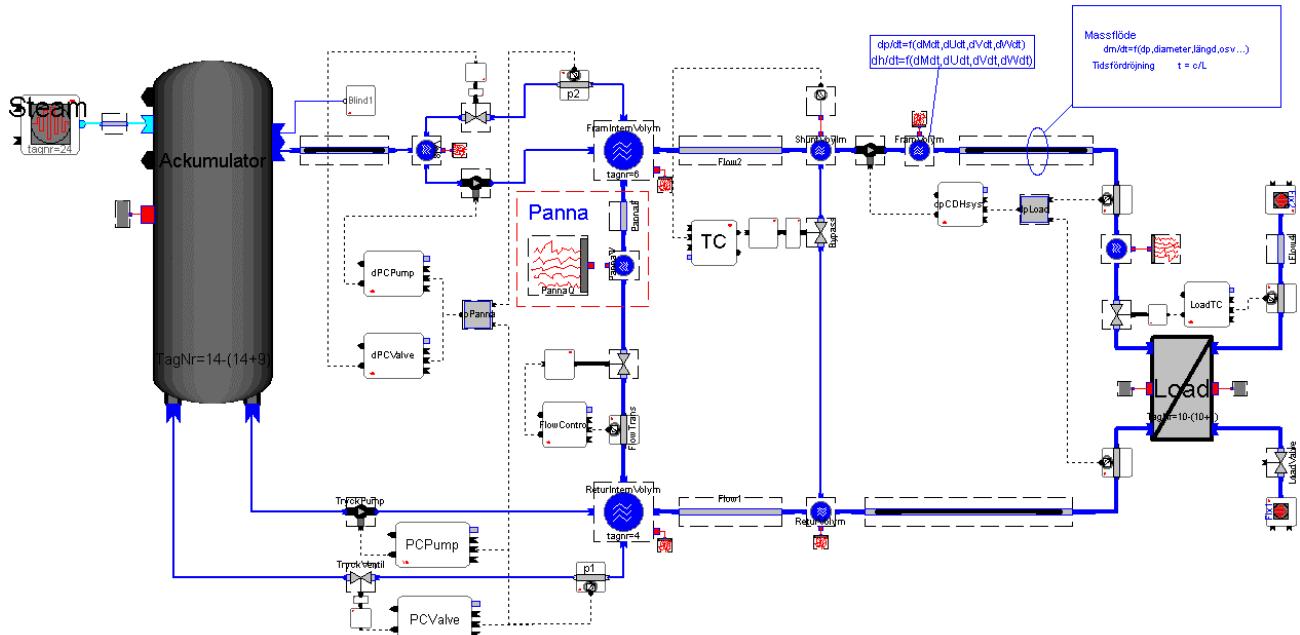


Figure 13 The model used to simulate the interaction of the atmospheric heat accumulator and the rest of the district heating system.

The model showed that some of the valves were too small and that there is a problem in determining the minimum pump speed. Besides this, the model delivers approximate controller parameters.

The load case shown in figure 14 and 15 is a boiler brake down during loading of the accumulator. The first transients are caused by the fact that the initial condition does not correspond with the load case.

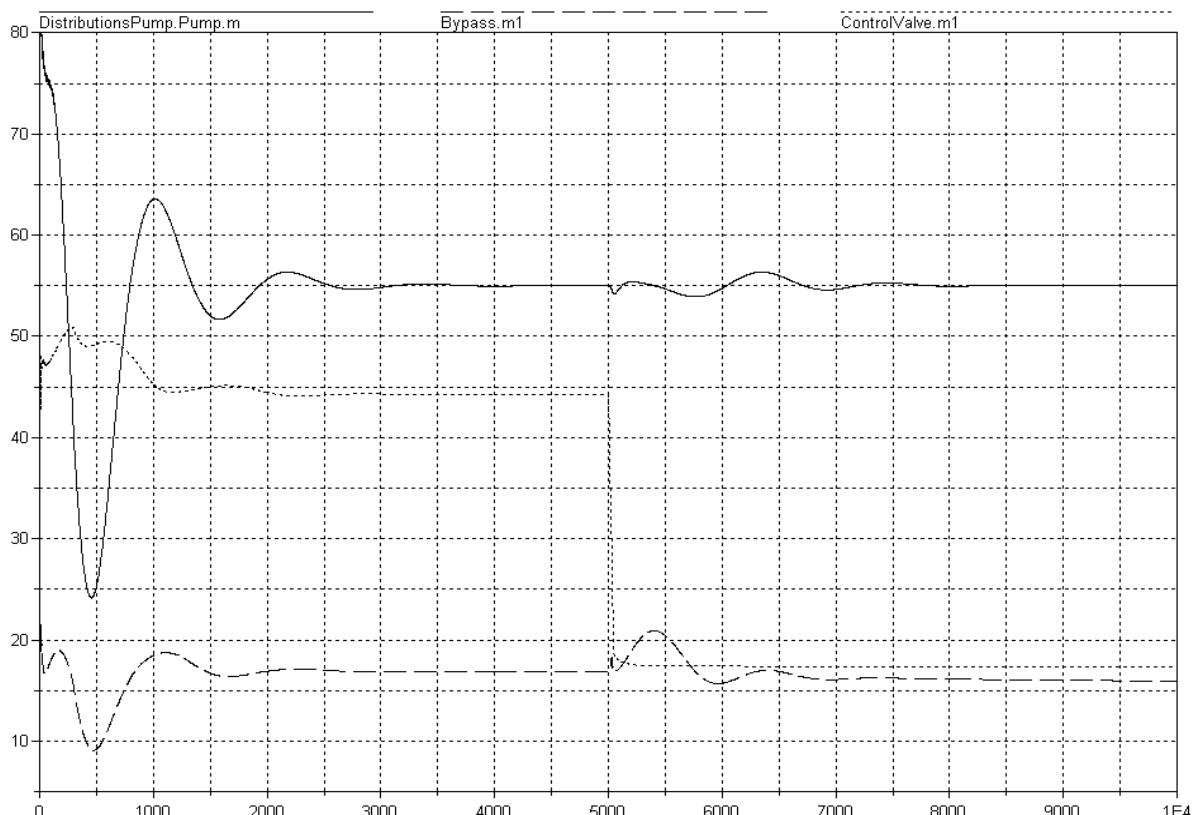


Figure 14 The mass flows in the district heating system.

The solid line is flow through the distribution pump, the dashed line is flow through the bypass valve and the dotted line is the flow through the boiler.

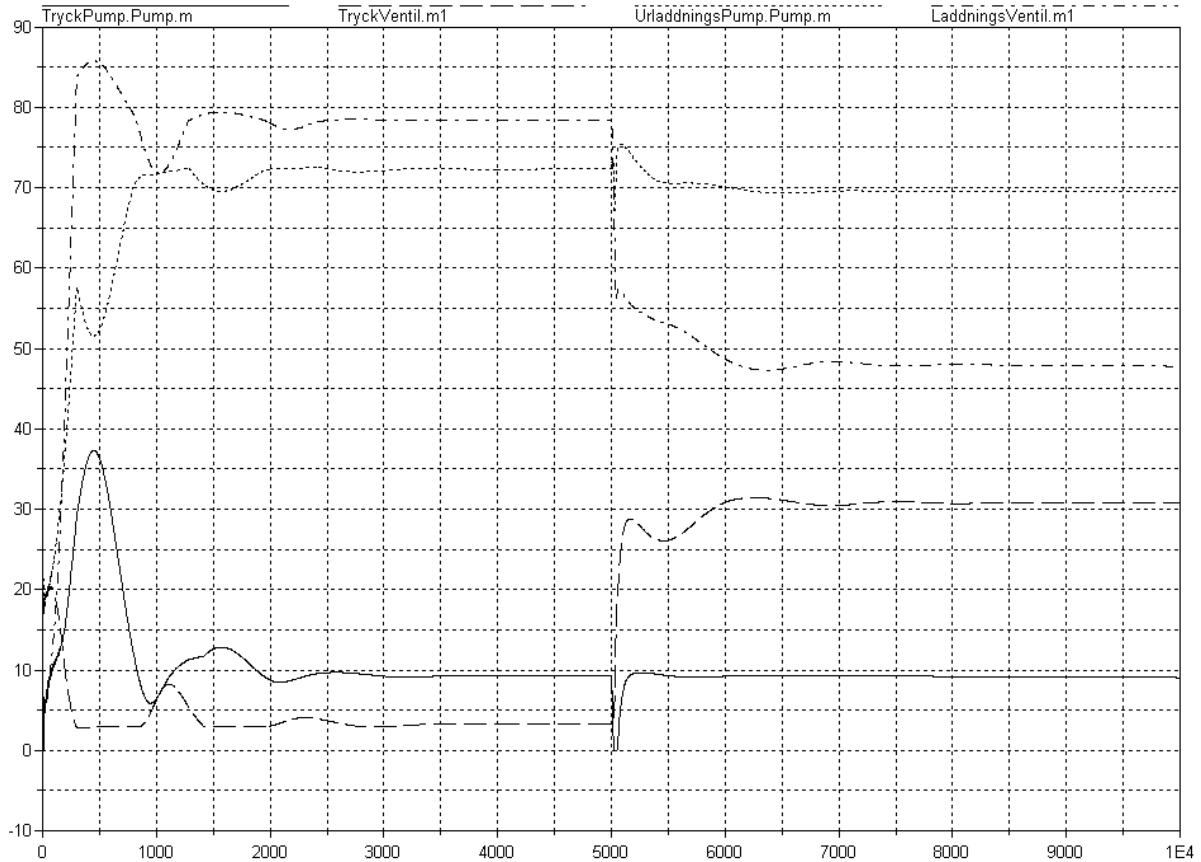


Figure 15 The mass flows in connection with the atmospheric heat accumulator

The solid line is flow through the pump used for pressurization and the dashed line is flow through the pressure control valve. The dotted line is the mass flow through the pump used for un-loading heat and the dash-dotted line is the flow through the valve used for loading heat.

Lessons Learned: The method works and the results were trusted.

4 Conclusions

This paper provides a number of examples that Dymola / Modelica is well suited to industrial modeling of Energy systems. Our experience shows that the technical and calculation issues can be addressed and solved, and that the simulations show a very high degree of correspondence between models and measurements.

In the projects above it has been proven that the method is commercially competitive. This is a possibility only thanks to the structured Energy library, providing not only reusable components but also thoroughly tested modeling methodology.

We still need to improve efficiency of the modeling, mainly in two areas. The first one is the degree of common understandability – here mainly making systems simple enough to allow process engineers to use models in their daily work of designing, validating and commissioning.

The second is in the area of tools facilitating modeling and simulations. A tool for calculation of the initial, start-up conditions of the complex systems we work with is our primary request.

Modelica development moves certainly in the direction fulfilling our needs, and we are today fully committed to base our future modeling and library development on both Dymola tools and Modelica.

Session 3A

Automotive Simulation – II

Development of a Vehicle Model Architecture in Modelica

Michael Tiller[†]

Paul Bowles[†]

Mike Dempsey[‡]

[†]Ford Motor Company, Powertrain Research Department

[‡]Claytex Services Limited

ABSTRACT

The real power and flexibility that comes from using Modelica for physical modeling stems from the combination of the acausal approach to formulating physical connections combined with sets of standard connector definitions in various engineering domains. These features are important because they help avoid *a priori* causality assumptions (which promotes reuse of components) and ensure physical compatibility across connections. However, complex systems are generally made up of several complex, multi-domain subsystems with numerous connectors. Such systems also benefit from having standardized subsystem interface definitions. This paper will focus on an initial proposal for a vehicle model architecture for vehicle system applications. Ultimately, we hope that feedback on this proposal from other groups doing vehicle modeling will lead to a consensus on the appropriate subsystem interfaces such that we can achieve the same level of flexibility and reusability for vehicle subsystem models that we currently have with component level models.

1 Motivation

Vehicle system modeling is an important part of optimizing overall vehicle performance. To avoid building up complete vehicle models from scratch repeatedly, it is useful to develop a pre-wired vehicle model architecture. We had two goals in mind when formulating such a vehicle model architecture. First, it should allow the exchange of subsystem models between different organizations (e.g. part/subsystem vendors, design organizations, universities) without the need to "rework" the models to fit into existing vehicle system models. Second, it should greatly simplify the handling of alternative vehicle system configurations by allowing substitution of one particular subsystem or strategy implementation for another.

Ideally, we hope that this architecture will develop to the point that other groups, outside of Ford, will adopt it. Given the growing number of automotive related libraries in Modelica [1-4], both freely available and commercial, such a vehicle model architecture will be a practical necessity to allow subsystem models from these libraries to be easily assembled into complete vehicle models.

Previous efforts at Ford have focused on providing a vehicle model architecture for models developed in Simulink [5]. While not disputing the value of a corporate standard for vehicle subsystem models, groups working with Modelica were not willing to give up the acausal flexibility in Modelica for an approach that required *a priori* causality assumptions. Furthermore, most existing vehicle level modeling applications using Modelica at Ford involved details (*e.g.* modeling the motion of the powertrain mounts) that were not possible with the Simulink framework.

As a result of internal discussions, it was agreed that an acceptable compromise would be to develop a purely Modelica architecture using essentially the same subsystem decomposition, as was done in Simulink, but avoiding *a priori* causality assumptions. In cases where Modelica models would be useful to someone working in Simulink, we hope to develop a set of standard "wrappers" for each subsystem that will allow us to impose the required causality on an otherwise acausal subsystem model and then convert these into an S-function using Dymola [6].

2 Architecture Structure

A complete vehicle system model must take into account the response of the various physical subsystems, the function of the controller modules (both subsystem and vehicle level) as well as other "external" influences like the environment and the driver. The following sections will discuss the decomposition in each of these categories.

2.1 Physical Subsystems

The first category we will be discussing includes all the physical subsystems in the vehicle. This section will provide some discussion for each physical subsystem and some explanation of what is contained within each subsystem. The order of the subsystems corresponds, roughly, to the order that they appear (from left to right) in Figure 1.

Note that each physical subsystem is connected to a subsystem controller. We will defer the discussion of this connection until Section 2.2.3 and instead focus, for now, on the physical connections associated with each subsystem.

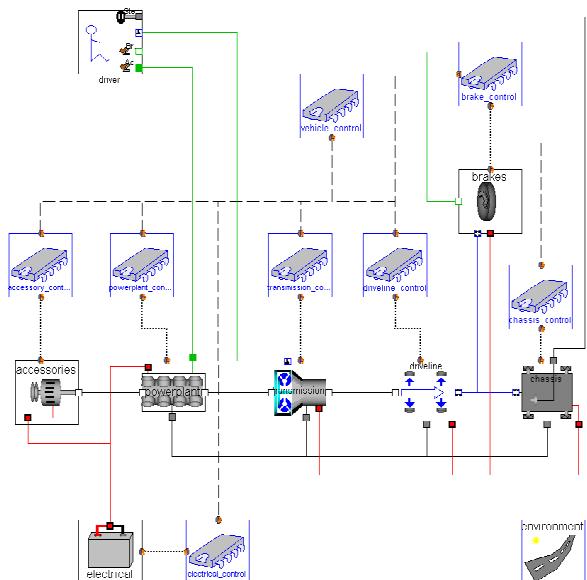


Figure 1: Vehicle Model Architecture

2.1.1 Accessories

The accessory subsystem is composed of those components typically connected to the front end accessory drive (FEAD) of an engine. Examples of such components would include an alternator or AC compressor. As shown in Figure 1, the accessories are connected to the front side of the powerplant. As a result, any torque required by these components will be taken from the powerplant. The accessories are also connected to the electrical subsystem and they typically represent a significant influence on the charging and discharging of the electrical system.

2.1.2 Electrical

The electrical subsystem is composed of the various purely electrical components in the vehicle. Typical examples would include the battery, radio and/or headlights. In addition to being the location

for all purely electrical components, the electrical system is also the source of electrical power for every other physical subsystem in the vehicle and, as such, is subject to "external" influences that may charge or deplete the battery (*e.g.* alternator, regenerative braking).

2.1.3 Powerplant

The powerplant subsystem represents the primary source of motive torque for the vehicle. Typically, this would be an internal combustion engine although it could also be, for example, an electric motor. Like the battery, the powerplant model provides power to the rest of the vehicle. As such, there are physical connections from the powerplant to the accessories and the transmission.

The powerplant is also connected to the electrical subsystem. Although the electrical influence of an internal combustion engine is normally quite small (*e.g.* spark plug energy, *etc.*), if the powerplant were an electric motor, the connection to the electrical system would become quite important. In the case of hybrid electric vehicles, additional electrical components, such as electric motors, may be included in the powerplant or they may be lumped into the transmission (depending on the powertrain topology).

The physical connection between the driver and the powerplant includes a signal representing the physical position of the accelerator pedal. Typically, this signal is translated directly into a throttle position. However, in "drive by wire" applications, it is assumed that the pedal position sensor would be associated with the powerplant subsystem and that sensor information would be relayed to the powerplant subsystem controller and/or vehicle controller where, for example, the commanded throttle position (or motive torque, in the case of an electric vehicle) would be calculated and returned as an actuator command.

Finally, Figure 1 shows that the powerplant has a third mechanical connection. This connection is to the powertrain mounts and accounts for reaction torque to the powertrain mount system.

2.1.4 Transmission

The transmission subsystem represents any "gearing" done to deliver power from the powerplant to the wheels. One side of the transmission is connected to the powerplant while the other side is connected to the driveline. Any hydraulic function associated with the transmission is assumed to be encapsulated within the transmission subsystem.

Like the powerplant, the transmission is also connected to the powertrain mounts. This is an important aspect that differentiates this architecture from most vehicle level models because it accounts for the influence of reaction torques in the powerplant, transmission and driveline on the motion of the powertrain. This 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 [11].

As with all the physical subsystems, the transmission subsystem is connected to the electrical subsystem. In addition, the transmission is also connected to the driver. The driver connection represents the shifting mechanism for either a manual or automatic transmission depending on the configuration options chosen for the vehicle (these will be discussed later in Section 3.3).

2.1.5 Driveline

The driveline subsystem is responsible for modeling 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).

Physically, the driveline is connected to the output side of the transmission and generally has the potential to influence each of the wheels. In order to avoid a complex series of graphical connections, all wheels are lumped into a single connector which is also physically connected to both the brake and chassis subsystems. Note that the driveline subsystem is also connected to the mounting system and the electrical system.

2.1.6 Brakes

The brake subsystem represents not only the friction used to decelerate the vehicle but also, as with the transmission, any encapsulated hydraulic function. The brake subsystem is physically connected to each wheel (*via* the single connector described in Section 2.1.5), the electrical subsystem and the brake pedal (associated with the driver). As with the powerplant, the connection to the driver could represent either direct actuator control by the driver or a "brake by wire" configuration where the brake pedal position sensor would be contained in the brake subsystem with pedal position information communicated to the brake subsystem controller and/or vehicle controller.

2.1.7 Chassis

The chassis subsystem represents the vehicle body, frame, wheels and suspension system. One remaining issue with the decomposition described in [5] is the handling of the steering mechanism. It is still an open issue what the physical interface between the steering mechanism and the suspension system should be. For now, we have kept the steering components inside the chassis while we collect feedback from experts on the best way to separate these two systems.

While for many applications the chassis may be modeled as a simple unsprung mass constrained to move longitudinally, the goal of this architecture is to provide sufficient flexibility to accommodate complex vehicle dynamics models ([1, 9]). The chassis subsystem is physically connected to the wheels and also to the powerplant, transmission and driveline through the mounts. The modeling of the mounts is handled inside the chassis system. Furthermore, the actual physical type of the mounting connections is configurable (*e.g.* 1D, 3D, *etc*). The modeling of the road-tire interface is also handled inside the chassis subsystem.

Physically, the chassis system is also connected to the electrical system and the steering wheel. As with the brake and powerplant models, the connection to the driver may represent a "by wire" connection.

2.2 Controllers

While analysis performed during the subsystem design process can sometimes be accomplished using simple open-loop control strategies for a single subsystem, it is much more important that vehicle level models include closed-loop control to capture communication between each subsystem plant and controller pair as well as physical interactions across the various physical subsystems.

The subsystem controllers are decomposed along similar lines as their physical counterparts. Rather than categorize the controllers by subsystem, we will focus on the controller hierarchy and how the controllers communicate both with each other and with the physical subsystems.

2.2.1 Vehicle System Controller

This vehicle architecture includes a hierarchy of controllers. At the top of this hierarchy is the vehicle system controller. The vehicle system controller exists to control vehicle level functions and deal with arbitration and apportioning of subsystem functions (*e.g.* balancing how much

motive torque is delivered from the internal combustion engine versus how much is delivered by electric motors in a hybrid electric vehicle).

In order to function, a vehicle system controller (if present, not all vehicles implement one) must communicate with each of the subsystem controllers on the vehicle. In an actual vehicle, this kind of communication would be done through a vehicle level communication bus (*e.g.* a Controller Area Network, or CAN, bus). Although the behavior of the bus itself can have a significant impact on overall vehicle performance, modeling of the bus is not currently within the scope of this architecture.

2.2.2 Subsystem Controllers

As shown in Figure 1, associated with each physical subsystem is a controller for that subsystem. These controllers are responsible for controlling the function of their particular subsystem. For example, for a vehicle with an internal combustion engine, the powerplant subsystem controller would be responsible for determining spark timing, injector timing and other specialized functions like cam phasing control.

Each subsystem controller must communicate with its associated physical subsystem to exchange sensor and actuator information. In addition, each subsystem may receive supervisory commands from a vehicle system controller. Finally, the architecture should accommodate any combination of continuous controllers (*e.g.* formulated using block diagrams) and/or discrete controllers (*e.g.* employing Petri-nets, z-domain blocks or embedded code).

2.2.3 Communication Buses

As mentioned previously, bus behavior can have a significant impact on vehicle performance. Although we would like to capture these effects, we feel it is important to focus initially on the interactions between the physical subsystems and controllers.

Even if we ignore the behavior of the communication bus, we still need to represent the information exchanged on the bus. This is complicated by the fact that each subsystem design can potentially have a wide variety of signals that must be communicated between the subsystem controller and its physical counterpart. For example, one powerplant may contain an internal combustion engine that has cam phasing while another one does not (while a third may have an electric motor as a powerplant and therefore an entirely different set of sensor and actuator signals).

For each case, the subsystem controller must have the appropriate architecture to deal with the varying sets of sensors and actuators in each case. As a result, the set of signals exchanged between the controller and its physical counterpart must be customizable on a per configuration basis.

In a similar way, the information exchanged between the vehicle system controller and each of the subsystem controllers will also depend on whether a vehicle system controller is present and, if so, what features are implemented at the system level.

2.3 External Influences

Apart from the physical subsystems and controllers, a vehicle system model must account for two important external influences. The first influence is the driver. While the driver is not strictly part of the vehicle, the driver obviously has a tremendous influence over the response of the vehicle. The other external influence is the environment. The environment could potentially influence things like air temperature and composition (used in predicting engine performance), road surface effects (*e.g.* changes in elevation, traction characteristics), obstacles or other vehicles (potentially necessary in evaluating intelligent cruise control and other active safety features).

In some sense, the driver is both a physical subsystem and a controller. Both of these functions are lumped into a single driver model. The environment is assumed to be purely autonomous typically based purely on time and vehicle position.

3 Modelica Features

3.1 Acausal Modeling

The rich set of physical modeling and configuration management features associated with the Modelica modeling language [10] provide great potential for vehicle system analysis [11].

Vehicle systems are typically modeled from either a "forward" [12] or "backward" [13] perspective. This limits the reusability of component models because they must be developed with these perspectives in mind. From a purely physical perspective, the ability to build components and subsystems without *a priori* causality assumptions allows these components and subsystems to be used in both "backward" and "forward" vehicle modeling applications. Beyond

the reusability of components that results from this acausal approach, the use of inheritance, subtype constraints and the ability to declare replaceable components and subsystems is often useful in practice for large scale modeling projects. In this section, we will discuss how these features allow us to satisfy important requirements for our vehicle model architecture.

3.2 Replaceable Subsystems and Controllers

The cornerstone of configuration management in Modelica is the ability to declare types and components as **replaceable**. In fact, all the physical subsystems, controllers and external influence components shown in Figure 1 are declared replaceable so that alternative configurations can be easily created. Furthermore, constraining types are also defined for each of these components to prevent inappropriate substitutions from being made.

One problem with making each component **replaceable** is that it leaves open the possibility that novice users will attempt to pair plant and controller models together that are not compatible with each other (*e.g.* the controller expects an automatic transmission but the actual transmission plant is a manual transmission). So, in addition to making each component in Figure 1 **replaceable**, the set of models associated with each subsystem (*i.e.* the plant, local controller bus signals, local controller and global bus signals) are grouped together (using replaceable packages) so that entire subsystem configurations can be changed in a single operation. This allows users to select from pre-packaged, consistent and compatible collections of these models that can be changed in a single operation.

Ultimately, vehicle level models will extend from the template shown in Figure 1 and then use redeclarations (as class modifications) to create each specific vehicle configuration. Furthermore, alternative vehicle configurations can then extend from each other *ad infinitum* to create many different variations on a baseline design. This approach allows users to easily control configuration options while at the same time maximizing reuse. In turn, this minimizes redundant code and/or configuration options across different configurations which greatly eases maintenance of the models.

3.3 Subsystem Configuration Options

As mentioned in Section 2.2.3, the set of signals communicated on each bus depends on the specific set of features implemented in each subsystem. To address this issue, our architecture contains a set of replaceable packages that are used to propagate specific definitions for connectors and/or records that are configuration specific.

For example, the powerplant configuration package includes a definition for the connector used to communicate information between the physical powerplant and the powerplant subsystem controller. That definition, in turn, can be customized (using replaceable type definitions) to specify what kind of information is required for each control feature. In this way, the fact that a particular powerplant has, for example, a dual independent cam phasing feature can be stated as a configuration option which then automatically adds the necessary signals to the connectors used on both the physical powerplant and the powerplant controller. In other words, for any given vehicle model there is a single top-level configuration option for each subsystem that ensures consistent bus definitions throughout the vehicle model.

This is essentially the same idiom, utilizing replaceable packages, that is sometimes used to model different media in fluid modeling applications [14].

3.4 Common Environment

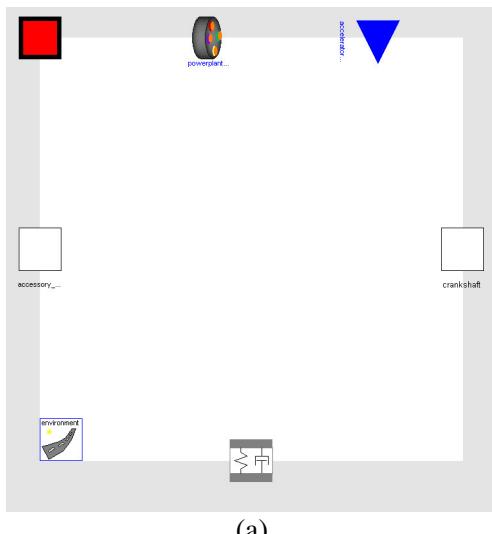
The ambient environment in this architecture contains information that is potentially relevant to every subsystem. Since the environment is a model (potentially with its own equations and states), it isn't possible to propagate the environment component through the vehicle hierarchy. Instead, an **inner** qualifier is used to make the information available to other components in the hierarchy.

3.5 Documentation

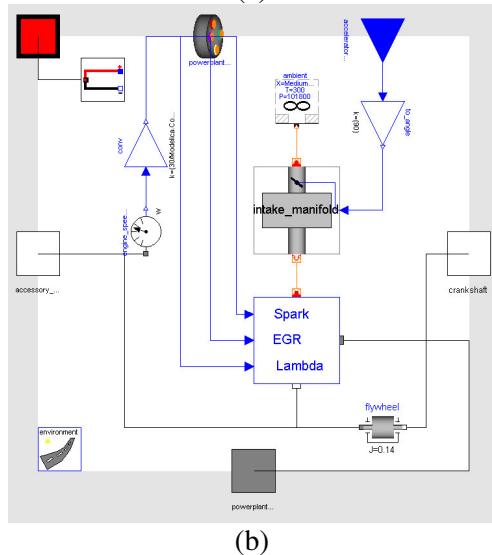
The ability to embed documentation about a package, subsystem, connector, etc. into its definition has already been utilized in this package to provide model developers with a useful online reference for the various interface definitions as well as HTML versions of the same information which can be posted, for example, on a corporate intranet site for reference.

4 Sample Application

To demonstrate how this architecture can be used to build a specific vehicle, we started from the base vehicle configuration shown in Figure 1 and added specific engine, transmission, driveline, brakes and chassis models. Along with these physical subsystem models, controllers for the engine and transmission were included to handle spark timing and gear shifting. The accessory and electrical subsystems were neglected in our example. The purpose of the model is to evaluate performance characteristics such as 0-60 MPH times and 0-400 meter times.



(a)



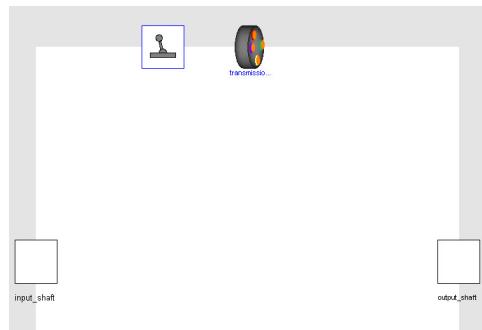
(b)

Figure 2: (a) Powerplant Interface; (b) Sample Engine

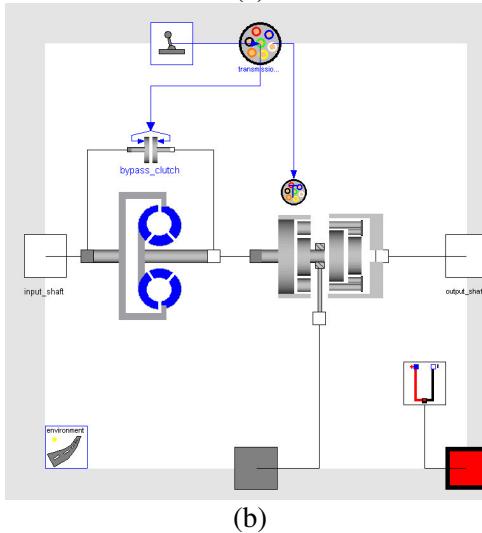
4.1 Engine

The engine model used in this example includes simple "filling and emptying" dynamics for the engine manifold and uses a table to lookup

engine torque as a function of spark timing, air fuel ratio and recirculated exhaust gas. Figure 2a shows the basic interface definition for a powerplant. Figure 2b shows our sample model which extends from the interface definitions so it can inherit all the physical and control system connectors required for compatibility with the overall architecture. Since, for this example, we are only interested in simple 1D rotational dynamics of the powertrain, the powertrain mount connection has been redeclared as a 1D rotational flange. Once this is done, the subsystem model is populated with component models which are connected to each other and to the interface connectors. Note that this particular subsystem translates driver accelerator pedal position directly into a throttle angle, reads the engine control parameters (*i.e.* spark, intended air-fuel ratio and command exhaust gas recirculation) from the subsystem control bus and writes the engine speed back onto the subsystem control bus.



(a)



(b)

Figure 3: (a) Transmission Interface; (b) Sample Transmission

4.2 Transmission

The transmission model represents a six speed automatic transmission. The basic transmission interface is shown in Figure 3a. By extending from the interface, redeclaring connectors and adding components we eventually end up with a complete transmission model as shown in Figure 3b which includes the torque converter, bypass clutch and gearbox. The gearbox is further composed of a series of planetary gear sets, inertias and clutches (not shown). Note that in this model we assume that the gear selection information is propagated back to the transmission subsystem controller which, based on this information, command the engaging and disengaging of specific clutches inside the gearbox.

4.3 Remaining Subsystems

The remaining subsystems do not contain much detail. Rather than presenting the interface and implementation for each subsystem, we will just summarize the behavior represented in each:

- Accessories – No accessory loads are considered in this analysis.
- Electrical – The electrical system provides a constant 12V to the other components (although none of these simple models draw any current).
- Brakes – The brakes are modeled as simple friction elements (from the Modelica standard library).
- Driveline – The driveline provides power to the front axle of the vehicle through a final drive gearset and a simple differential element
- Chassis – The chassis response is purely longitudinal. The tire behavior uses the Pacejka characterization [7] and the vehicle mass is represented by a single lumped mass. No weight distribution effects are included.

4.4 Control

The only control functions required for this analysis are spark control (to maximize mean engine torque), shift scheduling and clutch control (*i.e.* engaging and disengaging clutches depending on the currently requested gear). In addition, the chassis subsystem provides vehicle speed to its local subsystem controller that transmits the information to the transmission subsystem controller via the vehicle level communication bus.

4.5 Results

The models used to demonstrate the capabilities of this vehicle model architecture are part of the training materials used within Ford to familiarize engineers and model developers with Dymola and Modelica. As such, it is important to point out that the subsystem specifications and system simulation results do not represent or reflect the performance of any particular Ford vehicles. In fact, the controller calibrations are intentionally made sub-optimal to allow students to further refine them.

The training exercise that these models were taken from focuses on vehicle acceleration performance. Figure 4 shows the vehicle acceleration plotted as a function of time. From this plot we can clearly see the "torque holes" that occur while the transmission is shifting. In addition, the upper limit on acceleration seen at the start of the simulation represents the limited longitudinal traction provided by the tires before they start to slip.

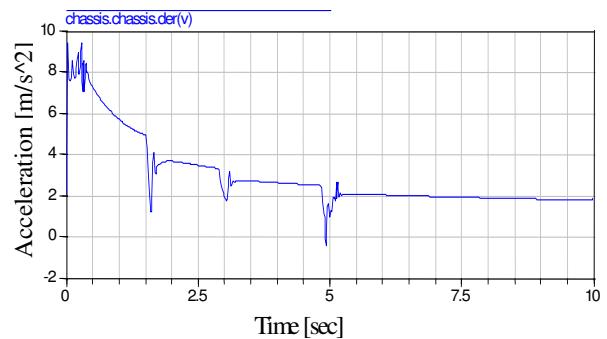
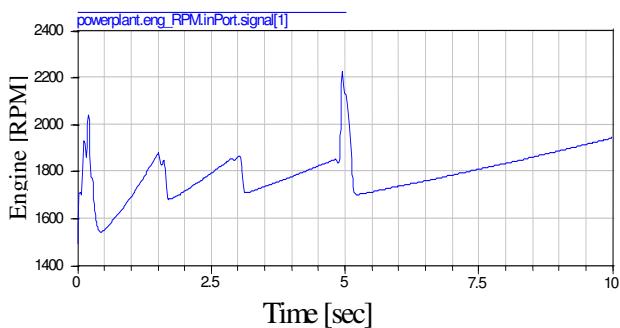
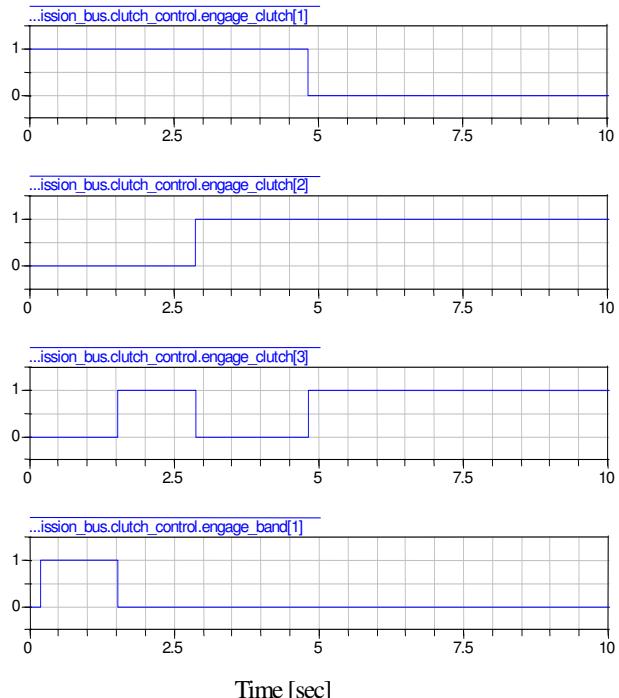


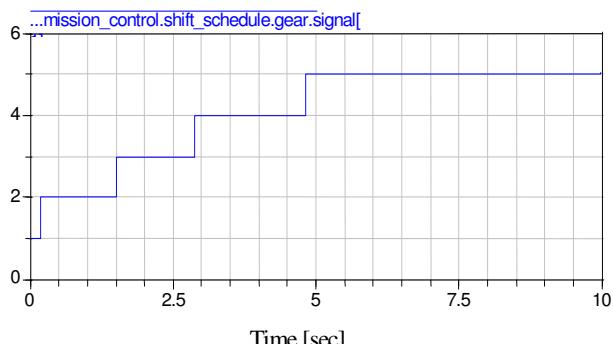
Figure 4: Vehicle Acceleration vs. Time

It is also interesting to examine the engine speed during the simulation as shown in Figure 5. Studying the RPM signal we can clearly see an "engine flare" at about 5 seconds into the simulation. Such flares occur when the shifting of the clutches in the transmission is not well controlled. As a result of poor control, the overall torque capacity of the transmission is less than the torque generated by the engine and the engine accelerates rapidly until the clutches engage.

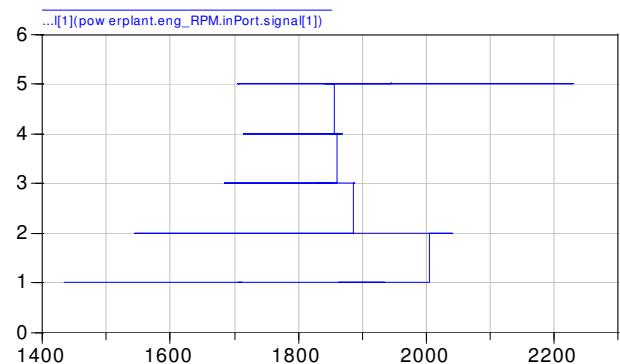
In addition to examining the physical signals within the system (*e.g.* torques, speeds, *etc*), it is also interesting to examine the communication between the controllers. Figure 6 shows the clutch and band engagement requests sent from the transmission controller to the physical transmission. These are actuator commands instructing the hydraulic controllers within the transmission to engage specific clutches and/or bands.

**Figure 5: Engine Speed****Figure 6: Clutch/Band Engagement**

Similarly, in Figure 7 we can see the internal decision making process of the transmission subsystem controller by plotting its selection of gear during the simulation. This information is what ultimately dictates the detailed clutch/band engagements shown in Figure 6.

**Figure 7: Gear Selection**

Finally, many insights can be gained by plotting some of the simulation variables with respect to each other. For example, if an engineer knows at approximately what speed the peak in the engine power curve appears, he might plot the commanded gear selection as a function of engine speed, as shown in Figure 8 for this example, to make sure that the shift strategy appropriately straddles that peak.

**Figure 8: Gear Selection vs. Engine RPM**

This section demonstrates just a few of the possible results that a vehicle level analysis can uncover. Having a standardized set of interfaces not only makes the exchange of models easier, it also assures, to some degree, that signals will have common names (at least those associated with the provided interfaces).

5 Usability Considerations

Some of the more advanced Modelica language features used in this architecture (*e.g.* replaceable packages, choice annotations, subtype definitions for classes, *etc*) are not necessarily accessible or intuitive for end users. In this section, we describe some ideas for representing the complex structure of the vehicle so that end users can easily configure and reconfigure vehicle models.

5.1 Handling User Choices

5.1.1 Link Choices to Component Icons

First, it should be possible to select a component in a vehicle model and browse a set of compatible alternative components. In other words, the set of alternatives should be easily accessible *from the graphical icon associated with that component* rather than requiring users to find components in, for example, the component browser (which requires knowledge of what classes the components were inherited from).

5.1.2 Consistent Handling of Choices

For complex "template" models (*i.e.* models that are designed so that end users can merely "fill in the blanks"), it is important that users be presented with a complete view of the model including all redeclarations/customizations they have made. Redeclarations can affect many different "visual" aspects of the model including its inheritance, its component hierarchy, the parameter dialogs, graphical appearance, results structure, associated scripts, etc. It is important for tools to make sure that all of these possibilities are always consistent with the choices made by the end user when customizing the models.

When interface definitions are influenced by top-level choices (*e.g.* the physical powerplant interface is altered by the choices made in the top level powerplant configuration package), this should influence the set of possibilities generated with the `choicesAllMatching` annotation in the models. For example, if the top-level configuration specifies a powerplant with dual independent cam phasing, the set of choices generated when redeclaring the powerplant should only include powerplant models that can satisfy that interface.

5.1.3 Carryover and Memory of Choices

While exploring alternatives, graphical tools should perpetuate user modifications for identical parameters and/or choices when possible and, when not possible, remember those modifications in case the same options reappear. For example, if a user configures a model to use one particular 5 speed transmission model and then switches to a different 5 speed transmission model, it should be possible to carryover any common parameters (*e.g.* gear ratios) or choices (*e.g.* torque converter model) between the two alternatives. In addition, if they explore the idea of a continuously variable transmission (CVT), the tool should remember the gear ratio settings if they decide to revert back to a 5 speed transmission.

5.2 Visualization

5.2.1 Decision Tree Visualization

With a template model as complicated as the one shown in Figure 1, the options and possibilities open to the end user can be quite disorienting. For these kinds of models, it would be very useful to have a compact representation of the tree of possible choices open to the user. Such a tree would need to be hierarchical and each decision that is made should be reflected in the tree (*i.e.* the tree should respond dynamically to user choices). Ideally, such

a tree should show, in a single comprehensive view, choices that influence topological changes (*e.g.* what transmission model is used) as well as parameters.

5.2.2 Visualizing Configurations

Another issue with template models is the proliferation of variations. It should be possible to visualize in a coherent way the modifications associated with a "tree" of configurations (in this case, a tree based on the inheritance hierarchy as opposed to the tree discussed in Section 5.2.1 which is based on the compositional hierarchy).

6 Limitations

While Modelica provides some powerful features to support the architecture described in this paper, there are still some areas where the existing features are still not sufficient. In this section, we will discuss some of the limitations we encountered and some ideas for overcoming those limitations.

As described in Section 3.3, we have chosen to propagate configuration information from the top down. In other words, decisions about connector definitions are made at the top level and then propagated to subsystems. This is awkward because it is often unnatural for this information to either appear or originate at the vehicle level. For example, information about signals exchanged between the powerplant and the powerplant controller is really determined by the set of sensors and actuators present on the powerplant itself but we were not able to find a way of expressing this in Modelica.

Along similar lines, the set of signals communicated on the vehicle control bus should be the union of all signals broadcast from each subsystem controller. From a user perspective, it would be best to simply choose the controller and physical subsystem and have the information about broadcast messages "propagate up" automatically to the vehicle level controller bus.

In the current design, the subsystem bus connector on the physical subsystems is always declared `inner`. This is done to allow the use of the `SignalBus` idiom [8] which allows sensors and actuators to reference only the specific signals they require (as opposed to all signals communicated in that subsystem). Unfortunately, the relationship between the bus connector and these sensors and actuators is not explicit because it relies on using `inner` and `outer` qualifiers. A better solution would be to allow direct connections.

Unfortunately, the current Modelica specification requires each connector to contain exactly the same signals. By relaxing this requirement and, for example, allowing one connector to be a subtype of the other, such connections would be possible and, as a result, clearer.

One of the biggest problems in developing such a framework is how to represent the fundamental engineering assumptions present. For example, the powertrain mounts might be represented as either 1D or 3D connections. Likewise, the electrical system may support multiple voltage levels. Several subsystem models can be impacted by these choices and there is no easy way of understanding what assumptions are made for particular models and how that affects the assembly and compatibility at the vehicle level. Rather than relying on complex nested replaceable type definitions and interfaces, the entire process might be more coherently represented with features (*e.g.* layers) that provide configuration based on a fixed set of possibilities.

7 Future Work

It is important to reiterate that the structure defined in this document is merely a proposal and that further discussion is required. Once a consensus is reached on the appropriate subsystem decomposition and interface definitions, there are several potential directions for this work. For example, it might be useful to extend the depth of the current hierarchy to define architectures for each of the various subsystems. For example, powerplant templates could be developed for internal combustion engines (*e.g.* I-4 or V-6 cylinder configurations) and transmission templates could be developed that decompose automatic transmissions into individual models for a torque converter, bypass clutch and gearbox (with interface definitions for each). Finally, other top-level architectures could be developed that reuse the subsystem interface definitions. These architectures may choose to use a subset of the subsystems shown in Figure 1 (*e.g.* an engine connected to a dynamometer) or they may choose to add additional subsystems for more exotic vehicle configurations (for towing applications, fuel cell vehicles, *etc.*).

8 Acknowledgments

The architecture presented in this paper is heavily based on a Ford Motor Company internal initiative, by Mark Jennings, Judy Che, Bradley Hieb, Tim Mortimer, Ken Butts, Chris Belton, Pete

Burchill, Peter Bennet, David Copp and Nick Darnton, to develop a vehicle model architecture for Simulink [5]. This work leverages a great deal from the system decomposition and thorough analysis that was done as part of that work. As a result, the authors would like to recognize the significant influence and impact that work had on the material in this paper.

The authors would also like to thank John Batteh, Chuck Newman, Erik Surewaard, Graham King, Johan Andreasson, Christian Schweiger, Martin Otter, Jonas Hellgren, Jonas Karlsson, Jonas Fredriksson, Bengt Jacobson and Lars Eriksson for their work in developing automotive component and subsystem models which we hope will, at some point, be compatible and freely exchangeable through this architecture.

9 References

1. J. Andreasson, A. Möller and M. Otter, "Modeling of a Racing Car with Modelica's Multi-Body Library", *Modelica Workshop 2000 Proceedings*, <http://www.modelica.org/workshop2000/proceedings/Andreasson.pdf>
2. M. Otter, M. Dempsey and C. Schlegel, "Package PowerTrain. A Modelica library for modeling and simulation of vehicle power trains", *Modelica Workshop 2000 Proceedings*, p. 23-32, <http://www.modelica.org/workshop2000/proceedings/Otter.pdf>
3. P. Treffinger and M. Goedecke, "Development of Fuel Cell Powered Drive Trains With Modelica", *Proceedings of the 2nd Modelica Conference*, p.125-131, http://www.modelica.org/Conference2002/papers/p16_Treffinger.pdf
4. J. Hellgren, "Modelling of Hybrid Electric Vehicles in Modelica for Virtual Prototyping", *Proceedings of the 2nd Modelica Conference*, p. 247-256, http://www.modelica.org/Conference2002/papers/p32_Hellgren.pdf
5. C. Belton, P. Bennet, P. Burchill, D. Copp, N. Darnton, K. Butts, J. Che, B. Hieb, M. Jennings and T. Mortimer, "A Vehicle Model Architecture for Vehicle System Control Design", *SAE Congress 2003*, SAE-2003-01-0092.
6. "Dymola 5.0 User's Manual", *Dynasim AB*, p. 206.

7. H. B. Pacejka and E. Bakker, "The magic formula tyre model.", *Proceedings of the 1st Tyre Colloquium, Delft*, October 1991.
8. M. Tiller, W. E. Tobler and M. Kuang, "Evaluating Engine Contributions to HEV Driveline Vibrations", *Proceedings of the 2nd Modelica Conference*, p. 19-24, http://www.modelica.org/Conference2002/paper_s/p03_Tiller.pdf
9. S. Drogies and M. Bauer, "Modeling Road Vehicle Dynamics with Modelica", *Modelica Workshop 2000 Proceedings*, p. 161-168, <http://www.modelica.org/workshop2000/proceedings/Drogies.pdf>
10. "Modelica Language Specification, Version 2.0", *Modelica Association*, 2002,
11. M. Tiller, "Introduction to Physical Modeling with Modelica", *Kluwer Academic Publishers*, 2001.
12. K. Wipke, M. Cuddy and S. Burch, "Advisor 2.1: A User-Friendly Advanced Powertrain Simulation Using a Combined Backward/Forward Approach", *IEEE Transactions on Vehicular Technology: Special Issue on Hybrid Electric Vehicles*, 1999, http://www.ctts.nrel.gov/analysis/pdfs/advisor_2_1.pdf
13. A. Rousseau, S. Pagerit, G. Monney and A. Feng, "The New PNGV System Analysis Toolkit V4.1- Evolution and Improvement", *SAE 2001 Future Transportation Technology Conference*, SAE 2001-01-2536.
14. C. Newman, J. Batteh and M. Tiller, "Spark-Ignited-Engine Cycle Simulation in Modelica", *Proceedings of the 2nd Modelica Conference*, p. 133-142, http://www.modelica.org/Conference2002/paper_s/p17_Newman.pdf

Modelling of Generic Hybrid Electric Vehicles

Leo Laine, CTH, Sweden, laine@mvs.chalmers.se
 Johan Andreasson, KTH Vehicle Dynamics, Sweden, johan@fkt.kth.se

Abstract

The software development of the control functions will be a large part of the work when developing future vehicles. Therefore, it is of great importance to be able to reuse the control architecture for different hardware configurations. In this work, a generic¹ control architecture for Hybrid Electric Vehicles has been modelled with Modelica. Functional decomposition was used to develop the generic control architecture. Functions are identified and placed into a hierarchical partitioning structure. Three functional levels are suggested; main control level, subsystem level, and actuator/sensor level. The main control contains a driver interpreter, energy management, vehicle motion control and a strategic control. These main functions are made independent of hardware and of hybrid configuration. The subsystem level contains driver interface, chassis, power supply and auxiliary systems. Two models, a parallel and a series hybrid electric vehicle, are used to demonstrate the implemented architecture.

1 Introduction

In order to handle the complexity of several actuators/sensors interacting in future Hybrid Electric Vehicles (HEVs) and to allow easy change of hardware configuration, a control architecture with suitable functional partitioning is necessary.

There are three main types of architectures for partitioning; centralised, hierarchical, and peer, as shown in Figure 1. The centralised architecture collects information from all sensors and computes references for all actuators. The benefit is that all signals are available simultaneously. The drawback is the lack of modularity that makes it hard to add new functionality. The hierarchical structure consists of a top level control block and several low level control blocks. This allows good modularity and also a central controller is available to coordinate the interaction between the actuators/sensors. The peer-to-peer architecture is the most modular one, but without a coordinator between

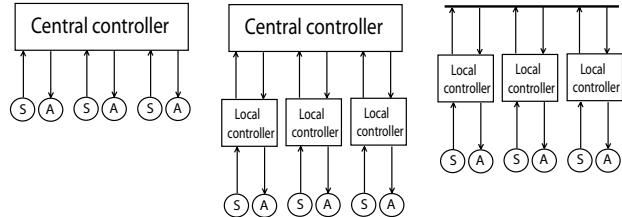


Figure 1: Centralised, hierarchical and peer-to-peer architecture.

the different actuators/sensors conflicts will be hard to avoid.

The architecture should be generic and work for several types of HEV configurations such as parallel, serial, and split etc and must therefore be modularised. It must also fulfill the requirements on interfaces between automotive suppliers and manufacturers so that brand specific qualities can be kept in-house. For both these demands, the hierarchical control architecture is suitable.

The purpose with the suggested control architecture is to easily handle the variety of vehicles that the authors believe will be found a decade from now and further on. These future vehicles could be serial HEVs with fuel cell as primary power unit, and with wheel units that can apply driving, steering, and suspension forces independently. However, to be really useful, the architecture must also be able to handle today's vehicle in a well defined way.

Modelica [1] was chosen as a platform for test and validation of ideas concerning generic modelling of HEVs. The aim is to study how HEVs can be modelled as a complete system and combine different areas of interest, such as: control, energy management, and vehicle dynamics. The first step is to evaluate if the suggested generic control architecture really is generic by modelling different hardware configurations with Modelica. The second step is to study how the specific strategies within Main Control should be designed. Finally, the sensibility to faults and inaccuracies will be studied. In this paper the ideas behind the architectures are first briefly described² and then the implementation

¹Generic: hardware and configuration independent

²See [2] for a more thorough explanation.

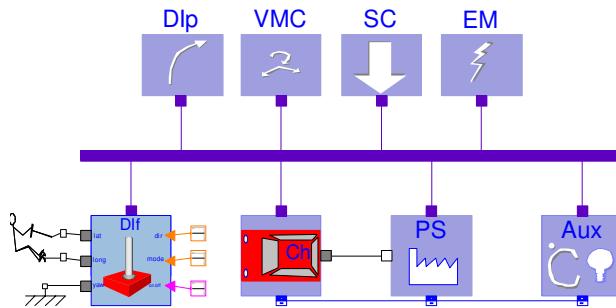


Figure 2: Main model architecture illustrating the main functions within functional levels 1 and 2.

in Modelica is discussed.

2 Main model architecture

The main model architecture is divided into different functional levels. The highest functional level is called main control and includes the following functions; Driver Interpreter (DIp) interprets the driver's demands as a desired path, Vehicle Motion Control (VMC) that controls the vehicle according to these demands and Energy Management (EM) assures that this is done in an energy efficient way. Additionally there is the Strategic Control (SC) which finalizes the orders from Vehicle Motion Control and Energy Management to the lower functional levels. It is only Strategic Control that can send orders to lower functional levels. This to uphold the causality of the orders. If a critical state is recognised by Energy Management or Vehicle Motion Control, Strategic Control will give priority to suggested signals from either part. The functional level 2 contains the following: Driver Interface (DIF), Chassis (Ch), Power Supply (PS), and Auxiliary Systems (Aux).

In Figure 2 the main model architecture implemented in Modelica shows functional levels 1 and 2. All functions exchange generic signals via a bus, and the chassis, power supply and auxiliary systems are coupled with standardised mechanical and electrical connectors. This allows each model to be changed without having to redesign the others. In Figure 3 this is illustrated by a menu that shows how different HEV configurations can be set up. Figure 4 shows the signal flow between functional levels 1 and 2. Auxiliary systems and Driver Interface are here excluded for simplicity.

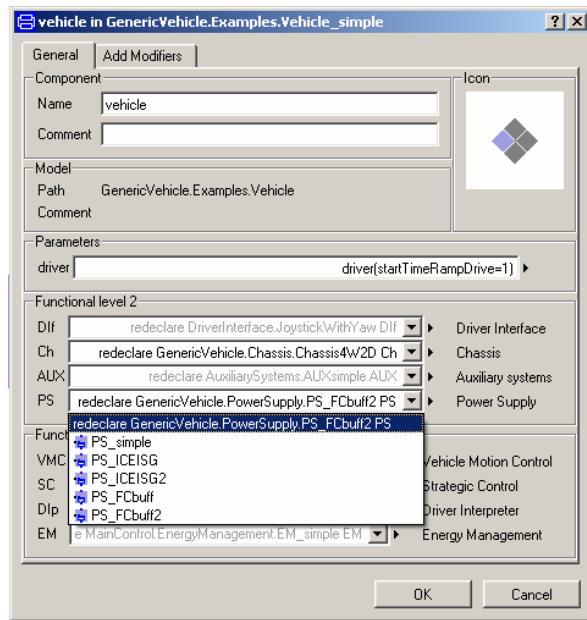


Figure 3: The generic vehicle menu easily allows changing the Power Supply.

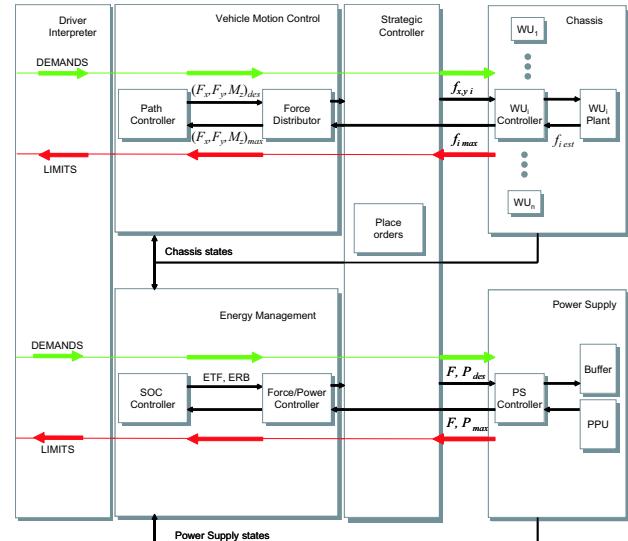


Figure 4: Signal between functional levels 1 and 2. Only signals to Power Supply and Chassis are shown for simplicity.

3 Modelica implementation

The Modelica implementation is gathered in the Modelica library `GenericVehicle`. According to Section 2 the main model consist of nine functions and in the library, these represent a sub-packages each. `DriverInterpreter`, `VehicleMotionControl`, `StrategicControl` and `EnergyManagement` cover the functional level 1. Additionally there are `DriverInterface`, `Chassis`, `PowerSupply`

and AuxiliarySystems for level 2. Finally the Bus package contain the models necessary for the information exchange.

3.1 DriverInterpreter

The Driver Interpreter communicates with the driver interface by interpreting the driver's signals and by sending proper feed-back. The driver's intentions are interpreted as a desired path, taking into account limitations set up by the Vehicle Motion Control and external inputs such as e.g. cruise control. The desired path is defined by the velocity v , the vehicle's slip angle β , and the curvature ρ .

3.2 VehicleMotionControl

The Vehicle Motion Control includes a controller that follows the desired path by the derivation of desired global forces (F_x, F_y, M_z). These forces are then distributed between the wheels within the allowed limits for each wheel unit. Thus, there is an optimisation task and a control task. These are currently handled as described in [3].

3.3 EnergyManagement

Energy Management controls the energy flow from the Primary Power Unit (PPU) and the flow to the Buffer (Bf). The simple version of Energy Management calculates a State-Of-Charge (SOC) target by considering the vehicle speed, see Equation 1. By comparing SOC target with actual SOC simple strategies are used to calculate how much Electrical Regenerative Braking (ERB) and how much Electrical Traction Force (ETF) should be applied. Both parameters are nominal values. The desired tractive force and the total desired power needed from PS are the signals sent to Strategic Control which places the orders to Power Supply. EM Simple also calculates a power limit value for Auxiliary Systems.

$$SOCTarg = C_0 - C_1 \cdot e^{\frac{-v(t)}{C_2}} \quad (1)$$

where $v(t)$ is the current vehicle speed, and $C_0 = 0.75$, $C_1 = 0.1$, and $C_2 = 6$ are constants.

3.4 StrategicControl

The Strategic Control is responsible for the commands from level 1 to level 2 and handles the priorities between VMC and EM. The simple SC only places the orders to functional level 2. Strategies about safety and reliability will be located at SC, checking the critical state signals from EM and VMC.

3.5 DriverInterface

The driver interface contains the actuators and sensors that the driver can influence. These could be steering wheel and pedals as well a joystick. DIf is here seen as a full drive by wire subsystem. The longitudinal, lateral and yaw signal are measured and then sent to DIp.

3.6 Chassis

The chassis (Ch) is thought of as a body onto which a number of wheel units are mounted. Each wheel is then considered as an autonomous unit and is by default decoupled from the other wheels. Depending on the linkage carrying the wheel as well as the available actuators, there are different possibilities to generate ground contact forces. A very simple example is a wheel with only brakes and no steering possibility and passive suspension, while other wheel units may have drive, steering, camber control and active damping.

The Modelica implementation is based on the `VehicleDynamics` library [4] components for three dimensional Multi Body System (MBS) chassis modelling. Additionally the `PlanarMultiBody` library [5] has been used to model simpler planar chassis models. The latter are suitable when influences of load transfer due to roll or pitch can be neglected since these models speeds up simulation time considerably.

The distributed forces from the SC is realised at each wheel unit that also sends information about maximum achievable force. For a future vehicle with independent wheel units, this is straightforward, but today's vehicles uses many passive components that in some case limits the wheel motion and also couples the wheels together. To deal with this, *restrictors* are introduced to limit the degrees of freedom (DOF) of the wheel.

3.6.1 Wheel Units

At each Wheel Unit (WU), the force commanded by the SC should be generated. To avoid saturation, the wheel unit provides the VMC with information about its current limitations. From the desired forces, the desired steering angle and wheel spin velocity are calculated³.

To generate the wheel spin velocity, the wheel unit checks how much rotational torque is available directly at the drive shaft from the PS and then coordinates the available actuators to meet the desired order.

In Figure 5, three different WU models are shown, illustrating the variety of modelling detail. The left-

³Details are found in [3].

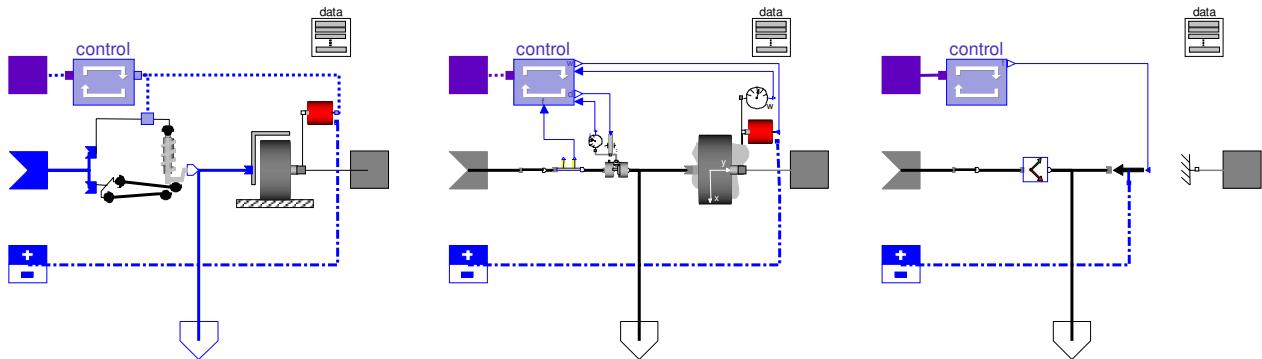


Figure 5: Wheel units with different level of detail. Left: A 3D MBS model of a control, linkage and wheel with an electric motor, middle: a 2D MBS model with linkage replaced by a steering joint, and right: an ideal Wheel Unit that generates the desired forces directly.

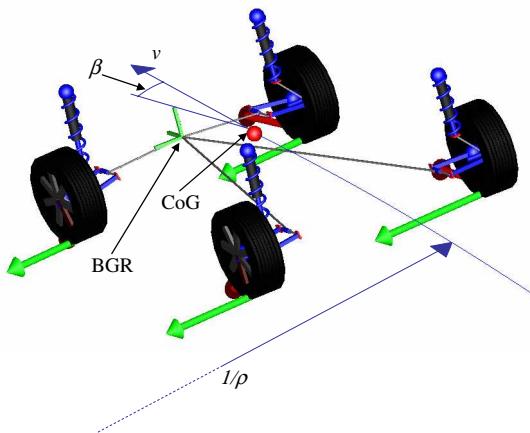


Figure 6: Screen shot of an animation showing a four wheeled HEV with independent wheel corners. The path (ρ, v, β) is indicated as well.

most example is a full 3D-model of a wheel and a linkage, e.g. a MacPherson or a DoubleWishBone. Here, a linkage suggested in [6] is used and an animation view of a vehicle with these wheel unit models can seen in Figure 6.

In many cases, when the details of the linkage are of less importance, simulation time can be reduced by using a simpler model as illustrated in Figure 5, middle. The linkage is reduce to an equivalent king-pin (steer) axis and no vertical motion is considered.

Still, these two models have in common the need to find steering angle and wheel spin velocity. The model in Figure 5, right, instead applies the desired forces directly.

3.6.2 Restrictors

As mentioned earlier, it is straightforward to use the WU concept as long at each wheel is independent of the others. This is not the case for today's vehicles and

the restrictors are used to describe these relations. Typical restrictors are rack steerings and differentials that constrains the steering angle and the force distributed from the PS, respectively. To make the VMC aware of their existence, they are connected to the bus and send information about a) between what WU they act and b) how they act. Active restrictors also receive information about the WU state and commands to figure out how they should act. In Section 4, the usage of restrictors is exemplified.

3.6.3 Bodies

The body is the frame that carries the WUs. It also sends information about its states to the VMC. The reason it is treated as a separate unit, and not just as a least common divider of all chassis, is because there will be an extension that handles more than one body, coupled by restrictors. Typical cases when this is relevant are tractor-trailer combinations, articulated buses and vehicles with a frame that cannot be considered as rigid.

3.7 PowerSupply

The conventional power train concept with a combustion engine, transmission, and drive line is not a valid description for a HEV. The HEV concept includes handling of a major electricity source in combination with a conventional or parts of a conventional power train. A more suitable name of this function is Power Supply. The PS includes both the Primary Power Unit and a buffer and can be anything from an internal combustion engine to a fuel cell. The buffer can be an electric buffer such as a battery, super capacitor or a mechanical one e.g. flywheel.

3.8 Auxiliary Systems

The Auxiliary systems is a gathering of all systems that are not involved in the vehicle's motion. Examples are air conditioning and lights. Aux calculates the actual power needed and sends this information to EM. EM limits the maximum power available for the Aux and PS provides the needed electricity by a standardised electrical connector.

3.9 Bus

The Bus contain generic information and orders that are exchanged between functional levels 1 and 2. The signals are named after their origin as exemplified below:

EM_Pauxlimit EM calculates a maximum power limit for Aux.

SC_Pauxlimit SC finalise the order to Aux.

Aux_Pactual The actual power consumption from Aux.

It is important that the signals are made hardware independent to allow easy change of functions. The Modelica implementation is based on the bus connectors available in the standard library. All models of a specific function e.g. EM, VMC, PS, and Ch share the same base, defining the send and receive signals.

The signals on the bus give an idea of what information is necessary for any kind of hardware configuration for the specific function.

4 Examples

To demonstrate the suggested architecture's ability to handle different hardware configurations, two different HEV configurations have been implemented. The first one is a parallel HEV with wheel motors on the front wheels, see Figure 7, left. As indicated in the figure, the front and rear wheels are constrained by restrictors. The front wheels have a rack steering that couples the steering angle of the two wheels. At the rear wheels there is also a rack steering, but in this case the range is set to 0, making the vehicle front wheel steered. Additionally, there is also a differential that distributes the driving torque from the PS.

The second case is a series HEV with wheel motors on all wheels, see Figure 7, right. Here no restrictors are used and each wheel is individually controlled.

The body weight and inertia for both cases is relevant for a sports utility vehicle. For both cases, the same models within functional level 1 are used.

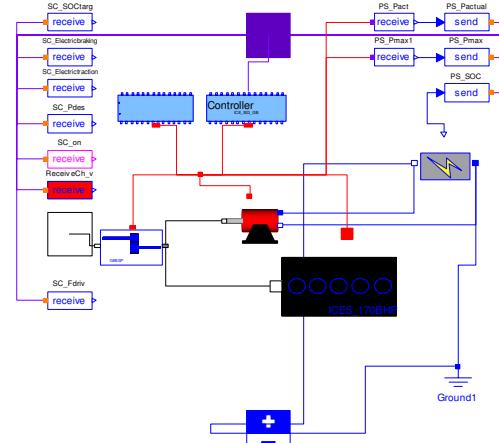


Figure 8: PS with ICE, ISG, and a GB.

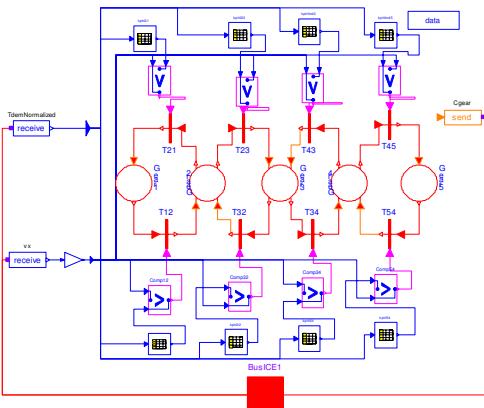


Figure 9: Petri net used for the shift strategy for the 5 speed gear box.

4.1 The parallel HEV case

The parallel HEV is equipped with PS containing an Internal Combustion Engine (ICE) as a PPU, and an Integrated Starter Generator (ISG), automated manual Gear Box (GB), and a battery, see Figure 8. There is a local controller that coordinates ICE, ISG and GB. The gearshift strategy is based upon a petri net which uses actual vehicle speed and desired torque for the boolean expressions. The petri net is shown in Figure 9.

The ICE model uses one dimensional look up tables for maximum and minimum torque. The fuel consumption is calculated by using the actual torque and rotational speed as input for a two dimensional look up table. The model is shown in Figure 10.

The Chassis contains wheel units with wheel motors for the front wheels and the rear wheels have a differential restrictor applying the torque provided by PS, see Figure 7, right. In this case the PS supplies both mechanical torque and electrical power to the chassis.

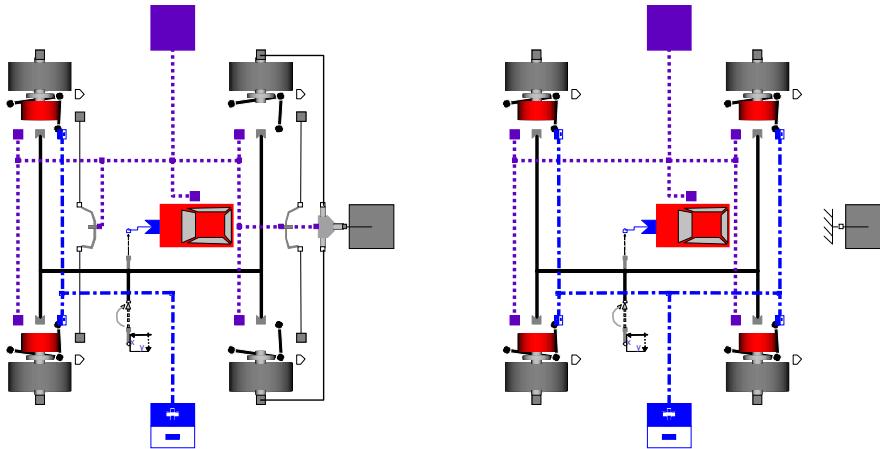


Figure 7: Chassis with independent wheel units used in the parallel HEV case, left and the serial HEV case, right.

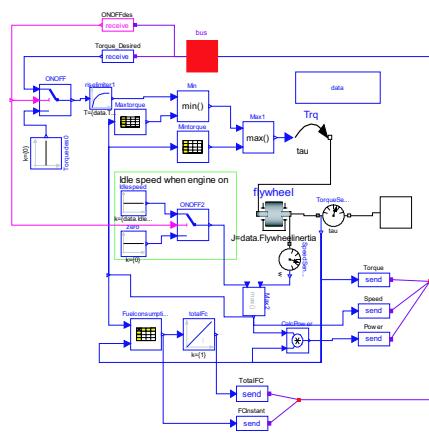


Figure 10: The ICE model.

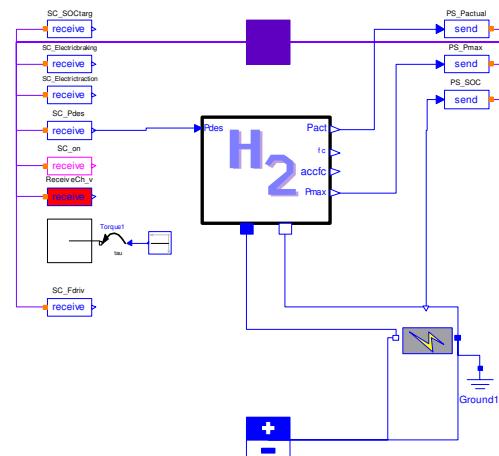


Figure 11: PS with FC and battery .

4.2 The serial HEV case

The Chassis contains wheel units with wheel motors on all four wheels. In this case PS submits only electric power to the chassis, see Figure 7, left. The PS contains a Fuel Cell (FC) as a PPU and a battery as buffer, see Figure 11.

4.3 Simulation

Figure 12 shows results from a ramp simulation of the parallel HEV vehicle starting from standstill. It is accelerated to 10 m/s in 4 s and then the velocity is kept constant for 1 s. Finally the velocity is decreased to stand still at $t = 8$ s. The first graph shows the desired speed from DIp and the actual speed. During the first 2 s of the deceleration the actual speed is higher than the desired. The second graph shows the actual torques from the ICE and ISG. Third graph shows the actual gear of the GB. Finally the fourth graph shows the SOC level of the battery.

The same simulation is also made for the serial HEV configuration, see Figure 13. The first graph shows the desired speed from DIp and the actual speed. The second graph shows the desired power and the generated power from FC. The third graph shows the actual SOC level of the battery. The fourth graph shows the instant and accumulated fuel consumption. During deceleration the FC is shut down.

The results show that it is possible to use the same VMC, EM, DIp and SC for both configurations. The performance of the models are not optimal since the scope of work have not been on sizing on components nor to find the optimal strategies.

5 Conclusions and discussion

Modelica has been a useful way to describe, model and test the architecture. It is a good platform because it allows easy interaction of different domains such as

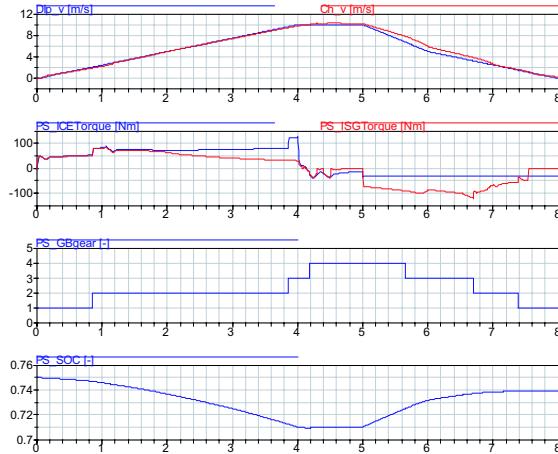


Figure 12: Ramp simulation for the parallel HEV configuration.

multibody, electrical, mechanical, and control.

The sample cases demonstrates the architecture. The results show that the architecture manage different hardware configurations and that exchanging hardware does not affect the highest functional level, i.e. Main Control.

Even though the over-all impression is positive, some limitations have been found. The size of the bus is dependent on the number of wheel units, bodies and restrictors and should thus be defined by the chassis itself. Since the size of the bus must be fixed, this is currently not possible. It would also be desirable to be able to send the equations defining the restrictors directly through the bus.

6 Future work

An extension of this work will mainly involve a) Development of a method to evaluate the reusability and constraints applied by using the suggested architecture. b) Verification of the reusability of the suggested architecture for different configurations of HEVs. Especially different configurations of PS. c) Studies on what control strategies within Main Control would apply for the foreseen HEV configurations. d) Studies on how critical states could be handled so that they are recognised by EM and/or VMC. e) More flexible description of restrictor information from functional level 2 to 1. f) Compensation for non-ideal sensors.

7 Acknowledgements

This work is financed by the Driving Dynamics and Main Control projects within the Swedish National Research Programme "The Green Vehicle/FCHEV".

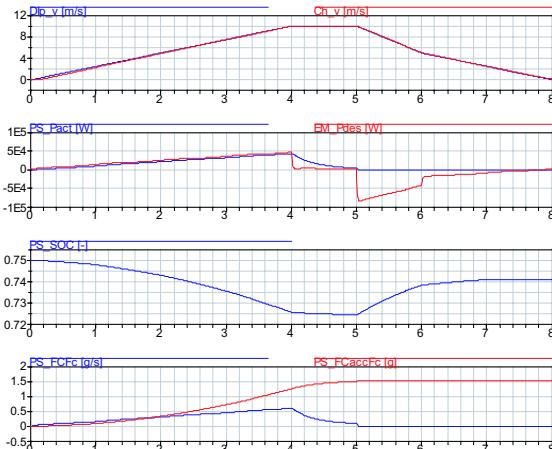


Figure 13: Ramp simulation for the serial HEV configuration.

References

- [1] Modelica Association. <http://www.modelica.org>.
- [2] L. Laine and J. Andreasson. Generic control architecture applied to a hybrid electric sports utility vehicle. *To be presented at the 20th International Electric Vehicle Symposium, Long Beach, CA, November 15-19, 2003*.
- [3] J. Andreasson and L. Laine. Driving dynamics for hybrid electric vehicles considering handling and control architecture. *18th Int. Symp. IAVSD, Dynamics of Vehicles on Roads and Tracks, Japan, August 2003*.
- [4] J. Andreasson. VehicleDynamics library. In Peter Fritzson, editor, *Proceedings of the 3rd International Modelica Conference*, Linköping, November 2003. The Modelica Association and Linköping University.
- [5] J. Andreasson and J. Jarlmark. Modularised tyre modelling in Modelica. In Peter Fritzson, editor, *Proceedings of the 2nd International Modelica Conference*, Oberpfaffenhofen, March 2002. The Modelica Association and Deutches Zentrum für Luft- und Raumfahrt.
- [6] S. Zetterstrom. Electromechanical steering, suspension, drive and brake modules. *VTC 2002-Fall, Vancouver, Canada, September 24-28, 2002*.

Advanced Electric Storage System Modeling in Modelica

Erik Surewaard and Eckhard Karden
 Ford Forschungszentrum Aachen (FFA)
 Energy Management Group
 Süsterfeldstrasse 200
 52072 Aachen, Germany

Michael Tiller
 Ford Motor Company
 Powertrain Research Department
 2101 Village Road
 48121 Dearborn MI, USA

1 Abstract

This paper will discuss two important components in the future electrical system of an automobile: the battery and supercapacitor. Models of these components have been developed in the Modelica language. The power of the Modelica language is shown by simulating a so-called dual storage system, consisting of a supercapacitor and battery. This paper also shows the comparison between the simulation and measurement results.

2 Introduction

Due to the increased amount of electric content in a vehicle, the electric powernet will have a significant influence on the fuel economy of a vehicle. In addition, new power supply/starting systems such as Integrated Starter Generators (ISG) will enable new features that improve fuel economy and emission attributes of a vehicle. It is therefore necessary to develop models that capture the detailed behavior of the electric powernet.

This paper will discuss models of two important components of the future powernet: the battery and supercapacitor. A description of the models will be given after which a simulation is performed with a so-called dual voltage storage system (also known as 14+x). This is an electric storage system consisting of a supercapacitor and battery in parallel, which allows a Belt-driven Integrated Starter Generator (B-ISG) to operate on two voltage levels. Such a system has been published by Sebille in [1]. Finally, simulation results will be compared with measurement results.

3 Battery

At the Ford Forschungszentrum Aachen (FFA), a battery model has been developed in Modelica, which is based on work of the Aachen University RWTH. This section describes both the model background as well as the implementation in Modelica.

3.1 Model Background

The battery behavior is characterized using impedance spectroscopy. As part of this process, the battery is excited with currents at different frequencies. Different operating points are also taken into account: temperature and State of Charge (SOC). A schematic of an impedance spectroscopy measurement of a battery is displayed in Fig. 1.

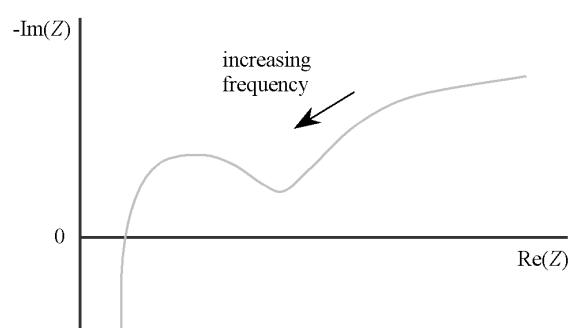


Fig. 1 Schematic plot of an impedance measurement of an automotive battery

A method has been developed by Buller *et. al.* [2], [3] to represent the impedance measurement into an electric equivalent circuit. This procedure is schematically displayed in Fig. 2a. The electric equivalent circuit for this representation is displayed in Fig. 2b.

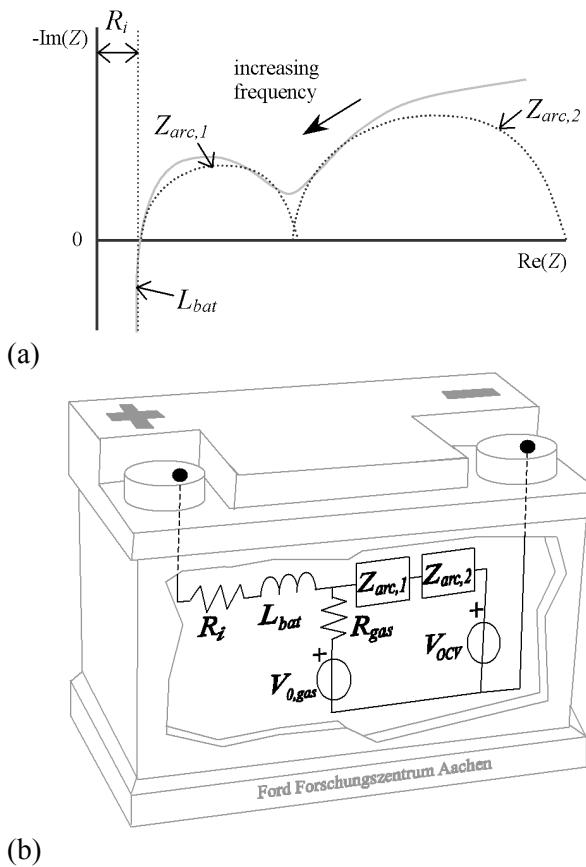


Fig. 2 (a) Approximation of a measured impedance spectroscopy line by electrical elements, (b) electric circuit representation for a battery

The measured impedance of the battery is approximated by an internal resistance R_i , an inductance L_{bat} and two depressed semi-circles in the complex impedance domain: $Z_{arc,1}$ and $Z_{arc,2}$. Inaccuracy arises at low frequencies where the modeled impedance does not approximate the measured impedance. Fig. 2b also includes the open circuit voltage V_{OCV} and the gassing reaction (R_{gas} and $V_{0,gas}$). The gassing reaction is mainly important for overcharging situations, where the charging efficiency of the battery decreases. This is the result of current that is lost in the gassing reaction. In the case of a valve-regulated lead-acid battery, e.g. in Absorbent Glass Mat (AGM) technology, other parasitic reactions have to be added in the gassing branch (especially oxygen recombination), but the topology remains valid.

The two depressed semi-circles ($Z_{arc,1}$ and $Z_{arc,2}$) are represented using specialized RC-circuits. The number of RC-circuits that are used in series to represent the depressed semi-circle is described by N_1 and N_2 (Fig. 3). This number of

RC-circuits is critical for both simulation speed and model accuracy.

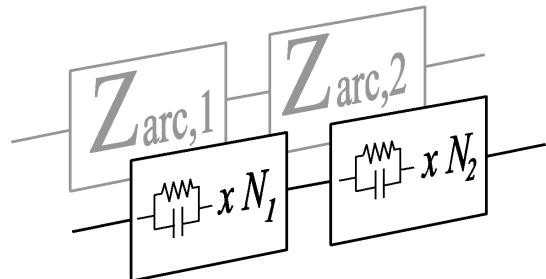


Fig. 3 Representation of the two semi-circles in the complex impedance domain ($Z_{arc,1}$ and $Z_{arc,2}$) by RC-circuit elements

3.2 Model Implementation

The model as displayed in Fig. 2b is constructed in Modelica. The result is displayed in Fig. 4.

The structure of the model is basically the same as the structure displayed in Fig. 2b. On the left you can see the internal resistance of the battery, after which the main branch is divided into two sub-branches. The upper branch shows the gassing reaction. The lower branch shows an element that calculates the SOC, the two depressed semi-circle elements ($Z_{arc,1}$ and $Z_{arc,2}$) and the Open Circuit Voltage (OCV) element. The battery inductance is not taken into account since the inductance of cabling and connectors to the battery are much more significant.

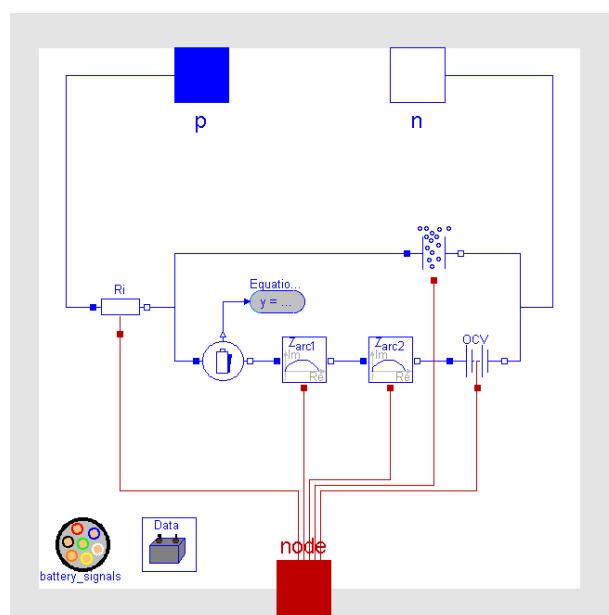


Fig. 4 Implementation of the battery model

The SOC element (circle in Fig. 4) has been added to monitor the energy content of the battery. Since

gassing current is not stored in the battery, this SOC element is positioned in the lower branch.

Also added to the components of the battery model are thermal connectors (going to the 'outside' thermal connector node). Not only the behavior of the battery is dependent on temperature, but the battery also generates a heat flow. If the heat capacity of the battery is known, the self-heating effect of the battery can be simulated. This self-heating effect is of minor effect (on the timescale of for instance a NEDC drivecycle) for a regular flooded battery. When however a more advanced lead-acid battery of the AGM-type is used, the self-heating effect can become significant. More detailed information of thermal battery modeling can be found in Berndt [4].

Since each battery type needs its own impedance spectroscopy measurement and parameterization, the battery model has been programmed in a way that allows to change the battery type (and its corresponding parameter set) in the parameter window (Fig. 5).

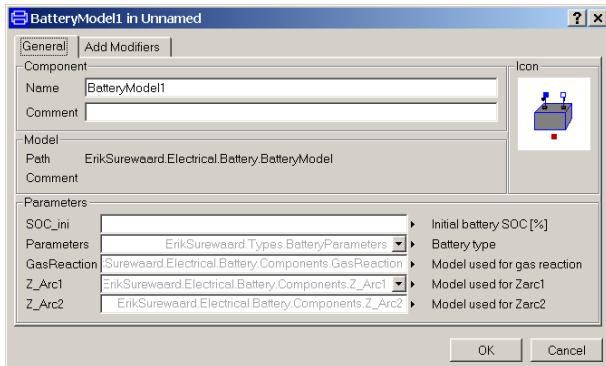


Fig. 5 Parameter window for the battery model

In this parameter window it is possible to change the battery initial charge (SOC_{ini}), its parameter set (Parameters), the models that are used for the gas reaction (GasReaction) and the description of the first and second depressed semi-circle (Z_{Arc1}, Z_{Arc2}). Currently, there are three types of parameter sets available:

- Ford Motorcraft SLI flooded battery, 12V, 70Ah
- Hoppecke AGM, 36V, 27.5Ah
- JCI Optima Red Top, AGM, 12V, 44Ah

It is also possible to 'design' new batteries by changing the voltage and capacity in the parameter set. This should however be done very carefully, since differences in technology and construction over different type of batteries exist.

The model enables replacing the model of the first and second depressed semi-circle ($Z_{arc,1}$ and $Z_{arc,2}$). This makes it possible to (i) change the number of RC circuits (Fig. 3) and (ii) remove the capacitance of the $Z_{arc,1}$ element. When larger simulation time-steps are taken (in the order of 0.01s), the capacitance of the first $Z_{arc,1}$ element can be neglected since their time constants are typically smaller than 0.01s. Removing this capacitance will increase simulation speed.

4 Supercapacitor

As was the case with the battery model, the Modelica supercapacitor model is based on work of the Aachen University RWTH. Both model background and the Modelica implementation are discussed in this section.

4.1 Model Background

As with the battery model, use has been made of impedance spectroscopy measurements to characterize the supercapacitor behavior. For this purpose, the supercapacitor is excited with AC currents in different operating points: temperature and voltage. Fig. 6 shows a typical impedance curve for a supercapacitor.

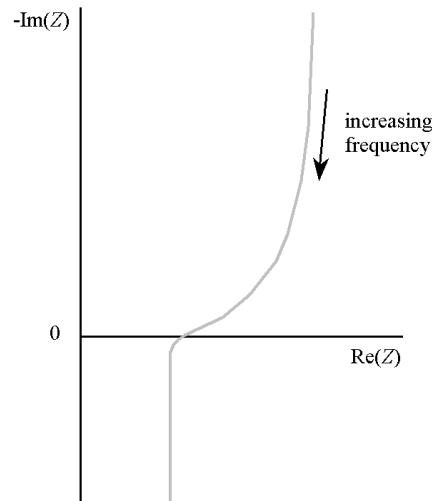


Fig. 6 Schematic plot of an impedance measurement of an automotive supercapacitor

To represent an impedance measurement of a supercapacitor with an electric circuit, Buller suggests in [5] to use the equivalent circuit shown in Fig. 7. For the pore impedance Z_{pore} there are two implementation forms possible: (i) with an RC-series networks and (ii) an RC-ladder network.

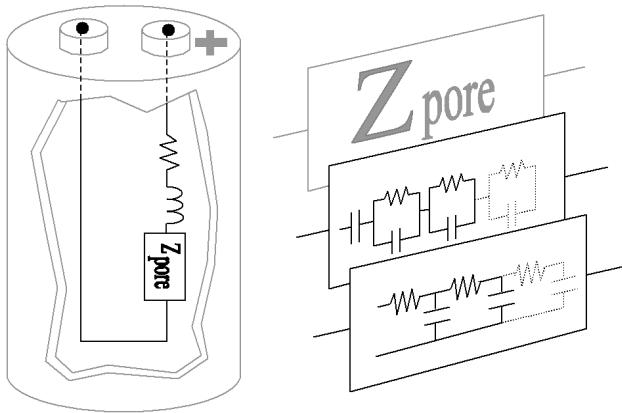


Fig. 7 Equivalent electric circuit for a supercapacitor cell and the two implementation forms for the pore impedance Z_{pore} (RC-series and RC-ladder circuits)

4.2 Model Implementation

The model, as described in the previous section, is constructed in Modelica. The number of RC-circuit in either the RC-series or RC-ladder network can be chosen. A for-loop has been used to connect these RC-circuits. A code fragment of the supercapacitor where the RC-circuits are connected is:

```
connect(p, R.p);
connect(R.n, Rpore[1].p);
for i in 1:numberRC loop
  connect(Rpore[i].n, Cpore[i].p);
  connect(Cpore[i].n, n);
  if (i < numberRC) then
    connect(Rpore[i].n, Rpore[i+1].p);
  end if;
end for;
```

First the positive connector on the supercapacitor is connected to the resistor (R). After that the pore impedance is represented by the RC-ladder method. The number of RC-ladders is determined by the parameter numberRC. The inductance of the supercapacitor is not taken into account in the model, since it is assumed that it can be neglected compared with the inductance of the connection and the cabling to the supercapacitor.

As with the battery, a parameter window is made available in which the parameter set (for the specific supercapacitor) can be chosen. This window is displayed in Fig. 8. The number of cells, initial cell voltage, number of RC-circuit for the approximation of the pore impedance Z_{pore} and the parameter set (type of supercapacitor) can be chosen. Currently, the following parameter sets are available:

- Montena 1400F
- Montena 2600F
- NESS 5000F

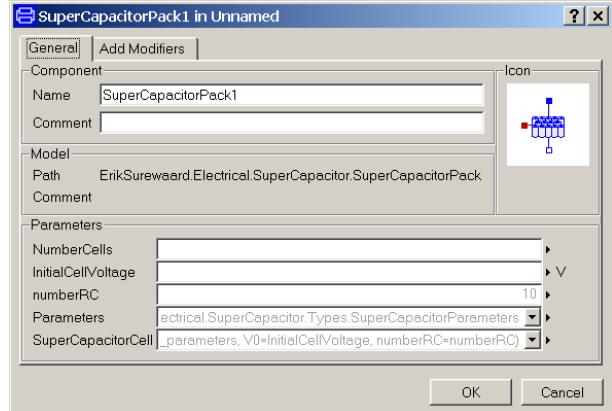


Fig. 8 Parameter window for the supercapacitor model

5 Simulation results

The battery and supercapacitor model will be simulated in a model of the so-called dual storage system. This section will first describe the dual storage system after which the Modelica implementation and the simulation results are displayed.

5.1 Dual Storage System

The dual storage system, also known as the 14+x system, is displayed schematically in Fig. 9. More information on the idea behind the dual storage system can be found in [1].

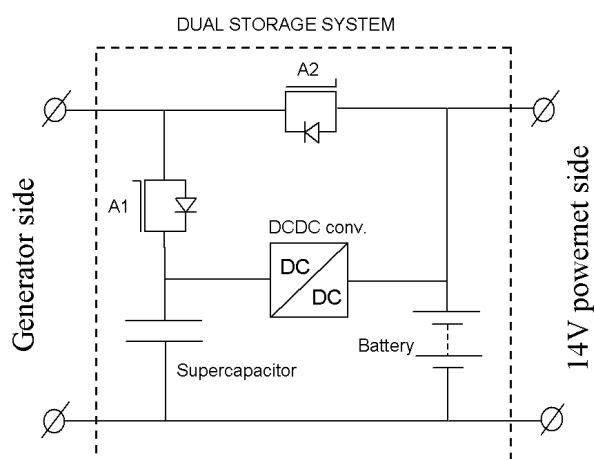


Fig. 9 Electric circuit representation for the dual storage system (14+x)

The dual storage system is particularly interesting for use with a B-ISG. The B-ISG is actually an advanced alternator, which has a higher efficiency and facilitates both generating and motoring mode. In this way the B-ISG can be used to crank the engine (*i.e.* the starter motor can be omitted). To deliver a high torque up to a high engine speed during engine cranking, it is beneficial to have the B-ISG operate on a higher voltage during cranking than is the case during generating. A storage system that can supply the B-ISG with two voltage levels for cranking and generating mode is the dual storage system:

- During motoring of the B-ISG, switch A1 is closed and A2 open. In this case the B-ISG is connected to the voltage of the supercapacitor. Since the supercapacitor voltage is not connected to the powernet, this voltage is therefore allowed to fluctuate significantly. The lower voltage-level of the supercapacitor is determined by the minimum required for cranking. The upper voltage level is determined based on the nominal supercapacitor voltage (lifecycle).
- During generating of the B-ISG, switch A1 is open and A2 closed. In this case the B-ISG is connected to the battery and the powernet, which are at a relatively constant voltage of 14V compared to the supercapacitor voltage, which is allowed to fluctuate.

A bi-directional DC-DC converter is used to enable a current flowing between the battery and supercapacitor and vice versa.

5.2 Modelica Implementation

A model of the dual storage system, as displayed in Fig. 9, is constructed in Modelica. The DC-DC converter is modeled with a table lookup model. The MOSFET switch has been modeled as the parallel connection of an ideal switch and an ideal diode with threshold voltage V_d . The code for this MOSFET switch is:

```
model IdealMosfetSwitch
  import Modelica.Electrical.Analog;
  import Modelica.Blocks.Interfaces;
  extends Analog.Interfaces.OnePort;
  parameter Real Ron(final min=0) = 1E-5;
  parameter Real Goff = 1E-5;
  parameter Real Vd;
protected
  Real s;
  Boolean on;
  Boolean u;
```

```
public
  Interfaces.BooleanInPort BooleanInPort1;
equation
  u = BooleanInPort1.signal[1];
  on = not (u) or not (s < Vd);
  if not (on) then
    v = s;
    i = s*Goff;
  else
    if u then
      v = Vd + (s - Vd)*Ron;
      i = s - Vd;
    else
      v = (s - Vd)*Ron;
      i = s - Vd;
    end if;
  end if;
end IdealMosfetSwitch;
```

The resulting model is displayed in Fig. 10. Since this model will be used in fuel economy studies in Simulink¹, input and output connectors have been used for the switching and sensoring signals.

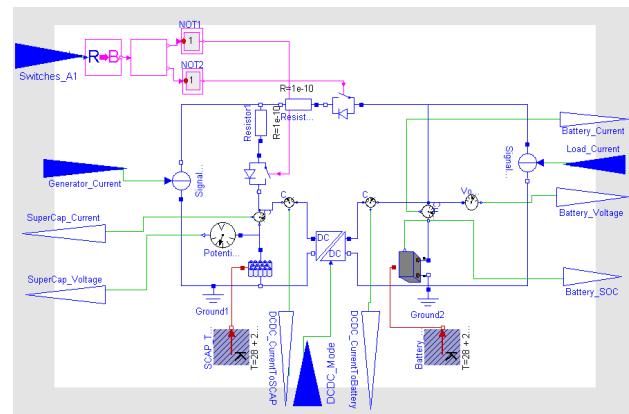


Fig. 10 Modelica model for the dual storage system

The model displayed in Fig. 10 is compiled to a Simulink native S-function block (Fig. 11):

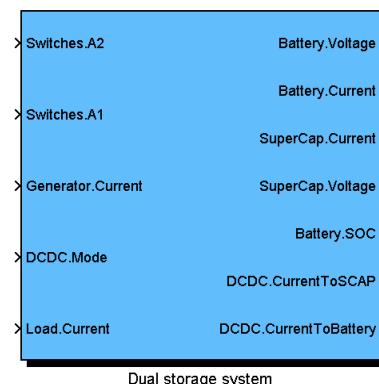


Fig. 11 Simulink block, compiled from the Modelica model, which represents the dual storage system

¹ Simulink is a registered trademark of The MathWorks, Inc.

The reason for making a Simulink S-function of the dual storage system, is that presently Simulink is the standard for control systems development within the Energy Management Group of FFA. That constructing the physical plant model in Modelica has advantages compared with plant modeling in Simulink is shown in [6].

5.3 Simulation Results

Simulation has been performed in Simulink using the block in Fig. 11. Model inputs are taken from a measurement that is performed with hardware of the dual storage system. The results for the battery are displayed in Appendix A. It should be mentioned that the results for the battery voltage do not completely agree due to the fact that the overcharging behavior of the battery has not yet been completely modeled. This will be improved in future versions of the battery model. In [6] it is already shown that the battery model shows excellent results in discharging operation.

6 Conclusions

This paper shows complex models for a battery and supercapacitor. These models are based on impedance spectroscopy and have been modeled in Modelica. Using these models, a dual storage system is constructed and simulated. The simulation results have been compared with measurement results.

Acknowledgement

The authors gratefully acknowledge the collaboration with Aachen University of Technology, Institute of Power Electronics and Electrical Drives (RWTH-ISEA), especially Stephan Buller, Marc Thele and Dirk Linzen who developed the theory behind the physical representation of the battery/supercapacitor model used in this paper, and the method of its parameterization.

The authors also wish to thank Daniël Kok, team leader of the Energy Management Group of FFA for his support and ideas.

Contact

Erik Surewaard is a member of the Energy Management Group of FFA. He graduated his studies of mechanical engineering at Eindhoven

University of Technology in February 2002 on a model, which he developed for describing what occurs during cold cranking of an internal combustion engine. He continued working for the Energy Management Group and now develops models, using Simulink and Dymola, to describe (i) the electric powernet and (ii) the engine cranking process. He can be reached by mail on:

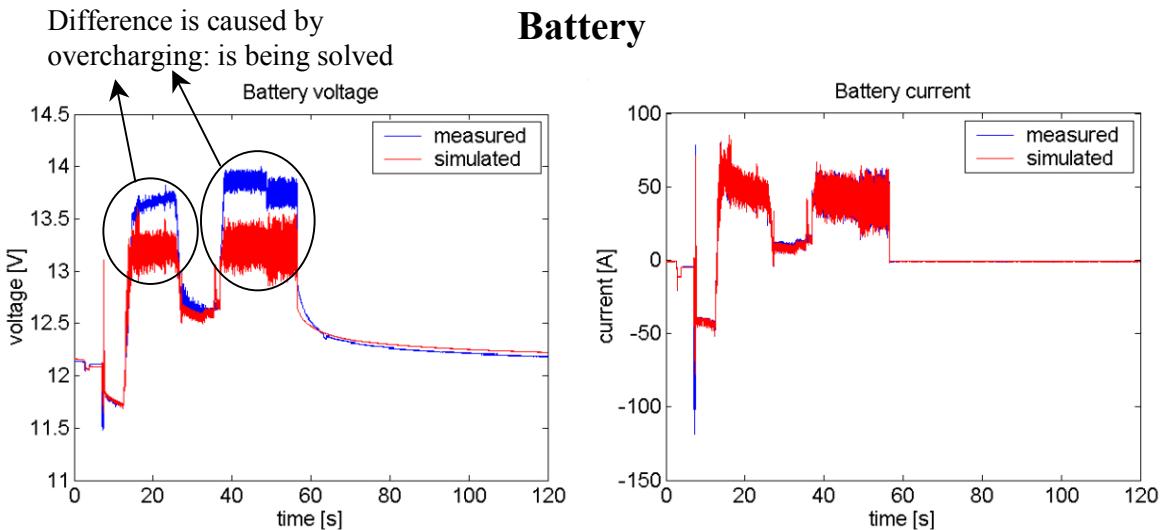
Erik Surewaard
Ford Forschungszentrum Aachen (FFA)
Süsterfeldstrasse 200
52072 Aachen, Germany

Email is also possible: esurewa1@ford.com

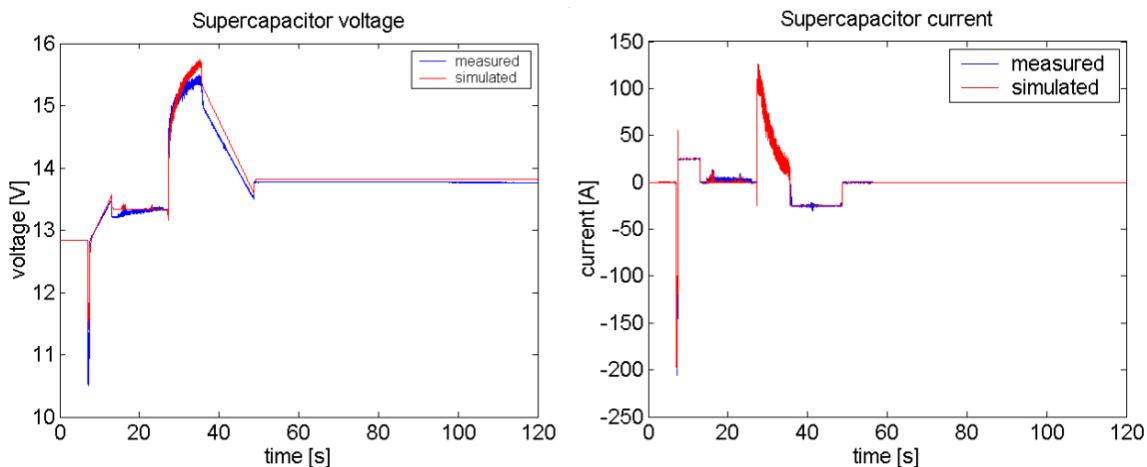
References

1. **Sebille, D.**, "Electrical Energy Management: 42V Perspective", MIT 42V meeting Dearborn, March 6th, 2003
http://mit42v.mit.edu/Members/Meetings/2003-03-Dearborn/Presentations/Sebille_Valeo.pdf
2. **Buller, S.**, "Impedance-based simulation models for energy storage devices in advanced automotive power systems", Shaker-Verlag, Aachen, 2003
3. **Buller, S., Thele, M., Karden, E., De Doncker, R.W.**, "Impedance-based non-linear dynamic battery modeling for automotive applications", Journal of Power Sources 113, pp. 422-430, Elsevier, 2003
4. **Berndt, D.**, "Valve-regulated lead-acid batteries", Journal of Power Sources 100, pp 29-46, Elsevier, 2001
5. **Buller, S., Karden, E., Kok, D. and De Doncker, R.W.**, "Modeling the dynamic behavior of supercapacitors using impedance spectroscopy", IEEE transactions on Industry Applications, Vol. 38 No. 6 Nov/Dec. 2002, pp 1622-1626
6. **Surewaard, E., Tiller, M. and Linzen, D.**, "A Comparison of Different Methods for Battery and Supercapacitor Modeling", SAE paper 2003-01-2290, 2003

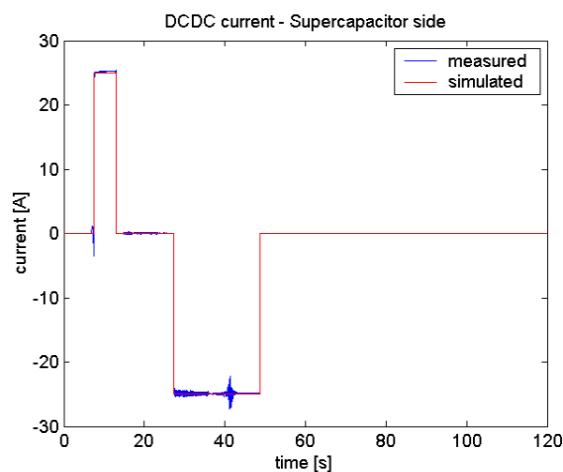
Appendix: Simulation Results



Supercapacitor



DC-DC converter



Session 3B

Tools – I

IDA Simulation Environment

a tool for Modelica based end-user application deployment

Per Sahlin

Pavel Grozman

Equa Simulation AB
 Box 1376, 172 27 Sundbyberg, Sweden
<http://www.equa.se/>

Abstract

A new Modelica implementation based on IDA Simulation Environment (IDA SE) is presented. IDA SE is primarily used for development of equation based simulators for end-users with limited modeling skills but provides interesting features also for the advanced user. A recently developed Modelica application for simulation of tunnel ventilation for commuter rail networks illustrates IDA usage. Excerpts of models from this application are presented in some detail as well as a list of present limitations of the IDA based Modelica implementation.

1 Introduction

Modelica has proven to be of excellent service to advanced modelers in several domains. However, presently, there is usually close contact between model developers and end-users. In fact, they frequently coincide in a single person. As Modelica uptake evolves, the need to deploy Modelica based simulators among less experienced users is likely to increase. IDA Simulation Environment (IDA SE) has been developed to facilitate this process. Originally based on a Modelica predecessor, NMF [1], IDA SE has been used for equation based end-user application development since the early nineties. Several real-scale simulation applications have been developed, some of which have earned leading roles in their respective markets.

IDA SE is based on the concept of pre-compiled component models, i.e. most IDA application end-users work only with fixed¹ component models that may be combined into arbitrary (input-output free) configurations without need for compilation. Simulators do not require a working compiler installation.

Encryption is not needed to preserve component model secrecy. The new Modelica implementation which has been included in the IDA SE package retains this structure, separating the typical roles of the model developer and end-user.

A large majority of potential simulation users have little appreciation of the beauty and generality of an advanced modeling language. They have a design problem to solve and want quality answers with minimum effort. Quite often the full mathematical formulation of the problem is of less interest. A good simulation application must communicate in terms natural to the user and in most situations this does not involve any modeling language but rather physical concepts from the target application. Pipes, pumps and valves may well be the optimal elements of communication rather than differential-algebraic equations.

The structure and main features of IDA Simulation Environment are presented in the next section. In Section 3, a sample IDA application is presented, followed by a discussion about the current state of the Modelica implementation. Some code details from train traffic modeling are discussed in an Appendix.

2 IDA Simulation Environment

Figure 1 shows the three main software modules of IDA SE:

IDA Modeler:	the interactive front-end
IDA Solver:	the numerical DAE solver
IDA Translator:	the model source code editor and processor

A development version contains all three, while a runtime installation lacks IDA Translator. The developer uses IDA Translator to generate a set of C

¹ array sizes, including connector arrays, can be modified after compilation

or F77 routines for each component², for equation evaluation, analytical Jacobian evaluation and general information about the model. The code is compiled from the translator into a Windows DLL which is then linked to IDA Solver. The Modelica (or NMF) source may or may not be shipped with the application, depending on the desired level of confidentiality. Also generated are native class descriptions for IDA Modeler, containing structural information about the model library. This code may then be complemented by application specific extensions.

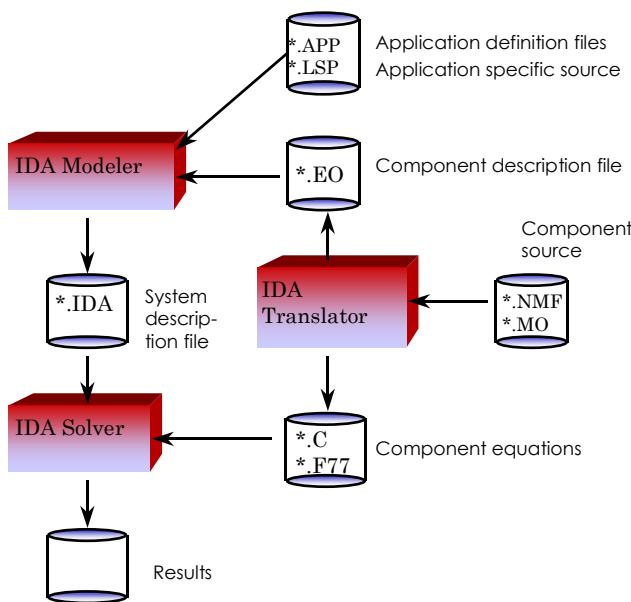


Figure 1: Structure of IDA Simulation Environment

Applications may be shipped stand alone, including an IDA runtime environment or as separate plug-ins for an existing IDA environment. Both the model library and the user interface of an application may be amended and altered by multiple extra separate installations, for customizations and application extensions. This allows efficient management of complex version structures.

The cost of the runtime environment for each installation is significantly lower than that of the full development environment, normally only a small fraction of the cost of the end-user product.

IDA Simulation Environment is presently available as an off-the-shelf product only with NMF for Microsoft Windows 98 or NT 4.0 and higher. IDA Solver and Translator have previously been ported to Unix platforms but are not maintained in this setting. Modelica is presently supported only for specific customers. We will return to the state of the Modelica implementation in Section 4.

2.1 IDA Solver

In tools, such as Dymola, where equations are globally reduced prior to integration, the numerical solver will deal with a fairly dense system of equations but where each equation can be quite complex. One can generally expect equation evaluations to take some time while factorization of Jacobian matrices is likely to be faster due to the dense problem structure. In a pre-compiled setting, the situation is the opposite: functions are rather simple (simple enough to differentiate analytically!) while Jacobians are typically large and sparse.

IDA relies on standard software components for sparse Jacobian factorization. Since large sparse matrices occur in many technical and scientific applications a range of powerful solvers are readily available for scalar as well as parallel architectures. Available solvers for IDA are: SuperLU [2], MUMPS [3] and UMFPACK [4]. The graph theoretical analysis of system structure is done by these external solvers rather than in the context of a global symbolic preprocessing.

There are many implications of this difference in solution strategy. A thorough discussion of this is beyond our current scope and we will merely point out a few aspects:

- + Component structure is maintained during integration. This allows for example: (1) Exploitation of special component structure by tailored methods. (2) Component level co-simulation with external tools such as FEMLAB (see Figures 2 and 3). (3) Component level debugging.
- + Equation topology may change during simulation. Since the graph theoretical analysis may be done in each timestep, discontinuities that alter the system structure can be accepted.
- + For few-timestep simulations, global compilation may take a significant part of the total execution time.

² A component or a *compilation unit* becomes an indivisible building block in the end-user application. The Modelica source of a component model may be a composite, hierarchical model. It is also possible to define hierarchical models in IDA Modeler containing multiple components.

- Pre-compiling component models precludes some operations that are natural in a setting where a global symbolic analysis is done. The most serious limitation concerns index reduction. Although index 2 systems generally can be simulated without any problems in IDA Solver, serious high-index problems are most likely better solved by means of global symbolic analysis.

A selection of methods for initial value computation are available: damped Newton, line-search, gradient and homotopy (embedding) methods

2.2 IDA Modeler

IDA Modeler provides a framework for interface development. It may be used to write simulation oriented applications of sufficient quality for competition with tools written from scratch but at a fraction of the cost. IDA Modeler exploits the fact that many tasks are common to most simulation applications: building and presenting models, editing parameters, interacting with a data base, making simulation experiments, viewing results as diagrams and reports, checking user licenses etc.

More elaborate IDA applications, divide the user interface into three levels, to serve users with different needs and capabilities:

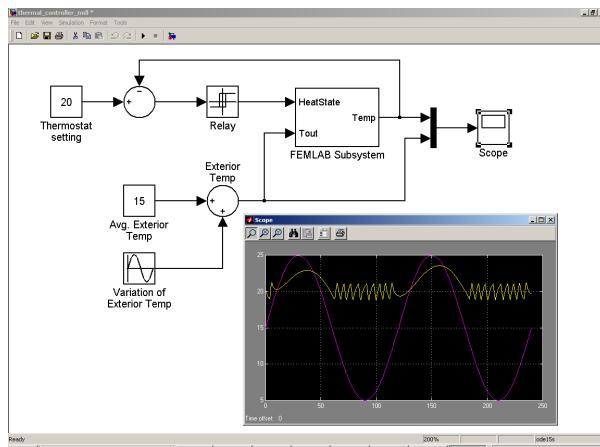


Figure 2: A FEMLAB-Simulink standard case “Thermal controller.” A heat source in a 2D region is controlled by a thermostat.

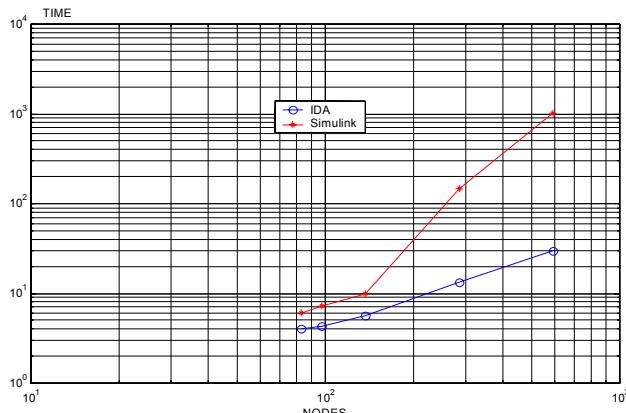


Figure 3: Execution time vs. FEMLAB spacial resolution in the “Thermal controller case”. The original Simulink model is compared to an identical FEMLAB-IDA model (from [5]).

IDA Solver is a variable timestep and order solver based on the MOLCOL implicit multistep methods, which include the most common implicit methods such as BDF. Explicit methods are currently not available for the global integrator but may be implemented for individual components.

Wizard level:	Least demanding. Each required input is presented in a sequence of user input forms.
Standard level	Intermediate. The user is required to formulate a model, but in terms that are natural to the domain.
Advanced level	The user builds a model using equation based objects. Facilities for model checking, automatic mapping of global data, selection of given variables and similar tasks are available.

In such an IDA application, the Advanced level interface offers a model-lab work bench similar to that offered by other DAE environments, providing the user with direct contact with the individual equations, variables and parameters of the mathematical model. However, a great majority of end-users prefer the tools of the Standard and Wizard level interfaces, where the basic mental concept is that of a physical system and not of a mathematical model.

The kernel of IDA Modeler is written in Common Lisp but most application programming is done interactively or by writing native scripts. Extensive facilities are available to simplify common tasks such as: building user interfaces in multiple natural languages; defining a data bases for input data objects; report generation; data mapping etc. Some user interface elements, such as dialog boxes with complex logic, may be written via an API in other languages.

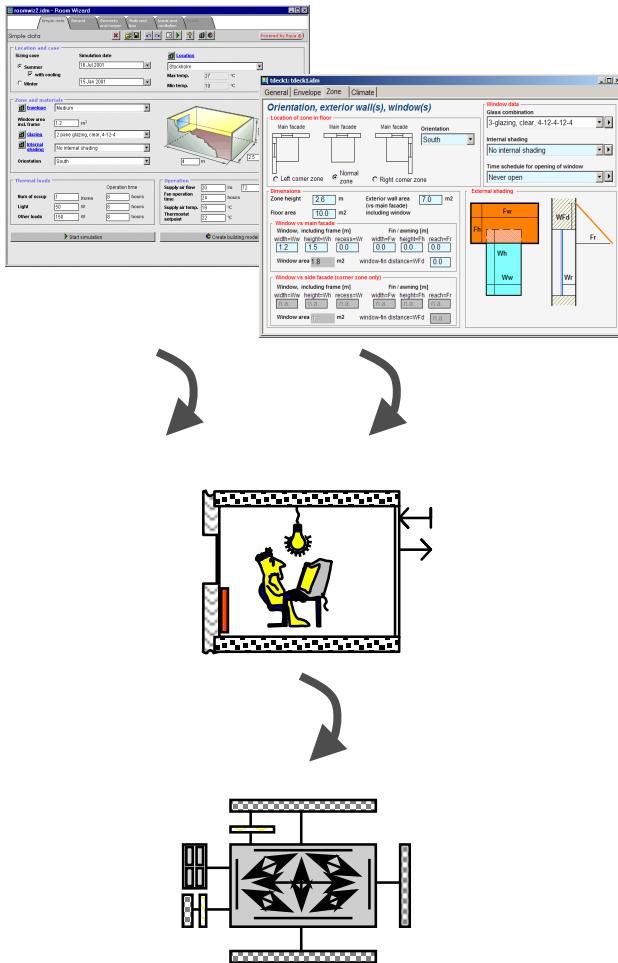


Figure 4: Applications may have multiple Wizard level interfaces for typical simulation tasks. Each interface has a separate data model and a tailored script language for data mapping between levels is provided.

Special emphasis has been laid on tools for development of web clients, running in a browser, powered by an IDA based simulation engine on the (Windows) server. A large portion of the native data structures have been mapped to Java script, facilitating advanced web development with minimum effort.

Several examples of full-complexity applications written in IDA Modeler are available. Equa markets two such applications: IDA Indoor Climate and Energy (IDA ICE) and IDA Road Tunnel Ventilation. Others have been developed for specific customers. IDA ICE is with more than 2000 users a leading international tool for thermal building simulation, available in six languages.

3 Ventilation and fire in commuter rail tunnel networks

The first full-complexity Modelica based IDA application concerns prediction of air flows in tunnels and on platforms of commuter rail networks. Results are needed for several reasons: hygienic ventilation, thermal comfort, smoke propagation in fire scenarios and for gas and particle dispersion studies.

A primarily pulsating air movement through the system is driven by train piston effect. Secondary driving forces are thermal stack effect, wind pressure on portals and openings and possible fan operation.

In this application, air has been modeled as, weakly compressible, i.e. propagating pressure waves have infinite speed but the temperature-density relationship is modeled (perfect gas law) in order to capture the stack effect. Solving the fully compressible equations is often required for rail tunnel studies to predict the effects of interacting pressure waves but this has not been done here since the solution of the resulting hyperbolic equations is likely to be time consuming and otherwise problematic.

Pressure drop in tunnels is modeled in 1D with conventional pipe flow theory: With the fluid is transported a series of fractions for computation of CO₂, age of air etc. Flows with altering directions, often fluctuating around zero, may be numerically difficult to handle in branched systems with high Reynolds number since coarse approximations of viscous losses tends to produce discontinuities. To overcome these problems Gardel [6] empirical formulae have been implemented for viscous loss coefficients, providing continuity around zero flow situations. Bulk air inertia is modeled leading to an index 2 system. Figure 5 shows a model of a four-station section.

A convenient way of expressing train traffic through the system is essential. A design principle has been to separate the models of the tunnel system from the traffic models. Input data for a train route through the system is depicted in Figure 6, including line segmentation, speed limits, accelerations, dwell times at stations etc. To add a new route, the user selects a sufficient number of objects in the direction of the traffic to unambiguously determine a path. The segmentation of the Route need not correspond to the segmentation of the physical tunnel. (The latter may e.g. depend on needed resolution of, e.g., a smoke front.)

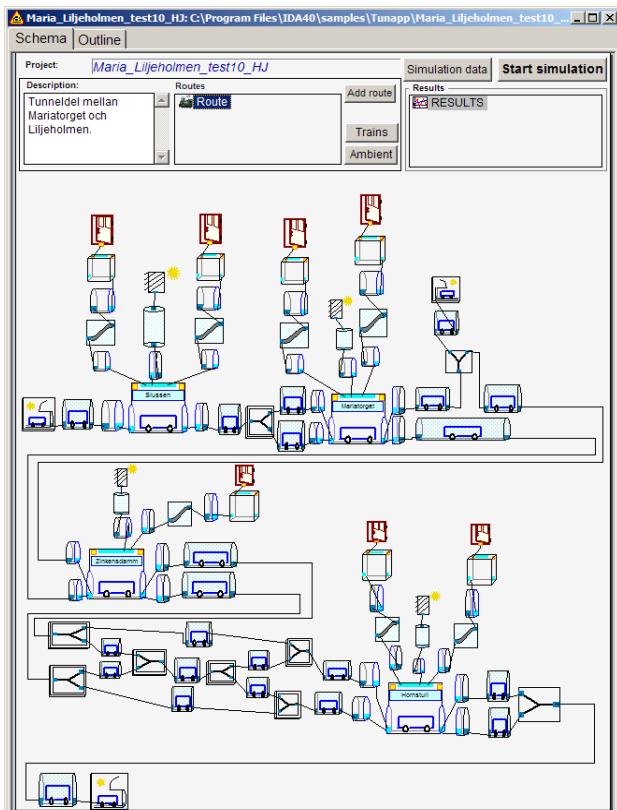


Figure 5: A model of a four-station underground section of the Stockholm subway. Tunnels and other airflow paths are modeled.

Each Route through the system is contained in a single instance of the Route block (code extract in Appendix). This block is then automatically connected to each segment of the physical tunnel using application specific code. The connection lines are not visible, since the number of tunnel segments may be exorbitant.

Seg. #	Length [m]	Max speed [km/h]	Acceleration [m/s ²]	Retardation [m/s ²]	End speed [km/h]	Dwell time [s]
1	190	72	1.0	1.1	0	40
2	965	72	1.0	1.1	0	40
3	710	72	1.0	1.1	0	40
4	740	72	1.0	1.1	0	40
5	825	72	1.0	1.1	72.0	0.0

Figure 6: The IDA form for description of a train route through the system.

The management of train routes is a good example of application specific programming, where the standard drag, drop and connect functionality needs to be complemented. The Route form in Figure 6 is an example of a native IDA form, which first has been automatically generated and then subsequently interactively altered. In the Outline tab, the user can see all available parameters, variables and interfaces of the block and in the Code tab, the Modelica code can be browsed (but not edited).

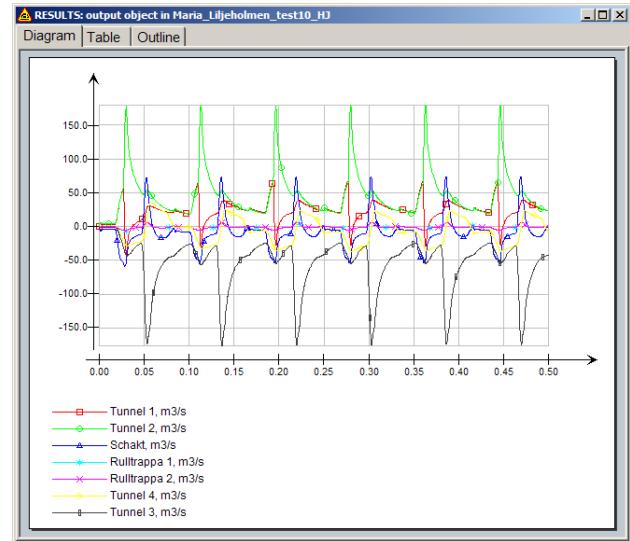


Figure 7: Computed airflows at station Mariatorget, with five minute traffic of C20 trains in one direction.

4 Present state of Modelica in IDA

The current IDA Modelica implementation has been developed to cater to the immediate modeling needs of ongoing projects like the mentioned subway ventilation study. It is our intention to continue to enhance the tools in the scope of cooperative modeling projects and then, at some future point, release an off-the-shelf product.

The design of the Modelica language itself has for natural reasons been centered around the only presently available implementation by Dynasim. In this section, we will outline a few issues where the present Modelica design is less well suited to usage in the pre-compiled setting of IDA and where Modelica extensions have been introduced. Present shortcomings of the implementation are also discussed.

4.1 Interpretation of Modelica code

The IDA Translator compiles classes, not complete systems. Compiled models normally contain:

- public connectors
- more variables than equations
- outer elements
- arrays with non-constant sizes

All public non-partial and non-local classes declared with keywords `class`, `model` or `block` are compiled to IDA components. Blocks are presently compiled to IDA algorithmic models. Public non-partial and non-local atomic types and connector classes are similarly compiled to IDA quantity and link types.

A compiled model may be extended after compilation by inserting and connecting submodels.

Public top-level connectors in compilation units are preserved by the compiler available for connections.

Compilation units may contain unresolved outer components. Such compiled models should be used only as elements of models that contain corresponding inner components. Unresolved outer classes are not supported.

For each compilation unit, a symbolic analysis is performed where as many variables as possible are solved for symbolically, effectively removing them from the global system of equations. Resulting equations are differentiated and code for evaluation of analytical Jacobians is generated. Although principally possible, no index reduction is currently done at this stage.

It is possible to allow the IDA Translator to process entire simulation problems, resulting in just a single compilation unit. However, this is not the intended usage of the tool since the topological flexibility of being able to re-configure pre-compiled units is an essential feature of most IDA applications.

4.2 IDA driven Modelica extensions

Events in functions and pre operator

The previous IDA language, NMF, supports events in functions, also in foreign functions. This is possible because the variables that monitor events are explicit in NMF models. In Modelica, these variables are automatically generated and not available for the programmer.

We have implemented events using the special function `mo_event(var, expr)`. The variable `var` is a special kind of variable (called *assigned state* in

NMF) that keeps its value from the previous timestep. The function modifies the value of `var` and generates an event whenever it changes sign. In order to be used in a function, the previous value of `var` should be passed to the function and the modified value should be returned. To make this possible, we have changed the semantic of the `pre` operator. In our implementation, `pre(v)` is always the value of `v` at the previous successful time step; this is also valid for non-discrete variables.

The modified `pre` operator may also be used for several other purposes, for example:

- To calculate a maximum value during the simulation:
 $xMax = \max(x, \text{pre}(xMax));$
- To break an algebraic loop in order to simplify solution of an equation with weak dependences:
 $RhoAir = 1/287.0 * \text{pre}(PAir) / Tair;$
- To implement local integration methods, for efficiency or for limiting numerical dissipation in PDE:s

A full account of the arguments for the extension of the `pre` operator is beyond the scope of this paper. However, uncontrolled numerical dissipation due to large and variable timesteps is a fundamental problem for many Modelica applications that should be further discussed.

Conversion to strings

In Modelica 2.1 there are functions that converts scalar values to strings, but there are no functions for converting arrays and matrices. We have implemented automatic conversion of non-strings to strings. Example:

```
assert(x>0, "x = "+ x + " should be positive")
```

Graphics

- More named colors
- Arrow: Closed, Left, Right, {type,side}. The size may be a vector
- lineThickness=0 - non-scaled minimal thickness
- Transformation: negative scale and aspectRatio may be used instead of flip.

4.3 Features yet to be implemented

The following list is intended to give a flavor of the present state of development.

Available variable and parameter types

- All variables and non-scalar parameter declared as Integer or Boolean are converted to Real. These variables cannot be used as arguments of function calls.

- Boolean scalar parameters are converted to Integer.
- String variables are not implemented (string parameters are supported)
- Attributes (except value and start) should be constant. They cannot be used in expressions.
- Attributes displayUnit, fixed, enable, nominal, stateSelect are not used.

Connections

- Connection of subconnectors is not yet supported

Modification and redeclarations

- Modifications of class elements are not supported (i.e., when instantiating or extending a class, it is not possible to modify local classes in that class).
- No subtype checking in redeclarations. The constraining clause is ignored.
- Choice annotations not supported.

Expressions

- Record constructors are not supported.

Iterations

- Multiple iterations (separated by “;”) not yet supported.
- Ranges with step from:step:to are not supported.
- Vectors in indices only partially supported.
- The index **end** is not supported.
- Deduction of range is not implemented.

Arrays

- Array expressions (not instances) may not be used as arguments to non-built-in functions.

Functions

- Optional arguments are not supported (except in some built-in functions)
- Record arguments are not supported.
- Protected variables in functions are not supported.
- The annotation derivative is not yet supported.
- Some restrictions on external functions.
- Not all Modelica utility functions are implemented.
- External objects are not implemented.

Initialization

- Initial equation/algorithm not implemented

Built-in functions and operators

- Not implemented functions: `initial`, `terminal`, `smooth`, `sample`, `edge`, `change`, `reinit`, `terminate`, `div`, `rem`, `integer`, `cardinality`.

Graphics

- Attribute visible and smooth is ignored.

- Cylinders and Sphere fill patterns are not supported.
- BorderPattern shown as rectangle with 3D border
- No line pattern if lineThickness ≥ 0.375
- Text rotation is not implemented
- Filled text is not implemented.
- Bitmaps: may be rotated by 90 degrees only, `imageSource` not implemented, `fileName` just copied (no directory information added).

5 Summary and further work

The present IDA Modelica implementation is a sufficient base for complex application development and delivery. Several partner projects are underway, where Equa supports developers with needed new functionality. Perceived user demand will determine when a public product is released.

Equation based simulation is presently limited by fragmentation into disparate single-vendor user communities. As a technology, Modelica is sufficiently neutral and powerful to break the present status quo. Hopefully, another reasonably complete independent implementation will aid this process. However, it is vital that the present Modelica community focuses on the truly critical success factors rather than on yet another intriguing technical issue.

References

1. P.Sahlin, E.F.Sowell, „A Neutral Format for Building Simulation Models“, Proceedings of the IBPSA Building Simulation '89 conference, Vancouver, Canada, 1989
2. J.W. Demmel, J.R. Gilbert and X.S. Li, “SuperLU User’s Guide”, Technical Report, UC Berkeley, USA, 1997
3. P.R. Amestoy, I.S. Duff, J.-Y. L’Excellent, “MUMPS Multifrontal Massively Parallel Solver v. 2.0”, Technical Report, CERFACS, France, 1998
4. T.A. Davis, “UMFPACK v. 4.0 User Guide”, Technical Report, Univ. of Florida, Gainesville, USA, 2002
5. C. Panagiotopoulos “Finite element models in a lumped model simulation environment. An interface between FEMLAB and IDA S.E.” Technical Report, KTH, Stockholm, 2001
6. Gardel, A. (1957), “Les Pertes de Charge dans les Ecoulements au Travers de Branchements en Te”, Bull. Tech. De la Suisse Romande, 83, 123-130, 144-148, 1957

Appendix - Structure of commuter rail model

The Traffic connector transmits information about train location, speed and acceleration between the Route block and the physical tunnel model:

```
connector Traffic "Traffic line in tunnel segment"
  Velocity speed(start=0) "traffic speed";
  Real nFront "no of vehicle fronts per segment";
  Real nBack "no of vehicle backs per segment";
  Length lBody "total length of vehicles per segment";
  Acceleration acc(start=0) "traffic acceleration";
end Traffic;
```

Below is the template for a Tunnel system. The end user may add instances of different models (sections, platforms, ventilation shafts, traffic routes) into a compiled Tunnel system and then connect and simulate the system (see Figure 5).

```
// The template for Tunnel document
model Tunnel "Tunnel Document"
  inner parameter ArraySize nFract = 2 "Number of air fractions";
  inner parameter ArraySize nVeh=1 "Number of vehicle types";
  inner parameter Vehicle[nVeh] veh "Description of vehicles";
  inner parameter Fraction[nFract] fract "Description of air fractions";
  Ambient amb "Properties of ambient air";
end Tunnel;
```

A traffic route is modeled as a Modelica block. Each instance describes a route in one direction. The model is connected (using traffic connector) with segments in tunnel sections and platforms (a tunnel section may consist of several segments). The connection is done by the application; the user only draws the route on the tunnel schema.

The route block is translated to an *algorithmic* model. It does not add equations to the tunnel system, but only supplies the system with input data series (like a table). IDA SE supports also *post-processing* algorithmic models, used for collecting and transforming measurements on a model.

```
block Route
// Array sizes
  parameter ArraySize
    nSched = 2 "Number of points in route schedule",
    nSeg = 1 "Number of tunnel segments",
    nRun = 5 "Max number of scheduled vehicles";
  final parameter ArraySize nPos = nSeg + 1 "Number of segment ends";

// Route schedule
  parameter Time tSched[nSched] = {0, 3600} "time column in schedule";
  parameter Velocity vSched[nSched] = {10, 10} "speed column in schedule";
  parameter Length xSched[nSched] "position column in schedule";
  parameter Length xSched0 = 0 "start position for schedule";
// Tunnel segments
  parameter Length lSeg[nSeg] "segment lengths";
  parameter Boolean reverse[nSeg] = fill(false,nSeg) "traffic direction";
  parameter Length xSeg[nPos] "segment ends";
// Time schedule
  Integer lastRun(start=0) "last scheduled vehicle";
  discrete Time
    nextDep(start=time.start) "Next departure time",
    interval(start=300) "departure interval",
    depTime[nRun] (each start=-1) "Departures time";

  parameter input Integer vehicleType = 1;
  output Traffic[nSeg] traffic;
  outer parameter ArraySize nVeh;
  outer parameter Vehicle[nVeh] veh "Description of vehicles";

protected
  Length xFront, xBack, xF, xB;
  Velocity v;
  Acceleration a;
  parameter Length lVeh = veh[vehicleType].length;
  parameter Time tMax "max route time";
  parameter Time tFront[nPos], tBack[nPos];
```

```

// parameter processing
algorithm
  // Calculate the train position at scheduled time points
  xSched[1] := xSched0;
  for i in 1:nSched-1 loop
    xSched[i+1] := xSched[i] + 0.5*(vSched[i]+vSched[i+1])*(tSched[i+1]-tSched[i]);
  end for;
  // maximal time per route
  tMax := tSched[nSched] +
    (if vSched[nSched]==0 then 0 else lVeh/vSched[nSched]);
  // segment lengths
  lSeg := xSeg[2:nSeg+1] - xSeg[1:nSeg];
  // the time then the train passes tunnel segments
  for i in 1:nPos loop
    tFront[i] := RouteTime(xSeg[i], nSched, tSched, xSched, vSched);
    tBack[i] := RouteTime(xSeg[i]+lVeh, nSched, tSched, xSched, vSched);
  end for;

algorithm
  // calculate the traffic parameters on each segment
  // the tunnel segments reads them (using traffic connector)
  // Launch the next train
  when time>=nextDep then
    lastRun := mod(lastRun, nRun) + 1;
    assert(depTime[lastRun]<0, "The max number of scheduled trains is exceeded");
    depTime[lastRun] := nextDep;
    nextDep := nextDep + interval;
  end if;
  // Initialize output variables
  for iSeg in 1:nSeg loop
    traffic[iSeg].speed := 0.0;
    traffic[iSeg].nFront := 0.0;
    traffic[iSeg].nBack := 0.0;
    traffic[iSeg].lBody := 0.0;
    traffic[iSeg].dSpeed := 0.0;
    traffic[iSeg].acc := 0.0;
  end for;
  // loop over all running trains
  for iRun in 1:nRun loop
    if depTime[iRun]>=0 then // if not removed
      if time >= depTime[iRun] + tMax then
        // the train is out of tunnel, remove it
        depTime[iRun] := -1;
      else
        // calculate the position, speed, and acceleration
        (xFront, v, a) :=
          RouteInt(time - depTime[iRun], nSched, tSched, xSched, vSched);
        xBack := xFront - lVeh;
        // loop over tunnel segments
        for iSeg in 1:nSeg loop
          // calculate the position of the train in the segment
          xF := xSeg[iSeg+1];
          xB := xSeg[iSeg];
          // is the train on the segments (with events)?
          if time>depTime[iRun]+tFront[iSeg] and time < depTime[iRun]+tBack[iSeg+1] then
            traffic[iSeg].speed := if reverse[iSeg] then -v else v;
            traffic[iSeg].acc := if reverse[iSeg] then -a else a;
            if time<=depTime[iRun]+tFront[iSeg+1] then
              // count the train fronts
              xF := xFront;
              traffic[iSeg].nFront := traffic[iSeg].nFront + 1;
            end if;
            if time>depTime[iRun]+tBack[iSeg] then
              // count the train backs
              xB := xBack;
              traffic[iSeg].nBack := traffic[iSeg].nBack + 1;
            end if;
            // count the total length
            traffic[iSeg].lBody := traffic[iSeg].lBody + (xF - xB);
          end if;
        end for;
      end if;
    end for;
  end if;
protected
  function RouteInt "Integrates the train movement along the route"
    input Time t
    "time elapsed from the start point";

```

```

input Integer n      "number of intervals in the schedule";
input Time tp[n]    "time column in schedule";
input Length xp[n]  "position column in schedule";
input Velocity vp[n] "speed column in schedule";
output Length x     "train position";
output Velocity v   "train speed";
output Acceleration a "train acceleration";
external;
end RouteInt;
function RouteTime "Returns the train time at given position"
  output Time t      "the calculated train time";
  input Length x     "the given train position";
  input Integer n    "number of intervals in the schedule";
  input Time tp[n]   "time column in schedule";
  input Length xp[n] "position column in schedule";
  input Velocity vp[n] "speed column in schedule";
  external;
end RouteTime;
end Route;

```

The tunnel segments and platforms are connected using TunnelCut connector:

```

connector TunnelCut
  outer parameter ArraySize      nFract  "Number of air fractions";
  Pressure                  P;
  flow          MassFlowRate   m_dot(start=0);
  flow          Temp_C         T(start=10);
  flow          HeatFlowRate_M Q;
  Real           vf[nFract];
  flow          MassFlowRate   vf_dot[nFract];
end TunnelCut;

```

The bi-directional flow of air with fractions (of CO₂, NO, dust, smoke etc.) is modeled in a similar way as in the Modelica Fluid package, but the implementation is different.

Here the end-user (working with pre-compiled components) is able to define media properties, especially number of air fractions. Therefore the number of fractions nFract is defined as a parameter and not as a constant as in the Modelica Fluid package.

Automatic translation of Simulink models into Modelica using Simelica and the AdvancedBlocks library

Mike Dempsey
Claytex Services Limited
5 Marston Close, Leamington Spa, UK
mike.dempsey@claytex.com
<http://www.claytex.com/>

Abstract

A new tool, SimelicaTM, is presented for converting Simulink[®] models into equivalent Modelica[®] models. The conversion is achieved while retaining the original structure of the Simulink model. The equivalent Modelica models are built from a new library of components, the AdvancedBlocksTM library.

The AdvancedBlocks library is designed to work with Simelica but also brings a new range of control system component models to the Modelica environment. The blocks are designed to enable the calculation method used to be varied in each particular instance that the block is required. For example, in the DiscreteIntegrator block you can choose from 3 different integration algorithms, whether to apply limits to the integrator or not, and how the initial condition is specified amongst many other options. The main focus is on delivering a user-friendly library to aid control system modelling.

Some example applications will be discussed to illustrate how effective the translation process can be.

1 Motivation

The use of system modeling and simulation is increasing in the automotive industry as we strive to reduce product development times whilst increasing the complexity and quality of the product. As the use of these simulation techniques increases so does the requirement to include more and more detail into the models and to ensure that the interaction between the different systems is being modeled adequately.

For many years Simulink has been the tool of choice for much of the automotive industry to develop both physical and control system models[1,2,3]. The main attraction of Simulink has been its flexibility and the range of toolboxes available to aid control system design, development and calibration. However, many users of Simulink are finding that as the physical system models increase in complexity, the task of developing these models further is becoming increasingly difficult and time consuming. Many are now looking at alternative systems and Modelica based tools are well placed in the market to meet these needs.

The adoption of the Modelica tools is currently limited to those departments within automotive manufacturers that are currently pushing forward the development of complex physical system models[4,5]. This is leading to problems within these companies where the control system engineers are still developing models in Simulink while the design engineers are developing physical system models using Modelica.

Currently tools such as DymolaTM provide methods to generate S-functions from the Modelica models[6] and this then enables the models to be simulated together in one environment. In our experience this method has not been completely successful. We have found that, with our more complex physical models, the Simulink solvers are unable to cope reliably with the generated S-function models. This has led to simulations effectively stalling where the time step becomes so small that the simulation is no longer making progress.

We then simply asked ourselves, why don't we make the process work the other way round? Why not convert the control system model into

Modelica and use that environment to simulate the interactions between control system and physical system. After all Modelica can support a block diagram modeling style and our physical models are working reliably in the Modelica environment.

2 Simelica

2.1 Overview

Simelica is a translation tool for converting Simulink models into equivalent Modelica models. It works as both a command line tool so that its use can be incorporated into scripts and also as a windows tool complete with graphical user interface (GUI).

The translation works by reading the Simulink .mdl file and interpreting this into a Modelica model based on the AdvancedBlocks library. Simelica is capable of dealing with all the modeling methods used in Simulink including:

- From-goto systems
- Signal Bus systems
- Muxed signals
- Data store read/write/memory systems

The majority of the standard Simulink library can be automatically translated into an equivalent Modelica block although there are some exceptions including the MatlabFcn, S-function and Stateflow® blocks.

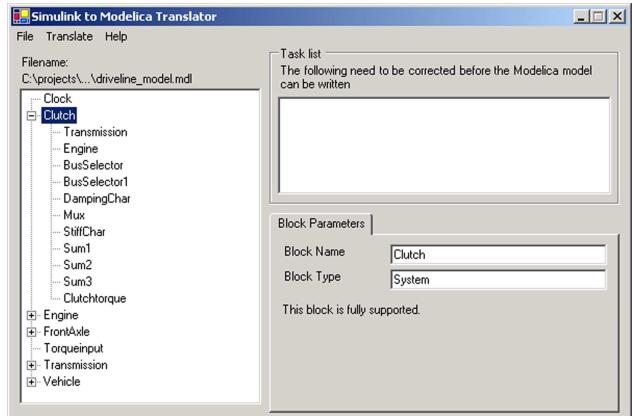
2.2 Using Simelica

The command line version of Simelica provides simple functions to translate a single Simulink file or all the Simulink files contained in a specified directory. This version is useful for incorporation into scripts but it does not provide many of the features available through the GUI that most users will find useful, such as highlighting unsupported blocks. Figure 1 shows a screen shot of Simelica.

When running in GUI mode after the Simulink file is read into the tool the structure of the model is presented to the user. Any unsupported blocks are highlighted to the user at this point along with a brief explanation of what action the user must take either now, or after the Modelica file is generated to ensure that the translated model can be used.

Following translation, a log of the work done is produced. This will list any problem blocks encountered and include their full path in the model. The user can then easily see what, if any, parts of the translated model need further attention before it can be used.

Figure 1: Screen shot of Simelica

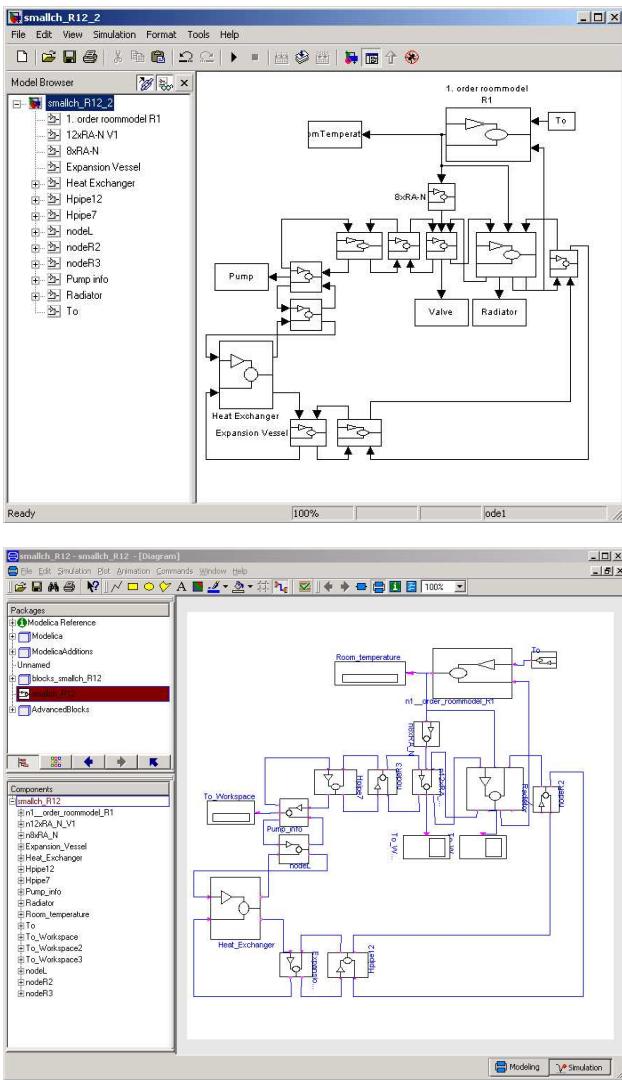


As well as the need to translate a model it is also essential to translate the data from the Simulink environment into the Modelica environment. Data can be imported and incorporated into a translated model using Simelica. The model data has to be stored as a Matlab® binary file, which can then be read by Simelica and the data incorporated into the model through the use of a record that is available in every subsystem.

An additional consideration in the translation of data is that Simulink can load different data files into different points of the model through the use of masked subsystems. In Simelica, masked subsystems are identified and the user is given the option of incorporating a data file directly into each masked subsystem. In this case each masked subsystem gains its own unique workspace record to replicate the fact that Simulink defines local workspaces for masked subsystems.

The Modelica models generated by Simelica are based on the AdvancedBlocks library rather than interpreting the model into a flat model file. This ensures that the model appears similar and maintains the same structure as the original Simulink model. Figure 2 shows a comparison of a translated model in Simulink and Dymola. It shows that the model structure is preserved and the layout and connection of blocks in the Modelica version is similar to the original model.

Figure 2: Comparison of a translated model in Simulink (top) and Dymola



3 AdvancedBlocks Library

3.1 Overview

A new Modelica library of control blocks has been designed to provide equivalent blocks in Modelica to those in the Simulink standard library. The design of the library has focused on providing a user-friendly library that can be used effectively as a modeling library. The main focus has been on providing simple ways to select the different options available for each block, for example the integrator method to be used, the port data type to be used, etc. There are a number of areas of interest in the design of the latest version of the AdvancedBlocks library and these are described in the following sections.

3.2 Connector Definition

The first step in developing the new library was to define the connector for the blocks. A new connector was required for a number of reasons; firstly, Simulink supports the use of matrix, vector and scalar signals whilst the original Modelica.Blocks.Interface.InPort and OutPort connectors[7] only support vector signals. Therefore we needed to change the connector definition to support matrix signals. During the translation process Simulink scalar signals are converted into Modelica matrices with only one value and vector signals are converted into Modelica matrices with only 1 row.

A second consideration was that Simulink cascades sample times along the connections. This means that a block can inherit a sample time from its driving block. To achieve this in Modelica we needed to add an additional signal to our connector to carry the sample times from block to block. It is necessary for this sample time signal to be a matrix because when muxed signals are used in Simulink it is possible for each signal to be carrying with it a different sample time. To replicate this behavior in the AdvancedBlocks library we actually pass a sample trigger along the connections that tells the connected block at which point in time it should calculate its output.

The final consideration for designing the connector was that Simulink signals could be different data types. We therefore needed to find a way to define a connector in which we could easily change the data type. We also needed to find a structure that would allow the connectors to be replaced even though the basic data type of the signal might be changing. The syntax for replaceable classes[8] would specifically prohibit the simple swapping of connectors if the basic types are different. Fortunately it is possible to replace classes that extend from the same base class.

To overcome the constraints of the language and to meet the design requirements the connectors are defined in packages and are created in a two-stage process. Each connector package specifies either an input or output connector for a specific data type. All the connector packages are extended from an appropriate base package that defines a base connector and a base data type conversion function. There is one base package for input connectors, and one for output connectors. Figure

3 shows the base package definition for the output connectors.

Figure 3: Base Connector Package Definition

```
partial package Base
  partial connector Outport "Output signal"
    parameter Integer n=1 "Dimension 1 of signal matrix";
    parameter Integer m=1 "Dimension 2 of signal matrix";
    output Integer samplereigger[n, m] "Sample trigger to be
    passed between blocks";
  end Outport;
  partial function Convert
  end Convert;
end Base;
```

The data type conversion function is used to apply the correct data type to the output signal. The blocks within the AdvancedBlocks library all use variables of type Real internally to handle the calculations. To correctly convert the internal signal type to that required in the connector a function is used that changes the signal data type and applies any limits to the value that may be required for a specific data type.

A connector for each required data type is then defined within its own package. This package must include a connector and function definition that extends from those in the base package. Figure 4 shows how the output connector for the uint8 (unsigned 8 bit integer) data type is defined in the AdvancedBlocks library.

Figure 4: Definition of the uint8 connector

```
package uint8 "uint8 (unsigned 8 bit integer) output signal"
  connector Outport "uint8 output signal"
    extends Base.Outport;
  output Types.uint8 signal[n, m] "Signal value";
end Outport;
function Convert
  extends Base.Convert;
  input Real u;
  output Types.uint8 y;
  algorithm
    y := integer(if u > 255 then 255 else if u < 0 then 0 else u);
  end Convert;
end uint8;
```

By declaring the different connectors within their own package it makes it possible to replace both the connector and conversion function using one redeclare statement. By ensuring that the connector and function names are the same in each package, the replaced package automatically changes the connector and conversion function to the chosen data type. In figure 5 the replaceable package Out1DataType is defined and then the

Outport connector is instantiated from this replaceable package. The constraint on the replaceable package ensures that we will only ever be able to replace the connector package with another valid package.

This structure to the design of the connectors and data type conversion function means that each connector in a block in the AdvancedBlocks library can use a different data type and this is achieved by simply redeclaring the relevant package that defines that connector to match the desired data type.

Figure 5: Example use of a connector

```
block OutputExample
  replaceable package Out1DataType =
    AdvancedBlocks.Interface.Connectors.Outputs.uint8 extends
    AdvancedBlocks.Interface.Connectors.Outputs.Base;
    Out1DataType.Outport out1(n=1, m=1);
  protected
    Real y1[1,1] "Result of internal calculation";
  equation
    out1.signal = Out1DataType.Convert(y1);
  end OutputExample;
```

Unfortunately this design cannot be implemented in the current version of Modelica because the data conversion function does not generate an event but integer values, such as those in the connector, are only allowed to change at events. This means that where we would like to use an Integer or Boolean data type in the connector we are unable to do so. The work around in the current version of the library is that all the connectors use a Real data type. The conversion functions also output a Real data type regardless of the actual data type desired but internally they apply the limits and round values as appropriate, i.e. round to the nearest integer if an integer data type is requested.

3.3 Continuous and Discrete time modes

A large proportion of the blocks in the Simulink standard library can run in different time-modes, i.e. either continuous or discrete time modes. In addition where blocks are able to run in discrete-time mode they can be defined to run at a set sample rate or they can inherit their sample time from their parent system or from their driving block.

To enable blocks within the AdvancedBlocks library to support running in these different time modes they have been defined so that they extend from a replaceable block that governs the calculation method used. Within each block that supports running in different time modes there is an encapsulated package that contains the different definitions required for operating in the different time modes. Figure 6 shows how the AdvancedBlocks.Math.Abs block is defined with the ability to switch between continuous and discrete time mode.

Figure 6: Block structure to support different time modes

```

block Abs "Abs block"
  extends TimeMode;
  replaceable block TimeMode = Options.Continuous extends
    Options.Base;

  encapsulated package Options
    import AdvancedBlocks.Interface;

  partial block Base "Base class and calculation function"
    extends Interface.BaseBlock "Icon and common properties";
    extends Interface.IOLayers.SI.Inports "Input definition";
    extends Interface.IOLayers.SO.Outports "Output definition";
    protected
      Real y[nout[1, 1], nout[1, 2]] "Result of internal calculation";
    equation
      y = abs(u1);
    end Base;

  block Continuous "Continuous time mode"
    extends Base;
    equation
      y1 = y;
      y1st = -ones(nout[1, 1], nout[1, 2]); // Sample trigger to next
    block
      end Continuous;

  block Triggered "Discrete time mode"
    extends Base;
    protected
      outer Boolean sampletrigger[1];
    equation
      y1st = if sampletrigger[1] then ones(nout[1, 1], nout[1, 2]) else
        zeros(nout[1, 1], nout[1, 2]); // Sample trigger to next block
      when sampletrigger[1] then
        y1 = y;
      end when;
    end Triggered;
  end Options;
end Abs;

```

When the user then drags the Abs block into their model for use they can simply switch time modes by redeclaring the block TimeMode to be any of the versions contained in the Options package. This is made even easier in tools such as Dymola where the version of TimeMode to be used can be selected from a pull-down menu. In the Abs block shown in Figure 6 it is possible to choose between a Continuous time mode and a Triggered time

mode. In the Triggered time mode the sample time is inherited from the parent system through the outer variable sampletrigger.

The structure of the Modelica code means that the actual equations defining the behaviour of the block are separate to the equations that force the block to act in a particular time-mode. This eases the maintenance of the library by not repeating blocks of code. This becomes a major consideration in the more complicated blocks.

3.4 Integrator Block

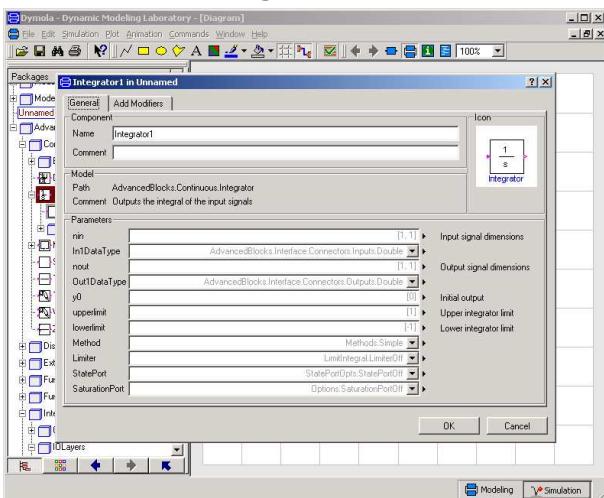
The continuous time integrator in Simulink is one of a number of blocks that can function in a variety of different ways depending on the choices made by the user each time the block is added to a model. The options for the integrator block include applying limits to the output, initialising with internal or external initial conditions, allowing for external reset signals, outputting state information and information on the limit condition[9]. To define all this in Modelica in a way that is easy to use has been achieved by extending the ideas described and used to change the time mode of the Abs block. This has led to the encapsulated package within the Integrator block becoming much more complex including several levels of hierarchy.

Each Integrator method is an extension of the same base class defined in the encapsulated package. The base block contains the definitions for the input and output connections and instantiates these from replaceable packages. This structure ensures that each integration option can redeclare the input and output layers to have the required number of connectors for this method. For example, if an external initial condition is required then two inputs are needed rather than one.

The result of this structure for the user is that they can easily choose what functionality they want within the integrator block in each instance. Figure 7 shows the dialog box produced by Dymola for the integrator block. Each option can be changed through the use of a pull-down menu showing the available options.

This same structure idea has also been used for many other blocks in the AdvancedBlocks library including the discrete integrator, math function block, trigonometric function block and many others.

Figure 7: Dymola dialog box for the Integrator block



3.5 Iterator Systems

The latest version of Simulink includes for-iterator and while-iterator subsystems. In these subsystems the blocks are executed a number of times at each time step. The actual number of times that the sub-system iterates at each time step can vary from time step to time step. The iterator subsystems have been introduced into Simulink to encourage its use as a control system software design and development tool. The key improvement for users in introducing these blocks is to facilitate the auto-coding of control system software. These subsystems along with the range of if-then-else and switch-case blocks make it much easier for controls engineers to design and develop the control system software.

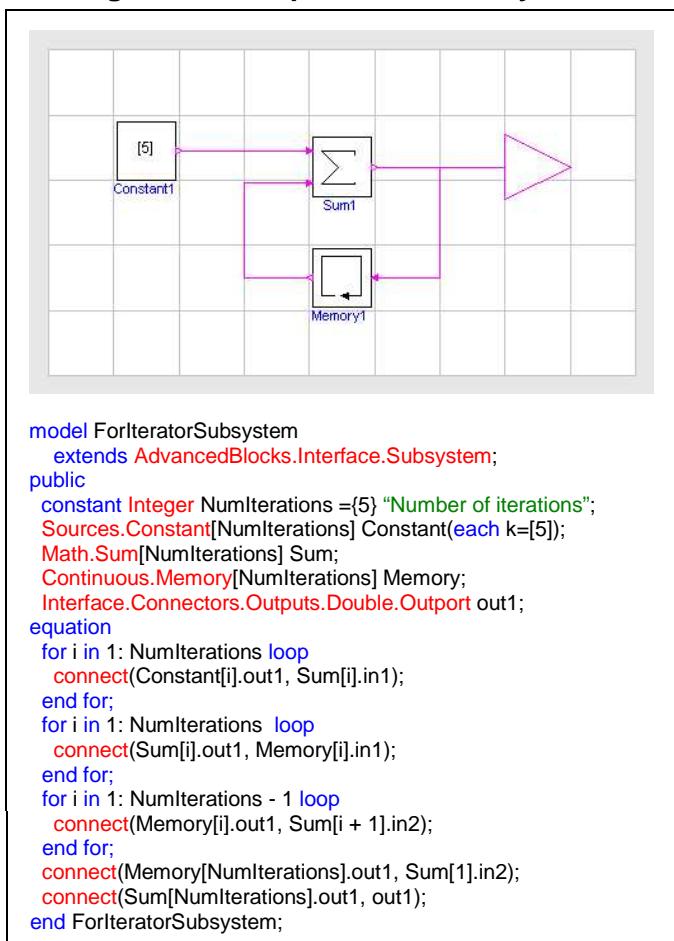
Iterator subsystems can be translated into Modelica where a fixed number of iterations are specified such as in some instances of for-iterator subsystems. In these cases the blocks within the subsystem are instantiated into an array of blocks where the size of the array equals the number of iterations to be performed. For example, figure 8 shows how a simple subsystem would be defined if it was required to iterate 5 times at each time step. The output from this subsystem at the first time step would be 25, after the second time step it would be 50, etc.

In this example the constant, sum and memory blocks are declared as component arrays where the size of the array is equal to the number of iterations. Each block within the component array forms a different iteration of the for loop. The subsystem output connector is only connected to

the Sum block in the final iteration of the for loop so that we get the full value of the loop passed out of this subsystem. The memory block is connected so that it effectively spans the iterations. The input to the memory block comes from the output of the Sum block in the current loop. The output from the memory block is connected to the input of the Sum block in the next iteration. In the final iteration of the loop the output from the memory block is connected to the input of the Sum block on the first loop.

To use this idea for while-iterator subsystems and for-iterator subsystems where the number of iterations can vary at each time step would require the component arrays to vary in size at each time step. It is not currently possible to implement this type of system in Modelica where the number of iterations varies at each time step.

Figure 8: Example Iterator subsystem



4 Example models

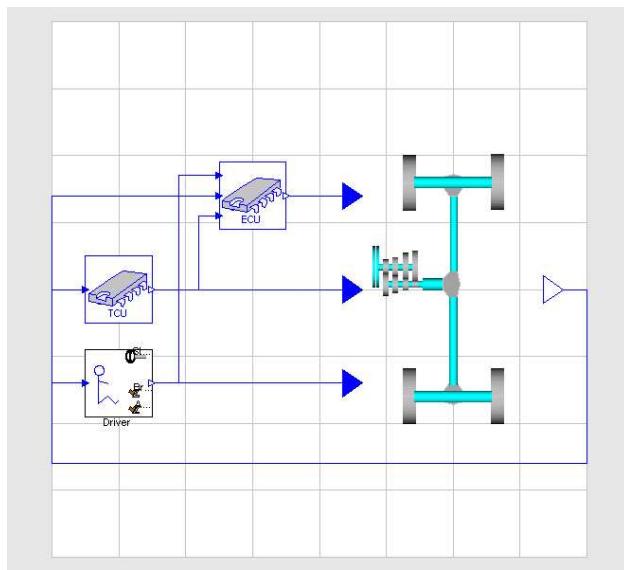
As well as a large number of relatively simple test cases a number of complex real-world examples

have been translated. Two examples of translating real-world models are presented in the following sections and the simulation performance and results have been compared.

4.1 Cruise Control Simulation

In this example we have combined a detailed physical powertrain model with the actual cruise control function from an engine control system, see figure 9. The cruise control function is developed by the system supplier in Simulink and then used by both the customer and supplier to develop and calibrate the system into the end product. Ultimately the actual code downloaded into the engine control unit is generated automatically from the Simulink model and so the latest version of the cruise control strategy will always be available in Simulink.

Figure 9: Powertrain model and converted controller system model



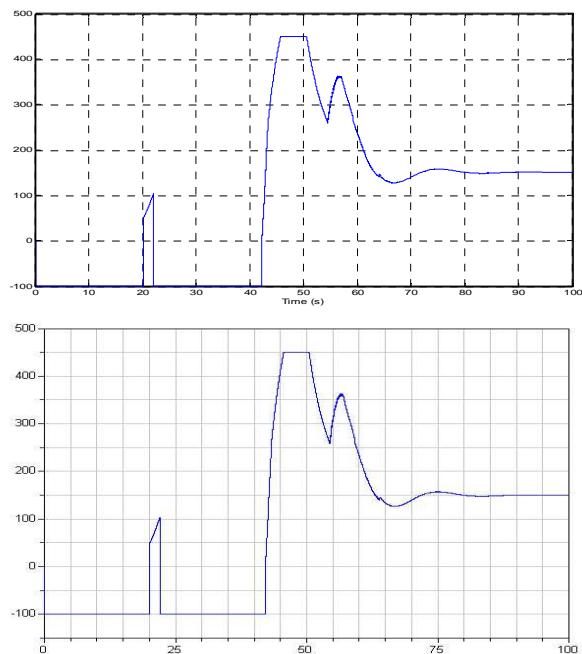
This cruise control function is designed to work as part of a torque structure engine management system. This means that when the cruise control function is active it demands an engine torque and feeds this into the torque structure function which converts this torque demand into a throttle position, spark timing and amount of fuel to inject. These quantities are determined so that the engine will produce as close to the demanded torque value as is physically possible within the constraints of the calibration.

For this example we have chosen to convert just the cruise control function from Simulink into Modelica using Simelica. This is then coupled to a

detailed powertrain model that does not include an engine model. The torque demand from the cruise control model is applied directly to the engine flywheel. In this way we can eliminate the need to calibrate the torque structure function on the assumption that this will be calibrated to translate the demanded torque into the actual engine torque produced at a later date.

The aim of this model was to enable the calibration of the cruise control function early in the development process. The task of calibrating the cruise control function traditionally requires a significant amount of test work to achieve good results. This is due to the difficulties involved in repeating each test exactly and the wide range of conditions that need to be tested. It is therefore an ideal candidate for applying simulation techniques which can reproduce the same test conditions repeatedly and help produce an initial calibration.

Figure 10: Comparison of Simulink (top) and Modelica Controller models

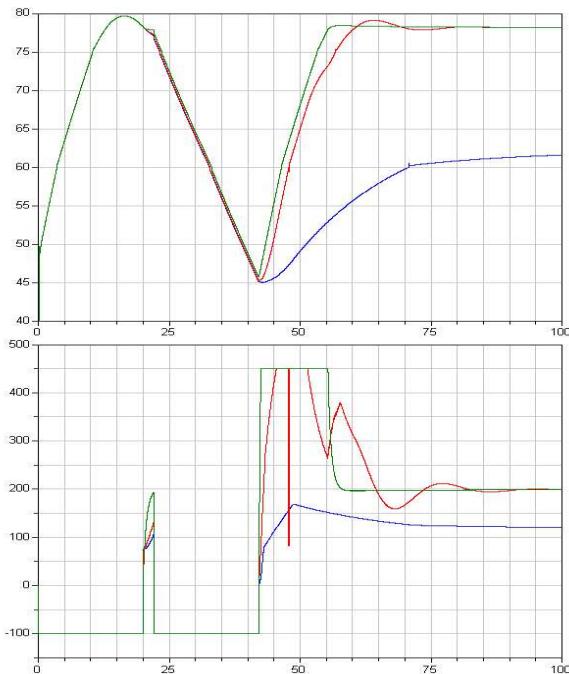


To translate the controller model and validate the generated Modelica model a Simulink model was generated that played measured data into the control system and recorded the outputs. This model, its parameter data and the measured data were then translated into Modelica using Simelica. Figure 10 compares the outputs from the controller function in both Simulink and Modelica. By ensuring that the Modelica controller model produces the same results as the original Simulink

model we can be sure that the translated model is accurate.

Once we are satisfied that the translated controller model was behaving in the same way as the original Simulink model the new Modelica model could then be used to attempt to calibrate the control system. There are many parameters within the control system that need to be calibrated and by repeating the same test exactly the effect of altering these parameters can be assessed and a calibration can be defined. Figure 11 shows the effect of altering one of the gains in the control system on a given test.

Figure 11: Effect of controller gain on a Cruise Resume Event



In this test the driver puts the vehicle into cruise mode at 20 seconds but then presses the brake at 22 seconds forcing the vehicle out of cruise mode and into a gentle deceleration. At 42 seconds the driver presses the Resume button and the vehicle enters back into cruise mode and attempts to regain the speed it was travelling at when the driver first put the vehicle into cruise mode. The three results traces demonstrate the effect of altering one of the gains in the cruise control function on the vehicle response.

4.2 Central Heating System

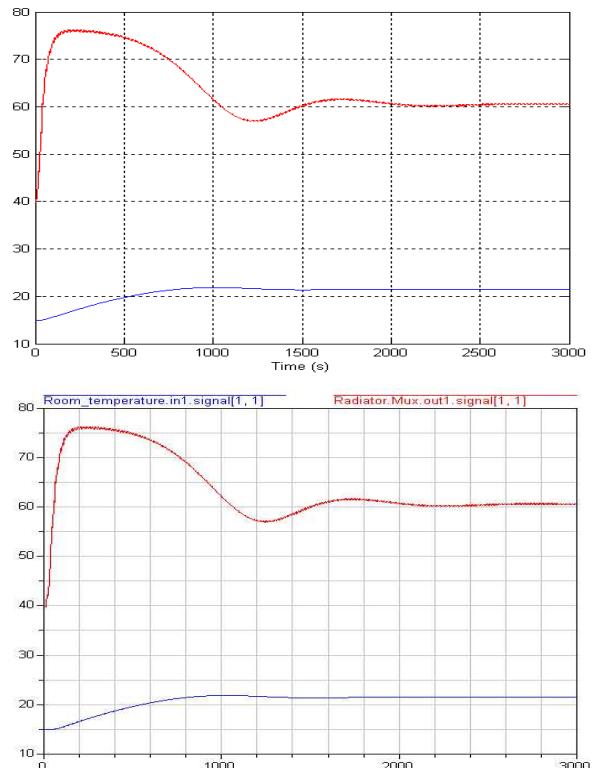
The model shown in figure 8 was developed in Simulink to predict the performance of a small

central heating system. The main motivator for attempting to translate this model into Modelica was to see if the simulation times would be improved. As Dymola generates efficient compiled models from the Modelica models and Simulink interprets the model at runtime it would provide an interesting comparison of simulation performance.

Using Simelica the model has been translated into Modelica and then compiled and simulated using Dymola 5.1a. Figure 2 shows this model in both Simulink and Modelica. It is clear from the diagrams that the same model structure and layout has been preserved during the translation process and any visible differences in the two diagrams are purely down to the way the two tools present the models graphically.

Figure 12 shows the results traces produced by both Dymola and Simulink versions of this model. It can be seen that although the model has been translated into Modelica the results obtained are the same. The time required to simulate a 24 hour period for the Modelica version of the model is 31 seconds but Simulink required just 9 seconds to carry out the same simulation on the same PC.

Figure 12: Comparison of Simulink (top) and Dymola simulation results



When comparing these simulation times it is also essential to consider that in Dymola 493 signals were stored but in the Simulink version only 12 signals were stored. In more complex systems the immediate availability of all this data would be very useful to help diagnose problems. To carry out the same investigation in the Simulink version of the model would require the user to manually add scopes to areas of the model that they suspect of causing the problem and then re-running the model. This process of adding scopes and re-running the model may have to be repeated several times before the problem can be correctly diagnosed.

5 Limitations

5.1 Limitations of Simelica

There are also some blocks available in Simulink that cannot be automatically translated into Modelica. These include blocks such as the MatlabFcn and S-function. The MatlabFcn block cannot be translated because it allows the user to use any Matlab script or command in the model, many of which do not have an equivalent in Modelica. The S-function block cannot be automatically translated because the c-code might need to be changed significantly to work as an external function in Modelica. It is possible to do this manually though. There are a number of other blocks that are currently unsupported but through the continual development of the tool the majority of these will be incorporated.

Another feature that cannot be automatically handled is the initialisation commands that can be fed into models and masked blocks. These cannot be supported because they allow any Matlab command to be used and executed during the model initialisation and many of these commands do not have an equivalent in Modelica. Rather than attempt to handle this and get it wrong, Simelica opts to simply copy all the commands from the initialisation layer into a comment in the block and then flag this to the user as a problem requiring attention.

The final limitation in the translation process currently is that matrix signals and signal data types are not supported. Although many of the features exist in the AdvancedBlocks library it is not yet possible for Simelica to correctly translate

models that include these features. Where data types other than the Matlab data type double are used in the model the different data type will be ignored by the translator and the converted model will use the double data type. Models that contain matrix signals will have the signal dimensions incorrectly set. From the point-of-view of the AdvancedBlocks library and Simelica a matrix signal is any signal that has more than one row. Many of the blocks within the AdvancedBlocks library will not currently function correctly when matrix signals are used. These issues will be addressed in future versions of Simelica and the AdvancedBlocks library.

5.2 Limitations of the Modelica language

There are some key differences between the Modelica language and what is possible in Simulink. Modelica does not support the same flexibility in block naming as Simulink does. For example Simulink can use any special character in the block names; names can also start with numbers; names can contain white space characters. Some transformations therefore have to be made by Simelica to ensure that a block name conforms to the Modelica specification. The difficulty here can be that blocks that were named differently in Simulink purely because of the inclusion of a special character, or series of characters that are prohibited in Modelica could end up with the same name in the Modelica model leading to errors.

Although many of the modelling methodologies used in Simulink can be translated into a form for use in Modelica it is not always possible to provide an equivalent methodology in Modelica. For example, signal buses are translated into simple muxed signal systems where the bus selector is defined to extract particular signals by index rather than by name. In Simulink the names of the signals are passed along the connection include the hierarchy within the bus system. Signals can then be extracted by selecting a particular signal name. This feature is widely used in Simulink[10] as it provides a powerful way to pass large groups of signals around a model.

A large number of the blocks within the AdvancedBlocks library contain encapsulated packages that would ideally be hidden from the user. This could be achieved by declaring the

package as protected but then all replaceable classes and parameters would not be visible in the GUI dialogs produced by Dymola. To get around this all the parameters and replaceable classes would then have to be declared in the block containing the encapsulated package but this would mean that the user is presented with options and parameters that might not be valid because of other selections they have already made. Another method of hiding these packages from the user whilst still making the parameters and replaceable classes visible in the tool dialogs is required. Ideally it would also be possible for the available options and required parameters to change as selections are made by the user.

6 Future

It is important to note that this paper refers to the current version of Simelica and the AdvancedBlocks library and that they will continue to evolve and support more features. They will both be continually developed to support the latest versions of Simulink and Modelica.

7 Acknowledgments

Many people have provided support during the design and development of Simelica and the AdvancedBlocks library and I would like to extend my thanks to them. Specifically Hilding Elmquist, Hans Olsson, Sven Erik Mattsson and Dag Bruck from Dynasim, Martin Otter from DLR and Mike Tiller from Ford Motor Company.

Matlab, Simulink and Stateflow are registered trademarks of The Mathworks Inc. Modelica is a registered trademark of The Modelica Association. Dymola is a trademark of Dynasim AB. Simelica and AdvancedBlocks are trademarks of Claytex Services Limited.

8 References

1. S.R. Anderson, C.R. Ciesla, D.M. Carey, R. Shankar, "A powertrain simulation for engine control system development", *1996 SAE International Truck and Bus Meeting and Exposition*, SAE 962171
2. P.M. Fussey, C.L. Goodfellow, K.K. Oversby, B.C. Porter, J.C. Wheals, "Integrated Powertrain (IPT) Model – Stage 2: Systems

Integration, Supervisory Control and Simulation of Emissions Control Technology", *SAE 2001 World Congress*, SAE 2001-01-0928

3. J.A. MacBain, J.J. Conover, A.D. Brooker, "Full-vehicle simulation for series hybrid vehicles", *Future Transportation Technology Conference*, SAE 2003-01-2301
4. M. Tiller, W.E. Tobler, and M. Kuang, "Evaluating Engine Contributions to HEV Driveline Vibrations", *Proceedings of the 2nd International Modelica Conference*
5. S. Soejima, "Examples of usage and spread of Dymola within Toyota", *Modelica Workshop 2000 Proceedings*
6. "Dymola 5.0 User's Manual", *Dynasim AB*.
7. "Modelica Standard Library 1.5", *The Modelica Association*, 2002
8. "Modelica Language Specification, Version 2.0", *The Modelica Association*, 2002.
9. "SIMULINK Release 13" (documentation), *The Mathworks Inc.*
10. C. Belton, P. Bennet, P. Burchill, D. Copp, N. Darnton, K. Butts, J. Che, B. Hieb, M. Jennings and T. Mortimer, "A Vehicle Model Architecture for Vehicle System Control Design", *SAE Congress 2003*, SAE 2003-01-0092

DrModelica

An Interactive Tutoring Environment for Modelica

Eva-Lena Lengquist Sandelin, Susanna Monemar, Peter Fritzson, Peter Bunus

PELAB, Programming Environment Laboratory
 Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
 Email: {evale, x02susmo, petfr, petbu}@ida.liu.se

Abstract

This paper states the need for interactive teaching materials for programming languages within the area of modeling and simulation. We propose an interactive teaching material for the modeling language Modelica inspired by existing tutoring systems for Java and Scheme.

The purpose of this new teaching material, called DrModelica, is to facilitate the learning of Modelica through an environment that integrates programming, program documentation and visualization. The teaching material is intended to be used for modeling and simulation related courses at the undergraduate and graduate level.

1. Background

The concepts of model, system, and experiment are central in the area of modeling and simulation. “A model of a system is anything an “experiment” can be applied to in order to answer questions about that system.” [1] “A simulation is an experiment performed on a model.” [1]

Tools that are used for modeling and simulation are becoming powerful aids in the product development process. Using advanced tools and languages to build a model of a product and then simulate its behavior, before producing a physical prototype, reduces the number of errors that can occur during fabrication. This reduction consequently leads to a decrease in the time needed to develop the final product. Furthermore, the earlier the errors are detected, the cheaper the corrections are.

Not too long ago in the history of modeling and simulation technology, mathematical models were implemented by hand. The models were usually

designed on paper using mathematical notation and the programs written manually in a high-level programming language, like C or Fortran, and stored in text files. Much manual work was needed, making not only maintenance of models expensive, but also the modification of models hard in order to adapt to new requirements [2].

2. Interactive Environments

Modelica helps solving problems concerning modeling and simulation. In order for Modelica to be used for this purpose, a modeling and simulation environment is needed. In this section the MathModelica environment is presented. MathModelica is partly built on Mathematica technology, which is also described below.

2.1. Mathematica

Mathematica [3] is a computer algebra system and programming environment for performing mathematical computations. The system can be used in many different ways; the most basic functionality involves using it as a “calculator”. The user types a calculation and Mathematica performs it immediately. However, there is a big difference between what a traditional calculator can do and what Mathematica can perform. The system seamlessly integrates a numeric and symbolic computational engine, graphics system, programming language, documentation system, and advanced connectivity to other applications.

Mathematica can also be used as a modeling and simulation environment. When a model is simulated in the environment, the results can be visualized in various ways, using the Plot function.

Mathematica is divided into two distinct parts: the computer algebra engine and interpreter (“kernel”) that receives and evaluates all expressions sent to it and the user interface (“front-end”). The front-end provides the programming interface to the user and is concerned with such issues as how input is entered and how computation results are displayed to the user.

Mathematica’s front-end documents are called notebooks. A notebook can contain specific computations, text (including hyperlinks to other notebooks), graphics, sounds and animations. Using a hierarchical structure divided into sections, subsections etc. A notebook can be made to look like a traditional typeset document, with the advantage that the calculations can remain active and can be re-evaluated at any time.

2.2. MathModelica

MathModelica, from MathCore Engineering AB [4], is a powerful engineering environment for physical modeling, simulation, analysis and design [5, 6]. In MathModelica, models are described using Modelica. Dymola [7], developed by Dynasim [8], is another powerful Modelica environment.

The MathModelica environment integrates modeling and simulation with graphic design, advanced scripting facilities, integration of code and documentation, and symbolic formula manipulation provided via Mathematica. Import and export of Modelica code between internal structured and external textual representation is supported by MathModelica. The environment extensively supports the principle of literate programming and integrates most activities needed in simulation design: modeling, documentation, symbolic processing, transformation and formula manipulation, input and output data visualization.

The user interface of MathModelica consists of the Model Editor, the Simulation Center and Mathematica notebooks. The Model Editor is a graphical tool for designing models using predefined library components. The Simulation Center is a graphical user interface for running simulations and plotting curves of the models. Mathematica notebooks provide a text based programming environment.

3. DrModelica

Understanding programs is hard, especially code written by someone else. For educational purposes it is essential to be able to show the source code and to give an explanation of it at the same time [9]. Moreover, it is important to show the result of the source code’s execution. In modeling and simulation it is important to have the source code, the documentation about the source code, the execution results of the simulation model, and the documentation of the simulation results in the same document. The reason is that the problem solving process in computational simulation is an iterative process that often requires a modification of the original mathematical model and its software implementation after the interpretation and validation of the computed results corresponding to an initial model.

Most of the environments associated with equation-based modeling languages focus more on providing efficient numerical algorithms rather than giving attention to the aspects that should facilitate the learning and teaching of the language. There is a need for an environment facilitating the learning and understanding of Modelica. Also, users are reluctant to using a programming language that does not provide an adequate programming environment [10]. All the above-mentioned facts constitute our reason for developing DrModelica [11], a teaching material for Modelica. DrModelica is based on MathModelica [4] and the ideas of Literate programming [12].

Literate programming is a programming methodology that was introduced by Donald E. Knuth. It represents the idea of organizing a source program in an “essay” manner by combining the source code with the corresponding documentation in the same document. By doing so it is easier to read and understand the program.

MathModelica has an interface allowing the user to write source code as well as documentation in the same document. The user does not have to switch to a command prompt to compile the source code, since this can also be performed in the environment. The same document also contains plots of the simulation results. Additionally, in DrModelica the whole Modelica language is available to the user, unlike many other tutoring systems, where it is common to provide a subset of the language. Furthermore, we have developed a web version of DrModelica, which

has a similar interface and includes most of the functionality that can be found in MathModelica. The interface for the web version of DrModelica is currently available at <http://www.DrModelica.org> although in order for the connection between the interface and the Modelica compiler to work, the

OpenModelica compiler has to be downloaded first. The difference between the web version and the MathModelica version of DrModelica is that the functionality of the web version is limited, for example there is no possibility to show plots of a simulated model.

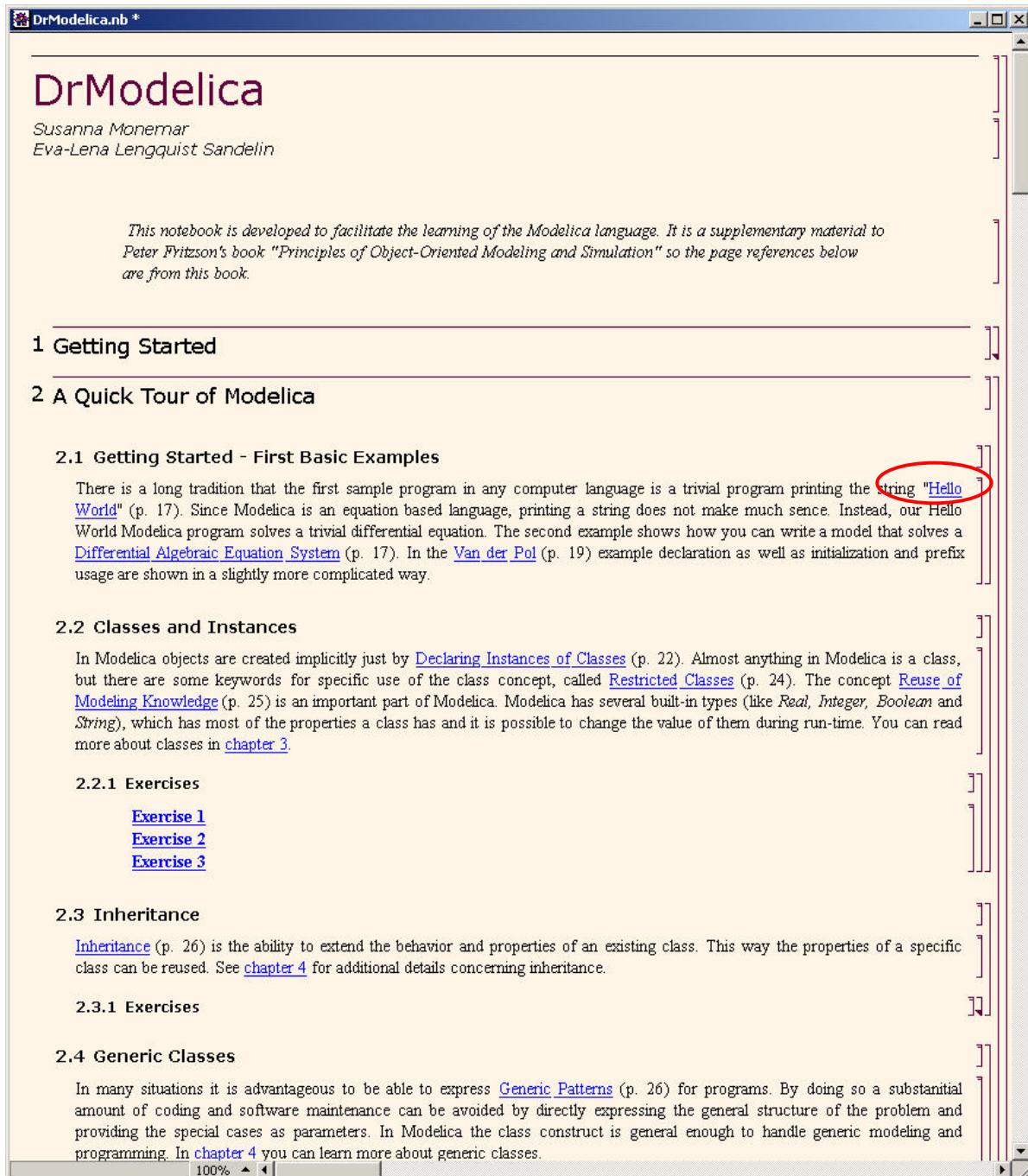


Figure 1. The front-page notebook of DrModelica.

Furthermore, the web version is intended to be used as a testing environment for evaluating Modelica code. It is not a teaching material, since there is no text or examples that the user can learn from.

DrModelica has a hierarchical structure represented as Mathematica notebooks. The front-page notebook is similar to a table of contents that holds all other notebooks together by providing links to them. This

particular notebook is the first page the user will see (Figure 1).

In each chapter of DrModelica the user is presented a short summary of the corresponding chapter of the book “Principles of Object-Oriented Modeling and Simulation with Modelica” by Peter Fritzson [1]. The summary introduces some *keywords*, being hyperlinks that will lead the user to another notebook describing the keyword in detail.

The screenshot shows a window titled "HelloWorld.nb *". The main content area displays the following text:

First Basic Class

1 HelloWorld

The program contains a declaration of a class called `HelloWorld` with two fields and one equation. The first field is the variable `x` which is initialized to a start value 2 at the time when the simulation starts. The second field is the variable `a`, which is a constant that is initialized to 2 at the beginning of the simulation. Such a constant is prefixed by the keyword parameter in order to indicate that it is constant during simulation but is a model parameter that can be changed between simulations.

The Modelica program solves a trivial differential equation: $x' = -a * x$. The variable `x` is a state variable that can change value over time. The `x'` is the time derivative of `x`.

```

class HelloWorld
  Real x(start = 1);
  parameter Real a = 1;
equation
  der(x) = -a * x;
end HelloWorld

```

2 Simulation of HelloWorld

```

SIMULATE [HelloWorld, {t, 0, 4}]

<SimulationData: HelloWorld : 2002-10-31 11:
 29:12 : {0., 4.} : 502 data points : 1 events : 3 variables>
{a, x, x'}

```

```

PlotSimulation[x[t], {t, 0, 4}, AxesOrigin → {0, 0}]

```

A plot window shows a blue curve representing the variable `x[t]` over time `t`. The x-axis ranges from 0 to 4, and the y-axis ranges from 0 to 1. The curve starts at (0, 1) and decays exponentially towards zero as `t` increases.

Figure 2. HelloWorld class.

Now, let us consider that the link “*HelloWorld*” in section 2.1 in Figure 1 is clicked by the user. The new notebook, to which the user is being linked (see Figure 2), is not only a textual description but also contains one or more examples explaining the specific keyword. In the class, *HelloWorld*, a differential equation is described.

No information in a notebook is fixed, which implies that the user can add, change or remove anything in a notebook. Alternatively, the user can create an entirely new notebook in order to write his/her own programs or copy examples from other notebooks. This new notebook can be linked from existing notebooks.

The screenshot shows the DrModelica interface with the title bar "DrModelica.nb". The main content area displays Chapter 9, titled "Algorithms and Functions".

9 Algorithms and Functions

9.1 Algorithms

In Modelica, algorithmic statements can only occur within [Algorithm Sections](#) (p. 221), starting with the keyword `algorithm`. [Simple Assignment Statements](#) (p. 222) is the most common kind of statements in algorithm sections. There is a special form of assignment statement that is only used when the right hand side contains a call to a [Function with Multiple Results](#) (p. 223).

The [for-Statement](#) (also called `for-loop`) is a convenient way of expressing iteration (p. 223). When using the `for`-loop for iteration we must be able to express the range of values over which the iteration variable should iterate in a *closed* form as an iteration expression. For cases where this is not feasible there is also a [While-loop](#) iteration construct in Modelica (p. 224). For conditional expressions the [if-Statement](#) (p. 225) is used. [When-Statements](#) (p. 226) are used to express *actions at event instants* and are closely related to when-equations. The [Reset](#) (p. 229) statement can be used in when-statements to define new values for continuous-time *state variables* of a model at an event.

The [Assert](#) (p. 231) statement provides a convenient means for specifying checks on model validity within a model. The most common usage of [Terminate](#) (p. 231) is to give more appropriate stopping criteria for terminating a simulation than a fixed point in time.

9.1.1 Exercises

- [Exercise 1](#)
- [Exercise 2](#)
- [Exercise 3](#)
- [Exercise 4](#)
- [Exercise 5](#)

9.2 Functions

The body of a Modelica function is a kind of algorithm section that contains procedural algorithmic code to be executed when the function is [Called](#) (p. 233). Since a function is a restricted and enhanced kind of class, it is possible to inherit an existing function declaration in the declaration of a new function. In this way we can declare the common structure of a set of functions as a [Partial Base Function](#) (p. 234) which can be inherited into the functions we want to define. A function with more than one output formal parameter has [Multiple Results](#) (p. 235). It is possible to call functions defined outside of the Modelica language, so called [External Functions](#) (p. 237).

9.2.1 Exercises

- [Exercise 1](#)
- [Exercise 2](#)
- [Exercise 3](#)

10 Packages

What is a [Package](#)?

10.1 Packages as Abstract Datatypes

The notion of a package partly originates from the notion of [Abstract Data Type](#) (p. 247).

At the bottom of the window, there is a toolbar with icons for zooming and navigating, and a status bar showing "100% ▲ ◀ ▶" and a scroll bar.

Figure 3. Chapter 9 in the main page of DrModelica.

When a class has been successfully evaluated the user can simulate and plot the result. These two actions are performed by the Mathematica commands `Simulate` and `PlotSimulation`. `Simulate` compiles the code and `PlotSimulation` shows a diagram of the result. Figure 2 shows how `HelloWorld` uses the Mathematica commands `Simulate` and `PlotSimulation`.

After reading a chapter in DrModelica the user can immediately practice the newly acquired information by doing the exercises that concern the specific chapter. We have written the exercises in

order to elucidate language constructs step by step based on the pedagogical assumption that a student learns better “*using the strategy of learning by doing*”. The exercises consist of either theoretical questions or practical programming assignments. All exercises provide answers in order to give the user immediate feedback.

Figure 3 shows Chapter 9 in the teaching material. Here, the user can read about language constructs, like algorithm sections, when-statements and reinit and then practice by solving the exercises corresponding to the recently read section.

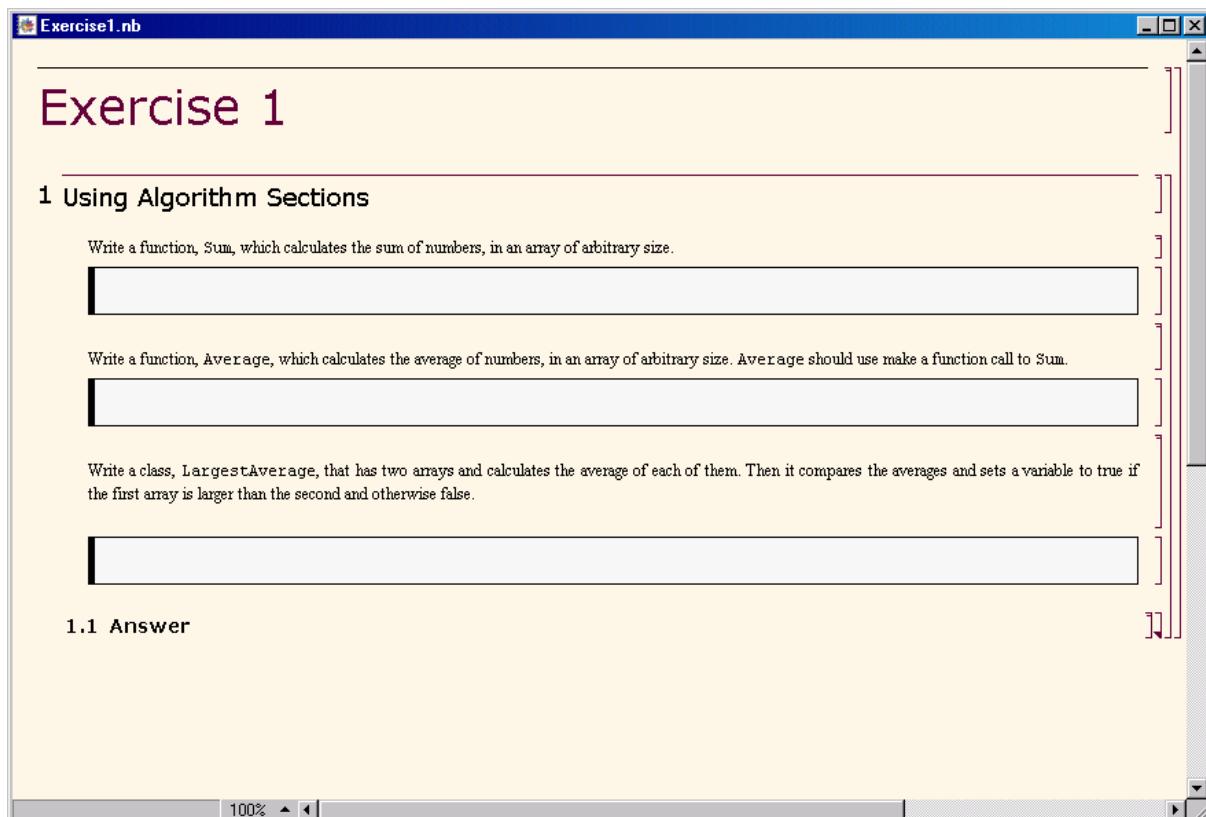


Figure 4. Exercise 1 in chapter 9.

Exercise 1 in section 9.1.1 is shown in Figure 4. In this exercise the user has the opportunity to practice different language constructs and then compare the solution to the answer for the exercise. Notice that the answer is not visible until the *Answer* section is expanded. The answer is shown in Figure 5.

Figure 6 shows that circuits created in the Model Editor of MathModelica can be inserted in DrModelica as pictures and it can be used to generate Modelica code from.

Exercise1.nb

1.1 Answer

1.1.1 Sum

```
function Sum
    input Real[:] x;
    output Real sum;
algorithm
    for i in 1:size(x,1) loop
        sum := sum + x[i];
    end for;
end Sum;
```

1.1.2 Average

```
function Average
    input Real[:] x;
    output Real average;
protected
    Real sum;
algorithm
    average := Sum(x) / size(x,1);
end Average;
```

1.1.3 LargestAverage

```
class LargestAverage
    parameter Integer[:] A1 = {1, 2, 3, 4, 5};
    parameter Integer[:] A2 = {7, 8, 9};
    Real averageA1, averageA2;
    Boolean AllLargest(start = false);
algorithm
    averageA1 := Average(A1);
    averageA2 := Average(A2);
    if averageA1 > averageA2 then
        AllLargest := true;
    else
        AllLargest := false;
    end if;
end LargestAverage;
```

1.1.4 Simulation of LargestAverage

```
Simulate[LargestAverage, {t, 0, 1}]
<SimulationData: LargestAverage : 2002-10-10 11:
 28:45 : (0., 1.) : 502 data points : 1 events : 13 variables>
{A1[1], A1[2], A1[3], A1[4], A1[5], AllLargest, A2[1],
 A2[2], A2[3], averageA1, averageA2, _derdummy, _dummy}
```

When we look at the values in the variables we see that A2 has the largest average (8) and therefore the variable AllLargest is false (= 0).

```
{AllLargest[1], averageA1[1], averageA2[1]}

{0., 3., 8.}
```

Figure 5. The answer section to Exercise 1 in chapter 9.

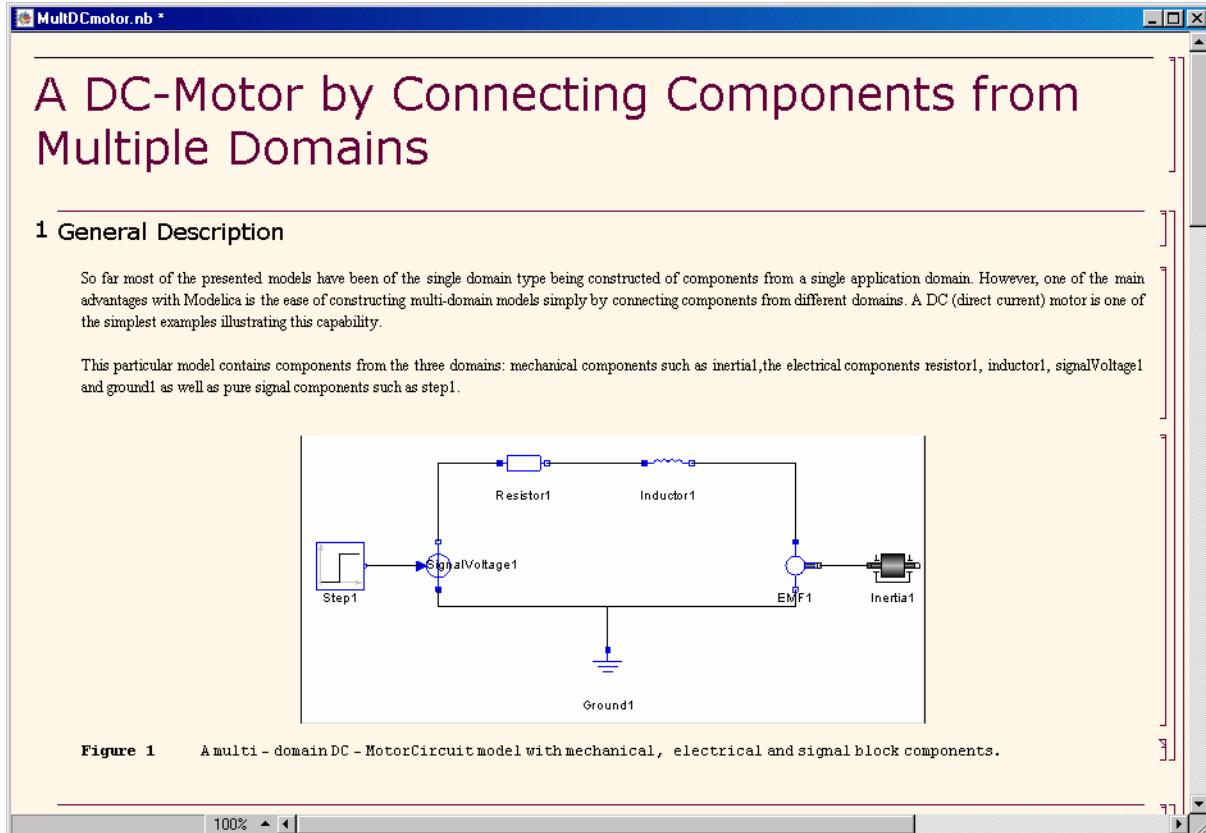


Figure 6. Pictures from the Model Editor in MathModelica can be inserted in the environment.

4. Related Work

During the last two decades interactive teaching materials have been developed with the purpose of facilitating the learning process. For example, DrJava and DrScheme are both interactive teaching materials for Java and Scheme respectively. These materials teach the language to the user both by explaining the concepts of the language and by letting the user write programs in a beginner-adjusted environment [13, 14].

DrScheme [14] is a programming environment for Scheme, providing a graphical user interface, in which it is possible to edit and interactively evaluate Scheme programs. The environment is especially useful for students learning Scheme, since it guides the student through Scheme in a way similar to an introductory course [14].

DrJava is an open-source, pedagogic programming environment for teaching Java. The environment is influenced by DrScheme, which has served as a

model for DrJava [13]. To facilitate the learning of Java, DrJava first introduces the concepts of coding, as well as testing and debugging the source code, and then focuses on the language semantics.

5. Evaluation of DrModelica

Evaluation methods are important tools for user interface design. Such methods can be divided into usability testing methods and usability inspection methods. The difference between them is that users are involved in usability testing methods but are not involved in usability inspection methods. For evaluation of DrModelica, both methods have been used, with specially developed questionnaires [15] and performing a heuristic evaluation [16].

Using a questionnaire is a usability testing method and reflects the users' subjective opinions. It is a cheap method for testing a system and can be distributed to many users.

Heuristic evaluation is a usability inspection method, which is performed by an evaluator, using a checklist

of guidelines to determine the usability of the user interface. This method is easy to learn and inexpensive to perform. Most of the general usability problems can be identified using a heuristic evaluation. The method requires some experience with heuristic evaluation principles for an optimal result. However, even a non-expert can find many usability problems using a heuristic evaluation.

5.1. Evaluation using Questionnaire

Twelve students attending a graduate Modelica course at Linköping University tested DrModelica. After a few weeks they were asked to answer a questionnaire. All testers were engineering students, either in the area of physics or computer science. The questions in the questionnaire concerned their expectations of the teaching material and if their expectations were fulfilled, what they felt about the approach using literate programming and the structure and layout of the material. The results of the questionnaire were positive. For example, Literate programming was appreciated when programming Modelica. The test group generally found DrModelica to be a better way of learning a programming language, compared to the way they were used to.

The structure of DrModelica and the way of navigating between the notebooks was, according to the test group, fairly easy. The exercises at the end of each chapter were also appreciated by the students. In this way the student was able to "directly use the collected knowledge", referring to one of the testers.

5.2. Heuristic Evaluation

Three usability experts from HCS (Human Centered Systems), at the Department of Computer and Information Science (IDA) have performed a heuristic evaluation on DrModelica. When performing the evaluation, the evaluators used the guidelines from "Ten Usability Heuristics" [17]. They are listed below:

1. Visibility of system status: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
2. Match between system and the real world: The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-

world conventions, making information appear in a natural and logical order.

3. User control and freedom: Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
4. Consistency and standards: Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
5. Error prevention: Even better than good error messages is a careful design which prevents a problem from occurring in the first place.
6. Recognition rather than recall: Make objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
7. Flexibility and efficiency of use: Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
8. Aesthetic and minimalist design: Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
9. Help users recognize, diagnose, and recover from errors: Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
10. Help and documentation: Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

The evaluation gave many valuable results. The evaluators found that learning how to use DrModelica was easy in general. However, realizing how some of the functionality works was, according to the

evaluators, not so intuitive. For example it can be hard to discover the ability to collapse and expand sections. Though, once it was known how to use the functionality they found easy. Furthermore, according to the evaluators it might be confusing that a link in some cases opens a new window and in other cases refers to another chapter in the same window. This is a problem concerning heuristic number 4. Another problem, when being linked to another page, is that there is no feedback telling the user that a new page has appeared in front of the previous one. This is a problem mostly concerning heuristics number 1, 2 and 3. When a new window is opened in front of the other the user is not properly informed about what is going on, since there is no feedback that the window was just being opened (see heuristic number 1). This involves another problem, taking the user back to the former window. This is currently resolved by closing the window, but it would be better solved by having a “back”-button, following real-world conventions (see heuristics 2 and 3). Heuristics number 5, 8 and 9 concern dialogues and error messages, none of which exist in neither DrModelica nor MathModelica, but that is why the environment does not have a need for it. Heuristic number 10 concerns help and documentation. There is a help section on how to start using DrModelica, which was appreciated by the users.

The evaluators also found that DrModelica was less intimidating than other programming environments, since the user is presented with an environment similar to a document showing only a small amount of functionality. This leads the user to believe that DrModelica is a reading material. However, after using the material for a while the user discovers that DrModelica could be used for programming as well. A common approach adopted by many programming environments is to lead the user in the opposite direction, by presenting all functionality from the beginning. This approach can have a discouraging effect on the user.

6. Future Improvements

Considering the results of the evaluation and comparing our work with related work we have discovered some possible improvements that can be implemented in the future. Here follows a list of these improvements:

A suggestion from the students, attending the Modelica graduate course, is to extend DrModelica to contain more exercises on simple as well as more complex constructs in order for the student to get more practice.

Since it can be difficult to learn how to use the functionality in DrModelica, an idea is to make an introductory exercise for practicing the basics step by step instead of just reading a long introductory text.

Links between files containing different variants of the same term should be added.

Currently the exercises in the material mainly concern language specific constructs, it would be desirable to add exercises reflecting the purpose of Modelica. The material needs to be extended with more exercises in general.

Features, like parenthesis matching and keyword highlighting, used in DrScheme and DrJava, would be helpful when programming.

7. Summary and Conclusions

In this paper we have presented the interactive teaching material for Modelica, based on MathModelica, called DrModelica. DrModelica has the goal of teaching Modelica in an environment that has the purpose of facilitating the learning process of the language. Because of the complexity of learning Modelica there is a need for such a material.

DrModelica is based on Literate programming, which enables the user to write, document and execute the source code in the same file or entity. This file or entity becomes a Literate program. In DrModelica the documentation about the source code is not embedded as comments in the code, but instead separated from the code in specific sections only with the purpose of containing text.

The Literate programming approach is extended in DrModelica, in such a way that the result of the executed Modelica program is included in the same file or entity. The results of the source code can be shown in the form of diagrams. This is a necessary part of DrModelica, since Modelica is a programming language used for creating models of complex physical systems and there is a need to check if these models’ behaviour follows the specification or comply with the user intent.

The evaluations of DrModelica resulted in many valuable opinions. The members of the test group, answering the questionnaire, generally found DrModelica to be a better way of learning a programming language compared to ways they are used to. One conclusion that can be drawn from the evaluation is that DrModelica is a good teaching material for Modelica. The evaluators also found that Literate programming is a methodology suitable for learning Modelica. DrModelica is developed with the programming environments DrJava (for Java) and DrScheme (for Scheme) in mind.

There is a need for a programming environment for Modelica and DrModelica will hopefully fill this need and increase the usage of Modelica by facilitating the learning process.

The interested reader can visit: <http://www.DrModelica.org>, where a short version of DrModelica is freely available for download. The full version of the material is included in the software MathModelica and in "*Principles of Object-Oriented Modeling and Simulation with Modelica*" by Peter Fritzson.

References

- [1] Fritzson, P., *Principles of Object-Oriented Modeling and Simulation with Modelica*. 2003: IEEE Press and John Wiley.
- [2] Grubb, P. and A.T. Armstrong, *Software Maintenance Concepts and Practice (Second Edition)*. 2003: World Scientific Pub Co.
- [3] Wolfram Research, *Mathematica*. 4 ed. 1999, Champaign, Illinois: Wolfram Research, Inc.
- [4] Fritzson, P., J. Gunnarsson, and M. Jirstrand. *MathModelica - An Extensible Modeling and Simulation Environment with Integrated Graphics and Literate Programming*. In *Proceedings of the 2nd International Modelica Conference*. 2002. Munich Germany.
- [5] Jirstrand, M. *MathModelica - A Full System Simulation tool*. In *Proceedings of the 6th Conference on Product Models, Global Product Development*. 2000. Linköping, Sweden.
- [6] Fritzson, P., et al. *The Open Source Modelica Project*. In *Preceedings of the 2:nd International Modelica Conference*. 2002. Munich, Germany.
- [7] Elmquist, H., D. Bruck, and M. Otter, *Dymola - User's Manual*. 1996, Dynasim AB, Research Park Ideon: Lund.
- [8] The Dynasim Home Page, *Dymola for Your Complex Simulations*. Available at: <http://www.dynasim.se>. Last accessed August, 2003.
- [9] Nørmark, K. *Requirements for an Elucidative Programming Environment*. In *Preceedings of the International Workshop on Program Comprehension, IWPC'2000*. 2000. Limerick, Ireland.
- [10] Ducassé, M. and J. Noyé, *Logic Programming Environments: Dynamic Program Analysis and Debugging*. 1994. **19/20**: p. 351-384.
- [11] Lengquist Sandelin, E.-L. and S. Monemar, *DrModelica - An Experimental Computer-Based Teaching Material for Modelica*, Master Thesis Department of Computer and Information Science. 2003, Linköping University, Sweden.
- [12] Knuth, D.E., *Literate Programming*. The Computer Journal 1984. **NO27(2)**: p. 97-111.
- [13] Allen, E., R. Cartwright, and B. Stoler. *DrJava: A Lightweight Pedagogic Environment for Java*. In *Preceedings of the 33rd ACM Technical Symposium on Computer Science Education (SIGCSE 2002)*. 2002. Northern Kentucky, USA.
- [14] Findler, R.B., et al. *DrScheme: A Programming Environment for Scheme. A Preliminary Version Appeared at Symposium on Programming Languages: Implementations, Logics, and Programs in 1997*. 2001.
- [15] Nielsen, J., *Usability Engineering*. 1993, San Diego: Academic Press Inc.
- [16] Nielsen, J. and R.L. Mack, *Usability Inspection Methods*. 1994: John Wiley and sons inc.
- [17] Nielsen, J., *Ten Usability Heuristics*. 1994. Available at: http://www.useit.com/papers/heuristic/heuristic_list.html. Last accessed September 2003.

Session 4A

Automotive Simulation – III

Simulation of Engine Systems in Modelica

John Batteh

Michael Tiller

Charles Newman

Ford Motor Company, Powertrain Research Department
Dearborn, MI USA
{jbatteh, mtiller, cnewman}@ford.com

Abstract

This paper details the use of the Modelica modeling language for the simulation of engine systems. The first part of the paper briefly outlines some of the challenging, multi-domain components of engine system modeling and is followed by a discussion of some of the connectors, interfaces, and model templates that enable robust, efficient model development. The remainder of the paper presents selected modeling examples with particular attention to the structure and implementation of the models that promotes model flexibility and re-use.

1 Introduction

As automobile manufacturers face increasing pressure to reduce emissions, increase fuel economy, reduce development costs, and enhance vehicle performance and driveability, it has become especially crucial to consider optimization opportunities at the system level. While it is conceptually possible to obtain system improvements via prototype hardware fabrication, this process is inefficient, costly, and sub-optimal. With the development of modeling tools that allow robust, multi-domain, system-level simulations, it is becoming increasingly attractive to perform this optimization process in the virtual environment.

Engine systems, in particular, contain a wide range of multi-domain physical modeling challenges [1]. Table 1 contains a partial list of physical processes and modeling domains that could be considered in the modeling of a spark-ignited (SI) engine system depending on the particular analysis and desired level of detail. Due to the wide variety of physical processes and modeling domains along with the inherent interactions, it is imperative to have a descriptive language that is capable of modeling across the different physical domains. This need only increases as more of the overall vehicle system and associated attributes (*e.g.* NVH, safety, *etc.*) are included.

Table 1. Physical processes and modeling domains for an engine system

Physical Process	Modeling Domain(s)
Intake and exhaust valve actuation mechanisms	M, F
Intake and exhaust flow past the valves	T
Piston and crankshaft motion	M
Manifold dynamics in the intake and exhaust systems	T, F
Injection and transport of liquid fuel and fuel vapor	T
In-cylinder fluid motion	T, F
Ignition and flame propagation in the combustion chamber	T, Ch
Heat transfer between the gas, fuel, coolant system, and metal surfaces	Th
Frictional effects in engine, valvetrain, and powertrain	M, Th
Emissions formation and mitigation	T, Th, Ch
Thermal response of the intake system, engine, and exhaust system	Th
Coolant and lubrication flow	F
Powertrain, chassis, and mount dynamics	M

Legend

Ch = Chemical

F = Fluid (distributed)

M = Mechanical

T = Thermodynamic

Th = Thermal

Modelica¹ [2] with its high-level, acausal, declarative formulation for physical modeling is an ideal language for multi-domain system simulations. The Modelica standard Mechanical, Rotational, MultiBody, and Thermal libraries contain the connector definitions, interfaces, and basic models that provide the framework for the modeling of engine systems. The sections that follow discuss the use

¹ Modelica is a trademark of the Modelica Association

of these standard libraries along with the supplemental connectors and associated models that enable the formulation and simulation of engine system models.

2 Physics Overview

For each of the physical processes described in Table 1, models of varying level of detail can be formulated. Due to the number of component models used in a typical engine systems simulation, it is impractical to discuss the physics of particular models in detail. This section is meant to give a very brief overview of some of the physics involved in engine systems modeling.

Mechanical modeling in an engine system includes a combination of 1D and multi-dimensional dynamics. Typically, the multi-dimensional dynamics are of interest in detailed models of the vehicle dynamics and mounting systems. A 1D approach is often used in modeling the engine itself. Within the 1D framework, the model of the valve actuation mechanism can either include kinematic relationships (*i.e.* cam motion constrained to the motion of the crankshaft with valve lift prescribed as a function of the cam motion) or dynamic behavior (see [3] for a discussion of a dynamic, camless valve actuator model). Similarly, the piston can be modeled as massless using kinematic relationships between the piston, crank-slider, and crankshaft or can include the effects of piston mass from a force balance.

Modeling the thermodynamics is a crucial part of engine systems modeling. Typically several control volumes are formulated for which fundamental equations for energy and mass conservation are applied:

$$\frac{dU}{dt} = \dot{Q} - \dot{W} \quad (1)$$

$$\frac{dM}{dt} = \dot{m} \quad (2)$$

A typical engine model might include one (or several) control volumes in the cylinder, the intake system, and the exhaust system with mass and energy exchange between the volumes. Flow past the valves in an engine is typically modeled using isentropic relationships for flow past an orifice with an experimentally determined discharge coefficient [1]. The calculations of the requisite thermodynamic properties come from models with varying treatments of the species (*i.e.* fuel, fresh air, *etc.*) and levels of detail (*i.e.* constant c_p and c_v ,

polynomial property functions, chemical equilibrium mixture calculations [4], *etc.*). Fluid modeling is similar to thermodynamic modeling but usually involves a larger number of distributed control volumes and may involve the conservation of momentum as well. For example, accurately capturing the pressure dynamics of the flow in induction and exhaust systems requires a high level of discretization, perhaps even with specialized numerical techniques for shock capturing.

Heat transfer and thermodynamics are intimately linked in engine systems via Eq. (1). Convective heat transfer between the gas and the metal surfaces affect the volumetric efficiency of the engine, heat losses during the power stroke, heat losses in the exhaust system, and the thermal response of the engine and exhaust system components. The convective heat transfer is modeled from the fundamental constitutive equation:

$$\dot{Q} = \bar{h}A(T_g - T_w) \quad (3)$$

where the average convective heat transfer coefficient comes from experimental correlations. Cold start thermal response of the engine components is key from the standpoint of both mixture preparation and emissions formation and mitigation.

Combustion is a highly complex process involving thermodynamics, heat transfer, fluid motion, and chemical kinetics. Combustion models come in many flavors and with varying levels of fidelity. The combustion process can be simplified to a prescribed heat release process, such as a Wiebe function [1] for mass fraction burned. More detailed, predictive combustion models typically can account for multi-zone combustion and heat transfer, the effects of charge motion on the combustion process, variations in the laminar flame speed for different cylinder conditions, *etc.* (see [4] and the references therein for a description of a detailed combustion model in Modelica).

3 Interfaces

Standard interfaces are a key element for developing flexible models. Experience has shown that the most powerful and flexible Modelica libraries are based on solid connector definitions. The remainder of this section discusses some of the modeling elements that comprise the engine architecture.

3.1 Thermal Architecture

The heat transfer process plays a significant role in engine systems modeling. The interaction between the air in the cylinder and the metal surfaces in the intake, exhaust, and cylinder affects the liquid fuel preparation process along with the volumetric efficiency, performance, and emissions of the engine.

One challenge in modeling the thermal effects in the engine is the variety of different models that can be used to represent the thermal response of the various pieces. For example, an engine thermal response model could be formulated on a cylinder-by-cylinder basis or could be a lumped model at the engine level. To allow for both of these formulations and to minimize the number of connections between the engine or cylinder and the thermal models, the special thermal connectors in Figure 1 were developed. Modelica code fragments for these connectors are shown in Figure 2. The CylinderTemperatures connector is a "mega connector"- a connector that is an aggregate of other connectors- and can be thought of as a thermal bus. It contains a number of thermal and friction connectors that comprise the pre-defined standard thermal cylinder architecture. This architecture defines the elements that are included in every cylinder thermal response model and is represented graphically in Figure 3. This breakout box explicitly shows all the connectors that are lumped into the single CylinderTemperatures connector and is used in the low-level cylinder heat transfer models to facilitate the graphical connection of the individual elements of the heat transfer model. The ThermalEnvironment connector is the engine-level connector and is an array of CylinderTemperatures connectors. This parametric representation scales with the number of cylinders being modeled and, by consolidating the signals onto one connector, allows for a single connection between the engine and the engine thermal response model at the top level. The cylinder and engine connectors will be seen repeatedly in the standard interfaces that follow.

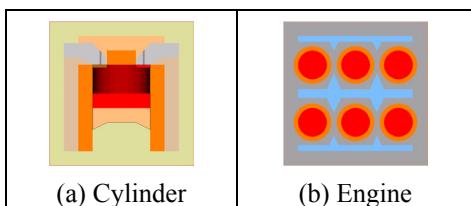


Figure 1. Thermal connectors

```

connector CylinderTemperatures
  import HeatTransfer=Modelica.Thermal.HeatTransfer;
  outer parameter Ford.Types.EngineTopology
    engine_topology;
    HeatTransfer.Interfaces.HeatPort_a head;
    HeatTransfer.Interfaces.HeatPort_a intake_valves[engine_topology.intake_valves];
    HeatTransfer.Interfaces.HeatPort_a block_coolant;
    HeatTransfer.Interfaces.HeatPort_a cylinder_liner;
    HeatTransfer.Interfaces.HeatPort_a piston;
    HeatTransfer.Interfaces.HeatPort_a oil;
    Ford.Engine.Interfaces.Friction valvetrain;
...
end CylinderTemperatures;
connector ThermalEnvironment
  outer parameter Ford.Types.EngineTopology
    engine_topology;
    CylinderTemperatures
    cylinder_temperatures[engine_topology.cylinders];
end ThermalEnvironment;

```

Figure 2. Excerpts from the thermal connectors models

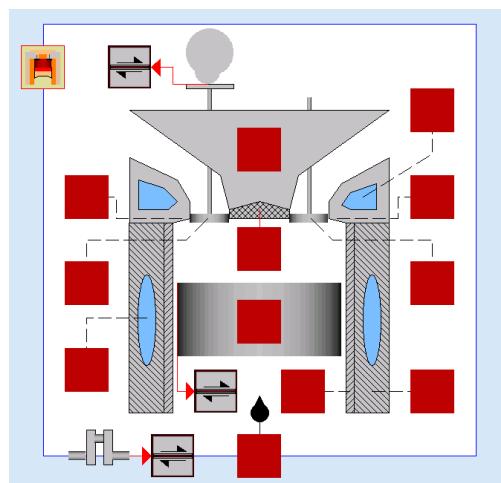


Figure 3. Breakout box showing elements of CylinderTemperatures connector

The thermal architecture in the engine provides the framework for the interactions between the cycle simulation models and the engine temperature models, thereby allowing independent selection of the either model. Roughly speaking, the cycle simulation models are responsible for computing the "metal-gas" thermal interactions while the engine temperature models calculate the "metal-fluid" interactions.

3.2 Cylinder Interface

The cylinder interface defines the framework for the cylinder implementation process. The standard interface is shown in Figure 4 and defines the exterior connection points for the cylinder. The `partial model` contains three 1D rotational connectors, one each for the crankshaft, camshaft, and engine block. The connection to the engine block allows for the rotational motion of the engine on the mounts. The interface also includes the

previously discussed `CylinderTemperatures` connector for the cylinder thermal environment along with thermodynamic connectors for both the induction and exhaust systems. The thermodynamic connectors contain pressure, temperature, species mass fraction, species mass flow rates, and convected energy along with information related to fluid properties. It is anticipated that these thermodynamic connectors will be replaced with those from the Modelica standard fluids library currently under development [5].

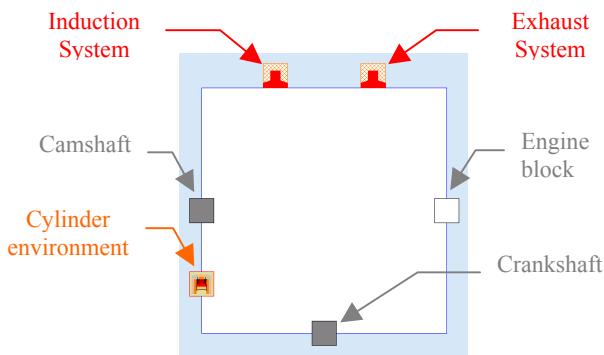


Figure 4. Cylinder interface

3.3 Engine Interface

The standard engine interface is shown in Figure 5. This `partial model` contains two 1D rotational connectors, one each for the crankshaft and the engine block. In addition, the interface contains a `ThermalEnvironment` connector to represent the engine thermal behavior. Note the absence of the induction and exhaust system thermodynamic connectors in the engine interface. These connectors have been omitted from the interface definition so that derived models can define their own plenum configurations (*i.e.* single plenum, dual plenum, *etc.*). Section 4.2 describes models that extend from this engine interface and instantiate the needed components (*i.e.* cylinders, *etc.*) for a complete engine implementation.

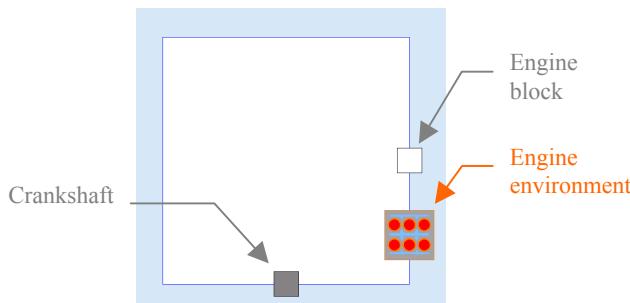


Figure 5. Engine interface

3.4 Medium Models

The working fluid is defined using the `MediumModel` idiom [4]. This approach defines a consistent set of models, functions, constants, and connectors that contain all the medium-specific information and thus define a particular implementation of the `MediumModel` idiom. For example, the material property calculations, equations of state, chemical species representation, combustion chemical kinetics, and associated helper functions could be included in the formulation. Implemented via replaceable packages, the `MediumModel` idiom enables the orthogonal development of property models and the components that use them (*i.e.* the decomposition of medium and machine) and provides an organized, consistent framework for the development of models with varying levels of detail.

Because the medium-specific information is contained wholly within the replaceable package, the working fluid specification can be changed at a single place at the highest level of the model with a consistent application of the change reflected throughout the model hierarchy. This "flip of a switch" flexibility is enhanced by the addition of the `choices` annotation in the Modelica language. The `MediumModel` concept is currently being used in the development version of the Modelica standard fluids library [5].

3.5 ModelData Structure

Populating hierarchical model structures with consistent data is a non-trivial task, especially considering the different data required for models of the same type but with varying levels of fidelity. To ensure a consistent application of data throughout the modeling structure, the `MediumModel` concept [4] has been adapted to organize data required for the engine models. A new `ModelData` package has been created to serve as the repository for the data required for the various models in the main library. Inside this package are sub-packages that correspond to the various subsystems in the vehicle (*e.g.* Engine, Transmission, *etc.*). Finally, packages exist that contain the particular data for a given entity (*i.e.* a vehicle, specific transmission, *etc.*). The various components that use the model data contain a `replaceable package` called `EngineData` from which specific elements are instantiated. Thus, a single redeclare of the `EngineData` package at the top-level of the model hierarchy populates

the entire hierarchy with a consistent data set for simulation of a particular system. The `redeclare` is simplified by the support for the choices annotation in the Dymola² [6] GUI.

3.6 SignalBus Concept

The SignalBus concept [7, 8] is used to pass control signals throughout the model hierarchy. This concept uses the `inner` and `outer` semantics to propagate the control signals without requiring connections at every level in the model hierarchy. This technique facilitates the propagation of the control signals for replaceable components which typically require varying control signals for different levels of model fidelity. The SignalBus concept requires a top-level definition that represents the union of all the control signals and is coupled with selective definition and use of the control signals at the lower model levels. The interested reader is referred to [7, 8] for more discussion of the implementation of the SignalBus idiom.

4 Model Templates

While the standard interfaces discussed in the previous section provide a nice framework for a flexible, reusable modeling system, it is highly desirable to have more extensive models pre-built to establish a higher-level starting point for the model developer. This section provides some sample template and configuration models with a focus on the key Modelica language features that contribute to the flexibility. Additional details of the templates and configuration options are given in [8].

4.1 Cylinder Configurations

The majority of the work in engine modeling is focused on establishing the proper model for the cylinder. This process involves choosing the intake and exhaust system models (including the valve actuation mechanism), the combustion and heat transfer models, and populating the models with the appropriate data (*i.e.* bore, stroke, compression ratio, valve timings, *etc.*). To streamline the effort in assembling the cylinder design model, it is desirable to create a baseline cylinder model that can be used as the starting point for many different variants via the Modelica `replaceable` feature. Figure 6 shows the MinimalCylinder model that serves as a base

model for various cylinder designs (note the components from the cylinder interface shown in Figure 4). An excerpt of the Modelica code is provided in Figure 7. Note the extensive use of `replaceable` types. Currently, the modifiers are applied to the instantiated components to ensure that the modifiers are picked up during a subsequent `redeclare`. In Modelica 2.1, the semantics of `redeclare` have been defined more explicitly to address the issue of modifiers with `replaceable` and `redeclare`. The combustion and heat transfer models are not included in MinimalCylinder and are left to be instantiated in an extending model. The MinimalCylinder template provides a flexible platform for creating cylinder models from different configurations and fidelity levels.

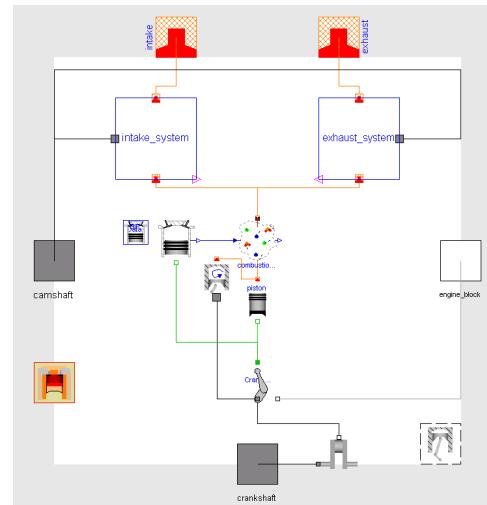


Figure 6. MinimalCylinder template model

```
partial model MinimalCylinder
  extends Ford.Engine.BaseClasses.Cylinder;
  replaceable model ControlVolume =
    Thermodynamics.VariableControlVolume;
  Control Volume combustion_chamber(modifiers);
  replaceable model Piston=Drivetrain.MasslessPiston
    extends Ford.Engine.Interfaces.Piston;
    Piston piston(modifiers);
  Mechanical.Crank crank(modifiers);
  InCylinder.ChamberVolume chamber_volume(modifiers);
  replaceable model IntakeSystem =
    Ford.Engine.Interfaces.IntakeExhaust;
  IntakeSystem intake_system(modifiers);
  replaceable model ExhaustSystem =
    Ford.Engine.Interfaces.IntakeExhaust;
  ExhaustSystem exhaust_system(modifiers);
...
end MinimalCylinder;
```

Figure 7. Code excerpt for MinimalCylinder

Figure 8 shows such an extension of the MinimalCylinder model with the intake and exhaust systems redeclared to be conventional, fixed valve timing models and the instantiation of

² Dymola is a trademark of Dynasim AB

Wiebe [1] combustion and Woschni-type [9] heat transfer models. Taking advantage of the `replaceable` components allows model variants to be quickly created with a minimum amount of model re-wiring, configuration, and code duplication. This sort of "plug and play" flexibility allows model assembly via simple `redeclare` statements for existing components. In terms of the valvetrain models, model variants exist to account for different valve actuation mechanisms, timing and phasing strategies, and configurations. The ideal piston could be replaced with a model that accounts for the effects of piston mass. Liberal use of the `replaceable` components is the key Modelica language feature for establishing these sorts of template models for "plug and play" configuration.

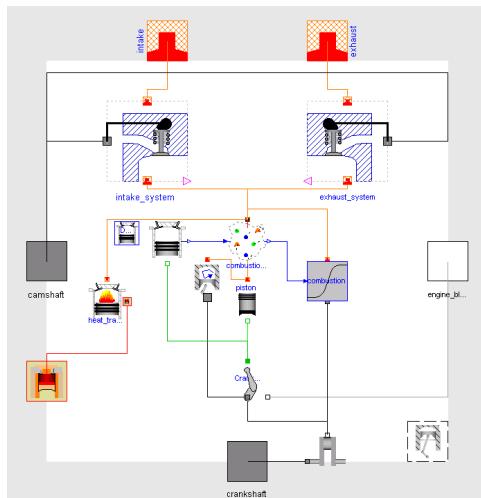


Figure 8. Fixed timing, Wiebe cylinder model

4.2 Engine Templates

Having established a flexible framework for the cylinder design process, it naturally follows that templates should be established for the various engine configurations. Again, these templates help to minimize the modeling effort for assembling model variants, which at the engine level means building an engine model using a new cylinder design. Templates exist for various engine/plenum configurations (*i.e.* single cylinder, I4, V6, V8, *etc.*) as shown in Figure 9. Each template extends from the engine interface in Figure 5 and includes all of the connections between the cylinder(s) and the external interfaces. The key feature in each of the engine configurations is the `replaceable` CylinderModel shown in the code excerpt in Figure 10. This CylinderModel is then instantiated repeatedly for multi-cylinder engines.

Therefore, creating a stand-alone engine model is simply a matter of extending from the appropriate engine template and redeclaring the CylinderModel. This single redeclare of the CylinderModel type is then used for the instantiation of each cylinder in the engine.

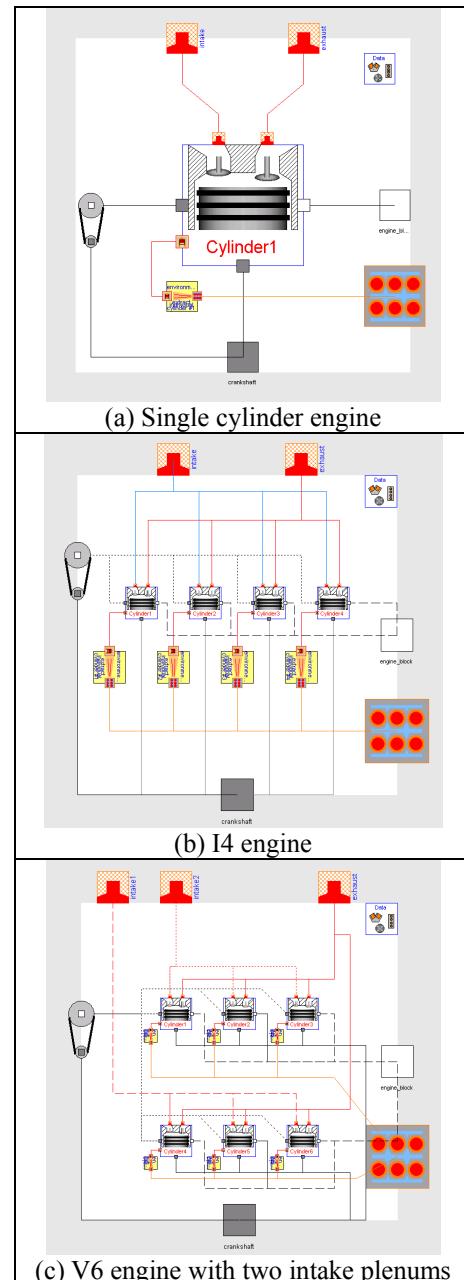


Figure 9. Engine configurations

```
...
  replaceable model CylinderModel =
    Interfaces.Cylinder;
  CylinderModel Cylinder1(shift=crank_shift[1],
  redeclare package MediumModel
  = MediumModel)
...

```

Figure 10. Code excerpt for engine templates

4.3 Experimental Templates

Extending the template abstraction even further, templates have been created for common types of simulation experiments. Figure 11 shows examples of an experimental setup for an engine on a dynamometer (a) and for a cranking engine (b). Code excerpts from the template base class are shown in Figure 12. These generic templates can be simulated for a particular engine configuration and cylinder design by simply extending from the appropriate template and adding a `redeclare` for Configuration and CylinderModel. This technique allows single templates to be used for every existing engine configuration and cylinder design that conforms to the interfaces in Figures 4-5.

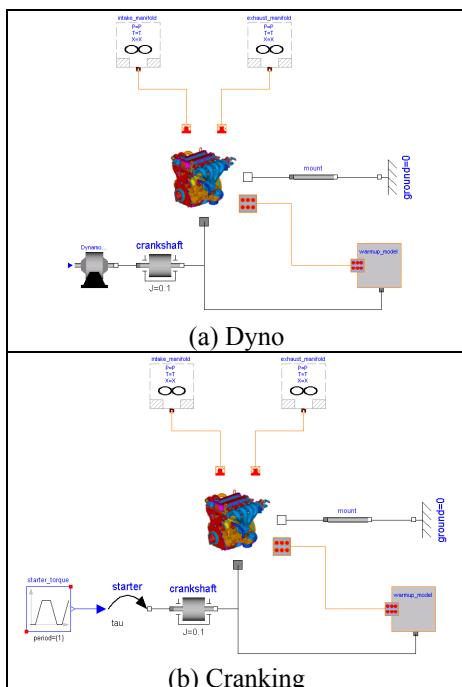


Figure 11. Templates for dyno and cranking experiments

```
...
replaceable model CylinderModel =
  Interfaces.Cylinder extends
  Ford.Engine.Interfaces.Cylinder;
replaceable model Configuration =
  Interfaces.Engine;
replaceable Configuration engine(modifiers);
...
```

Figure 12. Code excerpt from experimental template base class

5 Model Examples

This section presents some examples of engine system simulations. These examples illustrate the use of the experimental templates and also show

how models of increasing complexity can be built using the modeling framework discussed previously. Each model was simulated using Dymola [6].

5.1 Engine Cranking

The key-on crank of the engine is a complex, dynamic process involving the electrical system and controls, along with the actual engine itself. Controlling and optimizing the engine cranking behavior is crucial from the standpoint of both emissions and customer feel. This section shows some results from a detailed, multi-domain model of a cranking engine.

The crank model shown in Figure 13 is built upon the cranking template in Figure 11b. The Configuration has been defined as a single-cylinder engine with a CylinderModel that includes detailed, multi-zone, predictive combustion [4]. The intake reservoir has been replaced by a dynamic model of the manifold and throttle. The engine warmup model is a simple, fixed temperatures model. The control and electrical systems have been simplified such that the starter applies the commanded torque for 0.5s at 0.25s. The treatment of the engine friction is simplified in this model to a constant opposing torque starting at 0.5s. In this simulation, the throttle is closed to represent idle conditions. During the cranking process, the liquid fuel dynamics are extremely important since mixture preparation is inhibited at low speeds, high manifold pressures, and under cold conditions. While these effects can be considered within this modeling framework [3, 10], they are not included in these simulations.

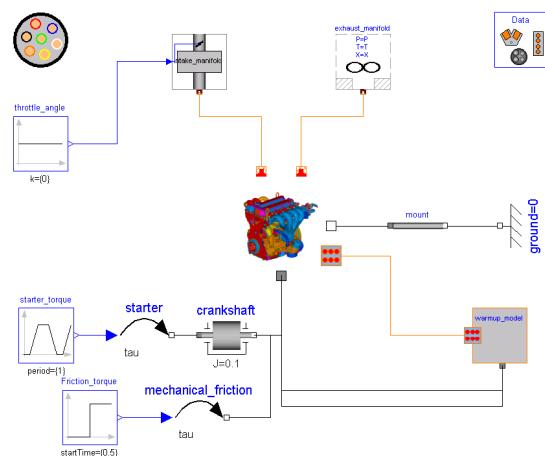


Figure 13. Model for cranking engine simulation

This single-cylinder cranking simulation with fixed metal temperatures has 207 components, 1033 time-varying variables, 1120 non-trivial equations, and 53 states. Figures 14-15 show the response of the engine speed and manifold pressure during the first 3s of the cranking simulation. The starter begins to spin the engine up at 0.25s. The manifold starts at approximately ambient pressure and then begins to pump down due to the emptying and filling process between the upstream intake reservoir (the ambient) and the engine. Note the "gulping" from the manifold due to the single-cylinder engine. A multi-cylinder engine results in the smoothing of the pumping down of the manifold due to the more frequent breathing from the multiple cylinders. The engine speed increases rapidly during the first few firing events since the manifold pressure is still high, resulting in a large amount of combustible mass in the cylinder. The engine speed starts to drop as the manifold pumps down and starts approaching a steady idle speed of 1700 RPM.

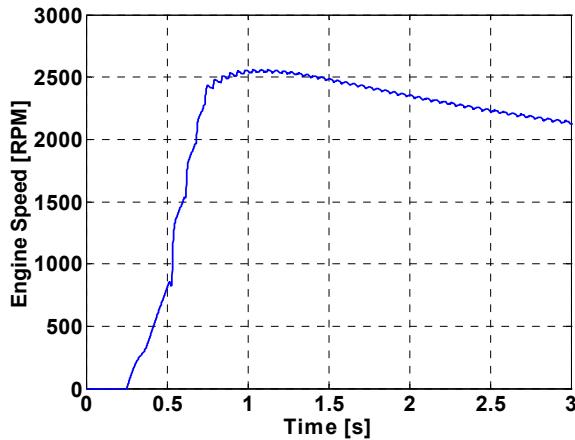


Figure 14. Engine speed response

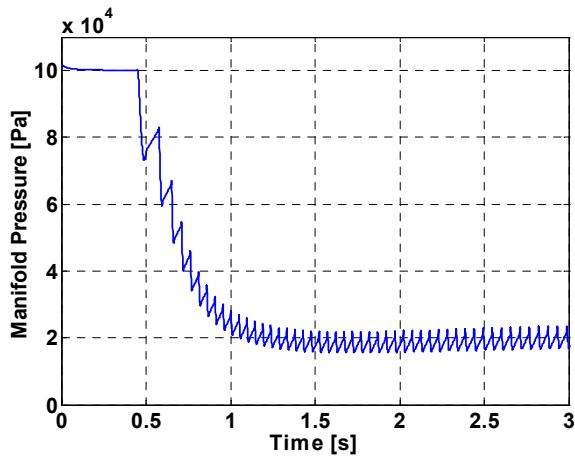


Figure 15. Manifold pressure response

5.2 Exhaust System Warmup

Vehicle thermal management is a critical issue in light of the recent legislation mandating lower emissions levels. The optimization of the engine system, from start-up strategy to component design of the intake, cylinder, and exhaust systems, is a key enabler to meeting more stringent emissions standards by reducing engine-out emissions and light-off time for the three-way catalyst. This section shows an engine system, cold start simulation from crank for evaluation of the thermal response of the exhaust system.

The model used in this simulation extends from the cranking engine model discussed previously (Figure 13). This version replaces the fixed temperatures model for the engine with the dynamic thermal response model shown in Figure 16. This model is extended from the work in [11] and includes models for the warmup of the piston, head, block, and valves along with a simplified representation of the oil and coolant loops. This simulation also includes a model, shown in Figure 17, of the exhaust system, including the exhaust manifold and downpipe leading to the catalyst. This model is based on [12] and includes distributed models for the thermal interaction between the exhaust gas and the pipe wall. The effects of forced convection between the gas and the wall, conduction along the pipe wall, and natural convection between the pipe outer wall and the ambient are included.

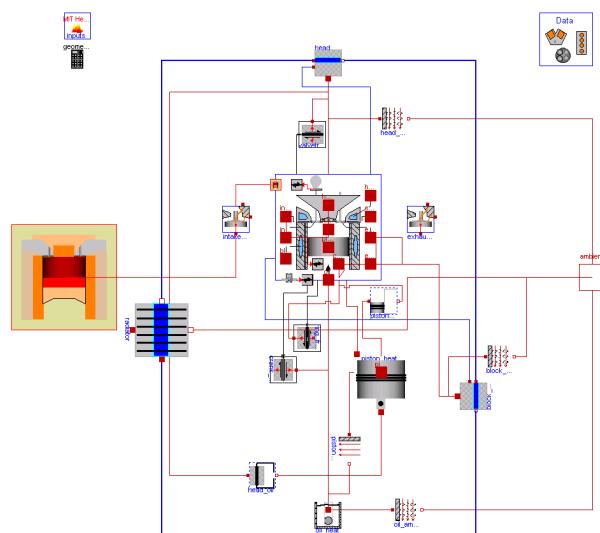


Figure 16. Engine thermal response model

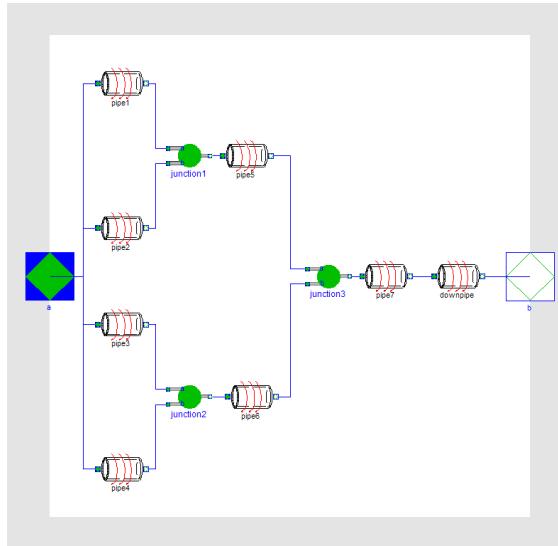


Figure 17. Exhaust system model

The cold start, cranking model with variable metal temperatures and an exhaust system has 924 components, 5476 time-varying variables, 5825 non-trivial equations, and 221 states. The increase in the number of equations and states from the cranking simulation discussed in Section 5.1 results mainly from the inclusion and discretization of the pipes in the exhaust system. Each of the 8 pipes in Figure 17 was divided into 10 elements along its length, and each element has 2 states (one each for the temperature of the exhaust gas and the temperature of the pipe wall in the element).

To simulate the start of the FTP drive cycle test for emissions, the model was run for approximately 20 seconds. This test begins with a cold crank and idle until approximately 20 seconds when the first acceleration occurs. Figure 18 shows the thermal response of some of the components in the engine thermal model (Figure 16). Note that the components that receive heat directly from the gas in the cylinder (*i.e.* piston, head, liner) start to warm first. The piston has a lower thermal capacitance than does the liner and the head so it warms more quickly. The temperature rise from ambient is fairly modest due to the large thermal capacitance of the engine and the short simulation time (typical engine warm-up occurs over several minutes).

The temperature of the exhaust gas as it traverses the exhaust system is crucial as the thermal energy in the gas is responsible for warming the three-way catalyst to the elevated temperatures at which it becomes effective. Figure 19 shows the transient temperature of the exhaust gas as various points in

the system. The highest temperatures are at the entrance to the exhaust port (just past the exhaust valve) with temperatures decreasing along the system due to heat loss to the cold pipe walls. The highest exhaust gas temperature occurs roughly at the maximum speed (see Figure 14) where maximum amount of combustible mass is trapped in the cylinder due to the high manifold pressure. Note the large drops in temperature throughout the system. Minimizing the amount of energy lost in the exhaust manifold and piping leading to the catalyst during a cold start is crucial for minimizing catalyst light-off times. This sort of engine system model can be used to effectively and efficiently evaluate different engine startup strategies and hardware designs and their effects on exhaust system thermal response.

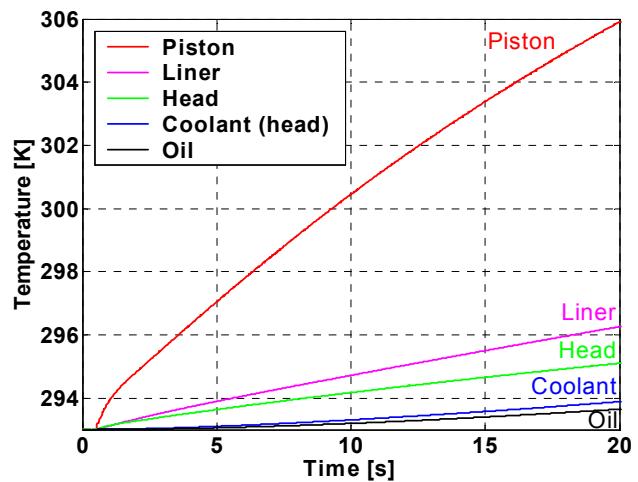


Figure 18. Thermal response of engine components

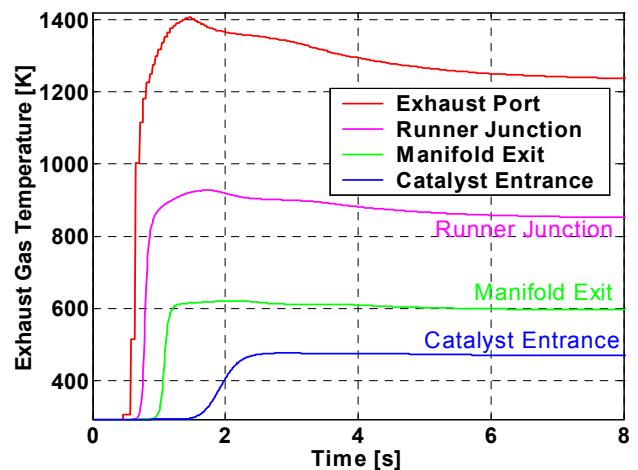


Figure 19. Thermal response of exhaust gas

6 Conclusions

This paper describes the use of the Modelica modeling language for engine system simulations. A robust, flexible, and re-usable modeling framework of connectors, interfaces and templates is described for multi-domain engine system modeling. Results from the detailed simulations of the engine cranking process yield some insight into the types of models that can be realized using this framework and the vast amount of information that can be obtained from these types of simulations. These multi-domain models are well suited for the evaluation and optimization of hardware design and control strategies, especially during the early concept assessment stage of the design process. Future work will focus on the validation of the individual submodels and system-level models.

References

1. Heywood, J.B., 1988, *Internal Combustion Engine Fundamentals*. McGraw-Hill.
2. Modelica Association, 2002, "Modelica Language Specifications (Version 2.0)", <http://www.modelica.org>
3. Puchalsky, C., et al., 2002, "Modelica Applications for Camless Engine Valvetrain Development", *2nd International Modelica Conference Proceedings*, pp. 77-86, http://modelica.org/Conference2002/papers/p1_1_Puchalsky.pdf
4. Newman, C., Batteh, J., and Tiller, M., 2002, "Spark-Ignited-Engine Cycle Simulation in Modelica", *2nd International Modelica Conference Proceedings*, pp. 133-142, http://modelica.org/Conference2002/papers/p1_7_Newman.pdf
5. Elmquist, H., Tummescheit, H., and Otter, M., 2003, "Principles of Object-oriented Modeling of Thermo-Fluid Systems", *3rd International Modelica Conference Proceedings*.
6. Dymola. Dynasim AB, Lund, Sweden, <http://www.dynasim.se>
7. Tiller, M., Tobler, W.E., and Kuang, M., 2002, "Evaluating Engine Contributions to HEV Driveline Vibrations", *2nd International Modelica Conference Proceedings*, pp. 19-24, http://modelica.org/Conference2002/papers/p0_3_Tiller.pdf
8. Bowles, P. and Batteh, J., 2003, "A Transient, Multi-Cylinder Engine Model Using Modelica", *SAE-2003-01-3127*, Society of Automotive Engineers.
9. Woschni, G., 1967, "A Universally Applicable Equation for the Instantaneous Heat Transfer Coefficient in the Internal Combustion Engine", *SAE-67-0931*, Society of Automotive Engineers.
10. Batteh, J.J. and Curtis, E.W., 2003, "Modeling Transient Fuel Effects with Variable Cam Timing", *SAE-2003-01-3126*, Society of Automotive Engineers.
11. Kaplan, J.A. and Heywood, J.B., 1991, "Modeling the Spark Ignition Engine Warm-Up Process to Predict Component Temperatures and Hydrocarbon Emissions", *SAE-91-0302*, Society of Automotive Engineers.
12. Laing, P.M., Shane, M.D., Son, S., Adamczyk, A.A., and Li, P., 1999, "A Simplified Approach to Modeling Exhaust System Emissions: SIMTWC", *SAE-199901-3476*, Society of Automotive Engineers.

Modelling 3D Mechanical Effects of 1D Powertrains

Christian Schweiger*

Martin Otter[†]

German Aerospace Center (DLR)
 Institute of Robotics and Mechatronics
 Oberpfaffenhofen, 82234 Weßling, Germany
<http://www.robotic.dlr.de/control/>

Abstract

It is described how to extend one-dimensionally modelled rotational mechanical systems such that they can be mounted on three-dimensional multi-body system models without neglecting any dynamic effects. This is performed by adding support torques to existing drive train elements and by introducing new components that take care of the gyroscopic torques and the transformation of one-dimensional into three-dimensional support torques. It is demonstrated that this approach is convenient for the user and leads to efficient simulation code.

1 Introduction

Dependent on the point of view, there are traditionally two ways of modelling and simulating powertrains. It is possible to obtain the complete dynamics of a powertrain using multi-body systems simulation. One drawback is the high effort needed in describing the powertrain in its complete geometry. Another disadvantage is the comparative low efficiency of this method concerning simulation of friction elements as used in clutches and gearboxes. Since the latter are the essential part of powertrains with a major impact on the dynamics, multi-body systems simulation seems to be not the appropriate method for realtime simulation.

For realtime simulation of powertrains, hybrid discrete-continuous modelling techniques became quite common. They allow modelling not only by differential equations, but by additional boolean equations. This combination is very useful for modelling of variable structure systems, especially of friction elements. As friction elements are considered only one-dimensional (1D), it was acceptable to neglect three-

dimensional (3D) mechanical effects in the past. Their neglect allowed to define the considered powertrain in simply one dimension with high efficiency.

With the growing level of detail in the vehicle dynamics area, there is an upcoming interest on the influence introduced by the powertrain dynamics. For example, the vehicle dynamics is influenced by the *support torques*, which act on the vehicle body over the mounting of the gearbox. Also *gyroscopic torques* could be important during gear shifts of a yawing vehicle.

The objective of the work described in this paper is to merge together the advantages of both 1D and 3D modelling of powertrains by using appropriate components, which provide the resulting torques of the powertrain to the 3D vehicle dynamics model without neglecting any dynamic effects.

The basic idea is as follows: The bearings of a 1D powertrain modelled with the 1D Rotational library (Modelica.Mechanics.Rotational) and/or the 1D PowerTrain library [1] are fixed rigidly on a *carrier body* which moves in 3D space and which is modelled as 3D multi-body system with the new Modelica MultiBody library [2]. The 3D movement of the whole system is described correctly and without any neglections in the following way:

- It is assumed that all rotating bodies in the powertrain have rotational symmetry.
- The carrier body on which the powertrain is fixed has to be defined in such a way that it has the common mass, the common center-of-mass and the common inertia tensor of the body together with the powertrain under the assumption that the rotating bodies in the powertrain are fixed relative to the carrier body. It does not matter at which angle the rotating parts are fixed, since it is assumed that the bodies have rotational symmetry.

*Christian.Schweiger@dlr.de

[†]Martin.Otter@dlr.de

This approach is also convenient for the user: It suffices to measure the mass, the center of mass and the inertia tensor of, say, a complete automatic gearbox or get this data from a CAD system. Additionally, only the inertias of the rotating shafts around their axis of rotation are needed, as required for 1D modelling of powertrains. This approach is simpler and more practical as requiring to get mass, center of mass and complete inertia tensor of every single piece of a powertrain.

In the next section it is described how to extend the existing 1D components such that support torques are computed. Section 3 introduces hybrid 1D/3D components. Section 4 discusses the implementation of a gearbox modelled solely with multi-body components in order to be able to compare the different modelling philosophies. An extract of the tests performed on the components are presented in Sec. 5. The paper closes with an application example in Sec. 6 and subsequent conclusions.

2 Support Torque in 1D

As discussed in the previous section, the 3D mechanical effects of a powertrain comprise the support torques for components which interact with the powertrain housing, e.g., gears and brakes. In the past, these support torques have not been considered neither in the `Modelica.Mechanics.Rotational` nor in the `PowerTrain` library. This was disadvantageous even for simple 1D powertrains: It was, e.g., not possible to model the dynamics of a gearbox housing, mounted on the ground via spring-damper-systems.

In order to overcome this deficiency, it is necessary to introduce an additional 1D connector representing the bearing flange. This bearing connector can be used to fix components on the ground or on other rotating elements or to combine it with force elements. As a side effect, the support torque is computed explicitly.

For backward compatibility and convenience reasons, it is desired not to be forced to connect this connector in every case. With the Modelica operator `cardinality` it is possible to inquire the number of connections to a connector. This information is used to provide different equations in case the connector is not connected. Otherwise, the duplication of many models would have been necessary.

In the following, it is shown for the model `Modelica.Mechanics.Rotational.IdealGear`, Fig. 1(a), how the respective components of the rotational library are adapted. In Lstg. 1 the flange

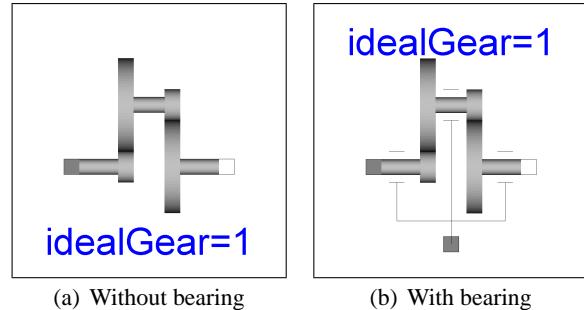


Figure 1: Ideal gearbox

connectors from the Rotational library are recalled.

Listing 1: Flange connectors from Rotational library

```
within Modelica.Mechanics.Rotational.Interfaces;
connector Flange
  import SI = Modelica.SIunits;
  SI.Angle phi "absolute flange angle";
  SI.Torque tau "cut-torque in flange";
end Flange;

connector Flange_a = Flange;
connector Flange_b = Flange;
```

The previous implementation of the `IdealGear` model is shown in Lstg. 2.

Listing 2: Previous gearbox implementation

```
model IdealGear
  import Modelica.Mechanics.Rotational;

  Rotational.Interfaces.Flange_a flange_a;
  Rotational.Interfaces.Flange_b flange_b;

  parameter Real ratio=1;
equation
  flange_a.phi = ratio*flange_b.phi;
  0 = ratio*flange_a.tau + flange_b.tau;
end IdealGear;
```

Since this change has to be carried out for a lot of models, it is advisable to create a superclass including common components and equations, cf. Lstg. 3.

Common equations are the torque balance and the computation of the relative angles with respect to the bearing flange.

If the bearing flange is not connected, i.e. `cardinality(bearing) == 0`, the Modelica default connection rule defines `bearing.tau = 0` and the additional equation supplied in the corresponding if-section sets the bearing angle to zero, i.e., the bearing does not move.

Otherwise, the support torque is identical to the torque of the bearing flange and the bearing angle

Listing 3: Superclass including bearing torque

```

partial model TwoFlangesAndBearing
  import Modelica.Mechanics.Rotational;

  Rotational.Interfaces.Flange_a flange_a;
  Rotational.Interfaces.Flange_b flange_b;
  Rotational.Interfaces.Flange_a bearing;

  Modelica.SIunits.Torque tau_support;
  Modelica.SIunits.Angle phi_a;
  Modelica.SIunits.Angle phi_b;

equation
  0 = flange_a.tau + flange_b.tau + tau_support;

  phi_a = flange_a.phi - bearing.phi;
  phi_b = flange_b.phi - bearing.phi;

  if cardinality(bearing) == 0 then
    bearing.phi = 0;
  else
    bearing.tau = tau_support;
  end if;
end TwoFlangesAndBearing;

```

`bearing.phi` is defined from the component connected to the bearing.

Using this superclass and inheriting from it, it is a straightforward procedure to adapt the components. The expression `flange_a.phi` has to be changed simply to `phi_a` and `flange_b.phi` to `phi_b`, which leads to the implementation given in Lstg. 4. The icon shown in Fig. 1(b) indicates the function of the bearing flange.

Listing 4: New gearbox implementation

```

model IdealGear
  extends TwoFlangesAndBearing;

  parameter Real ratio=1;
equation
  phi_a = ratio*phi_b;
  0 = ratio*flange_a.tau + flange_b.tau;
end IdealGear;

```

For components which had up to now only one flange (Torque, Move etc.), the procedure is similar.

The described changes have been performed directly to the components of the Modelica Standard Library and the PowerTrain Library, since these changes are backward compatible, i.e., existing user models are not affected by this addition.

3 Hybrid 1D/3D Components

The 3D mechanical effects induced by a powertrain on its carrier body are in fact additional torques, see Sec. 1. In order to produce these torques on the car-

rier body, some components are needed with both 1D (flanges) and 3D (frames) connectors. They are described in this section.

3.1 3D Connector “Frame”

The definition of the needed 3D connector of the MultiBody library is shortly sketched. For more details, see [2]. The variables of the Frame connector are displayed in Fig. 2.

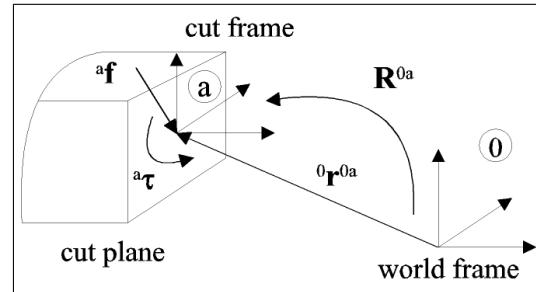


Figure 2: Variables of 3D connector Frame

A coordinate system frame a is rigidly fixed at an attachment point of a 3D mechanical part. This frame is described with respect to the world frame (i.e. an inertial coordinate system) by the

- position vector ${}^0\mathbf{r}^{0a}$ that is directed from the origin of the world frame to the origin of frame a and is resolved in the world frame and by the
- orientation object \mathbf{R}^{0a} describing the relative orientation between the world frame and frame a .

To ease usage, the MultiBody library is designed such that knowledge about the actual description form of orientation is not necessary. This is achieved by providing a pre-defined type `MultiBody.Frames.Orientation` and utility functions in `MultiBody.Frames` operating on instances of this type. In the sequel, only the utility function `Frames.AngularVelocity2` is needed to compute the angular velocity of the frame, resolved in the local coordinate system attached to the part.

It is assumed that a cut is performed between mechanical parts that shall be connected together at frame a . In the cut plane a resultant cut force ${}^a\mathbf{f}$ and a resultant cut torque ${}^a\tau$ act on frame a . Both vectors are resolved in this frame.

The four previously defined variables are used in connector `Frame`, see Lstg. 5. The additional connectors `Frame_a` and `Frame_b` have the identical definition as connector `Frame`. The only difference is that

Listing 5: MultiBody connector Frame

```

connector Frame
  import SI = Modelica.SIunits;
  SI.Position[3] r_0; // =  ${}^0\mathbf{r}^{0a}$ 
  MultiBody.Frames.Orientation R; // =  $\mathbf{R}^{0a}$ 
  flow SI.Force [3] f; // =  ${}^a\mathbf{f}$ 
  flow SI.Torque[3] t; // =  ${}^a\tau$ 
end Frame;

connector Frame_a = Frame;
connector Frame_b = Frame;

```

Listing 6: Implementation of Mounting1D

```

model Mounting1D
  import Modelica.Mechanics.Rotational;
  Rotational.Interfaces.Flange_b flange_b;

  parameter Modelica.SIunits.Angle phi0 = 0;
  parameter Real[3] n={1,0,0};
equation
  flange_b.phi = phi0;
  frame_a.f = zeros(3);
  frame_a.t = -n*flange_b.tau;
end Mounting1D;

```

Frame_a and Frame_b have different icons in order to be able to distinguish Frame connectors more easily in a composition diagram. The cut force and cut torque are flow variables in order that the force and torque balance at a point where several components are connected together is fulfilled. Note, that two connected frames (a and b) coincide, since $a.r_0 = b.r_0$ and $a.R = b.R$ due to the connection rules of Modelica.

3.2 New Component “Mounting1D”

In order to acquire support torques from the powertrain and to propagate them to the carrier body, a new component called Mounting1D is used, see Fig. 3. It has the equations of the Rotational.Fixed component and in addition a 3D frame connector for the mounting on a multi-body component, as well as a parameter vector n that defines the direction of the axis of rotation of the 1D flange connector. At the same time, n defines the direction of the support torque.

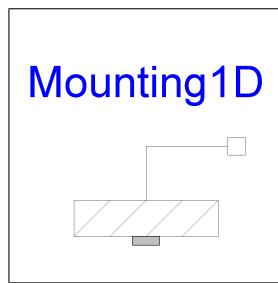


Figure 3: Component Mounting1D for propagating support torques to carrier body

This component transforms the 1D bearing torque into 3D space, see Lstg. 6, and enables 3D movement of all 1D elements connected to it.

All components of a powertrain that are connected to a common Mounting1D element need to have the same axis of rotation along parameter vector n. This means that, e.g., bevel gears where the axis of

rotation of flange_a and flange_b are different cannot be described properly by connecting to the Mounting1D component. It is discussed later, how to treat this case.

3.3 New Component “Rotor1D”

Powertrain parts rotating relative to their carrier body exert gyroscopic torques on this body, if the carrier is rotating. This effect can be mathematically described with a so-called gyrostat as illustrated in Fig. 4. It consists of a carrier and a body with rotational symmetry, called *rotor*, that is mounted on the carrier by rigid bearings.

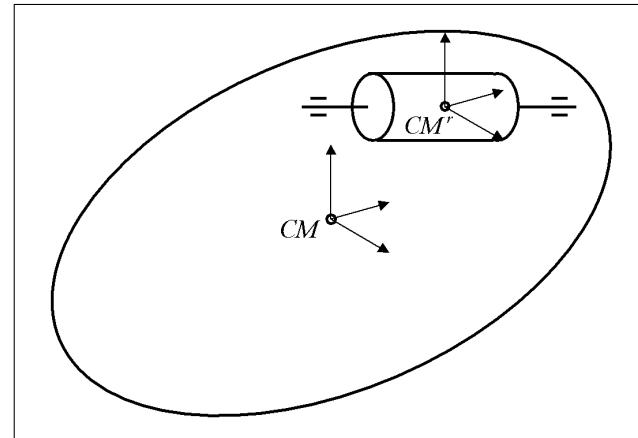


Figure 4: Gyrostat consisting of carrier and symmetric rotor

Two coordinate systems are present: The carrier frame is fixed in the carrier at the common center of mass CM of the total system. The rotor frame is also fixed in the carrier but at the center of mass CM' of the rotor. The rotor frame is parallel to the carrier frame.

According to, e.g., [3, 4], the equations of motion of this combined system are described by

$$\mathbf{J}\dot{\omega} + \mathbf{J}'\dot{\omega}_{\text{rel}} + \boldsymbol{\omega} \times (\mathbf{J}\boldsymbol{\omega} + \mathbf{J}'\boldsymbol{\omega}_{\text{rel}}) = \boldsymbol{\tau} \quad (1)$$

with the absolute angular velocity of the carrier, ω , the angular velocity of the rotor relative to the carrier, ω_{rel} , the inertia tensor of the total system with respect to its common center of mass CM , \mathbf{J} , the inertia tensor of the rotor with respect to the center of mass of the rotor CM^r , \mathbf{J}^r , and the external torque with respect to CM , τ . All vectors and tensors are resolved in the carrier frame. Reordering of terms yields

$$\underbrace{\mathbf{J}\dot{\omega} + \omega \times \mathbf{J}\omega}_{\tau_{\text{body}}} + \underbrace{\mathbf{J}^r\dot{\omega}_{\text{rel}} + \omega \times \mathbf{J}^r\omega_{\text{rel}}}_{\tau_{\text{gyro}}} = \tau. \quad (2)$$

By comparison with the equations of motion for a rigid body, it can be seen, that the term τ_{body} represents the contribution of the total system with the rotor fixed on the carrier and τ_{gyro} represents the additional contribution caused by the rotation of the rotor relatively to the carrier. Since the properties of the total system with the non-moving rotor are modelled completely by the carrier body, see Sec. 1, τ_{gyro} has to be considered in the Rotor component for gyroscopic torques.

The property of the rotor axis of rotation coinciding with one of its principal axis of inertia yields the simplification

$$\mathbf{n}^T \mathbf{J}^r \dot{\omega}_{\text{rel}} + \omega \times \mathbf{n} \mathbf{J}^r \omega_{\text{rel}} = \tau_{\text{gyro}} \quad (3)$$

with \mathbf{n} being a unit vector in direction of the axis of rotation of the rotor, \mathbf{J}^r the moment of inertia around \mathbf{n} and ω_{rel} the absolute value of the relative angular velocity of the rotor with respect to the carrier.

Whereas (3) considers solely the rotational degrees of freedom of the total system, the rotor has an additional degree of freedom of its own, as an external torque $\tau^r(t)$ is exerted on it. In [3, 4], the respective equation of motion is derived as

$$\mathbf{n}^T (\mathbf{J}^r \dot{\omega} + \mathbf{J}^r \dot{\omega}_{\text{rel}}) = \mathbf{n}^T \tau^r. \quad (4)$$

In a similar way as above, this equation can be simplified to

$$\mathbf{J}^r \mathbf{n} \dot{\omega} + \mathbf{J}^r \dot{\omega}_{\text{rel}} = \tau^r(t). \quad (5)$$

The new component `Rotor1D` is constructed by using the `Modelica.Mechanics.Rotational.Inertia` model as a basis, attaching a 3D Frame connector and adding the equations from (3) and (5). The Modelica implementation is shown in Lstg. 7.

The parameter `NeglectCoupling` was introduced in order to optionally neglect the term $\mathbf{J}^r(\mathbf{n}^T \dot{\omega})$, which corresponds to $\mathbf{J}^r \mathbf{n} \dot{\omega}$ in (5). This term is usually negligible if the powertrain accelerates much faster as the base body (this is, e.g., the case in

Listing 7: Implementation of Rotor1D

```

model Rotor1D
  import Modelica.Mechanics.Rotational;

  Rotational.Interfaces.Flange_a flange_a;
  Rotational.Interfaces.Flange_b flange_b;
  MultiBody.Interfaces.Frame_a frame_a;

  parameter Modelica.SIunits.Inertia J = 1;
  parameter Real n[3] = {1,0,0};
  parameter Boolean NeglectCoupling = false;

  Modelica.SIunits.Angle phi;
  Modelica.SIunits.AngularVelocity w_a[3];
  Modelica.SIunits.AngularAcceleration z_a[3];
  Modelica.SIunits.AngularVelocity w;
  Modelica.SIunits.AngularAcceleration a;

equation
  flange_a.phi = phi;
  flange_b.phi = phi;

  w = der(phi);
  a = der(w);
  J*a = flange_a.tau + flange_b.tau -
    (if NeglectCoupling then 0 else J*(n*z_a));

  w_a = MultiBody.Frames.angularVelocity
    (frame_a.R, der(frame_a.R));
  z_a = der(w_a);

  frame_a.f = zeros(3);
  frame_a.t = n*(J*a) + cross(w_a, n*(J*w));
end Rotor1D;

```

vehicle powertrains). The essential advantage is that an algebraic loop is removed since then there is only an action on acceleration level from the powertrain to the base body but not vice versa.

3.4 New Component “BevelGear1D”

A new component is needed for 1D modelling of gearboxes with non-parallel axes, see Fig. 5. In general, the axes of rotation of `flange_a` and `flange_b` and the direction of the support torque vector are different in this case. Therefore, it is necessary to attach the 3D connector directly to this component.

In accordance to Fig. 6, the bevel gear is characterized by

$$i = \frac{\omega_{\text{in}}}{\omega_{\text{out}}} = -\frac{\tau_{\text{out}}}{\tau_{\text{in}}} \quad (6)$$

with the gear speed ratio i , shaft angular velocities ω and shaft torques τ [5]. As illustrated in Fig. 6, the indices refer to the input and output shaft respectively.

With \mathbf{n}_{in} , \mathbf{n}_{out} vectors in direction of the input and output shaft, respectively, a 3D torque balance results in

$$\mathbf{0} = \tau_{\text{in}} \frac{\mathbf{n}_{\text{in}}}{|\mathbf{n}_{\text{in}}|} + \tau_{\text{out}} \frac{\mathbf{n}_{\text{out}}}{|\mathbf{n}_{\text{out}}|} + \tau_{\text{support}}. \quad (7)$$

The implementation is shown in Lstg. 8.

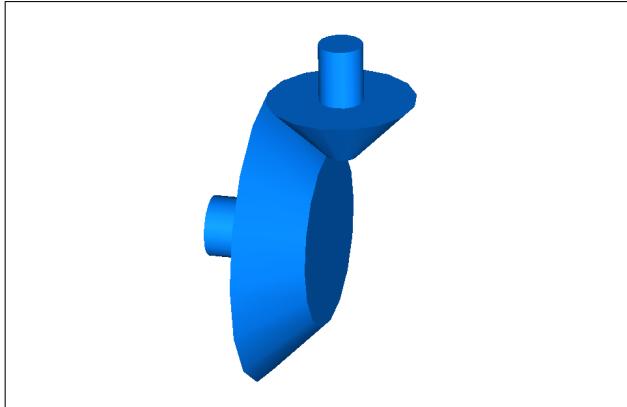


Figure 5: Gearbox with non-parallal axes

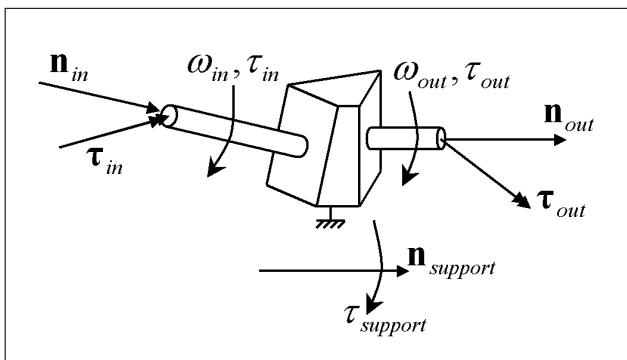


Figure 6: Quantities of a bevel gear

Listing 8: Implementation of BevelGear1D

```

model BevelGear1D
  import Modelica.Mechanics.Rotational;

  parameter Real ratio=1;
  parameter Real n_a[3]={1,0,0};
  parameter Real n_b[3]={1,0,0};
protected
  parameter Real e_a[3]=n_a/sqrt(n_a*n_a);
  parameter Real e_b[3]=n_b/sqrt(n_b*n_b);
public
  Rotational.Interfaces.Flange_a flange_a;
  Rotational.Interfaces.Flange_b flange_b;
  MultiBody.Interfaces.Frame_a frame;
equation
  flange_a.phi = ratio*flange_b.phi;
  0 = ratio*flange_a.tau + flange_b.tau;
  frame.f = zeros(3);
  frame.t = -flange_a.tau*e_a - flange_b.tau*e_b;
end BevelGear1D;

```

given as Modelica text taking into account

$$\tau_{in} = \tau_{in} \frac{\mathbf{n}_{in}}{|\mathbf{n}_{in}|}, \quad \tau_{out} = \tau_{out} \frac{\mathbf{n}_{out}}{|\mathbf{n}_{out}|} \quad (8)$$

with the corresponding frame cut torque τ and axis of rotation \mathbf{n} as shown in Fig. 6. The constraints (6) implicitly define the torques to be applied at the two revolute joints.

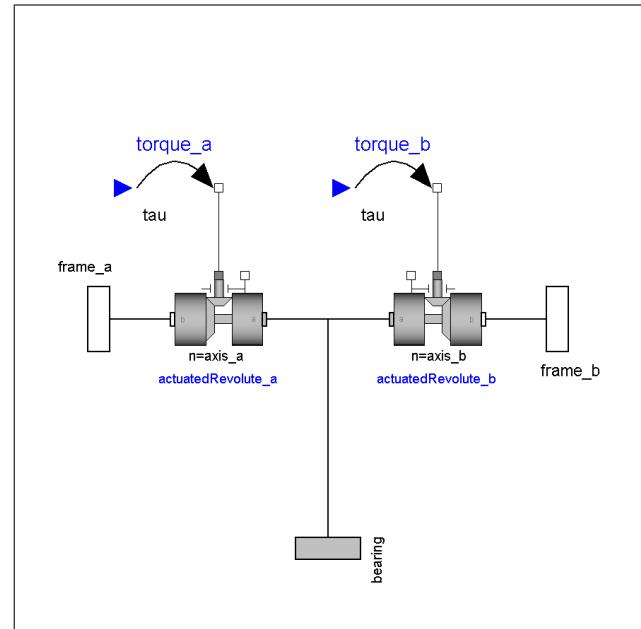


Figure 7: 3D gear constraint without 1D rotational components

3.5 Animation

Most components of the PowerTrain library are animated. All these components have been equipped with a 3D Frame connector to provide the information needed for deducing the 3D position and orientation of the animation shapes. The possibility to switch off animation and to completely remove the corresponding code was conserved.

4 New Component “GearConstraint”

In order to be able to model gearboxes completely in 3D, a component to provide the gear constraint (6) for multi-body systems was introduced. By default, the component allows to model gearboxes with nonparallel shafts as well.

In a first step, see Fig. 7, the 3D gear constraint was implemented without using 1D rotational components in order to be not forced to take care for support torques. Therefore, the constraint equations have been

Since equivalent equations are provided by `Modelica.Mechanics.Rotational.IdealGear`, in a second step the 3D gear constraint was imple-

mented as shown in Fig. 8. No additional equations are necessary.

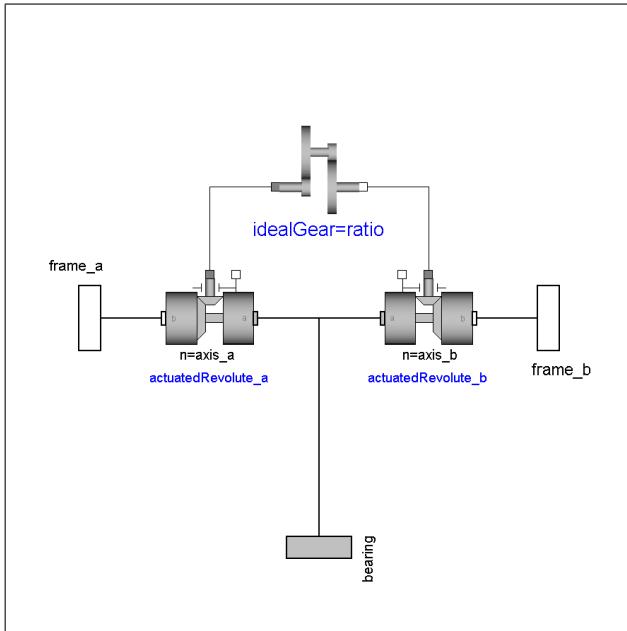


Figure 8: 3D gear constraint using model IdealGear

5 Validation Examples

The following examples illustrate the usage of the introduced components and are used in order to validate the design by comparison of 1D with 3D realizations.

5.1 Gyrostat

As mentioned above, a so-called gyrostat consists of a carrier body and a rotor, which is mounted on the carrier body. There are two possibilities of modelling such a system, especially the rotor. One possibility is the usage of a rotating 3D body. Alternatively, the rotor could be modelled by a non-rotating 3D body in combination with the Rotor component described in Sec. 3.3. It was intended to show the equivalence of both possibilities. Several simulations have been carried out with different joint combinations, axis directions and driven or free rotor. In the following, a selected setup is presented.

In the example shown in Fig. 9, the two rotors to be compared are mounted in both cases on a carrier cylinder of their own which is able to move in three rotational degrees of freedom, since the latter is mounted to the ground by a spherical joint.

At the start of the simulation, the carrier cylinders are in an elevated position and start moving due to

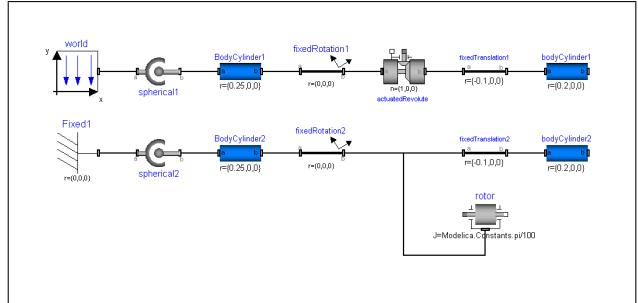


Figure 9: Modelica model of rotating body and non-rotating body with rotor

gravitation. The rotors are not actuated and have an initial angular velocity $\omega = 10 \text{ rad/s}$. All relevant signals, especially the mounting forces (Fig. 11) and the body orientations, are identical.

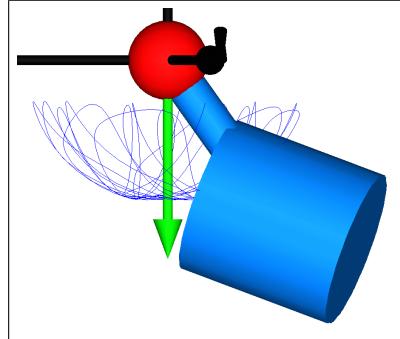


Figure 10: Animation of the two gyrostat systems

5.2 Body-mounted Actuator

Another example is shown in Fig. 12. It consists of a robot arm that is connected with a revolute joint to the base (i.e. world frame). On the (moving) robot arm a gearbox and a motor is present that drive the revolute joint. One wheel of the gearbox is rigidly attached to the axis of rotation of the revolute joint.

Again, two different Modelica models of this system are compared. One model was implemented using solely 3D components, cf. Fig. 13(a). The other model replaces some 3D components by 1D equivalences, cf. Fig. 13(b).

Both implementations yield identical results.

6 Application: Automatic Gearbox

The basic intention of modelling 3D mechanical effects of 1D powertrains was to be able to examine the

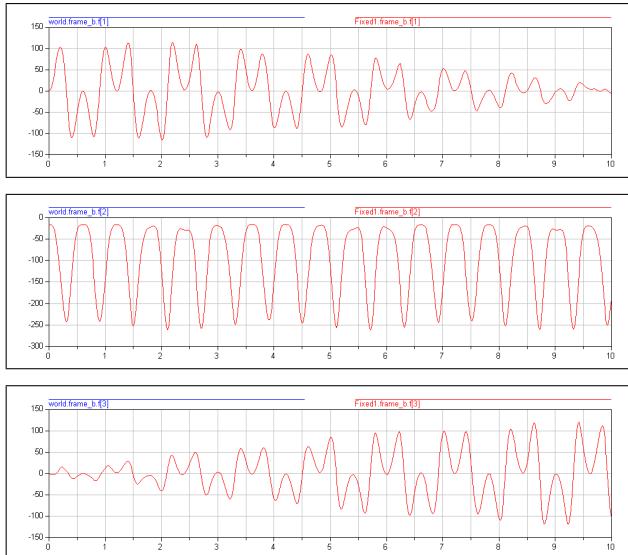


Figure 11: Mounting forces of the two gyrostat systems in x-, y-, z- direction in the world frame

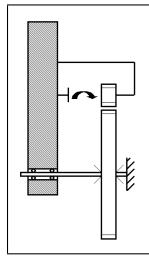


Figure 12: Sketch of body-mounted actuator

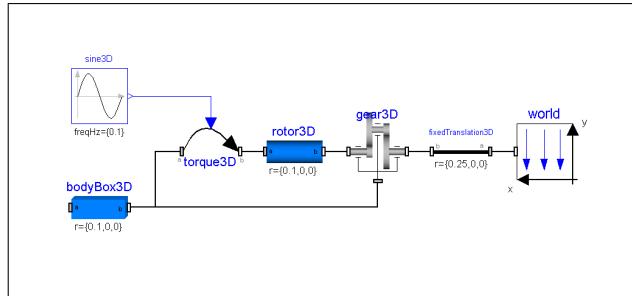
interaction of vehicle and powertrain dynamics. The example discussed in this section sketches the idea of merging both modelling areas.

A six-speed automatic gearbox based on a Lepelletier wheelset [6], Fig. 14, is considered. A corresponding model is available in the PowerTrain library.

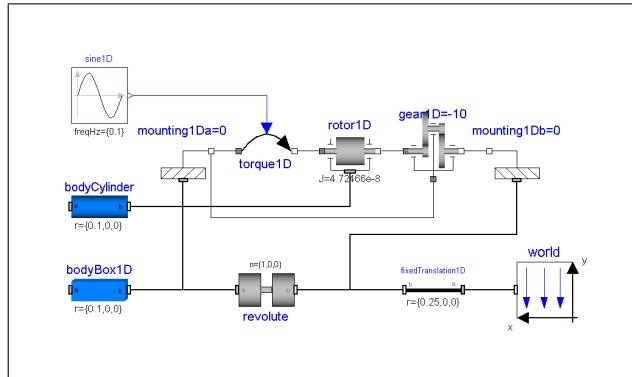
This model was changed such that the previous 1D rotational mechanical components have been replaced by these introduced above, cf. Fig. 15. As a consequence, the Lepelletier wheelset model was equipped with a MultiBody frame connector.

The obtained component is then used in the setup shown in Fig. 16. A carrier body is connected to the ground by a revolute joint and moved similar to a yawing vehicle. The yaw rate was increased up to 2 rad/s and then held constant. The automatic gearbox is mounted on the carrier body and connected with two rotor components at the input and output shafts, respectively. The drivetrain is accelerated by a torque of 10 Nm.

Figure 17(a) shows the angular velocities of the ro-



(a) Solely using 3D MultiBody components



(b) Using both 3D and 1D components

Figure 13: Modelica models of body-mounted actuator

tors at the input and output shaft. At a simulation time of 4 s, a gear shift is initiated from the third to the fourth gear, reducing the system's ability to accelerate. The influence exerted on the carrier body can be seen in Fig. 17(b).

In Fig. 18, an animation screenshot illustrates the assembly of the gearbox.

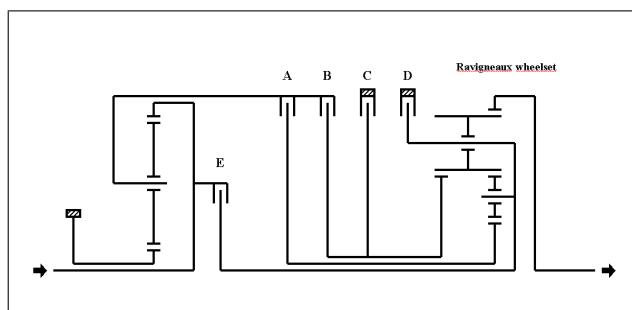


Figure 14: Schematic of Lepelletier wheelset

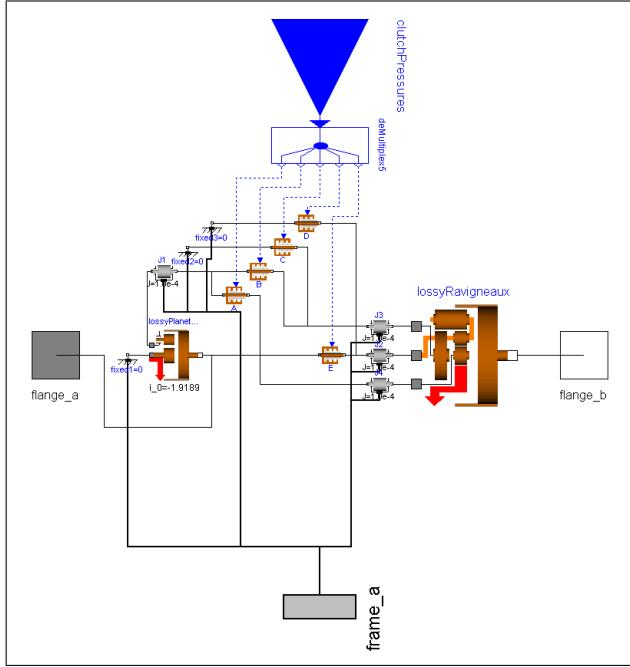
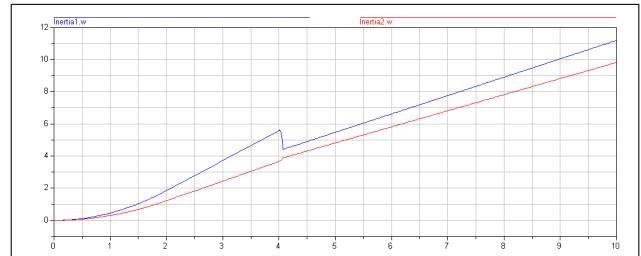
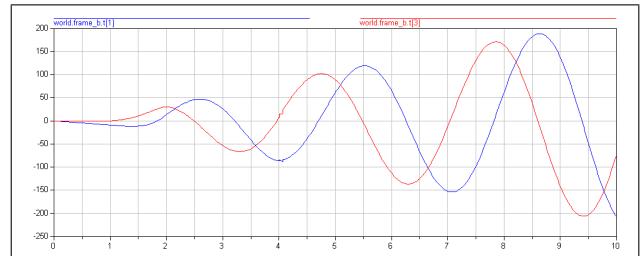


Figure 15: Modelica model of Lepelletier wheelset



(a) Angular velocities of rotors



(b) Support torques of carrier body in x- and z-direction of world frame

Figure 17: Simulation results for automatic gearbox on yawing carrier

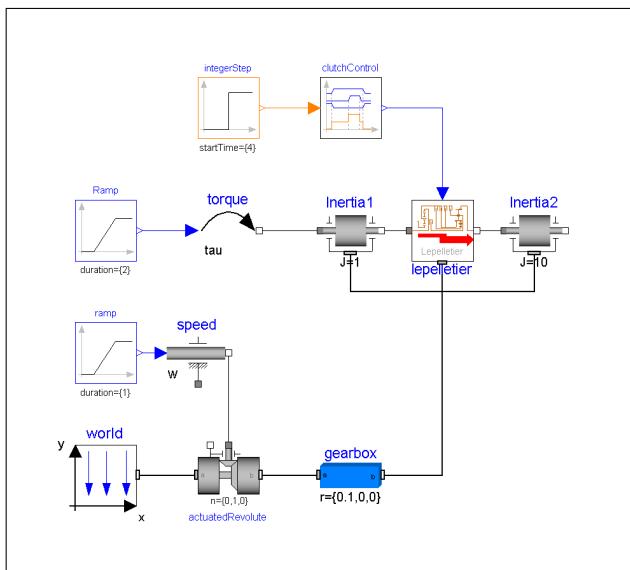


Figure 16: Test setup similar to yawing vehicle

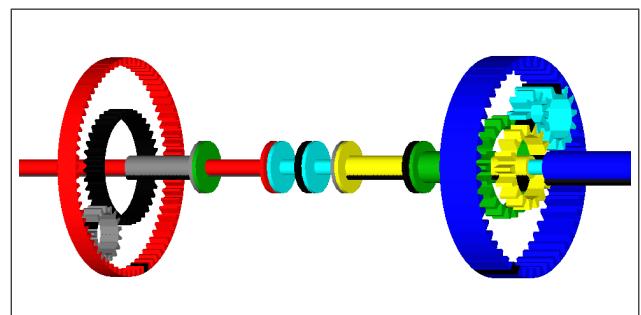


Figure 18: Animation screenshot for Lepelletier wheelset

7 Conclusions and Outlook

All the details have been described how the one-dimensional rotational Modelica libraries have been extended in order that drive trains modelled with these libraries can be mounted on parts of the MultiBody library moving in three dimensions. All dynamic effects, such as support and gyroscopic torques, are taken into account.

This approach has several advantages:

- Only the mass, the center of mass and the inertia tensor of a complete powertrain, such as an automatic gearbox, has to be provided by the modeller together with the rotational inertias along the axes of rotations. For a conventional modelling with multi-body components, data for mass, center of mass and inertia tensor is required from every single piece of a powertrain.
- The powertrain can be modelled and tested first as a pure one-dimensional system.
- Mounting a powertrain on a three-dimensionally moving system just requires to connect the three-dimensional Frame connectors of the powertrain components to appropriate mounting objects (Mounting1D) that are fixed on the multi-body parts.
- A hybrid 1D/3D model does not have problems with possible overconstraining that is a major problem for a 3D model of a powertrain.

Future work will include integration of powertrains into vehicle dynamics models [7] and examination of the interaction between vehicle and powertrain dynamics.

Acknowledgements

For fruitful discussions, the authors would like to thank Bill Tobler and Michael Tiller from Ford Motor Company. This work was in parts supported by *Bayerisches Staatsministerium für Wirtschaft, Verkehr und Technologie* under contract AZ300-3245.2-3/01 for the project *Test und Optimierung elektronischer Fahrzeug-Steuengeräte mit Hardware-in-the-Loop-Simulation*.

References

- [1] German Aerospace Center (DLR), Oberpfaffenhofen, *PowerTrain Library 1.0 – Tutorial*, Dezember 2002. <http://www.dynasim.se/www/PowerTrainTutorial.pdf>.
- [2] M. Otter, H. Elmquist, and S. E. Mattsson, “The New Modelica MultiBody Library,” in *Proceedings of the 3rd International Modelica Conference* (P. Fritzson, ed.), (Linköping), The Modelica Association and Linköping University, November 2003.
- [3] J. Wittenburg, *Dynamics of Systems of Rigid Bodies*, vol. 33 of *Leitfäden der angewandten Mathematik und Mechanik*. B. G. Teubner Verlag, 1977.
- [4] M. Otter, *Objektorientierte Modellierung mechatronischer Systeme am Beispiel geregelter Roboter*. Dissertation, Ruhr-Universität Bochum, 1994.
- [5] J. Loomann, *Zahnradgetriebe*. Springer, 3rd ed., 1996.
- [6] H. Dach, W.-D. Gruhle, and P. Köpf, *Pkw-Automatgetriebe*, vol. 88 of *Die Bibliothek der Technik*. Landsberg/Lech: Verlag Moderne Industrie, 2nd, revised ed., 2001.
- [7] J. Andreasson, “VehicleDynamics Library,” in *Proceedings of the 3rd International Modelica Conference* (P. Fritzson, ed.), (Linköping), The Modelica Association and Linköping University, November 2003.

Session 4B

Electrical and Chemical Systems

SPICELib - Modeling & Analysis of Electric Circuits with Modelica

Carla Martin, Alfonso Urquia and Sebastian Dormido

Department of Computer Science and Automatic Control, UNED, Madrid (Spain)
 {carla,aurquia,sdormido}@dia.uned.es

Abstract

SPICELib is a set of model libraries, written in Modelica language, that supports some of the modeling and analysis capabilities of the circuit simulator PSpice. An upgraded version of *SPICELib* will be released in the near future: version 1.1. The Modelica code and the documentation of *SPICELib* 1.1 will be freely available. The purpose of this contribution is to discuss the capabilities supported by *SPICELib* version 1.1, and to illustrate its use by means of two examples.

1 Introduction

SPICELib is a set of model libraries developed at the Computer Science and Automatic Control Department of UNED for academic purposes. Programming of *SPICELib* version 1.0 was completed in January, 2002. It supported OP, AC sweep and TRAN analyses for the following analog device types: linear resistor and capacitor, independent voltage and current sources and NMOSFET (LEVEL1 model).

SPICE models "translation" into Modelica is performed as faithfully as possible. *SPICELib* device models are the PSpice models described in [6][4]. In addition, *SPICELib* names for device parameters and part types are equal to the PSpice ones [6], so that people used to PSpice models are not required to make an additional model-understanding effort. The procedure used to validate *SPICELib* is by comparing the analysis results obtained with *SPICELib* with the results obtained using OrCAD PSpice version 9.1 [6].

Two procedures were supported by *SPICELib* 1.0 for setting the initial conditions [7]: IC1 and IC2 pseudo-components, and the capacitor IC property. IC1 is a one-pin symbol that allows setting the initial

voltage on a circuit node. IC2 is a two-pin symbol that allows setting the initial voltage between two nodes. Capacitor IC property allows setting the initial voltage-drop across the capacitor.

Bias point calculation is the most problematic analysis from the numerical standpoint. PSpice [6] first tries to solve the static model of the circuit using the Newton-Raphson algorithm ("static model iteration" algorithm, abbreviated: SMI). If a solution is not found, and "GMIN stepping" is enabled, the GMIN algorithm is applied. If it also fails or it is not enabled, then "static model ramping" algorithm (abbreviated: SMR) is applied. *SPICELib* version 1.0 implemented four algorithms for solving the circuit static model [7]: the three algorithms supported by PSpice [6][3][4] and, in addition, the algorithm "dynamic model ramping" (abbreviated: DMR), proposed in [1]. *SPICELib* allows the user to choose among these four algorithms in order to [7] perform the OP analysis, calculate the operating point prior to the AC sweep analysis, and evaluate the steady-state initial condition of the transient analysis.

SPICELib 1.0 implemented the two transient-analysis initialization methods supported by PSpice [6][7]: with and without bias point calculation. If bias point calculation is not skipped, prior to performing the transient analysis, the bias point of the circuit is computed, with the independent sources set to their respective transient-analysis starting values. The calculated steady-state is the initial condition for the transient analysis. On the other hand, if the bias point calculation is skipped, the bias conditions are fully determined by the IC specifications for capacitors.

An upgraded version of *SPICELib* will be released in the near future: version 1.1. The purpose of this contribution is to discuss the capabilities supported by *SPICELib* version 1.1, and to illustrate its use by means of two examples.

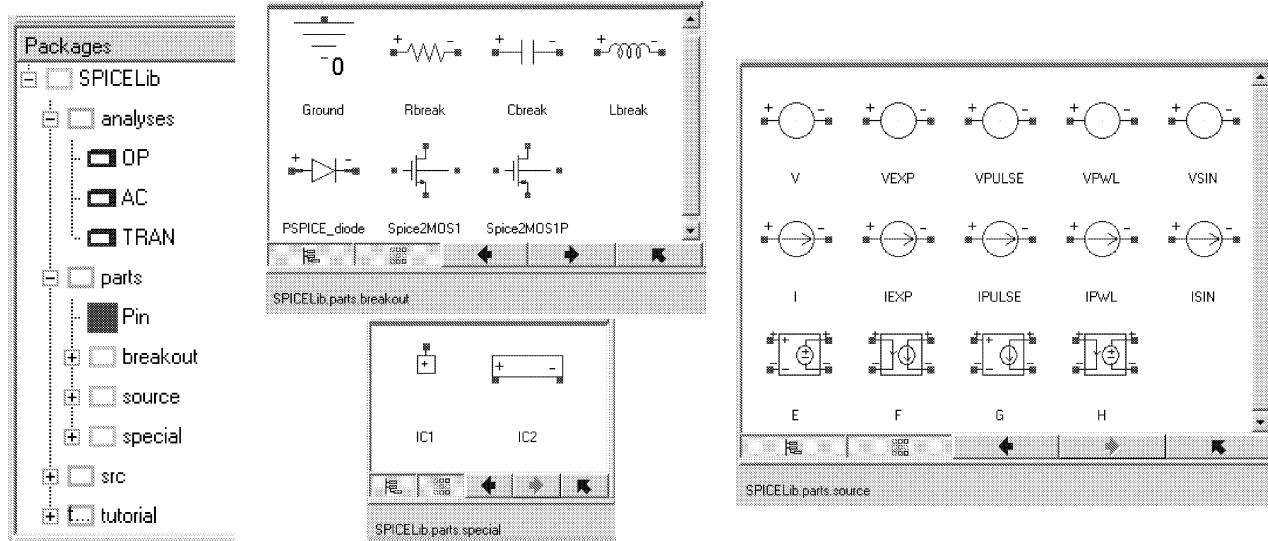


Figure 1: *SPICELib* 1.1 libraries. Device models in *SPICELib.parts* library.

2 New features in *SPICELib* 1.1

The model libraries have been reorganized, in order to facilitate their use and maintenance (see Fig. 1). A clear separation is made between those libraries to be used by *SPICELib* users and those libraries to be used only by *SPICELib* designers.

- *SPICELib* models describing the structure and behavior of parts and analyses are gathered in the *src* library. This library is intended to be used and modified only by *SPICELib* designers. Its documentation is oriented to the designers, explaining implementation details without interest to *SPICELib* users.
- Two libraries are defined, *parts* and *analyses*, in order to gather the "final" models that *SPICELib* users need to compose or analyze their circuits. These two libraries are a "mirror" of a reduced set of *src* library models, inheriting the structure and the behavior, and adding the class documentation oriented to the library user.
- *tutorial* library has been added (see Fig. 1). It contains some examples discussing how to use the *parts*-library models to compose the circuit schematic, and how to use the *analyses*-library models to analyze this previously defined circuit.

SPICELib capacitor and NMOS models [7] have been modified, in order to fix some numerical problems and improve the simulation speed of the transient analysis. This point will be discussed in Sections 3.2 and 3.4. In addition, the following device models have been

included in the new version of the library (see Fig. 1):

- Linear inductor: *Lbreak* model. It supports the inductor IC property [6]: current through the inductor during the bias point calculation can be set.
- Controlled linear sources (E, F, G, H) [6].
- PSpice diode [4] and SPICE2 LEVEL1 p-channel MOSFET [4].

SPICELib 1.0 analysis models have not been modified. All the discussions in [7] concerning analyses are valid for *SPICELib* 1.1, and therefore they will not be reproduced in this manuscript.

3 Devices models

SPICELib device models are composed of three different device descriptions: static, AC small-signal and large-signal. Each of these three descriptions has its own set of equations and variables, and its own contribution to the device-model interface. As a consequence, a circuit model built using *SPICELib* library contains the static, the large-signal and the AC small-signal descriptions of the circuit [7].

SPICELib analysis models describe the sequence of control-signal value transitions required to perform the analyses. Control-signals are *inner* variables [5] of the analysis models and *outer* variables of the device models. Analysis models inherit the circuit model, which is composed of device models. *SPICELib* control-signals trigger instantaneous changes in the

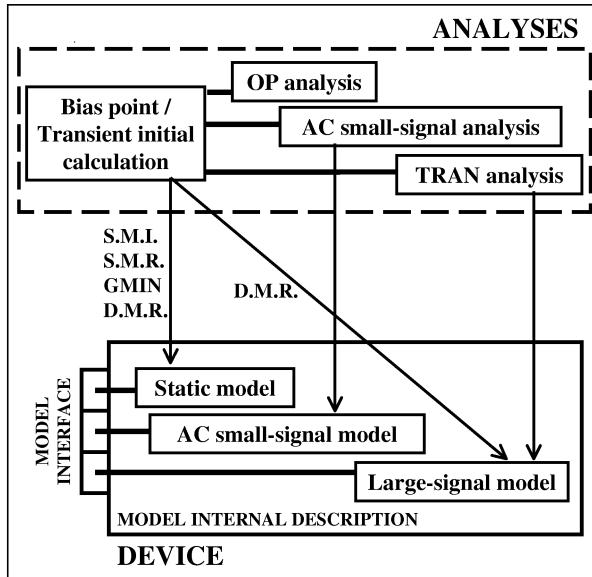


Figure 2: Structure of *SPICELib* device descriptions and analyses.

device model state-variables and in the mathematical structure of the device descriptions. These structural changes are carried out by opening and closing the switches internal to the device descriptions [7]. These changes facilitate setting the analysis initial conditions. Also, they facilitate the exchange of information among the device descriptions, required when an analysis use several device descriptions (see Fig. 2) [7]:

- *SPICELib* performs an OP analysis before the AC sweep analysis in initiated, in order to calculate the bias-point. Then, *SPICELib* calculates the AC model of the circuit by substituting, in the small-signal (i.e., linearized) model of each component, the bias-point current and voltage values.
- If *SKIPBP* parameter is set to false, then *SPICELib* carries out an OP analysis prior to the transient analysis, and the calculated bias-point is the initial condition for the transient analysis.
- "Dynamic model ramping" algorithm requires the combined use of the static and large-signal device descriptions.

In addition, control-signals "disable" the model descriptions once they are no more useful during the analysis [7]. As a consequence, the simulation computational effort is not unnecessarily increased.

Relevant aspects of the new models supported by *SPICELib* 1.1, and the modifications made on the

capacitor and NMOS models, are discussed next. In particular, the large-signal and the static descriptions of the controlled sources, capacitor, inductor, PN-junction diode and MOS transistor will be discussed. The reader interested in all the mathematical details of the descriptions is referred to the documentation included in the *SPICELib.parts* library.

3.1 Controlled linear sources

SpiceLib 1.1 supports the following four types of controlled linear sources (see Fig. 1):

- Voltage-controlled voltage source (E).
- Current-controlled current source (F).
- Voltage-controlled current source (G).
- Current-controlled voltage source (H).

SPICELib controlled sources are devices with four pins: two controlling pins, p1 (+) and n1 (-), placed at the icon left side; and two output pins, p2 (+) and n2 (-), placed at the icon right side. In the three model formulations (i.e., static, AC small-signal and large signal [7]), the source output current or voltage is proportional to its controlling current or voltage. The proportionality constant is a model parameter: *Gain*.

3.2 Capacitor and inductor

SPICELib 1.1 inductor and capacitor models are analogous. The partial models *Capacitor* and *Inductor* are defined in the library *SPICELib.src.BREAKOUT*. They describe the static, large-signal and AC small-signal behavior of capacitors and inductors respectively, except their large-signal and AC small-signal capacitance and inductance. The linear capacitor (*Cbreak*) and the capacitors internal to the PN-junction diode and MOS transistor models are sub-classes of *Capacitor*. The linear inductor (*Lbreak*) is a sub-class of *Inductor*.

SPICELib 1.0 linear capacitor model (*Cbreak*) has been modified in the new version of the library: an small-value resistor has been included (by-default value: $R_EPS2 = 2 \cdot 10^{-4}\text{ohm}$). See Fig. 3, in comparison with Figs. 3 and 4 in reference [7]. Analogously, *R_BIG2* resistor has been included in the linear inductor model, *Lbreak* (see Fig. 3).

The reason behind this model modification is to avoid that the circuit model index is greater than one. Models with an index greater than one,

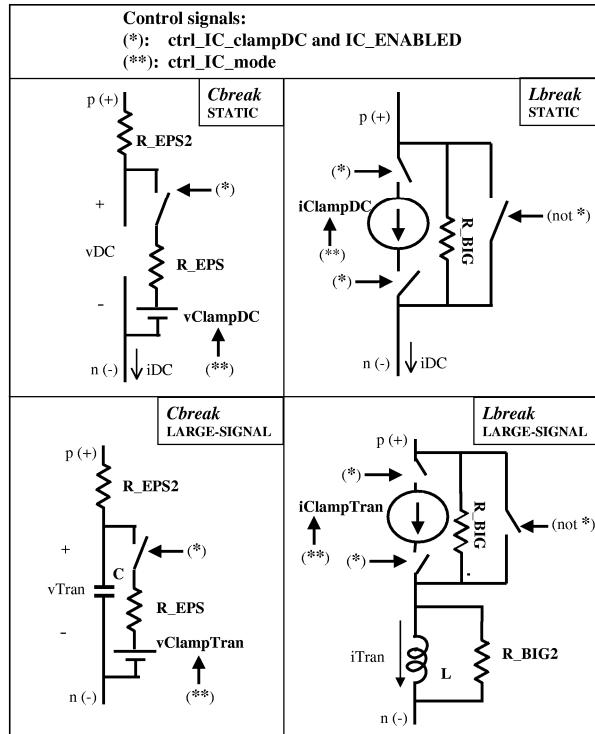


Figure 3: Static and large-signal descriptions of *Cbreak* and *Lbreak*.

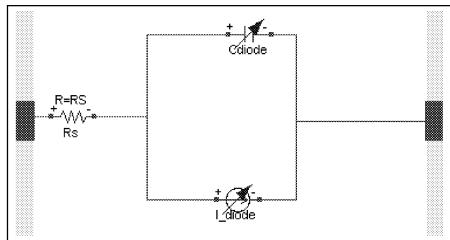


Figure 4: Diagram of SPICELib diode.

due to the parallel connection of *Capacitor* subclasses, must be avoided, because index reduction involves differentiating discontinuous equations. In that situation, Dymola [2] shows an error message during the simulation run when the differentiated-function discontinuity is detected.

The static and large signal descriptions of *Cbreak* and *Lbreak* devices are shown in Fig. 3. The purpose of the control signals *ctrl_IC_clampDC* and *ctrl_IC_mode*, and the parameter *IC_ENABLED* is discussed in [7]. Applying an IC property for a capacitor has the same effect as applying one of the pseudocomponents *IC2* across its nodes [6]. *SPICELib* attaches (closing the switch) a voltage source (*vClampDC* and *vClampTran*) with a 0.0002

ohm series resistance (*R_EPS*) in parallel with the capacitor. In the case of the initial current through the *Lbreak* inductor, the internal implementation is analogous to the capacitor *Cbreak*. *SPICELib* attaches a current source (*iClampDC* and *iClampTran*) with a 1 Gohm parallel resistance (*R_BIG*) in series with the inductor (see Fig. 3).

3.3 PN-junction diode

The PN-junction diode model in *SPICELib* 1.1 is the PSpice model. This is based on SPICE2 diode model, with new features for modeling the recombination and high-level effects [4]. *SPICELib* diode model, *PSPICE_diode*, is composed of the following devices (see Fig. 4): a linear resistor (of *Rbreak* class [7]), a voltage-dependent capacitor, and a voltage-controlled, non-linear current source.

3.3.1 Voltage-dependent capacitor

It extends the partial model *Capacitor*, defining the large-signal and the AC small-signal capacitance (as described in [4]). The mathematical relationship between the large-signal capacitance of the diode, *C*, and the large-signal voltage drop across the capacitor, *v*, is a two-branches function, that can be conveniently described using the Modelica expression *if-then-else*:

$$C = \begin{cases} C_d(v) + C_j(0) \cdot \left(1 - \frac{v}{\phi_0}\right)^{-m} & \text{for } v < FC \cdot \phi_0 \\ C_d(v) + \frac{C_j(0)}{F_2} \cdot \left(F_3 + \frac{m \cdot v}{\phi_0}\right) & \text{for } v \geq FC \cdot \phi_0 \end{cases} \quad (1)$$

where $C_d(v)$ is a continuous, highly non-linear function of *v* (sum of several exponential terms); and $C_j(0)$, ϕ_0 , m , FC , F_2 and F_3 are model parameters. By-default value for parameters *m* and *FC* is 0.5; and by-default value for ϕ_0 is 1 volt [4]. In order to illustrate the shape of the C-V characteristic described in Eq. (1), the C-V curve of the D1N4002 diode is represented in Fig. 5.

The following consideration is important from the numerical standpoint. Acceptable values for *FC* guarantee that if $v < FC \cdot \phi_0$, then the term $\left(1 - \frac{v}{\phi_0}\right)$ is greater than zero. However, numerical problems are experienced (i.e., square root of a negative number) unless the *if-then-else* condition is surrounded by *noEvent*, as explained in [2][5].

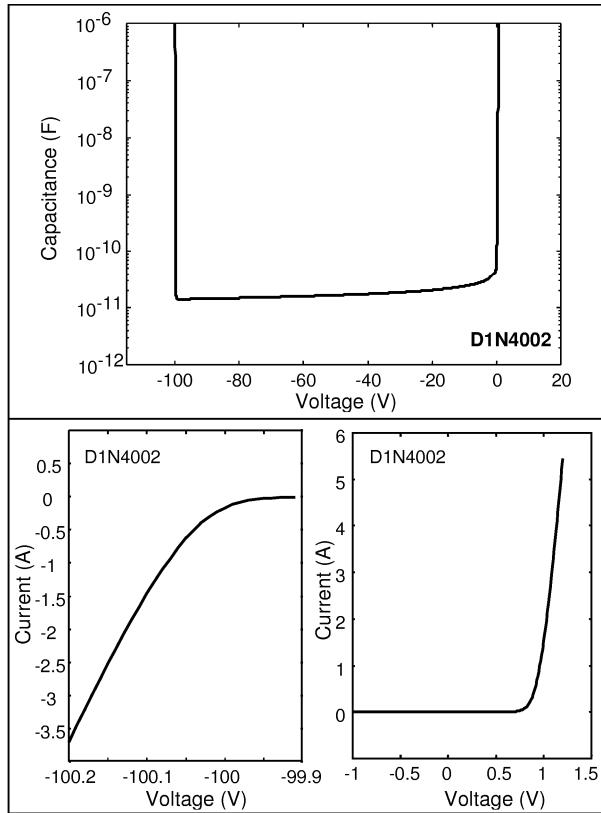


Figure 5: C-V and I-V characteristics of D1N4002.

3.3.2 Voltage-controlled current source

The static and large-signal constitutive relation of the source is (see I-V characteristic of diode D1N4002 in Fig. 5) [4]:

$$\begin{aligned} i_d &= K_{hli}(v) \cdot I_S \cdot \left(e^{\frac{q \cdot v}{n \cdot k \cdot T}} - 1 \right) \\ &+ K_{gen}(v) \cdot I_{SR} \cdot \left(e^{\frac{q \cdot v}{n_R \cdot k \cdot T}} - 1 \right) \\ &- IBV \cdot e^{-\frac{q \cdot (BV+v)}{k \cdot T}} \end{aligned} \quad (2)$$

$$K_{hli}(v) = \begin{cases} \sqrt{\frac{I_{KF}}{I_{KF} + I_S \cdot \left(e^{\frac{q \cdot v}{n \cdot k \cdot T}} - 1 \right)}} & \text{for } I_{KF} > 0 \\ 1 & \text{otherwise} \end{cases} \quad (3)$$

$$K_{gen}(v) = \sqrt{\left[\left(1 - \frac{v}{\phi_0} \right)^2 + 0.005 \right]^m} \quad (4)$$

Transient analysis. R_S resistor guarantees that the capacitor voltage can always be a state variable of the large-signal model (see Fig. 4). As a consequence, the large-signal constitutive relation of the current source is used, during the transient analysis, to calculate the current generated by the source.

The numerical efficiency of the SPICELib transient analysis has been significantly improved by including in the diode model an equation setting that the voltage drop across the capacitor (state variable) is equal to the voltage drop across the current source. The large-signal constitutive relation of the current source is formulated in terms of this voltage drop, directly calculated from a state variable, instead of in terms of the difference between the voltage at the source pins. Although mathematically these two modeling approaches are equivalent, they are not from the numerical standpoint. In the first case, the current generated by the source is directly calculated from a state variable. In the second case, it is necessary to calculate the voltage values at the two source pins in order to obtain the current generated by the source. This second approach commonly leads to non-linear algebraic loops.

DC analysis. As the static model of *Cdiode* capacitor is an open circuit, the computational causality of the source constitutive relation is not known in advance. In fact, numerical problems can be experienced when the voltage drop, v , needs to be calculated from Eq. (2). The current-source conductance $\frac{di_d}{dv}$ falls to zero in the reverse-bias region of operation (see Fig. 5): the current is constant and no longer a function of the diode voltage. Often, very small conductance values force the Newton-Raphson algorithm to significantly overshoot the correct solution voltage, and a large number of algorithm iterations are required to work back to the correct solution voltage. Worse yet, if the conductance value ever reaches zero, the next Newton iteration will cause a floating divide-by-zero error.

This problem is solved in SPICE by placing a shunt resistor in parallel with the PN-junction [3]. This resistor has a default conductance value of $GMIN = 10^{-12}$ mhos. The GMIN resistor is built into the source model, and always provides a small voltage-dependent current for the devices. The constitutive relation of the current source, with the GMIN resistor term included, is the following: $i = i_d + v \cdot GMIN$. Consequently, the conductance under reverse-bias conditions is $\frac{di}{dv} = 0 + GMIN$.

To help avoid nonconvergence problems, $GMIN$ should be set as large as possible without affecting the accuracy of the simulation output. The larger the value of $GMIN$, the faster the Newton-Raphson algorithm will converge on a solution. $GMIN$ is a global parameter for the entire SPICELib circuit, so when selecting the value of $GMIN$, the most current

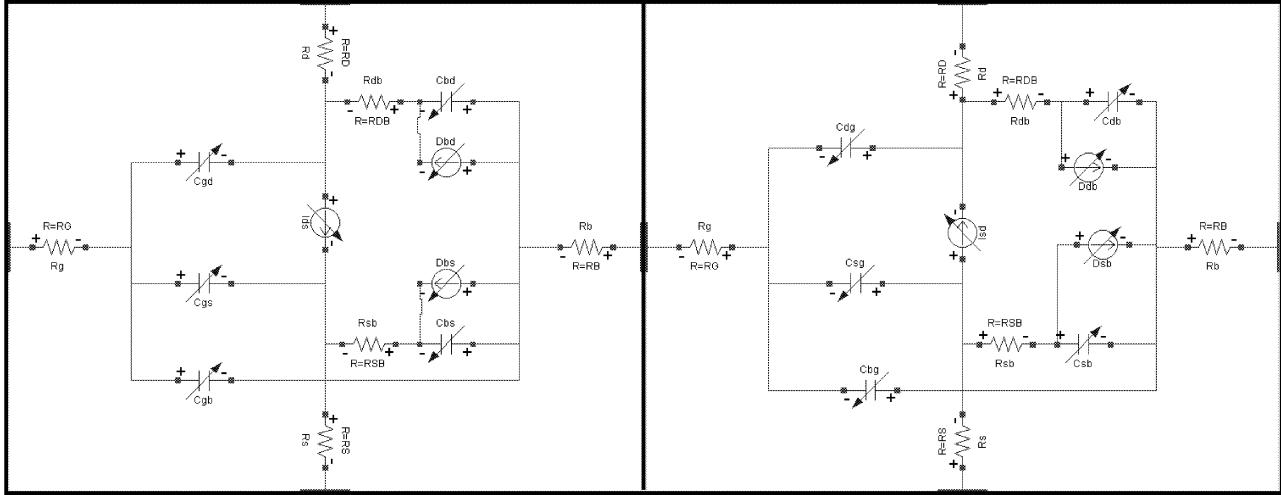


Figure 6: SPICElib NMOS (left) and PMOS (right) model diagrams.

sensitive parts of the circuit should be considered. The following recommendation is found in [3]: to set $GMIN$, determine the smallest parasitic resistance value (R_p) which could be placed across any two nodes without influencing the behavior of the circuit. Set $GMIN = \frac{1}{R_p}$.

"GMIN stepping" algorithm attempts to find a solution for the static model by starting with a large value of $GMIN$, initially 10^{10} times the nominal value. If a solution is found at this setting, $GMIN$ is reduced by a factor of 10 and a new solution is found. This continues until $GMIN$ is back to the nominal value, or the repeated cycle fails to converge [6][3]. SPICELib implementation of "GMIN stepping" algorithm is described in [7]. The $scaleGMIN$ variable is included in the static constitutive relation of the SPICELib current source: $i = i_d + v \cdot GMIN \cdot scaleGMIN$.

Just as the Newton-Raphson algorithm has problems with very small values of conductance, very large values of conductance can lead to problems also [3]. Under these conditions, the next voltage iteration (v_{n+1}) will be only slightly different than v_n , and if the device is biased far from the proper solution voltage, many iterations will be required to work back to the solution. Conductance values become unrealistically large when PN-junctions have a forward bias of more than 0.8 volts (see Eq. (2) and Fig. 5). If a high forward bias is applied to the diode during the iterative process, then the Newton-Raphson algorithm may fail to converge. SPICE (and SPICELib) addresses this problem including a series resistance in the diode model (see R_S in Fig. 4) [3]. Under high forward-

bias conditions, the series resistance dominates the device conductance and helps reduce the occurrence of nonconvergence.

3.4 MOS transistors

PSpice provides six MOSFET device models, which differ in the formulation of the I-V characteristic [6]. *SPICELib* implements LEVEL1 model [4]. The *SPICELib* model diagrams of the NMOSFET and PMOSFET devices are shown in Fig. 6. They are modeled as intrinsic MOSFETs using ohmic resistances in series with the drain (R_g), source (R_s), gate (R_g) and bulk (R_b). The model of the two substrate junctions is composed of a linear resistor, a voltage-dependent capacitor and a voltage-controlled, non-linear current source (see Fig. 6). For the sake of conciseness, only NMOSFET model will be discussed in this manuscript: PMOSFET model is completely analogous (all the details can be found in *SPICELib* library documentation).

The drain-to-source current (I_{DS}) and the gate capacitances (C_{GB} , C_{GS} and C_{GD}) are a function of the variables V_{DS} , V_{GS} and V_{BS} . However, V_{GS} and V_{BS} cannot be calculated from the terminal variables of the I_{DS} source model (i.e., the voltage and current at their pins). Gate capacitors modeling presents a similar problem. This situation is solved in SPICElib defining the variables V_{DS} , V_{GS} and V_{BS} as *inner* variables [5] of the transistor model, and as *outer* variables of its components: the I_{DS} current-source and the gate capacitors. These variables are defined as

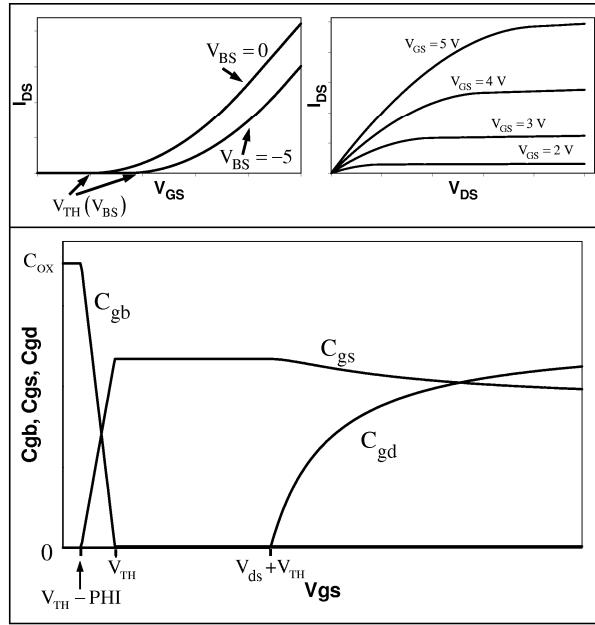


Figure 7: Transfer and output characteristics (above), and gate capacitance (below) of LEVEL1 MOS implemented in SPICELib. Axes scaling is linear.

follows: $V_{DS} = \text{noEvent}(\text{abs}(C_{GS}.v - C_{GD}.v))$, $V_{GS} = \max(C_{GS}.v, C_{GD}.v)$, and $V_{BS} = \max(C_{BS}.v, C_{BD}.v)$; where $C.v$ represents the voltage drop across the capacitor C . As all terms $C.v$ are state-variables of the large-signal MOSFET model, these definitions tear the computational-causality loops of the MOSFET large-signal description.

Drain-to-source current is described by Eq. (5) when the transistor is in the linear region and by Eq. (6) when it is in the saturation region. Equations (5) and (6) are valid for $V_{DS} > 0$ (normal mode). For $V_{DS} < 0$ (inverted mode), SPICELib switches the source and drain in the above equations.

$$I_{DS} = \frac{KP \cdot W}{L - 2 \cdot X_{jl}} \cdot \left(V_{GS} - V_{TH} - \frac{V_{DS}}{2} \right) \cdot V_{DS} \cdot (1 + \lambda \cdot V_{DS}) \quad (5)$$

$$I_{DS} = \frac{KP \cdot W}{2 \cdot (L - 2 \cdot X_{jl})} \cdot (V_{GS} - V_{TH})^2 \cdot (1 + \lambda \cdot V_{DS}) \quad (6)$$

Gate capacitance models used by SPICELib are represented in Fig. 7 (as described in [4]). The three gate capacitances (i.e., C_{GB} , C_{GS} and C_{DS}) are nonlinear, continuous functions of V_{GS} . In addition, C_{GS} and C_{DS} are functions of V_{DS} when the transistor operates in the linear region (i.e., when $V_{GS} > V_{TH} + V_{DS}$). For $V_{DS} < 0$ (inverted mode), SPICELib

switches the source and drain in the capacitance calculations. The transition between C_{GS} and C_{DS} at $V_{DS} = 0$ is discontinuous in the SPICE2 model described in [4]. The model introduced in SPICELib differs from that described in [4], because a continuous link has been introduced in SPICELib between C_{GS} and C_{GD} in the vicinity of $V_{DS} = 0$.

The substrate junctions model differs slightly from the PN-junction model discussed in Section 3.3 [4]. The series resistances (see R_{db} and R_{sb} in Fig. 6) guard against extremely large values of the conductance and to avoid that the MOSFET model index is greater than one.

4 Modeling and analysis of an AC to DC quadrupler

This first example discusses the modeling of an AC to DC quadrupler (i.e., full wave rectifier in series with a low pass filter) using *SPICELib.parts* library. The results of OP, AC sweep and TRAN analyses using *SPICELib.analyses* and OrCAD PSpice are compared.

4.1 Circuit modeling

The AC to DC quadrupler circuit is composed of one independent voltage source, four diodes, one capacitor and one inductor (see Fig. 8). The steps to build the model are the following:

- To open SPICELib library, select in the Dymola window "File/Open". Open the file *SPICELib/package.mo*.
- Create a new package. Enter *circ1* as name of the new package. The circuit model and the analysis models will be inserted in this package.
- Create a new partial-model. It will describe the circuit schematic. Enter *Schematic* as name of the new model, and insert it in package *circ1*.
- Drag the voltage source model from *SPICELib.parts.source* library and drop it into *Schematic* model window. Drag and drop the other circuit components. Resistor, capacitor, inductor, diode and ground models can be found in *SPICELib.parts.breakout* library.
- Connect the components as shown in Fig. 8.
- Double-click on the component icons to enter the component parameters.

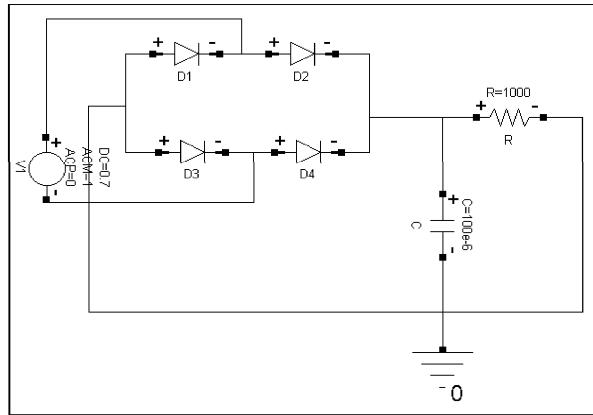


Figure 8: SPICELib model of an AC to DC quadrupler.

Device model documentation is included in *SPICELib*. To get documentation for a device model, place the cursor on it, press the right button and select "Info".

4.2 Bias point analysis

Once the circuit schematic has been described, follow these steps to perform the OP analysis:

- Create a new model, and insert it in *circ1* package. Enter *circ1_OP* as name of the new model.
- Drag *OP* model from *SPICELib.analyses* library, and drop it into *circ1_OP* model window.
- Double-click on the component icon. Enter the name of the circuit model to analyze (*Schematic*) and the value of the OP analysis parameters: *TimeScale*, *LogResults* and *SOLVE_STATIC*. The meaning of these parameters is discussed in [7]. *TimeScale* parameter value is intended for providing an approximate value of the circuit time-constant (in this example, *TimeScale* = *CLOCK* = 0.1s). *LogResults* value sets the amount of information to be logged during the bias point analysis. The bias-point algorithm to be used in determined by *SOLVE_STATIC* parameter value (0, 1, 2 or 3).
- Change to the Dymola window "Simulation" and translate *circ1_OP* model.
- Set Dymola's variable *StopTime* to a value large enough not to interfere with *SPICELib* analysis [7]. A valid setting is *StopTime* = 10 · *TimeScale*. In this example *StopTime* = 1s.
- Run the simulation. *SPICELib* forces the simulation to end when the analysis is finished

```

OP1_R_p_vDC(1e-010) = 0.0115765230984542
OP1_R_iDC(1e-010) = 1.15765230984542e-005
OP1_V1_IDC(1e-010) = -1.160495529906962e-005
OP1_V1_powerDissipation_dissipatedPower(1e-010)= 8.123468709348735e-006
OP1_D2_Rs_p_vDC(1e-010) = 0.3557882615492271
OP1_D1_p_iDC(1e-010) = -1.421610030771139e-008
OP1_D4_Rs_p_vDC(1e-010) = -0.3442117384507729
OP1_D3_p_iDC(1e-010) = 1.159073919876191e-005
OP1_D2_Rs_p_vDC(1e-010) = 0.3557882615492271
OP1_R_p_vDC(1e-010) = 0.0115765230984542
OP1_D2_vDC(1e-010) = 0.3442117384507729
OP1_D2_p_iDC(1e-010) = 1.159073919876191e-005
OP1_D4_vDC(1e-010) = -0.3557882615492271
OP1_D4_p_iDC(1e-010) = -1.421610030771285e-008
OP1_ctrlx_0logx_0DC(1e-010) = 1
OP1_totalPowerDissipation(1e-010) = 8.123468709348735e-006

```

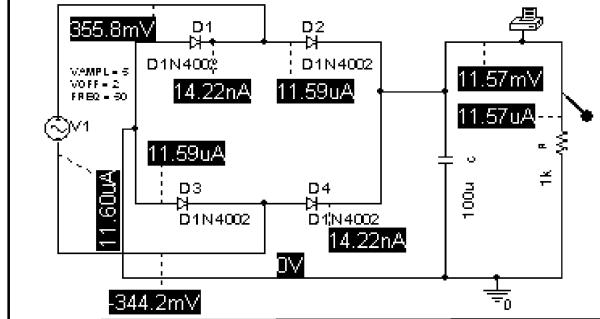


Figure 9: OP analysis results. *SPICELib* using SMI algorithm (above). ORCAD PSpice (below).

(always before *StopTime* is reached, otherwise increment *StopTime* value and repeat the analysis simulation) [7]. Once the simulation is finished, the analysis results are logged out to *dslog.txt* file.

The results of *SPICELib* OP analysis using SMI algorithm are compared in Fig. 9 with the results obtained using OrCAD PSpice [6].

SPICELib OP analysis is repeated using SMR algorithm. The static-description value of the independent power supply is ramped by *SPICELib* from zero to its target value, during *TIME_SCALE* seconds. Then, *SPICELib* logs the analysis results out to *dslog.txt* file. *CLOCK* seconds are elapsed before *SPICELib* triggers the simulation end. The evolution in time of the power supply voltage and the evolution of the voltage drop across the resistor ($\times 10$) are shown in Fig. 10.

Continuing with the example, *SPICELib* GMIN algorithm is applied. The results are shown in Fig. 11. *SPICELib* solves the circuit static formulation starting with a large value of *GMIN*. Every *CLOCK* seconds *SPICELib* reduces *GMIN* by a factor of 10, until *GMIN* nominal value is reached. Then, *SPICELib* logs the voltages and currents of the circuit static formulation out to *dslog.txt* file.

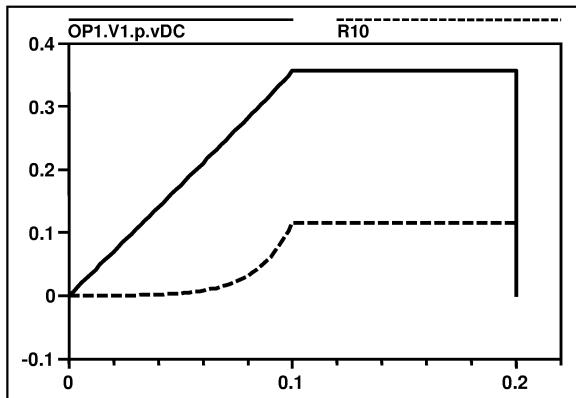


Figure 10: Power supply ramping and voltage drop across the resistor ($\times 10$).

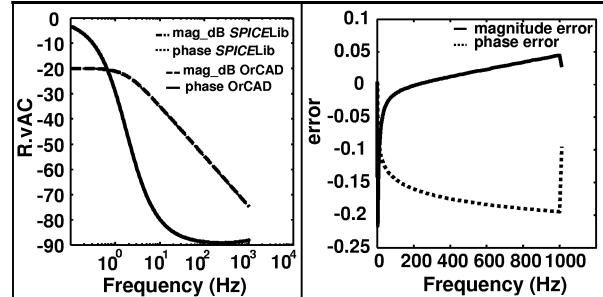


Figure 13: AC sweep analyses.

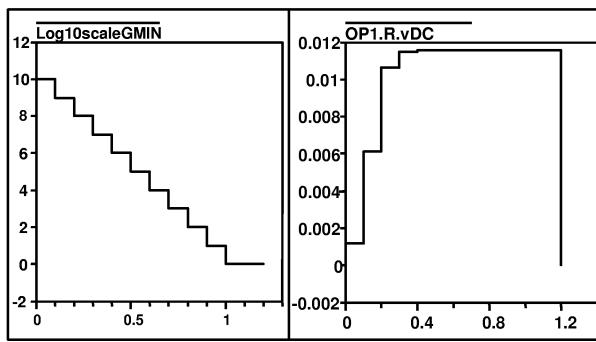


Figure 11: $\log_{10}(scaleGMIN)$ (left). Voltage drop across the resistor (right).

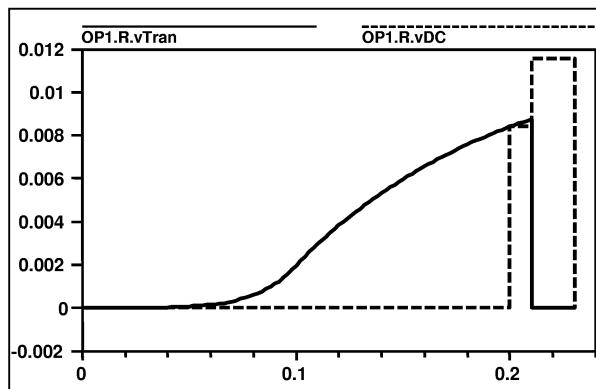


Figure 12: OP analysis using DMR algorithm.

Finally, DMR algorithm is applied (see Fig. 12). The value of $CLOCK$ is set to 0.01. The initial condition to iterate the circuit static description is obtained by simulating the circuit large-signal description [7][1]. A transient analysis is performed: the power supply is ramped from zero up to its initial value during

$TIME_SCALE = 0.1s$ and this value is held for $TIME_SCALE = 0.1s$ to allow the circuit to stabilize. Then (at $time = 0.2s$), the large-signal description voltages are transferred to the static description. This static circuit setting is held for $CLOCK = 0.01$ second (until $time = 0.21s$). Then, the circuit static formulation is solved. The OP analysis results are logged out to *dslog.txt* file at $time = 0.22s$. $CLOCK$ seconds latter, SPICELib triggers the simulation end.

4.3 AC sweep and TRAN analyses

Follow these steps to perform an AC sweep analysis:

- Create a new model, and insert it in *circ1* package. Enter *circ1.AC* as name of the new model.
- Drag *AC* model from *SPICELib.analyses* library, and drop it into *circ1.AC* model window.
- Double-click on the component icon. Enter the name of the circuit model to analyze (*Schematic*) and the value of the AC analysis parameters: *TYPE_AC_SWEEP* (linear or logarithmic by decades), *POINTS_NUMBER*, *START_FREQUENCY* and *END_FREQUENCY* [7]. In addition, the parameters of the OP analysis prior to the AC sweep have to be set.

The results of the AC sweep analysis using SPICELib are compared in Fig. 13 with the results obtained using OrCAD PSpice.

The steps to perform a TRAN analysis are analogous. In this case, drag *TRAN* model instead of *AC* model. SPICELib TRAN analysis of the AC to DC quadrupler, with bias point calculation, is shown in Fig. 14, and it is compared with OrCAD PSpice analysis.

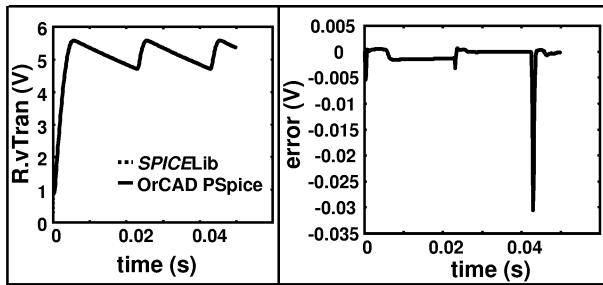


Figure 14: TRAN analyses with bias point calculation.

5 Analysis of digital circuits

The precision obtained from LEVEL1 model is limited; however, the simplicity of these equations is useful in many situations, for instance the analysis of digital circuits. NAND and NOR logical gates have been modeled using *SPICELib* (see Fig. 15), and the results of a transient analysis with bias point calculation have been compared with the results obtained using OrCAD PSpice (see Figs. and 16 and 17).

Conclusions

The new capabilities of *SPICELib* 1.1 have been discussed, and illustrated by means of two examples.

References

- [1] Cellier, F. E.: *Continuous System Modeling*. Springer-Verlag, 1991.
- [2] Elmquist, H. et al.: *Dymola. User's Manual*. Dynasim AB. Version 5.0a, 2002.
- [3] Kielkowski, R. M.: *Inside SPICE*. McGraw-Hill. 2nd edition, 1998.
- [4] Massobrio, G. and Antognetti P.: *Semiconductor Device Modeling with SPICE*. McGraw-Hill. 1993.
- [5] Modelica Association: *Modelica. Language Specification*. Version 2.0, July 10, 2002.
- [6] OrCAD, Inc.: *OrCAD PSpice A/D. Reference Guide & User's Guide*. OrCAD, Inc. 1999.
- [7] Urquia, A. and Dormido, S.: *DC, AC Small-Signal and Transient Analysis of Level 1 N-Channel MOSFET with Modelica*. In proc: 2nd Modelica Conf. pp. 99-108.

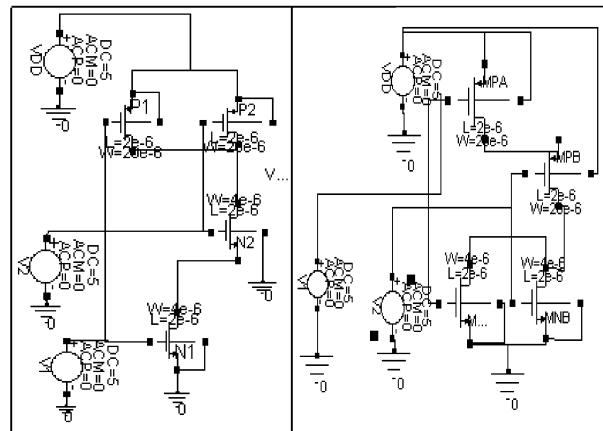
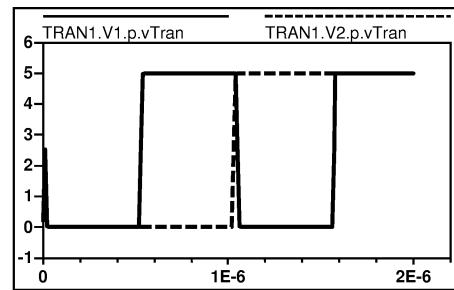
Figure 15: *SPICELib* NAND and NOR models.

Figure 16: Input stimuli (equal for both gates).

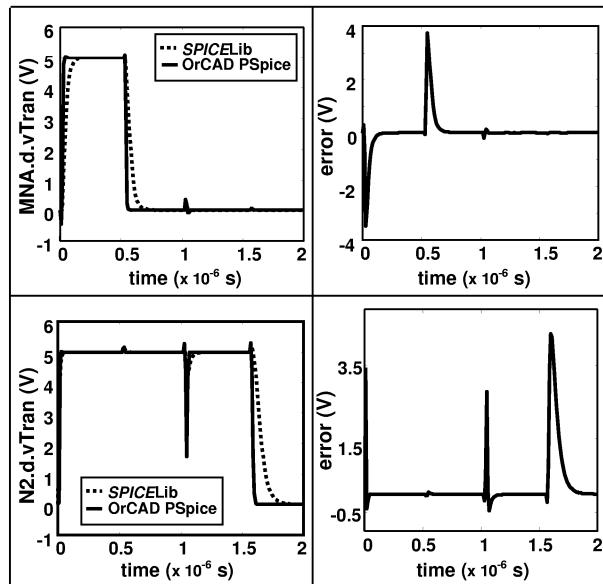


Figure 17: NOR and NAND output voltages.

WasteWater a Library for Modelling and Simulation of Wastewater Treatment Plants in Modelica

Gerald Reichl

Technische Universität Ilmenau

Department of Automation and Systems Engineering

P.O.Box 10 05 65, 98684 Ilmenau, Germany

email: gerald.reichl@tu-ilmenau.de

Abstract

The Modelica application library *WasteWater* containing three Activated Sludge Models of different complexity with the essential components of municipal wastewater treatment plants is presented. Component models are got due to the physical and biochemical modelling of activated sludge basins and secondary clarifiers. The library is verified for different operational situations by a benchmark simulation study. Simulation results of an example real-world wastewater treatment plant are shown.

Keywords mathematical models, simulation, object-oriented modelling, wastewater treatment

1 Introduction

From the point of view of a sustainable management of water and its quality, multidisciplinary teams are currently working to model, to simulate, and to optimize the design and the operation of wastewater treatment plants (WWTPs) with the global goal to reduce the pollution of the environment (receiving water) as well as the operational costs. Among other things this is due to national and international regulations, e.g. the Council Directives concerning urban wastewater treatment (91/271/EEC and 98/15/EEC) of European Commission. Nowadays large efforts are undertaken to extend the consideration of wastewater treatment to a plant wide scope, including such processes as sludge dewatering, waste sludge disposal, energy transformation by bio gas production, etc. Even including the whole or main part of the sewer system is subject of investigations.

To achieve the goals mentioned above a better understanding of microbiological behavior is needed,

and its effects on wastewater control and management processes must be evaluated. That's why a number of mathematical models were developed in the past, e.g. [5, 6]. Most models are used for simulation purposes. Sometimes they are used in connection with simple control algorithms. This is also reflected in the simulation tools available.

Because of the growing effort in establishing computer models of large, complex, and heterogeneous physical systems, e.g. [8, 9, 11], an object-oriented approach has been chosen. The advantages are the suitability for multi-domain modelling, the usage of general equations of physical phenomena, the re-usability of model components, and a hierarchical model structure. The main goal consists in establishing object-oriented system models and furthermore in utilization of the automatically generated, efficient simulation code suitable both for simulation, and later on for control and optimization purposes.

Therefore the *WasteWater* library for Modelica was created that contains widely used and international accepted Activated Sludge Models (ASM) and models for secondary clarifier describing the processes at wastewater treatment plants (WWTP) according to its physical laws with different mathematical complexity. Currently three Activated Sludge Models which are the ASM1, ASM2d and ASM3 [6] and five secondary clarifier models for each ASM are included within the library *WasteWater*. The most important parts at a WWTP are the biological part (activated sludge basin) and the secondary clarifier (settler). Components belonging to these parts are modelled for each ASM.

The verification of the approach is performed with the benchmark plant proposed by the COST benchmark study, [1]. The results published there could exactly be reproduced. Following the library *WasteWater* has been successfully applied to a real-world wastewater treatment plant.

2 Process and Models

Municipal wastewater treatment consists of two stages (a biological and a secondary clarification) and removes carbon, nitrogen, and phosphorus from the wastewater. Mechanical pre-treatment takes place prior these stages. There are activated sludge tanks of different properties. Nitrification takes place in an aerated section where ammonium-nitrogen ($\text{NH}_4^+ \text{-N}$) is converted into nitrate-nitrogen ($\text{NO}_3^- \text{-N}$) by special bacteria under consumption of dissolved oxygen and denitrification takes place without dissolved oxygen and removes the nitrate-nitrogen. Both processes use the carbon compounds in the wastewater as energy source. Biological phosphorus removal occurs under absence of both dissolved oxygen and $\text{NO}_3^- \text{-N}$ e.g. anaerobic conditions. In the secondary settler the activated sludge is separated from the cleaned water by gravity and is returned to the biological stage. Several models exist that describe the processes taking place in the biological part of a wastewater treatment plant and a few models describing the settling process of the activated sludge within the secondary clarifier. Mostly used and accepted are models from the ASM model family [6] by the International Water Association (IWA) and layer sedimentation models. Therefore the Activated Sludge Model No. 1 [5], the ASM2d, and the ASM3 as biological process models and the secondary settling tank models by Takács [13], Härtel [4], Otterpohl [10] and Krebs [7] are collected in a *WasteWater* library. Simulation results of the library were verified by the COST Benchmark plant configuration [1] that uses the ASM1 in connection with the secondary clarifier by Takács.

2.1 Activated Sludge Models

To model a wastewater system object-oriented it is useful to introduce the terms ‘potential variables’ and ‘flow variables’. The dissolved (S_i) and particulate concentrations (X_i) considered by an ASM are the potential variables in a WWTP model. The volume flow rate Q of the wastewater is considered as the flow variable. These variables will be included into the components interfaces (see 3.1) and determine the mass flow rate between connected control volumes (basins). It is assumed that a basin is fully mixed and has a constant volume V . For such a basin the mass balance equations of an ASM define the model equations as follows:

$$\frac{dS_i}{dt} = (S_{i,\text{in}} - S_i) \frac{Q_{\text{in}}}{V} - r_i, \quad (1)$$

$$i \in \{I, S, NO, NH, ND, ALK\}$$

$$\frac{dS_O}{dt} = (S_{O,\text{in}} - S_O) \frac{Q_{\text{in}}}{V} - r_O + r_{\text{air}}, \quad (2)$$

$$\frac{dX_i}{dt} = (X_{i,\text{in}} - X_i) \frac{Q_{\text{in}}}{V} - r_i, \quad (3)$$

$$i \in \{I, S, BH, BA, P, ND\}$$

The index i here stands as example for the concentrations modelled in the ASM1 which are in equation (1) the different dissolved concentrations like inert organic matter (S_I), substrate (S_S), nitrate nitrogen (S_{NO}), etc. and in equation (3) the particulate concentrations which are among others the heterotrophic (X_{BH}) and autotrophic (X_{BA}) biomass. Variables subscripted by index ‘in’, e.g. $S_{i,\text{in}}$, indicate concentrations carried by the flow Q_{in} entering a considered tank. Equation (2) describes the balance of the dissolved oxygen and has an additional term for the oxygen uptake (aeration r_{air}) caused by the blowers. The reaction rates r_i resp. r_O in the balance equations (1 – 3) are given by the model matrix of the Activated Sludge Models. The ASM1 models 13 relevant concentrations (state variables) and eight processes (p_i), the ASM2d is the most complex model with 19 concentrations and 21 biological processes, and the ASM3 has 13 wastewater components with 12 processes. The complete description of the models and their development is available in [6].

2.2 Settler System Models

The settler system models that are provided basically rely on a layer theory [4, 10, 13]. Here the settler is divided into horizontal layers of different properties with mass exchange (hydraulic and sedimentation flux) between the layers. The basis on which the sedimentation flux is modelled makes the difference in the clarifier models included in the *WasteWater* library. As example the double-exponential settling velocity function (4) by [13], that is based on the solids flux concept and applicable to both hindered and flocculant settling conditions is given as follows:

$$v_{sj} = v_0 e^{-r_h X_j^*} - v_0 e^{-r_p X_j^*} \quad (4)$$

$$0 \leq v_{sj} \leq v'_0$$

with v_{sj} - settling velocity in layer j , X_j^* - suspended solids concentration in layer j subject to the limiting condition $X_j^* = X_j - X_{\min}$, X_j - suspended solids concentration in layer j , $X_{\min} = f_{ns} X_{\text{in}}$ - minimum attainable suspended solids concentration, f_{ns} - non-settleable fraction, X_{in} - mixed liquor suspended solids concentration entering the settler.

A clarifier layer model contains at least of three layers. The clarifier models provided in the library are

composed of ten settler layers. All layers are characterized by flux exchanges of adjacent layers caused by hydraulic and settling mass transport. The feed layer (clarifier inflow) receives the wastewater stream from the biological part of a WWTP. There is an upward directed hydraulic flow above the inflow caused by the wastewater flow and a downward directed hydraulic flow below the inflow caused by the return and waste sludge flow at the bottom of the clarifier. In all layers a sedimentation flux occurs due to the gravity that is calculated by e.g. settling velocity function (4) multiplied by the corresponding suspended solids concentration X_j^* .

3 Object-Oriented Modelling

The library *WasteWater* consists of sub-libraries for each implemented Activated Sludge Model, e.g. ASM1, ASM2d and ASM3 besides a sub-library for icons and one for wastewater units. An ASM library itself has an interfaces sub-library for partial models and connectors, sub-libraries for pre-clarifier and the secondary clarifier models, a sub-library for example wastewater treatment plant models, and contains the necessary components for modelling of wastewater treatment plants.

3.1 Definition of Connectors

In order to built up an Activated Sludge Model component library the first step is to define the component interfaces. The proper definition of the interfaces is an essential part because the connectors determine the independent parts of a complex model. After definition of the connectors, library components can be developed and tested independently. The main connector of an ASM library within *WasteWater* is that one between the different basins of a WWTP and consists of the flow and potential variables described in section 2.1. For example, this reads in Modelica modelling language for the ASM1 as follows:

```
connector WWFlowASM1
  package WWU = WasteWaterUnits;
  flow WWU.VolumeFlowRate Q;
  WWU.MassConcentration Si;
  WWU.MassConcentration Ss;
  ...
  WWU.Alkalinity Salk;
end WWFlowASM1;
```

Within the sub-libraries of the several secondary clarifier models different interfaces to connect and inter-

change information between adjacent layers are provided.

3.2 ASM Library Components

In this section an overview over the components inside an ASM sub-library of *WasteWater* shall be given. An ASM library consists of components describing the processes taking place in the biological stage of a WWTP (e.g. *Nitri*, *Deni*), a blower, flow mixer, flow divider, measurement devices (concentration sensors), a source and sinks for the wastewater stream, and a sub-library *SecClar* containing the clarifier models which each having classes that describe the sedimentation processes in the different secondary clarifiers. First of all the ASM parameters and equations (process rates, reactions and derivatives of the states) and the connector information that are needed in different model classes are defined in a *partial* model which gives this information to the components. In extracts the partial model for the ASM1 reads as follows:

```
partial model ASM1Base
  package WWU = WasteWaterUnits;
  parameter Real mu_h=4.0;
  ...
  WWU.MassConcentration Si,...;
  Real p1...p8 "process rates";
  Real r1...r13 "reactions";
  Real inputSi,inputSo,...;
  Real inputXi,inputXp,...;
  Real r_air;
  equation
    p1 = ...;
    r1 = ...;
    // derivatives
    ...
    der(Xp) = inputXp + r7;
    der(So) = inputSo + r8 + r_air;
    ...
    // Outputs
    Out.Q + In.Q = 0;
    Out.Si = Si;
    ...
  end ASM1Base;
```

Following components are available for each ASM:

Deni: It inherits graphic information and the information from the respective partial model e.g. *ASM1Base* and extends it by a specific tank volume to model a denitrification tank ($r_{air} = 0$).

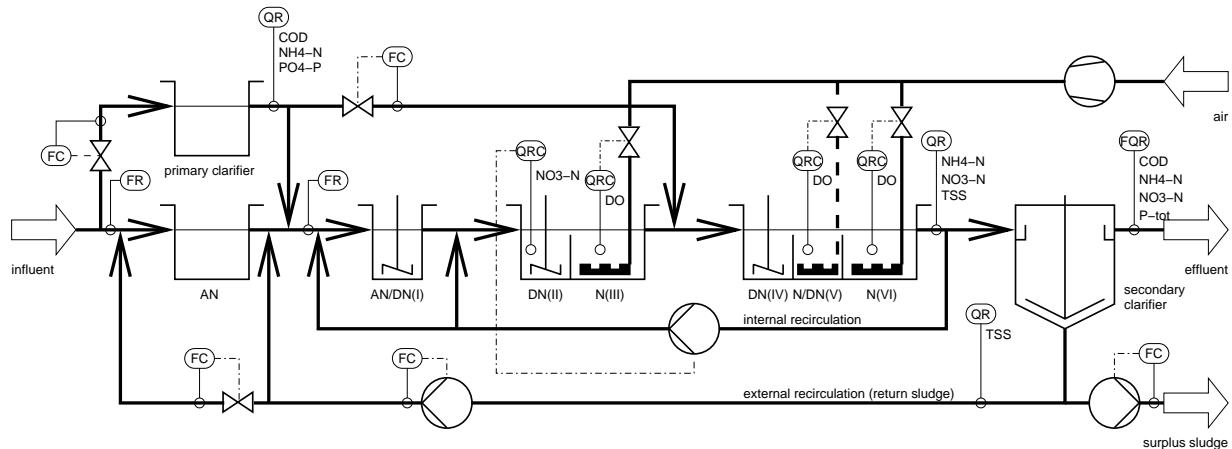


Figure 1: Simplified scheme of the example wastewater treatment plant

Nitri: This component is used to model a nitrification (aerated) tank of a WWTP which as well inherits its graphic information and e.g. ASM1Base and is extended by the tank volume and aeration system dependent parameters.

```
model Nitri
  extends Icons.nitri;
  extends Interfaces.ASM1Base;
  import SI = Modelica.SIunits;
  parameter SI.Volume V "tank volume";
  //aeration system parameters
  parameter Real alpha=0.7;
  ...
  Interfaces.AirFlow AirIn;
equation
  r_air = ...*AirIn.Q_air*...;
  // Volume dependent dilution
  inputSi = (In.Si - Si)*In.Q/V;
  inputXi = (In.Xi - Xi)*In.Q/V;
  ...
end Nitri;
```

SecClarModTakacs: Is a prepared component which describes a ten-layer secondary clarifier model based on Takács [13] using the sub-library *SecClar:Takacs*.

Blower: The blower can be used to model an air flow between a minimal (Q_{\min}) and a maximal (Q_{\max}) blower capacity as input to the nitrification tank based on a control signal.

Pump: This component models a wastewater pump. It generates a wastewater flow between Q_{\min} and Q_{\max} that is controlled by an external control signal.

Mixer: There are two components available which mix two respectively three different flows of wastewater of

different amount and different concentration. The output is a single mixed wastewater stream.

Divider: These two elements divide one flow of wastewater into two separate flows of same concentration either by known flows or externally controlled by a signal.

OxygenSensor: The concentration of oxygen in a tank or a wastewater stream is measured and transformed into an output signal $y(t)$ that can be further processed. Similar sensors for the concentration COD, nitrate-nitrogen (S_{NO}), ammonia-nitrogen (S_{NH}), and others are provided.

WWSource: Provides all ASM data at the influent of a WWTP. The dimension depends on the used ASM. The information can also be read from a file.

EffluentSink: Is the receiving water at the effluent of a wastewater treatment plant and terminates a WWTP model. A similar component is the *SludgeSink*.

4 Example of use

For verification and validation purposes of the *WasteWater* library's components first of all the COST benchmark plant layout was used. The results published in [1] could exactly be reproduced using ASM1 components of *WasteWater*. But this is not discussed in more detail here.

Following the library *WasteWater* is applied to a real-world WWTP. The plant is situated in Jena, Germany, and has a size of 145,000 population equivalents. A model of this plant is available in each *ASM_Examples* sub-library as complex plant example. The configuration of this WWTP is shown in Figure 1. The continuous flow WWTP is a cascade type denitrification with

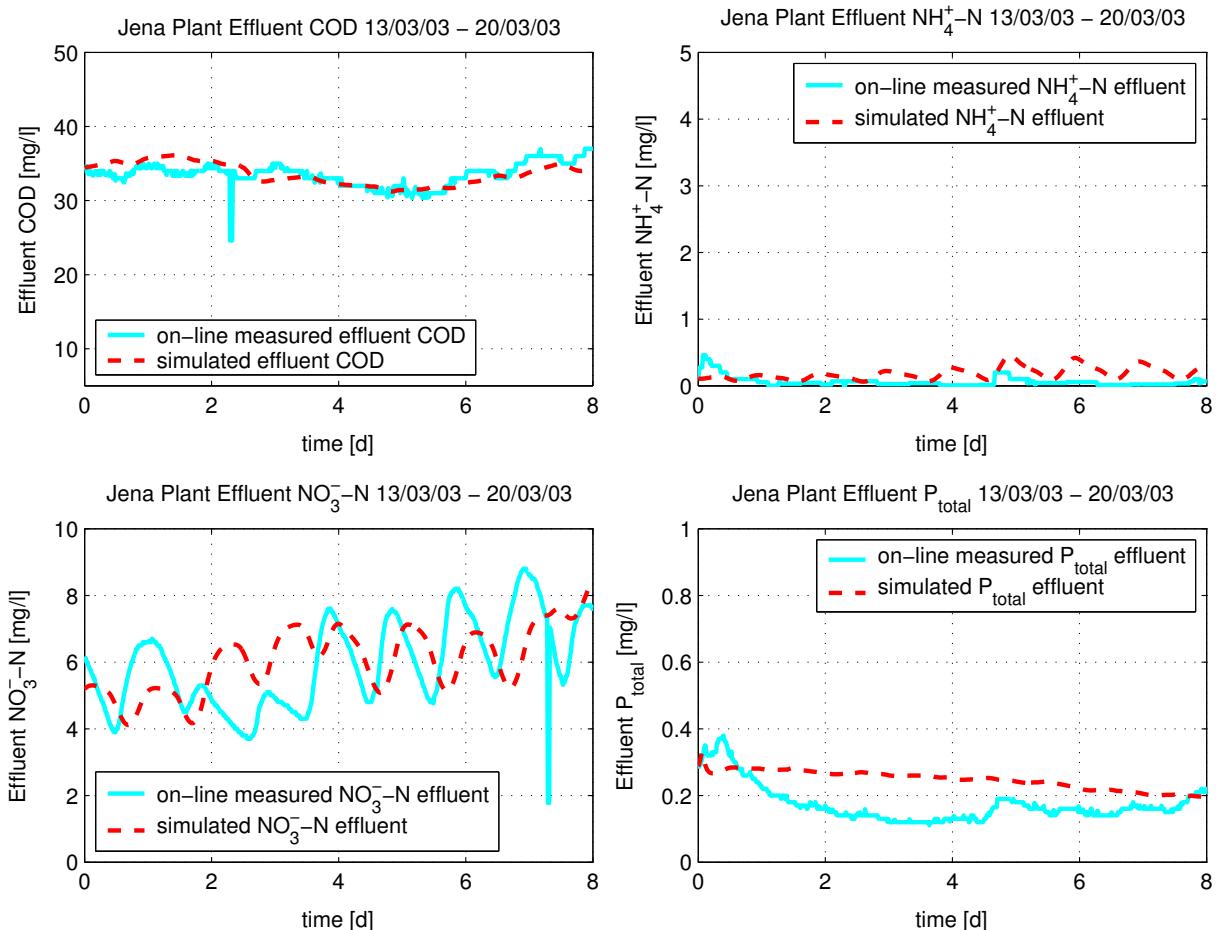


Figure 2: Simulated and on-line measured effluent values for the calibrated ASM2d WWTP example model

pre-clarification, biological and chemical phosphorus removal, and secondary settling. The plant is designed for a mean dry weather inflow of 28,500 m³/d. The total volume of all activated sludge tanks is approx. 24,000 m³, of which 14,000 m³ can be aerated. There are two flow feedbacks, one internal recirculation from the last biological tank, and a return sludge flow from the bottom of the secondary settler, see Figure 1. An additional outflow, the surplus (waste) sludge flow, occurs at the bottom of the settler. The effluent of the WWTP is discharged to the receiving water and is located at the settlers surface.

On-line measurements are available for the influent flow rate and concentrations (chemical oxygen demand (COD)), ammonia-nitrogen, and phosphate), internal and external recycle flow rates and total suspended solids (TSS) concentrations, dissolved oxygen in the aerated tanks, effluent quality (COD, ammonia- and nitrate-nitrogen, and phosphate) as well as phosphate, ammonia- and nitrate-nitrogen at the outflow of the cascade.

The software package DYMOLA [2] is used to implement the *WasteWater* library and to perform the example simulation scenarios. A simulation diagram can be established by drag and drop of the several components of the *WasteWater* library and linking the elements together via the connectors. A system of differential-algebraic equations (DAE), in the described plant configuration using the ASM2d, with 3081 unknowns and equations and 252 state variables is established automatically by DYMOLA. The DASSL integration procedure implemented in DYMOLA is used to solve the DAE system.

Simulating real wastewater treatment plants normally needs a model calibration procedure, as the provided ASM set of parameters by IWA that is implemented in *WasteWater* has to be adapted and does not match all WWTPs. Many of the biological and kinetic parameters may vary in a limited range. Such a model calibration has been done for the ASM2d complex example plant using genetic algorithms.

Figure 2 shows the simulated effluent values COD,

ammonia- (NH_4^+ -N) and nitrate-nitrogen (NO_3^- -N), and total phosphorus of the calibrated ASM2d example model (dashed line) compared to the on-line measurements from the SCADA system (solid line). The simulation results are satisfactory so far for the precision of wastewater treatment process models. Obvious differences occur due to several assumptions and simplifications which need to be done during the validation phase. Further improvement of the results is expected by applying initial state estimation which is subject of current investigation.

5 Conclusion

An application library *WasteWater* for Modelica that collects the Activated Sludge Models ASM1, ASM2d, and ASM3 by the International Water Association (IWA) including several secondary clarifier models was developed. It contains essential WWTP components according to an object-oriented approach and based on physical modelling. The *WasteWater* library presented and its application to plants of several complexity show the usefulness and the advantages of an object-oriented modelling approach.

The compiled system model can be used for solving parameter and state estimation problems and especially as basis for ongoing control and optimization applications, see [3].

Future work is directed to use the automatically compiled system model (DAE system) of an calibrated WWTP model inside a model-predictive control algorithm within a decision support system in order to optimize the plant behavior. First open-loop optimization results are already available.

6 Acknowledgment

This publication has been made possible by the technical and financial support of the SMAC project, EVK1-CT-2000-00056, under EC's 5th Framework programme.

References

- [1] J. B. Copp. The COST simulation benchmark. <http://www.ensic.u-nancy.fr/COSTWWTP/>, 2000.
- [2] H. Elmquist, D. Brück, S. E. Mattson, H. Olsson, and M. Otter. Dymola – dynamic modeling laboratory. User’s manual. Dynasim AB. Sweden, 2001.
- [3] R. Franke. Formulation of dynamic optimization problems using modelica and their efficient solution. In M. Otter, editor, *Modelica 2002*, Proceedings of the 2nd International Modelica Conference, pages 315–323, 2002.
- [4] L. Härtel. *Modellansätze zur dynamischen Simulation des Belebtschlammverfahrens*. PhD thesis, TH Darmstadt, 1990.
- [5] M. Henze, C. P. L. Grady Jr, W. Gujer, G. v. R. Marais, and T. Matsuo. Activated sludge model no. 1. Scientific and technical report no. 1, IAWQ, 1987.
- [6] M. Henze, W. Gujer, T. Mino, and M. v. Loosdrecht. Activated Sludge Models ASM1, ASM2, ASM2d and ASM3. Technical report, IWA Task Group on Mathematical Modelling for Design and Operation of Biological Wastewater Treatment, 2000.
- [7] P. Krebs and M. Armbruster. Numerische Nachklärbeckenmodelle. *Korrespondenz Abwasser*, 47(7):985–999, 2000.
- [8] S. E. Mattsson, M. Anderson, and Åström, K. J. Object-oriented modeling and simulation. In D. A. Linkens, editor, *CAD for Control Systems*, pages 31–69. Marcel Dekker, New York, 1993.
- [9] M. Otter. Objektorientierte Modellierung Physikalischer Systeme, Teil 1. *at - Automatisierungs-technik*, 47(1):A1–A4, 1999.
- [10] R. Otterpohl and M. Freund. Dynamic models for clarifiers of activated sludge plants with dry and wet weather flows. *Water Science and Technology*, 26(5-6/90):1391–1400, 1992.
- [11] H. Puta, G. Reichl, and R. Franke. Model based optimisation of a waste water treatment plant. European Control Conference ECC’99, Karlsruhe, Germany. Summaries vol. p. 189 (Full paper on CD of Conference Proceedings),, 1999.
- [12] G. Reichl, S. Hopfgarten, and H. Puta. Objekt-orientierte Modellierung von Abwasserreinigungsanlagen. In *Frontiers in Simulation*, 16. Symposium Simulationstechnik ASIM 2002, pages 424–432, 2002.
- [13] I. Takács, G. G. Patry, and D. Nolasco. A dynamic model of the clarification–thickening process. *Water Research*, 25(10):1263–1271, 1991.

Session 5

Poster session

Adaptive signal management

- A Modelica and C++ interaction example, the SignalFlow Library

Jörgen Svensson and Per Karlsson

Lund University
 Dep. of Industrial Electrical Engineering and Automation
 Box 118
 SE-221 00 Lund
 Sweden
 jorgen.svensson@iea.lth.se

Abstract

This paper presents an adaptive signal management Modelica library, "SignalFlow", interconnected with a C++ class library. The objective is to simplify the signal exchange in large simulation models based on modular designs, which should correspond to the signal flow for real applications by representing common networks as models with general interfaces. The library enables automatic configurations during simulation using dynamic vectors and has additionally functions for exchanging several types of signals in both continuous and discrete mode. The work is an outcome for enabling "plug and produce" capabilities in scalable distributed power system applications that is exemplified.

1. Introduction

Communication interfaces are used in almost every technical application that needs signals to be exchanged. Control units may be embedded in components of varying sizes, where a component itself might be aggregated of others according to design and structure. Dependent on complexity, there are several signal levels both for horizontal and vertical interconnections [1,2]. This is complicated in real systems but even more in simulation environments (SE). Several SEs have hierarchical possibilities in modeling and define terminal types for signal exchange. However, numerous possibilities easily become a trap when using a multilevel hierarchy of signal interconnections. In Figure 1a it is shown that several different IO terminals easily become disordered as the system becomes larger. Different terminals represent various groups of signals that need to be used for connecting the models. It is easy to complicate the model structure by extending the number of terminal types, which cause many and tricky connections. For example, if adding a new type of terminal in model M12 in Figure 1a, each model at all levels need to be reconstructed by adding new terminals. If modeling a large model with many levels of aggregations it becomes even more complex. This is simplified by using a model representing a general communication bus, as shown in Figure 1b, where every instance is interconnected to the same bus independent of information level if so desired. Dependent on the

number and types of communication interfaces there are alternative configuration opportunities for the signal exchange in the models. For example, if throughout using the same interface it might be practical for the user to be spared assigning identities to every single communication node. One model solution for the Figure 1b case requests a vector based signal bus, where the bus vector merges together all the terminal vectors. Although, this brings in a problem with always keeping in mind the correct number of indexes dependent on the number of connected components.

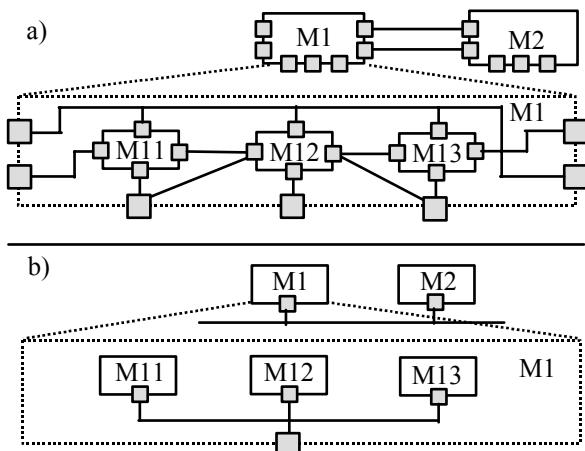


Figure 1. A combination of interconnections with different types of terminals at several levels (a) and a general signal bus using one type of terminal enabling a plain structure (b)

Even the index of each component has to be determined if a general approach is used. In this case it is necessary to define the exact number of signals in every terminal to be connected and the bus needs a pre-defined vector index according to number of connected components. If a uniform terminal is used with a large number of signals where several models are not using but a small amount of terminal signals, the SE will have an unnecessary high number of signals to handle. It is therefore desirable to be able to choose which internal module signals to be exposed by limiting the terminal signals. The goal of a general structure is a signal bus that automatically assigns the components with their identities and that enables the user to mainly focus on the signal to be exchanged and not on the underlying

structure and functions that manage the signal communication. The structure should be adaptive to different user specific desires and also able to resemble several types of communication, e.g. between software processes, computers or in automation systems. One solution based on Modelica interconnected to a C++ library will be further described and exemplified.

2. Design

The design is meant to work both in a SE and for real applications. This calls for either an automatic translation from the modeling language to the target application or a smooth software interface between the SE and the chosen program language. As the Modelica language has nice facilities for external function calls the main functions of the library is implemented in a C++ library [3,4,5].

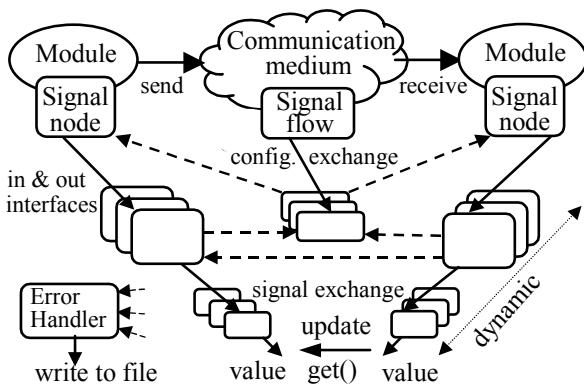


Figure 2. Signal management design overview

The basic idea is that independent of different types software modules there should be simple means to establish signal configuration and connection via some type of communication media to another module as depicted in Figure 2. A module could be a software process, thread or a model in a SE where the communication media could be an external or internal communication link [6]. The design is divided in a hardware and software structure, where the physical part, communication lines, transmitters and receivers are developed in the Modelica languages and the software functions in C++. The principal structure in Modelica builds upon signal nodes and signal flows where the signal nodes may represent a temporary storage, transmitter or receiver. The signal flows represent the communication lines between the signal nodes with the main tasks, in initiating mode, to inform the connected nodes about the configuration and, in operation mode, to control that the physical line is in order for signal transmitting. The signal nodes creates in- and output interfaces according to the signal flow configurations as shown. The interfaces in turn create individual signal objects for each specified signal. An input interface then points to an output interface of another signal node where the configurations are checked before switching to operation mode. During operation, each signal object

updates the data when triggered by the signal node. Every failure or configuration error is reported to an error manager that writes the needed information in a file or to the log window in the SE.

3. Signal classification and configuration

An important issue concerning signal classification is how to enable several types of configurations without making it too complex. The following signal classifications are used to configure the signal flows.

- The **Signal Identity** (SI, SIdentity), which enables a signal to have a unique identity, but this might imply obstacles regarding dynamical capabilities
- The **Signal Type** (ST, SType), where each signal must be specified by a unique type (e.g. command, power set point, etc)
- The **Signal Block Type** (SBT, SBType): a predefined number of signal types, which could be uniformed (protocol)
- The **Signal block Group Identity** (SGI, SGIdentity) can be used if several SBTs are connected between the same source and destination. The unique group identity enclosures SIs, STs or/and SBTs.

Signal flow configurations must at least have a specification on a SIdentity or SType. Normally the SIdentity is used as a unique identity that can be found anywhere in a system model. However, if using a model with several components of the same type and signal interface together with a higher-level control unit collecting and distributing signals, the SBTypes and SGIdentities are requested as depicted in Figure 3.

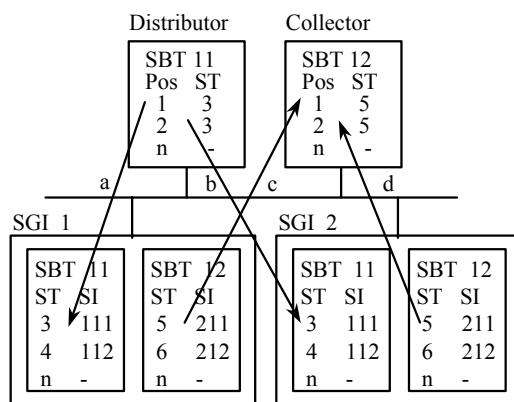


Figure 3. Example of the signal classification and their signification

First, by examine each component it is desirable to use the same set of identities for the signals in each component. As in this case there is at least two SBTypes (11 & 12), one for input and one for output signals, where each component signal flow need to be unique by enclosure all the SBTypes in a SGIdentity as shown. Further, it is possible to use only the STypes and not the SIdentities at higher level, as the intention is a dynamic

distributor and collector at the higher level. Arrow “a” and “b” in the Figure point out that it is the SType 3 at position 1 and 2 that should be transmitted to SGIdentities 1 and 2 respectively. For the collector it is similar the SType 5 at position 1 for both SGIdentities 1 and 2 that should be transmitted to the two first positions of the collector. If a new component is added with SGIdentities 3, the distributor and collector per automatic should extend the vector signal with respective STypes.

4. Modelica library

The signal classifications are the basic parameters for configuring the signal flows in the SE. The SignalFlow library is built in Dymola, using the Modelica language. Dymola is an object-oriented SE for modeling transient physical systems that has good support for interconnecting other object-oriented languages such as C++, which is well exploited in this library [3,4,5,7].

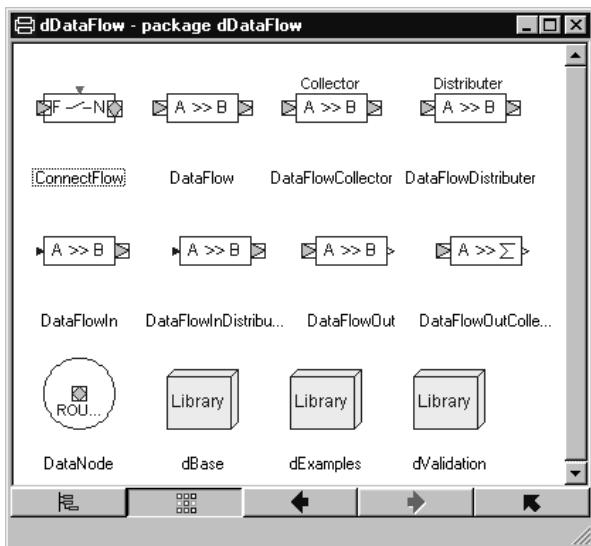


Figure 4. The SignalFlow library

The SignalFlow library is depicted in Figure 4 and mainly contains the following component models:

- The **SignalNode** (SN, SNode) model manages all incoming and outgoing signal interfaces initiated by the SignalFlows.
- The **SignalFlow** (SF, SFlow) model represents the signal configuration between the SNodes.
- The **SignalFlowOut** (SFO, SFlowOut) model has a standard Modelica output terminal to interconnect with other library models.
- The **SignalFlowIn** (SFI, SFlowIn) model has a standard Modelica input terminal that supplies the signal system with signals.
- The **SignalFlowDistributor** (SFD) model is similar to the SFlowIn but initiates an automatic search for SFlowOut models that are configured according to a predefined SType.

- The **SignalFlowCollector** (SFC) model initiates an automatic search among SFlowOut models for signal types to be collected (model with sum-sign).
- The **SignalResourceManager** (SRM) model interacts by connecting a SNode where it accesses a predefined resource type.

The main component models are constructed several sub-libraries within the Base library, which contain the dConnector, dInterface, dIcon, dRecord, and dFunction library. The sublibrary dConnector contains two connector (terminal) types that are defined by the following Modelica semantics:

```
connector SignalNodePort
  Real signalNode;
  flow signalLine;
end SignalNodePort;
```

The second connector “SignalFlowPort” is identical except for the icon, which is a triangle instead of a quadrangle as in the “SignalNodeConnector” case. The model interfaces, within the dInterface sublibrary, are composed as below where the node and flow have one or two connectors respectively.

```
partial model SignalNodeInterface
  extends dicons.SignalNodeIcon;
  dConnectors.SignalNodePort nodePort;
  Real nodeS = nodePort.signalNode;
  Real lineS = nodePort.signalLine;
end SignalNodeInterface;
```

```
partial model SignalFlowInterface
  extends dicons.SignalFlowIcon;
  dConnectors.SignalFlowPort flowPortA;
  dConnectors.SignalFlowPort flowPortB;
  Real nodeA = flowPortA.signalNode;
  Real lineA = flowPortA.signalLine;
  Real nodeB = flowPortB.signalNode;
  Real lineB = flowPortB.signalLine;
end SignalFlowInterface;
```

The user interface assigning the SNode parameters are depicted in Figure 5 where the node type can be used to force the node to be of storage type. The node identity is normally automatically assigned but could also be forced to a specific value, and the last parameter determines if the node should be continuously or discrete. The SNode in the SignalFlow library is initiated and automatically assigned an identity by the zNodeInit function. The function argument is a configuration vector (cv) where all predefined parameters are placed. Dependent on whether the SNode should be discrete (sampled) or continuous the variable nodeD or nodeC is assigned. When initiated the node updates every time interval according to the SE and extended equations are only one technique solving the discrete or continuous options. The SNodes are always initiated at simulation start and during simulation the only input is the “lineS” that is one of the arguments in the zNodeUpdate function.

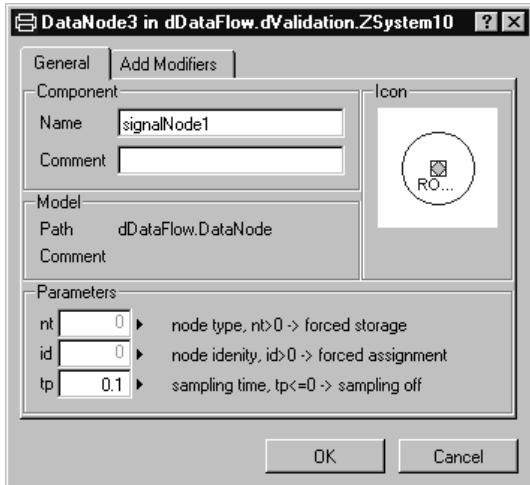


Figure 5. User interface for assigning the parameters of the SignalNode model

As the “lineS”, per Modelica-definition, is declared as a “flow”, all connected SFlow models are summarized in this variable enabling the node to examine which lines that are active.

```
model SignalNode
  extends dRecords.SignalNodeRecord;
  extends dlInterfaces.SignalNodeInterface;
protected
  Real nodeID(start=0.0);
  Real nodeC(start=0.0); // continuous
  Real nodeD(start=0.0); // discrete
  Boolean sampleTrigger;
equation
  when initial() then
    nodeID = dFunctions.zNodeInit(cv, cvSize);
    reinit(nodeC, nodeID);
  end when;
  sampleTrigger = if samplingON then
    booleanPulse1.outPort.signal[1] else false;
  when sampleTrigger then
    nodeD = dFunctions.zNodeUpdate(nodeID,
      lineS, time);
  end when;
  der(nodeC) = if samplingON then 0.0 else
    nodeID - dFunctions.zNode_
    Update(nodeID, lineS, time);
  nodeS = if samplingON then pre(nodeD)
    else nodeC;
end SignalNode;
```

By definition, the SFlow model has always a flow of signals from A to B as shown by the icon and in the “SignalFlowInterface” declaration where the two terminals are denoted A and B. The SFlow model is initiated as soon as the variables “nodeA” and “nodeB” are positive. The initiating function, zFlowInit, then automatically returns the line identities lineAID and lineBID. In normal operation the SFlow only checks that the line is correct for transmitting. If a failure occurs on the line, the ““lineA” and “lineB” are assign to an error code. When the line is restored the initiating process once again is performed.

```
model SignalFlow
  extends dRecords.SignalFlowRecord;
  extends dlInterfaces.SignalFlowInterface;
  Real flowID(start=0);
  Real lineAID(start=0);
  Real lineBID(start=0);
equation
  when (nodeA*nodeB > 0) then
    flowID = dFunctions.zFlowInit(nodeA, nodeB,
      cv, cvSize, signalType, sTypeSize,
      signalAID, sAIDSize, signalBID,
      sBIDSize);
    lineAID = dFunctions.zTryConnectFlowOut(flowID,
      nodeA, time);
    lineBID = dFunctions.zTryConnectFlowIn(flowID,
      nodeB, lineAID, time );
  end when;
  lineA = dFunctions.zFlowUpdate(flowID, nodeA,
    lineAID, nodeB, lineBID, time);
  lineB = dFunctions.zFlowUpdate(flowID, nodeB,
    lineBID, nodeA, lineAID, time);
end SignalFlow;
```

In the SignalFlowOut model the lineB is not used and in the SignalFlowIn model the lineA is not used. They are replaced by the “value” variable that is connected to the standard Modelica Input or Output connectors.

```
-----SignalFlowOut ---
lineA = zFlowUpdate(flowID, nodeA, lineAID);
for index in 1:sTypeSize loop
  value[index] = zFlowGet(nodeA, lineA, index);
end for;
-----SignalFlowIn ---
lineB2 = zFlowUpdate(flowID, nodeB, lineBID,...);
lineB = zFlowSet(nodeB, lineB2, value, valueSize);
```

In the zFlowInit function, the argument is equivalent to the “SignalFlowRecord” that corresponds to the signal classification in section 3 and in Figure 6.

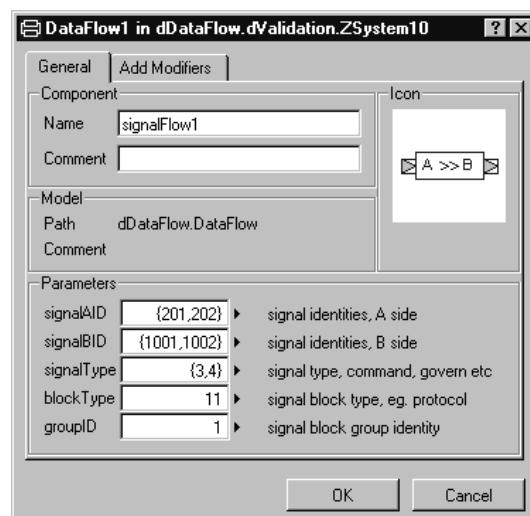


Figure 6. User interface for assigning the parameters of the SignalFlow model

Both side A and B are represented with one identity vector each enabling different identity assignments of the same signal in different communication areas. Using the library in the simplest case the identity is the only parameter to be assigned.

```
record SignalFlowRecord
  parameter Integer[:] signalAID ={-1};
  parameter Integer[:] signalBID ={-1};
  parameter Integer[:] signalType={-1};
  parameter Integer groupID;
  parameter Integer blockType;
protected
  parameter Integer[:] cv={0,0,0,groupID,blockType};
  parameter Integer cvSize=size(cv, 1);
  parameter Integer sTypeSize=size(signalType, 1);
  parameter Integer sAIDSize =size(signalAID, 1);
  parameter Integer sBIDSize =size(signalBID, 1);
end SignalFlowRecord;
```

The SNode model has no direct limitation in connecting the number of SFlow models, as the structure is dynamic. Each new connection creates a new object that might be automatically removed if disconnected a predefined amount of time. The structure is simple and can be connected with unlimited SFlows between SNodes, Figure 7, and at any hierarchy level. In the simplest case one SNode is used as a communication bus where all SFlows are connected to that single bus. The SFlows are normally embedded in some user specific model hiding the pre-defined communication interface interacting the bus.

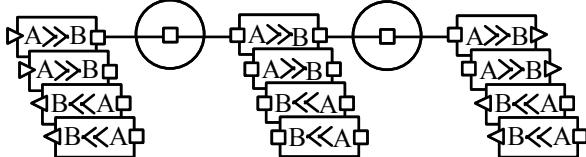


Figure 7. Dynamic number of connections

The basic Modelica structure for connecting SNodes and SFlows, depicted in Figure 8, prevents algebraic loops, which is easily caused with a high degree of control levels in a SE. Even if the SE can handle this, as in the Dymola case, it might become a complicated problem in large system models. The SNodes have a state (node) corresponding to capacitors (voltage) summarizing the variables from the SFlow models. The SFlow assigns the lineA and lineB, identities, corresponding to currents in the electrical case. The line identities are then identified by using a “modulo 2” function both when assigning and decoding the identities, e.g. if four SFlows are connected and the identities of them are 2, 4, 8 and 16 with the sum of 30, it is easy for the SNode to decode the SFlows to determine both if a new SFlow has to be configured and if a line is broken. Referring to Figure 8, each node can be connected to numerous signal flows where the SNode assigns the terminal variable “nodeS” and the line identities are summarized in the terminal variable “lineS” of the node. In the SFlow, the lineA and lineB

are separated enabling the responsible node to change the line identity if needed. In case of disconnecting one SFlow model, it is not likely that the identities will be the same in a dynamic environment. There is also a cross coupling between the nodes by the arguments in the update function, which uses the corresponding node identities in order to avoid losing the equation tail in the SE. The SE initiating process automatically assigns all identities of the SNode and SFlow models.

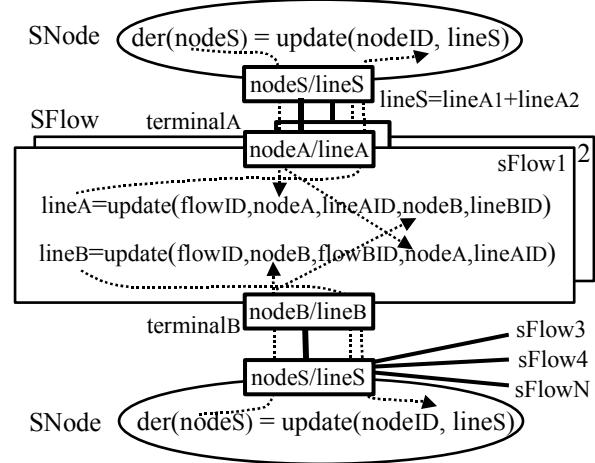


Figure 8. Terminal assignments between SNodes and SFlows

5. Interconnection to C++ library

The Modelica models in the SignalFlow library have several functions for interacting the with C++ library. All included functions are similarly declared as here exemplified for the “zNodeUpdate” function.

```
function zNodeUpdate
  annotation (Library={"Libcore"});
  input Real nodeID;
  input Real flowSum;
  output Real y
external "C" y = zNodeUpdate(nodeID, flowSum);
end zNodeUpdate;
```

The “Libcore” assignment is the actual C++ library that is linked to the SE by the “Libcore.lib” file, which is placed under the “dymola\bin\lib” directory.

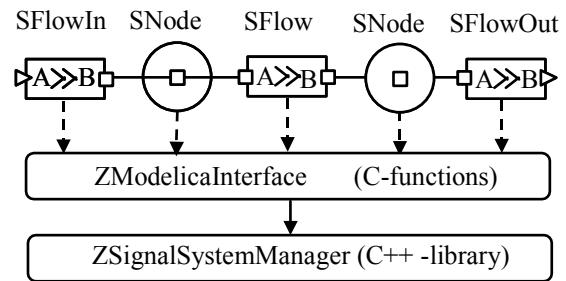


Figure 9. Modelica and C++ interconnection

Each SNode and SFlow model creates a separate object in the C++ layer where the Modelica-layer models use function calls to the ModelicaInterface and method calls to the C++ library, as depicted in Figure 9. The header declarations for the “ZModelicaInterface.h”, enables method calls from the C++ implementation, as shown below. By using the “ZModelicaInterface.cpp”, the Modelica function calls are translated to method calls from the “ZSignalSystemManager” class where all methods can be found in each step interfacing the C++ library.

```
// --- ZModelicaInterface.h ---
#ifndef ZMODELICAINTERFACE_H
#define ZMODELICAINTERFACE_H

#include "ModelicaUtilities.h"
#include "ZSignalSystemManager.h"

#ifdef __cplusplus
extern "C" {
#endif
double zNodeInit(int cv[], int cvSize);
double zNodeUpdate(double nodeID, ....);
..... remaining functions

#ifdef __cplusplus
}
#endif
#endif // end
```

The description of the specification is here exemplified, where the remaining functions are declared as the two presented.

```
// --- ZModelicaInterface.cpp ---
#include "ZModelicaInterface.h"

ZSignalSystemManager sys; // C++ class

double zNodeInit(int cv[], int cvSize) {
    return sys.nodeInit(cv, cvSize);
}
double zNodeUpdate(double nodeID,...) {
    return sys.nodeUpdate(nodeID, ....);
}
..... remaining functions
// end
```

The “ZSignalSystemManager” class is the actual interface between the Modelica-layer that manages all the ZSignalNode and ZSignalFlow objects using dynamic vectors as is briefly shown here. The JVector class is a dynamic vector (DV) equivalent to the CVector class in the C++ standard library except for some modifications making a smooth conversion to the JAVA environment. The dynamic properties in the library are based on the DV that is used for pointing out all needed objects for the specific application. There are facilities in Modelica allowing to declare void*-pointers and external objects by defining a partial class “ExternalObject” with constructor and destructor functions that would make the ZSignalSystemManager

excessive. However, this is not used in this version but might be implemented in the next.

```
typedef JVector<ZSignalNode*> ZSignalNodeVector;
typedef JVector<ZSignalFlow*> ZSignalFlowVector;

class ZSignalSystemManager
{
private:
    ZSignalNodeVector* m_nodeVector;
    ZSignalFlowVector* m_flowVector;
    int m_nodeCounter, m_flowCounter;
    ZOutFileManager* outfile;
public :
    ZSignalSystemManager();
    ~ZSignalSystemManager();
    double nodeInit(int cv[], int cvSize);
    double nodeUpdate(double node, flowS, time);
    double flowInit(double nodeA, nodeB, int .....);
    double flowUpdate(double flow, nodeA, .....);
    double tryConnectFlowIn(double flow, node, flowA);
    double tryConnectFlowOut(double flow, node);
    double flowSet(double node, flow, *value, .....);
    double flowGet(double node, flow, int index);
    void outputManager(const char* text,int type);
}; // end ZSignalSystemManager
```

6. The C++ class library

The key to manage the dynamic design interconnected to a SE is to use higher levels of abstraction as in object-oriented languages [6,8]. An example for this approach is implemented in a C++ class library based on implementation corresponding to the SNode and SFlow models used in the Modelica layer.

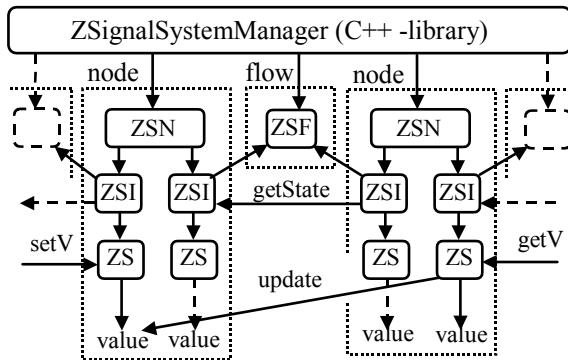


Figure 10. C++ structure

Using the C++ class libraries, a designer can describe components at a broad range of abstraction levels, which result from the ability to perform signal and control modification separately. As pointed out in section 1, there are different levels of abstraction at which C++ can be used for the signal management system as depicted in Figure 10. The C++ library mainly includes the following classes:

- The **ZSignalNode** (ZSN, ZSNode) class is responsible for searching and joining together every signal in each connected input and output signal

interface. The class is receptive to changes of new interconnections and continuously controls all connections for not being defected.

- The **ZSignalInterface** (ZSI, ZSInterface) manages a pre-defined number of signals in a block that, for example, could correspond to a protocol. This object could be of several types such as inputs and outputs for communication but also parameters for configurations and specifications.
- The **ZSignal** (ZS, ZSignal) class is the object containing the particular value of the signal and its configuration. It could also be a reference pointing at another ZSignal.
- The **ZSignalFlow** (ZSF, ZSFlow) class is the actual configuration of the signals between two SNodes including the needed types and identities.

As depicted in Figure 10, the SNode model is interconnected to the ZSNode class and the SFlow model to the ZSFlow class. The ZSNode includes DVs pointing at ZSInterface objects that in turn also have DVs pointing at ZSignal objects.

```
class ZSignalNode
{
private:
    ZIntegerVector*      m_cv;
    ZSignalInterfaceVector* m_iv;
    ZSignalInterfaceVector* m_ov;
    int                  m_state;
    bool                 m_change; m_storage;
public:
    static int m_signalNodeCounter; // object counter
    // constructor and destructor
    ZSignalNode(int cv[], int cv_size);
    ~ZSignalNode(void);
    // ----- configuration functions -----
    int verifyInInterface(int cv[], int cv_size);
    int addInInterface(ZSignalFlow* flow, dou. time);
    void removeInInterface(int iID);
    // corresponding functions for OutInterface
    void tryConfiguration(void);
    void configureAllSignals(void)
    void verifyAllSignals(void);
    void findSignalByID(ZSignalInterface* di);
    void findSignalByType(ZSignalInterface* di);
    void findSignalByBlock(ZSignalInterface* di);
    // ----- operational functions -----
    void update(int nodeID, double ntime);
    void setValues(interfaceID, dou* value, int size);
    double getValue(int interfaceID, int index);
    // ----- error and information functions -----
    void checkConfiguration(int cv[], int cv_size);
}; // end ZSignalNode
```

At simulation start, the ZSNode is created and initiated. A unique identity is then returned to the SNode model, which represents a pointer to the ZSNode object. The ZSNode object is at start in configuration state and awaits method calls for SFlow connections. As soon as the SFlow model is properly interconnected, it attempts to call respective ZSNodes by the “zTryConnectFlow” method. This checks the SFlow reference and then calls

for the “addInInterface” or the “addOutInterface” of the ZSNode object dependent on if connected to the A- or B-side. The “addInterface” method then creates a new ZSInterface object according to the configuration of the ZSFlow model and returns a unique line identity according to the ZSInterface object. Every time a new event occur in the SNode, the internal variable “change” is set, which start internal methods to find in- and output signals that are matching and then connects them dependent on configurations. As soon as the ZSInterface is correct interconnected, it is turned over to “operation” state and starts updating the signals continuously or according to a sample rate. Dependent on output interface configurations the ZSNode has methods to find input signals according to signal identity (findSignalByID), or type (findSignalByType). In the type case, there are also more specific methods searching particular block types. This is, for example, used when the output interface is of collection type, which is expanding according to the number of input interfaces including the requested signal type. In operation state the ZSNode only uses the “update” method that propagates the call to the affected ZSInterface objects.

```
class ZSignalInterface
{
private:
    // Internal variables
    ZIntegerVector*      m_cv;
    ZSignalFlow*         m_flow; // configuration
    ZSignalVector*       m_sv;
    ZSignalInterface*   m_siConnected;
    Int                  m_state
    bool                 m_active, m_change;
    bool                 m_storage, m_destination;
public:
    static int m_signalInterfaceCounter; // object counter
    // constructor and destructor
    ZSignalInterface(int cv[], int cv_size, ZSFlow* flow);
    virtual ~ZSignalInterface();
    // ----- configuration functions -----
    int getSize(void); // number of signals
    int getType( void ); // Interface type
    int getStorageType(void), getState(void);
    void setState(int state);
    int getGroupID(void);
    ZSignal* getSignalRefByIndex( int index );
    ZSignal* getSignalRefByID( int id );
    ZSignal* getSignalRefByType( int type );
    void connect(ZSignalInterface* di);
    void tryConfiguration(void);
    void verifyConfiguration(void);
    void addSignalElement(ZSignal* sObject);
    void removeAllSignalElements(void);
    bool change( void );
    // ----- operational functions -----
    void update(void);
    void setValues(double* value, int valueSize);
    double getValue(int index);
    double getSum(void), getMax( void ), getMin( void );
    // ----- check functions ( throwable ) -----
    void checkConfiguration(int cv[], int cv_size);
}; // end ZSignalInterface
```

The ZSInterface is based on a vector (`m_sv`) including a pre-defined number of ZSignal objects. The ZSInterface has several functions for configuration, mainly to interconnect and verify all included ZSignal objects to the intended destinations for updating the flow when turning to operation mode. If the ZSInterface object is of input type it has also a reference (`m_siConnected`) to an output ZSInterface of another ZSNode object. This gives that an input ZSInterface can never leave the configuration state until the reference-pointer points at an output ZSInterface in operational state. This is the actual control chain that implies that the source ZSInterface is the first one turning in operational state and then, step by step, permits the chain of ZSInterfaces to the last instance, the destination ZSInterfaces, to become operational. In the ZSInterface class, there are additional methods (`getSum`, `getMax`, `getMin`) normally used if utilizing the collector type, SFC model, which collects specific signal types.

The bottom class, ZSignal, includes the value of one signal and its signal specification. There are built-in functions to determine whether the signal should be locally stored or only point at another ZSignal object. The ZSignal class and its methods are shown beneath.

```
class ZSignal
{
private:
    int      m_signalID, m_signalType;
    int      m_blockType, m_groupID;
    double   m_value;
    double*  m_valueRef;
    bool     m_refOK, m_storage;
public:
    ZSignal( int bid, int id, int type, bool storage );
    virtual ~ZSignal();
    // ----- configuration functions -----
    void    setValueRef(double* value);
    double*  getValueRef(void);
    bool    refOK(void);
    bool    active(void);
    int     getGroupID(void);
    int     getBlockType(void);
    int     getSignalType(void);
    int     getSignalID(void);
    // ----- operational functions -----
    void    update(void);
    void    setValue(double value);
    double  getValue(void);
    // ----- check functions (throwable) -----
    void    checkConfiguration();
}; // end ZSignal
```

The update, `setValue` and `getValue` methods are the actual methods that the higher-level classes call for in operation state. Consequently, all signal management is only handled between the ZSignal objects that, in fact, are not aware of the other classes. They are only utilized to keep track and be prepared to change connections according to the signal configurations and routings.

7. Verification by samples

The “SignalFlow” library is verified by using all the components in several connections as depicted in Figure 11, which corresponds to the configuration example in Figure 3. At the left hand, there are two identical areas with internal control using the signal facilities. The SGIdentities are assigned 1 respective 2 that in this case also represents the two units. Unit 1 is not connected until 0.3 second after simulation start for testing of components added during simulation. Each unit has a SNode (SN) corresponding either to a communication intermediate storage area or a complete database for the unit where a number of signals are selected. All SNodes are assigned to 10 Hz sample rate. The SFlowOut of unit 1 and 2 are only assigned with the SIdentities and the SFlow models are assigned with SGIdentities, SBTypes, STypes and SIdentities enabling SFDistributors and SFCollectors to be used. The SFDistributor distributes two signals that, at start, are assigned 1.0 and 2.0. Reaching 0.8 second, the signals are increased 0.5 to 1.5 and 2.5.

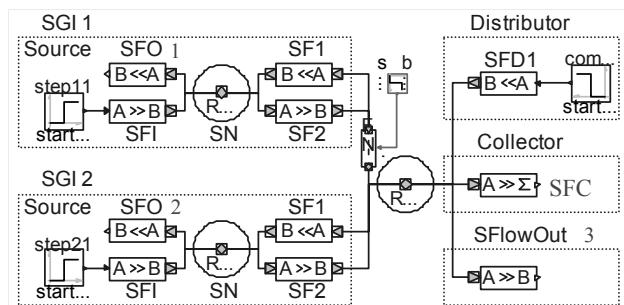


Figure 11. Signal system setup for validation of several different signal exchange possibilities

The signals are distributed to the SFlowOut (SFO1, SFO2) models of respective unit, which is depicted in the upper graph in Figure 12, where the initiating for configuration requires 3 samples. At this point, the SFlowOut2 is in operation state and updates the signal to the value 1.0. The SFlowOut1 should have been operated in the same manner but is not connected until time equal to 0.3 second, which then takes another 2 samples to be configured before turning to operating state. At 0.8 second, the two signals are directly increased to verify that there is no delay time in operation state. This example shows that the SNodes can handle altered configuration during simulation with only a few samples of delay and that the signals are distributed to the intended units. Moreover, in the other signal flow direction, the sources of unit 1 and 2 are constant 1.0 respectively 2.0. The SFlowOut3 are configured by SIdentities to connect these to signals. In the middle graph, Figure 12, this is also shown by first being delayed 3 samples before operating the signal from unit 2 and then additionally 2 samples for unit 1 due to the afterward connection at 0.3 second. Assigning the SBTypes and STypes configures the SFCollector in the third case. The bottom graph in

Figure 12 shows that the configuration events are the same as in the previous case. The value 1-3 corresponds to the sum, max and min functions, which are correct at 0.5 second, where the sum is equal $1+2=3$, max=2 and min =1. However, this example is configured with a small number of SNode and SFlow models that imply that few samples are needed for the configuration state. Consequently, in more complex system models the configuration sample delay increases but not necessarily in time, dependent on the sampling rate.

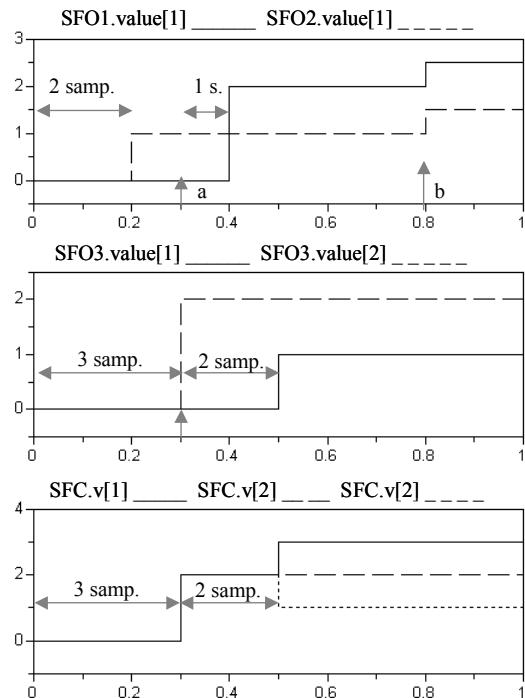


Figure 12. Simulation results for the validation model

A more complex illustration, where the SignalFlow library is frequently used is depicted in Figure 13, which represents a modular wind power plant (WPP) model in Dymola. The model is built from several model libraries, the “SignalFlow”, the “ControlFlow” mainly including control system related units, the “PowerFlow” (converters, cables etc.) and the “WindPower“ library (wind turbine units). The WPP model has several control unit levels and consequently several communication levels (CL). The four CLs included are the production control level (CL4), plant control level (CL3), process control level (CL2) and field control level (CL1). The principles are similar at each level including several controllable power units connected to a control unit via a communication bus. At each level, the SignalFlow library is used for exchanging signals. A single SNode represents different types of communication buses (B) dependent on the CL, and is also used for inter-process communication within a control unit. Between each CL, a control unit is interconnected that contains two signal exchange (E) units (SNode, SFlows) separating the CLs, and a number of software modules (M) controlling the

connected units. A module includes several function blocks (FB) that are connected by the SFlowIn and SFlowOut models to a SNode representing the local database. A selected number of signals are then exchanged from the database to the external bus using two SFlow models.

8. Conclusion

An adaptive signal management structure, based on object-oriented dynamical programming, is presented. A simulation library, “SignalFlow”, is developed in Modelica, where the model components are interconnected to the signal management structure. The presented results verifies that the structure meet the requirements. The structure forms a base level layer enabling adaptation for higher-level control and information flows. It is also a good example on how to develop function calls, interacting external programming languages with a simulation environment.

References

- [1] Svensson, J. and Karlsson, P., "Wind Farm Control Software Structure", *Third International Workshop on Transmission Networks for Offshore Wind Farms*, Royal Institute of Technology, Stockholm, Sweden, April 2002
- [2] Svensson, J., Karlsson, P. and Johnsson, A., "Information Structures for Scalable Distributed Power Systems", *3:rd IASTED International Conference on Energy and Power System*, Marbella, Spain, September 2003
- [3] Freiseisen W.; Keber R.; Medetz W.; Pau P.; Stelzmueller D., "Using Modelica for Testing Embedded Systems", *2:nd International Modelica Conference*, Proceedings, pp. 195-201
- [4] Pereira Remelhe, M.A., "Combining Discrete Event Models and Modelica - General Thoughts and a Special modeling Environment", *2:nd International Modelica Conference*, Proceedings, pp. 203-207
- [5] Modelica™ – *A Unified Object-Oriented Language for Physical Systems Modeling, Languages Specification, Version 2.0*, <http://www.modelica.org>
- [6] Svarstad, K.; Ben-Fredj, N.; Nicoleson, G.; Jerraya, A., "A Higher Level System Communication Model for Object-Oriented Specification and Design of Embedded Systems", *Conference on Asia South Pacific Design Automation 2001*, Yokohama, Japan
- [7] Dymola, Dynamic Modeling Laboratory, Dynasim AB, Lund, Sweden, <http://www.dynasim.com>
- [8] Al-Agtash, S., Al-Fayoumi, N. "A Software Architecture For Modeling Competitive Power system", *IEEE Transactions on Power Systems*, 2000, pp. 1674-1679

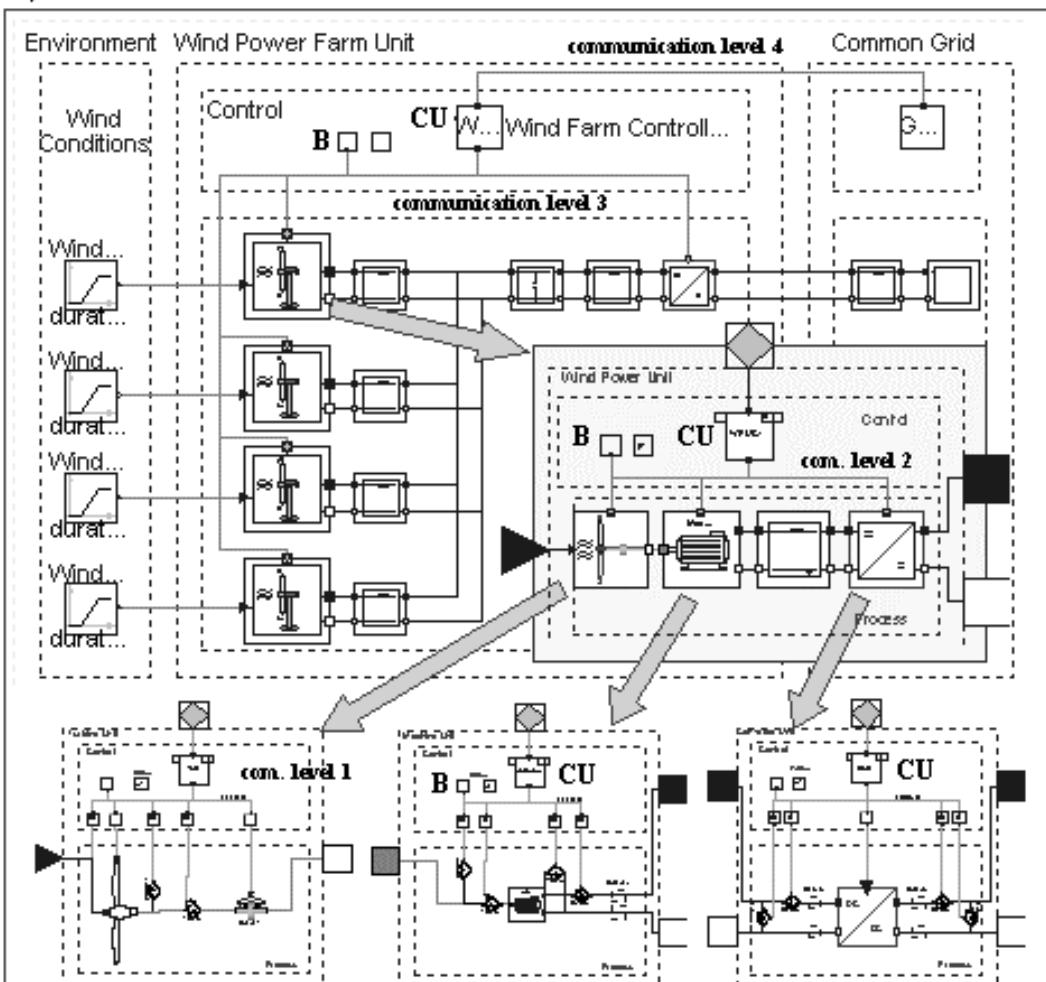
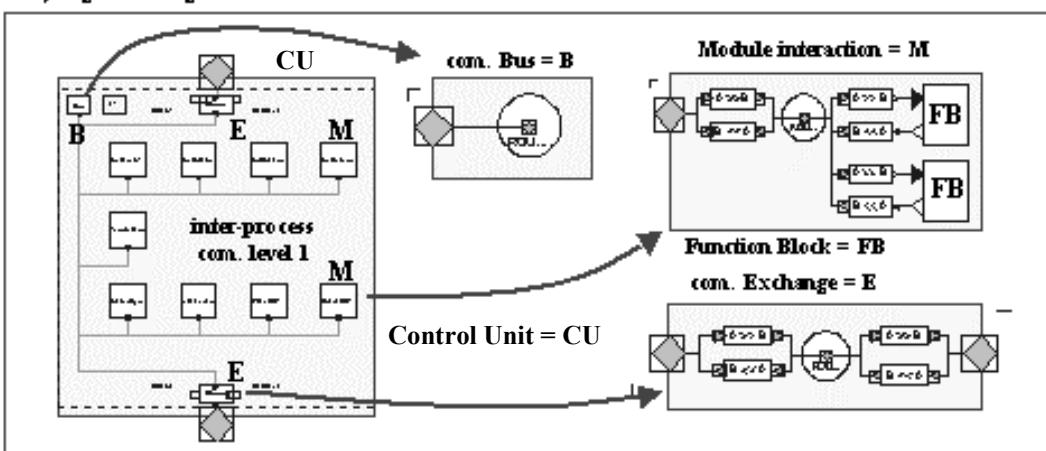
a) Wind Power Plant Model**b) Signal Management**

Figure 13. Modular simulation model of a Wind Power Plant including several communication levels (a), and signal management models used in the WPP model (b)

Object-oriented simulation of energy supply systems on the basis of renewable energy

Christian Hoffmann¹Jens Kahler²¹Technische Universitaet Ilmenau

P.O. Box 10 05 65, 98694 Ilmenau, Germany

Christian.Hoffmann@systemtechnik.tu-ilmenau.de

²De Montfort University

Water Software Systems, Queens Building, LE1 9BH, Leicester, UK

JKahler@dmu.ac.uk

Abstract

The usage of renewable energy sources has become more important in recent times mainly due to shortage of fossil fuels. Thus new challenges arise for the control and planning of heat supply systems. These systems work with different renewable energy sources (earth heat, solar radiation, exhaust air heat from building), which therefore lead to control problems with a higher rank of difficulty. The main goal of these control strategies is the reduction of operational cost as well as an increase in the rate of return for the investment.

This paper describes the simulation of a heat supply system using Dymola [13] by means of object-oriented modelling with Modelica. The final aim of such a simulation is to develop a model predictive control strategy on the basis of the proposed object-oriented model in Modelica. To develop such a control strategy it is necessary to gain knowledge about the heat demand of buildings and the power usage of its different components. The use of Computer Aided Engineering (CAE) can lead to a further reduction of development costs. A Dymola library named **RECOMB** has been created to support of modelling and simulation of such heat supply systems. The name **RECOMB** stands for **R**enewable **E**nergy **C**omponents **m**odelling and **optiM**isation of **B**uildings. The library itself consists of sub-libraries:

- Weather (chapter 1)
- Predict (chapter 2)
- HESYS (chapter 3)
- Buildings (chapter 4)

The components and the resulting models for the thermodynamic models of the sub-library HESYS

have been validated by comparing them with existing simulation software [11], [12], while the building models have been validated with help of German guidelines [9] as well as with other simulation software [8], [10].

1 The sub-library Weather

The sub-library Weather contains measured weather data for several German cities and a selection of other cities for validation purposes. The data has been obtained with the help of the database program Meteonorm. This program can supply various climatic information (e.g. total radiation, diffuse radiation, relative air humidity, wind speed, environmental temperature, etc.) in a desired output format (every second, hourly, daily).

Unfortunately, solar radiation is only available as radiation on a horizontal surface. But outside walls of building models normally are vertical and solar panels are also tilted to maximise radiation input. Therefore, a component is needed that allows correct calculation of radiation on a tilted surface from the available horizontal radiation data.

This component is called SolarRadiationTransformer (SRT). The SRT allows correct calculation of the radiation on a tilted surface. The formula of Liu and Jordan [4] has been implemented:

$$G_{\text{total, tilted}} = R_b \cdot G_{\text{hor, beam}} + \frac{1}{2} \cdot (1 + \cos \beta) \cdot G_{\text{hor, diffuse}} + \rho_{\text{env}} \cdot G_{\text{hor, total}} \cdot \frac{1 - \cos \beta}{2} \quad (1.1)$$

where

$G_{\text{total, tilted}}$ total radiation on tilted surface

$G_{hor,beam}$	direct radiation on horizontal surface
$G_{hor,diffuse}$	diffuse radiation on horizontal surface
$G_{hor,total}$	total radiation on horizontal surface
ρ_{env}	reflection factor of environment
β	angle of tilted surface

The calculated weather data is then passed on as information values to the connected models.

2 The sub-library Predict [6]

The sub-library Predict can be used for short-term prediction of climate data. This is essential for developing control strategies for regions where none or very few weather data can be provided. The library contains different methods to forecast climate values. Sometimes difficulties arise in data rededication, when climate data is not clearly correlated as this causes problems for most algorithms used for prediction. To handle these difficulties three different prediction methods had to be implemented:

- Persistence
- Stochastic methods, AR-modeling
- Heuristic methods
 - fast fuzzy based forecasting
 - Neural network

Prediction using the method of persistence is the simplest method of forecasting climate data. No model knowledge is needed to apply it, but it is only useful for slowly changing processes as the environmental temperature.

Autoregressive prediction (AR) is a model-based algorithm (figure 2.1). The predicted value for the next time-step is calculated by taking the preceding values into account and applying weighting factors to them. Nevertheless the most powerful method of prediction used in this library is the fast fuzzy prediction (figure 2.2.) This method compares the actual situation with situations already known from the past, which are similar to the

current one and then generates the new value. Since the algorithm uses known patterns from situations already occurred in the past, the method belongs to the group of pattern based prediction methods.

Finally, a method using neural networks has been implemented. This neural network works on the basis of a multi-layer perceptron to forecast the actual weather data (figure 2.3). Important filters to remove trends as well as modules for scaling values and to calculate statistical values for time series have been implemented into the sub-library Predict. To make full use of the Modelica language options, some parameter calculations are carried out using external C/C++ functions. The implemented methods of forecasting have been validated on short-term prediction with the available data from the program Meteonorm. In connection with the long-term data provided by weather institutes and databases, the combination of the two forms of climate prediction leads to a powerful tool for controlling and maximising the output of solar thermal and photovoltaic constructions. Again the predicted data is passed on as information values to the connected models.

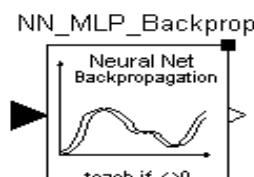


Fig. 2.3 Neural network prediction

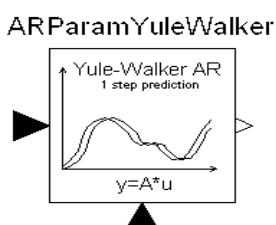


Fig. 2.1 Yule-Walker Parameter estimation

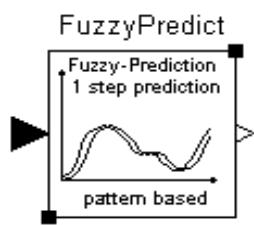
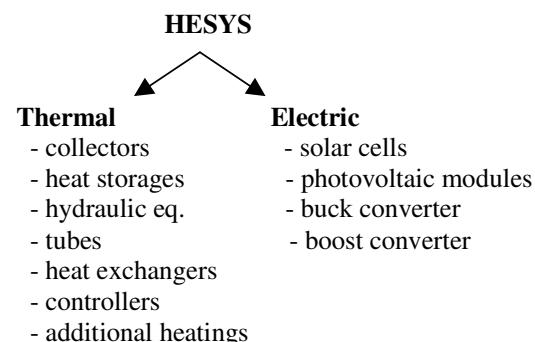


Fig. 2.2 Prediction using Fuzzy logic

3 The sub-library HESYS

This library contains all components necessary to model and simulate **Heat** and **Energy Supply SYStems**. Together with the sub-library **Buildings** (chapter 4) this is the core of the library **RECOMB**. The library itself is divided into two sub-libraries to separate those elements necessary to model and simulate solar-thermal systems, heat pumps and conventional heating from elements necessary to model photovoltaic systems.



Unlike other model libraries, the components of HESYS have been modelled especially for controlling aspects, as it is the main aim to develop efficient control strategies for the use of renewable energies. This aspect is also shown in the design of the connectors. For the use of the sub-library Thermal a HeatPort Connector as it can be seen in figures 3.1 – 3.3 has been implemented, which contains three variables. To completely describe the energy flow, only two variables would be sufficient (temperature T , heat flow \dot{Q}), but in this case the mass flow m has also been integrated. This is due to the fact that the mass flow later will become the actual control variable:

```
partial connector HeatPort
  "Thermal port for 1-dim. heat
  transfer"
  SIunits.Temp_K T "Port
  temperature";
  flow SIunits.HeatFlowRate Q_dot;
  flow SIunits.MassFlowRate m_dot;
end HeatPort;
```

Fig. 3.1 Heat Port Connector in Modelica

```
connector HeatPort_in
  "Thermal port for 1-dim. heat
  transfer"
  extends HeatPort;
end HeatPort_in;
```

Fig. 3.2 Heat Port In Connector in Modelica

```
connector HeatPort_out
  "Thermal port for 1-dim. heat
  transfer"
  extends HeatPort;
end HeatPort_out;
```

Fig. 3.3 Heat Port Out Connector in Modelica

The connector for the photovoltaic components of the sub-library Electric is based on the connectors from the general electric library from Modelica PositivePin and NegativePin. This also allows the use of components defined in that library. To demonstrate the use of some of the basic elements of the sub-library HESYS two simple examples, one for each of the underlying sublibraries, are shown.

Figure 3.4 shows a collector array with a size of $A = 4\text{m}^2$. The basis for the model of the collector is the static general collector model [1]:

$$\dot{q}_{abs} = a_0 \cdot G_0 - a_1 \cdot (T_{out} - T_{env}) - a_2 \cdot (T_{out} - T_{env})^2 \quad (3.1)$$

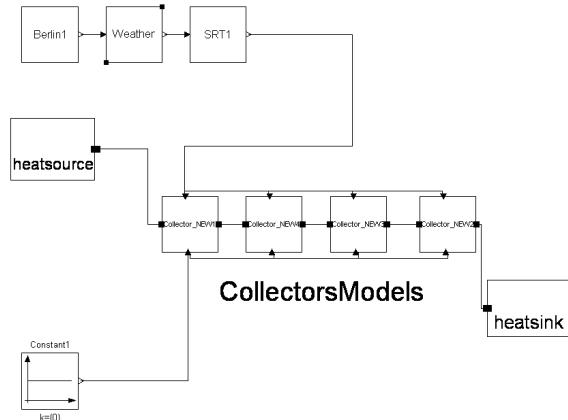


Fig. 3.4 Dymola/Modelica model of a collector field with 4 collectors in series

where

a_0 optical efficiency factor
 a_1, a_2 thermal loss coefficients
 \dot{q}_{abs} specific absorbed heat flow per m^2 collector size
 G_0 total radiation on surface
 T_{out} fluid temperature at collector exit
 T_{env} environmental temperature

The parameters used to obtain the results shown in figure 3.5 resample a Prinz Lux 2000 collector. The

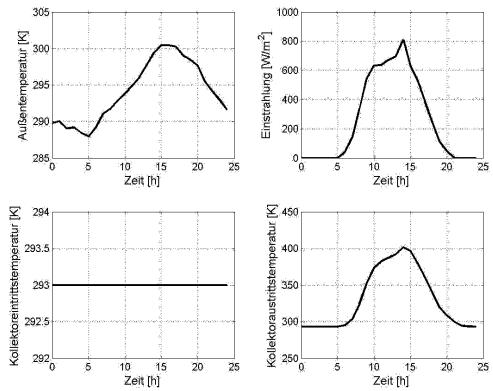


Fig. 3.5 Simulation results of a collector field

upper left graphic shows the environmental temperature over a one-day horizon and the graphics in the upper right the radiation on the collector surface. The two lower graphics show the temperature of the fluid entering the collector (left) and at the exit. The benefits can be seen clearly, as the fluid reaches temperatures around 100 degrees Celsius at the collector exit during noon.

The photovoltaic module as it can be seen in figure 3.6 consists of 36 solar cells connected in series.

The simulation model looks similar to the model of the collector array seen in figure 3.4.

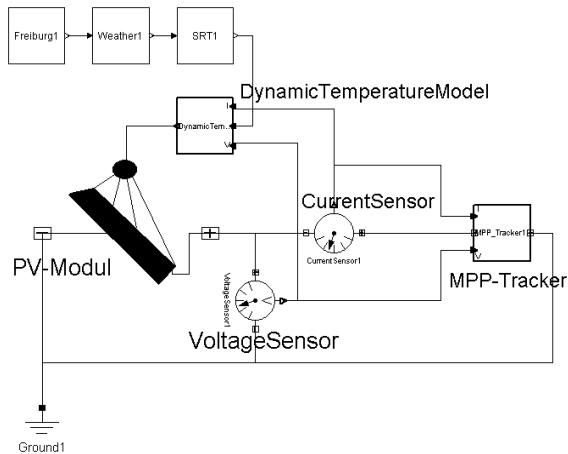


Fig. 3.6 Dymola simulation model of a photovoltaic module

The simulation shows, that several factors influence the output of a photovoltaic module, e.g. the temperature of the module itself and the point of maximum performance, obtained from the specific solar cell characteristics. For simulation purposes a solar cell can be considered a diode in its behaviour and 2-diode-model can be used to calculate current and voltage of the cell [1]:

$$0 = I_{ph} - I_{s1} \cdot (\exp(\frac{U + I \cdot R_s}{m_1 \cdot U_T}) - 1) - I_{s2} \cdot (\exp(\frac{U + I \cdot R_s}{m_2 \cdot U_T}) - 1) - \frac{U + I \cdot R_s}{R_p} - I \quad (3.2)$$

where

- R_p current losses along the borders of the cell
- R_s voltage drop
- I_{ph} photo current
- I_{s1}, I_{s2} satisfactionary currents
- m_1, m_2 diode specific parameters
- U_T voltage drop depending on temperature
- I current through cell
- U voltage drop over cell

The results can be seen in figure 3.7.

Again, the two upper graphics show the environmental temperature (left) and the radiation on the surface of the photovoltaic module (right) over 24 hours. The lower left graphics shows the temperature on top of the photovoltaic module and the power produced during that particular day can be seen in the graphics in the lower right corner of figure 3.7.

As it is the overall aim of the project to maximise the use of renewable energy sources both introduced models are only a smaller part of a system. In figure

3.4 the heat sink in reality is in most cases a water tank, the main purpose of which is to provide heated water to a connected household. This so heated water is then available for domestic use or to supply floor or ceiling heating, both becoming more popular in low-energy houses.

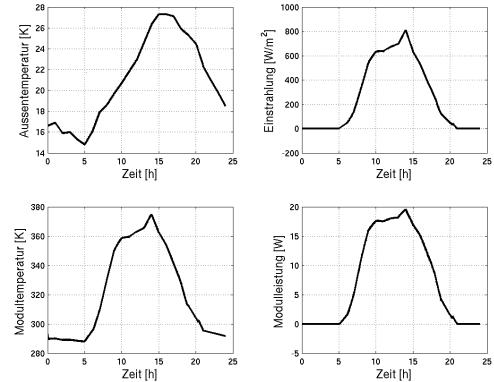


Fig. 3.7 Simulation results of a photovoltaic module

To simulate the heat demand of a regular household, the sub-library also offers components to model complete heat supply circuits, including tubes, pumps, flow-heaters and storages. One of the major parts in these systems is of course a water storage tank. At the moment two common types of storages tanks are available. A mixed-liquid storage tank, where the hot water is supposed to be mixed ideally with the colder water of the storage as well as the more frequently used stratified storages including a stratification charger to minimize heat losses. The latter ones are better in terms of cost and energy savings, since hot water is injected into a corresponding layer of water with almost identical temperature in the tank. This leads to less extra conventional heating. Nevertheless mixed-liquid tanks are still widely used due to their lower investment costs. If there is no internal or external heat exchanger used, the change in storage temperature can be calculated [3]:

$$\frac{dT_{storage}}{dt} = \frac{\dot{Q}_{collector} - \dot{Q}_{used} - \dot{Q}_{loss}}{c_{p,storage} \cdot \rho_{storage} \cdot V} \quad (3.3)$$

where

- $\dot{Q}_{collector}$ heat flow from the collector
- \dot{Q}_{used} heat flow from storage to user
- \dot{Q}_{loss} heat loss through walls, top and bottom of tank
- $c_{p,storage}$ specific heat capacity of stored fluid

$$\begin{aligned}\rho_{\text{storage}} & \text{ density of stored fluid} \\ V & \text{ Volume of storage tank}\end{aligned}$$

A complete solar thermal model with storage tank can be seen from figure 3.8.

In the same sense, the energy produced by the photovoltaic components can be used. The energy can be used directly to feed domestic energy sinks (e.g. TV, bulbs) using buck or boost converters.

The results obtained from both the solar thermal and the photovoltaic models have been compared with results from other common simulation software packages like T-Sol [12] and the Simulink Toolbox CARNOT [13]. The results justify an approach using the modules from the sub-library HESYS.

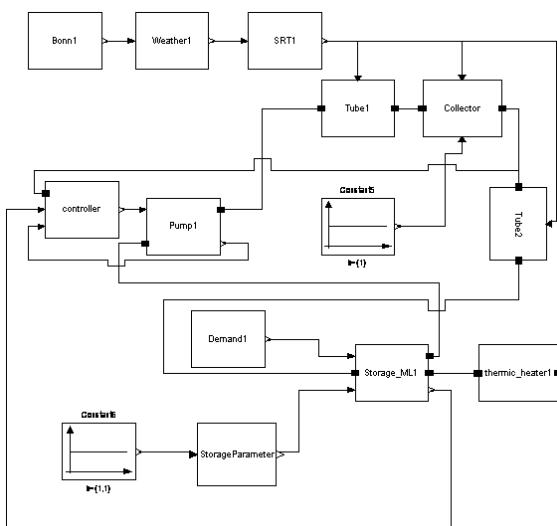


Fig. 3.8 Dymola simulation model of a simple heat supply system

4 The sub-library Buildings [5]

An essential part in developing efficient control strategies to work with renewable energy components is to have high knowledge about the thermal behaviour of the actual energy sink. In this case, a building, not necessarily a low-energy house, is considered to be the main energy consumer. Therefore, the sub-library Buildings contains components (walls, windows, models of exchange of radiation inside rooms) to build realistic models of any kind of buildings.

There exist already several libraries to simulate thermal behaviour, even one designed with Modelica [10]. Still the authors decided to develop one of their own. This again was driven by the control approach, as it was necessary to connect the components from the HESYS with building components as well as with the controllers.

Nevertheless, both libraries share similarities, for example the modelling of heat transmission between walls or other heat emitting elements inside a room (machines, persons). The library allows the use of the two-star model [7], which approximates the heat transmission. This model is very efficient for non-rectangular rooms or rooms with additional heat emitters. For validation purposes, exact models to calculate heat exchange between opposite or perpendicular heat radiation emitting elements [9] are implemented as well.

Major differences can be found in the implementation of walls of windows. Even though a wall can easily be divided into objects for different layers, it is considered to be a better approach to implement a wall as a single object. This allows a more exact realisation of a wall consisting of different layers.

Starting from the partial differential equation from Fourier:

$$\rho \cdot c_p * \frac{\partial T}{\partial t} = -\lambda \cdot \frac{\partial^2 T}{\partial t^2} + \dot{q}_i \quad (4.1)$$

with

$$\begin{aligned}\rho & \text{ density of material} \\ c_p & \text{ specific heat capacity} \\ \lambda & \text{ heat conductivity} \\ \dot{q}_i & \text{ heat flux density} \\ T & \text{ absolute temperature}\end{aligned}$$

it is useful to then generate a discrete approximation of this equation. In [5] it is shown that it is more accurate to use an implicit method for differentiating. Therefore, the reverse quotient for differentiating is used to calculate the temperature distribution through the wall. Since a wall normally consists of layers of different materials, it can be physically justified to consider each wall layer a numerical layer. This leads to the following system

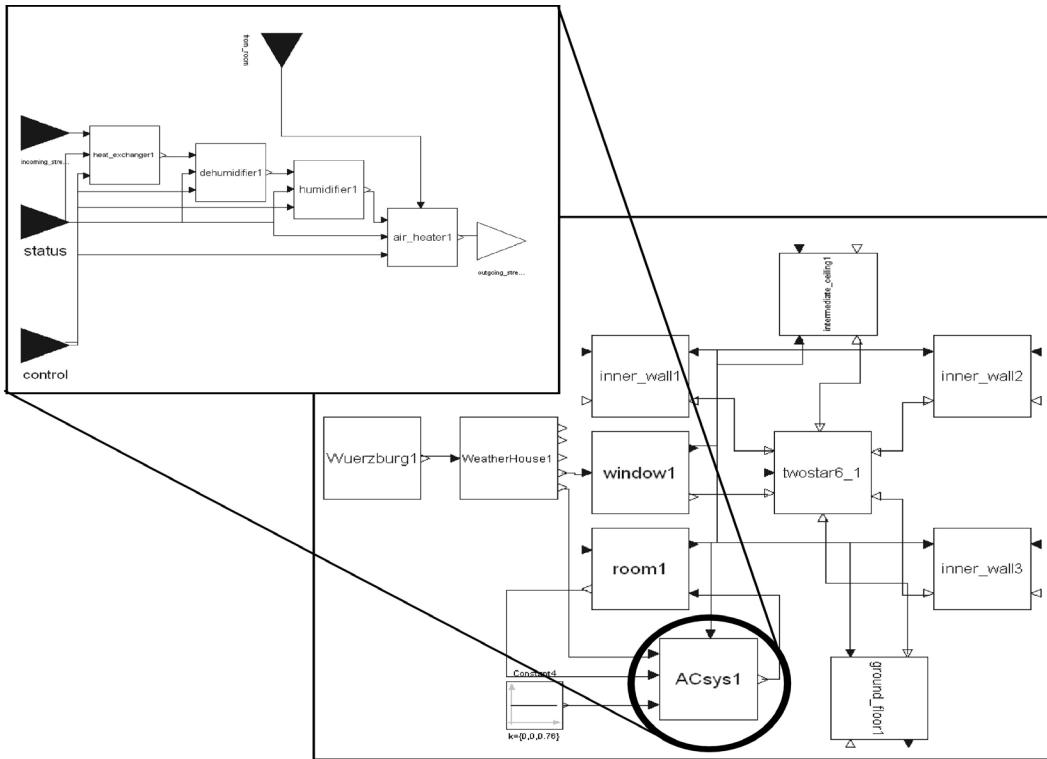


Fig. 4.1 Dymola simulation model of a simple room with air conditioning

of equations for heat conduction inside a wall element:

$$\frac{dT_i}{dt} = \begin{cases} \frac{a_1}{d_1^2} \cdot (T_2 - 2 \cdot T_1 + T_{\text{outside}}) & \text{if } i=1 \\ \frac{a_i}{d_i^2} \cdot (T_{i+1} - 2 \cdot T_i + T_{i-1}) & \text{if } 1 < i < n \\ \frac{a_n}{d_n^2} \cdot (T_{\text{inside}} - 2 \cdot T_n + T_{n-1}) & \text{if } i=n \end{cases} \quad (4.2)$$

with

- a thermal diffusivity
- d thickness of layer
- n number of layers

The temperatures on the inside and the outside of the wall can be calculated using the heat balances on these surfaces:

$$0 = \dot{Q}_{\text{convection, outside}} + \dot{Q}_{\text{longwave}} + \dot{Q}_{\text{shortwave}} \quad (4.3)$$

$$+ \lambda_1 \cdot \frac{A}{d_1} \cdot (T_1 - T_{\text{outside}})$$

$$0 = \dot{Q}_{\text{convection, inside}} + \dot{Q}_{\text{longwave}} + \dot{Q}_{\text{shortwave}} \quad (4.4)$$

$$+ \lambda_n \cdot \frac{A}{d_n} \cdot (T_{\text{inside}} - T_n)$$

where

$\dot{Q}_{\text{shortwave}}$ heat transmission caused by short-wave radiation

$\dot{Q}_{\text{longwave}}$ heat transmission caused by long-wave radiation

$\dot{Q}_{\text{convection}}$ heat transmission caused by convection on inside or outside of element

A area of wall

d_1, d_n thickness of first/last wall layer

λ heat conductivity of wall layer

T absolute temperature

The same approach is used for the model of the windows. As originally low-energy houses have been considered, a window can also be seen as a wall element with, in case of double-glazing, at least three layers, where the intermediate layer is a vacuumed space or filled with gas. But heat transmission through a layer of glass is different to that of a wall. To generate an exact model this

has to be considered. Shortwave radiation does normally not get absorbed in glass layers. Nevertheless, after being absorbed by objects in the room it is transformed and emitted as long-wave (heat) radiation. For this radiation, glass layers are impenetrable. This leads to a natural heating of the room and this effect is commonly known as the greenhouse effect. Furthermore, for glass layers reflection and absorption between layers has to be considered. This is accomplished by using the ray-tracing method.

Another major part of the sub-library next to model buildings is to model the interior behaviour of a room. This is again done with respect to the overall aim of developing the control strategies. To minimise conventional heat sources, low energy houses also make use of heat recovery in air conditioning systems. Hence, these systems obviously have an influence on the total heat absorbed or emitted inside a room. Therefore, it has to be implemented in a complete model as well as conventional heat suppliers like radiators. The sub-library allows to model realistic air conditioning systems, which can be extended to a heat recovery system due to the object-oriented nature of the simulation environment.

Another extension of the available library is the possibility to simulate the room behaviour with respect to hygienic factors. The general purpose of the library was also to generate a room climate that is generally considered comfortable by a human being. Therefore in addition to heat transmission, emission and absorption, factors for relative humidity, moisture on the insides of walls and air pollution by CO₂ have to be considered. Furthermore, it is possible to control the room behaviour with simple PID – controllers as well as with state space controllers. This already leads to a good impression of the total heat consumption of a single room or house under different requirements of comfort.

The models generated using the sub-library Buildings have been validated using an exemplary configuration for a room under certain conditions obtained from [9] as well as comparing the results with those from the “Baukonstruktionslehre” (BKL) – house introduced by Feist [7]. The Verband Deutscher Ingenieure (VDI) offers a range of examples in one of their guidelines to compare total heating and/or cooling load or for a selected week of rooms under defined conditions. Figure 4.2 shows the complete model of the VDI example 13 as it appears in Dymola.

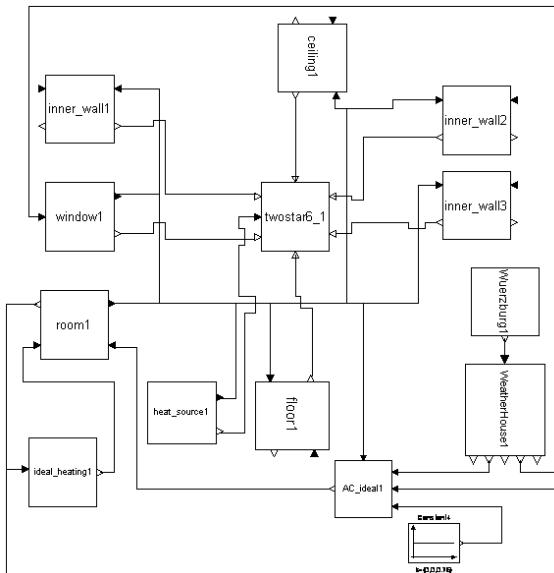


Fig. 4.2 Dymola simulation model of the VDI example room 13

The VDI has used five different simulation programs to get results for the maximum heating and cooling load as well as for the total heating and cooling load over a one-year period. The results have been grouped into a minimum, a maximum and an average. A model can be suggested as validated, if the simulated values for the mentioned are in between the range of the other five simulation programs.

As it can be seen from figure 4.3 the model build with components from RECOMB fulfil this requirement, as only the maximum value for the cooling load violates the constraint by eight per cent.

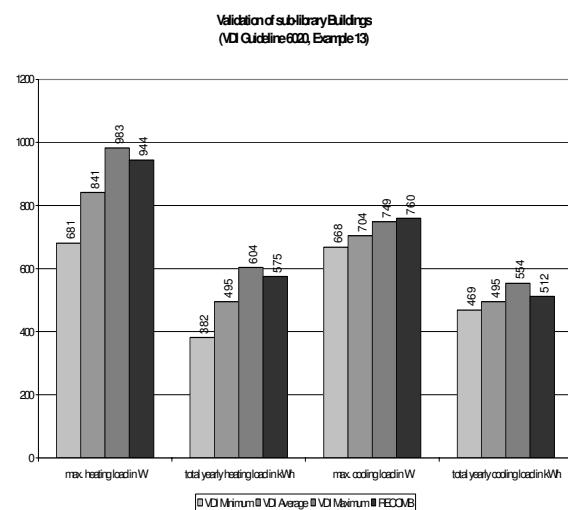


Fig. 4.3 Validation results VDI - RECOMB

Comparing the results with the BKL – Vergleichshaus, has also validated the model. The difference is, that the BKL Vergleichshaus is actually a three 2-level houses in a row, sharing one side. Thermodynamically speaking, the one in the middle is the most interesting one, as it is connected to the other houses on its east and west side. Only the front (facing south) and the back (facing north) are in contact with the environment as well as the roof. Figures 4.4 shows the different temperatures in the ground floor and the 1st floor as they appear in the middle section of the BKL – Vergleichshaus. It can be seen that the temperature in the top room (upper trajectory) varies more than the temperature in the lower room. Also due to extra heat gains during the summer through the flat roof the temperature in 1st floor is higher than in the ground floor room.

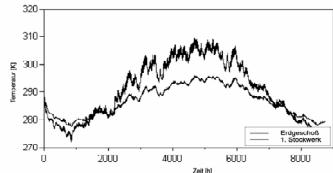


Fig. 4.4 Room temperatures for the BKL - Vergleichshaus

5 Further work

The development of the introduced library RECOMB is an ongoing process and is far from completion. As has been mentioned briefly in the introduction, the overall aim of this library is to develop powerful and efficient control strategies to minimise the use of conventional heat resources. Therefore, a major part in the near future will involve developing modules to calculate optimal control trajectories to work with the heat and energy supply systems and/or air conditioning systems inside rooms. This might eventually lead to re-designing some of the already existing modules.

Also the library aims to give help to another area important for maximising the benefits from renewable energy resources. Already being developed but far from being presentable, the authors currently work on modules to help determine the optimal size of a heat and energy system, e.g. optimal size of a collector field for a normal houses.

Finally, as seen from the introduced figures of the library, the graphical output of the models has to be improved to offer the user an easier way of generating simulation models. Well-designed

icons for buildings and heat and energy supply components will further improve the usability of the library.

6 Acknowledgements

This research was partially supported by the EPSRC Grant No. GR/N26005.

7 References

- [1] V. Quaschning, (1999), Regenerative Energiesysteme, 2. Auflage, Carl Hanser Verlag
- [2] Duffie and Beckmann, (1991), Solar Engineering of Thermal Processes
- [3] H. P. Garg, S.C. Mulick and A.K. Bhargava, (1985) Solar Thermal Energy Storage, D.Reidel Publishing Company
- [4] B.Y.H. Liu and R.C. Jordan (1963), The Long-Term Average Performance of Flat-Plate Solar Energy Collectors, Solar Energy Vol.7
- [5] J. Kahler, (2002), Entwicklung einer Gebäudekomponentenbibliothek für kontrolliert-belüftete und -entlüftete Niedrigenergiehäuser, Diplomarbeit TU Ilmenau, Inst. für Automatisierungs- und Systemtechnik
- [6] M. Kissler, (2002), Entwicklung einer Prognoseverfahrensbibliothek für Klimadaten in Dymola/Modelica, Diplomarbeit TU Ilmenau, Inst. für Automatisierungs- und Systemtechnik
- [7] W. Feist, (1994), Thermische Gebäudesimulation: kritische Prüfung unterschiedlicher Modellierungsansätze
- [8] C. Nytsch-Geusen , (2001), Berechnung und Verbesserung der Energieeffizienz von Gebäuden und ihren energietechnischen Anlagen in einer objekt-orientierten Simulationsumgebung, Dissertation, TU Berlin
- [9] VDI Richtlinie 6020, (2001), Anforderungen an Rechenverfahren zur Gebäude und Anlagensimulation
- [10] F. Felgner et al., (2002), Simulation of thermal building behaviour in Modelica, Proceedings of the 2nd International Modelica Conference
- [11] Solar-Institut Juelich, (1999), CARNOT Blockset Version 1.0 User manual
- [12] Dr. Valentin EnergieSoftware GmbH, (1999), T-Sol 2.0 User manual
- [13] Dymola User's Manual, (2002), Dynamsin AB Lund

Implementation of a Modelica Library for Simulation of Refrigeration Systems

Torge Pfafferott* Gerhard Schmitz†

Technical University Hamburg–Harburg, Department of Technical Thermodynamics
Denickestr. 17, D–21073 Hamburg

October 2003

Abstract

The physical modelling and transient simulation of refrigeration systems can be useful within the specification, development, integration and optimisation. Therefore, a model library for vapour compression cycles has been implemented. The library is based on the free Modelica library ThermoFluid and contains basic correlations for heat and mass transfer and pressure drop, partial components for control volumes and flow resistances and advanced ready-to-use models for all relevant components of refrigeration systems like pipes, heat exchangers, compressor, expansion devices and accumulator. The library currently enables the use of two refrigerants (CO_2 , R134a), but due to the structure of the library the extension to other refrigerant medium models is quite easy to realise. The modelling approach, the structure of the library and some validation results are presented in this paper.

1 Introduction

The modelling and simulation of refrigeration systems is of interest for several problems:

- Development and testing of control strategies and controller configurations
- Prediction and investigation of cycle dynamics like cool-down performance, start-up behaviour, pressure gradients and torque at the compressor
- Prediction of power consumption and COP (Co-efficient Of Performance)
- Design and evaluation of heat exchangers

- Determination of optimal refrigerant charge
- Integration of refrigeration system as a subsystem within other systems like air-conditioning systems of automotives, buildings and aircrafts
- Development of combined heat pump systems for cooling and heating
- ...

This listing is incomplete but it clarifies the need of transient analysis of refrigeration systems by dynamic simulation.

The development of a Modelica library for refrigeration systems is being realised within two joint research projects founded by Airbus Deutschland GmbH, Hamburg, and DaimlerChrysler AG, Stuttgart. The aim of both projects is the development of a tool for transient simulation to support the research and development of new refrigeration technologies and to optimise currently used systems. In general, both projects focus on vapour compression cycles. A schematic of a vapour compression cycle is shown in Figure 1 (left figure). The main components are compressor, condenser, accumulator, expansion valve and evaporator. The working fluid is compressed in the compressor from suction line state to the high pressure line. In the following condenser the heat is rejected isobaric from the working fluid to the ambient or to a secondary coolant. After the condenser the accumulator is placed, where the subcooled fluid is dried by a desiccant. The expansion valve throttles the fluid isenthalpic to the low pressure level. In the evaporator the fluid is evaporated and superheated isobaric by removing a heat flux from the cooling medium.

The working fluid has to fulfil several requirements depending on the area of application. One of these

*pfafferott@tuhh.de

†schmitz@tuhh.de, tel.: +49–40–42878–3144

requirements is the environmental sustainability of refrigerants. At the moment, this fact is an important driving force for the development of new refrigeration technologies, since the prohibition of currently used refrigerants is discussed. In the beginning of the 1990s the HCFCs based refrigerants were prohibited due to their ozone depletion potential (ODP). The used substitutes, the so called CFCs, have no ODP but the global warming potential of these refrigerants is very high in comparison to other, so called natural refrigerants like water, CO₂, ammonia or hydrocarbons (propane, butane) [1]. Therefore, the prohibition of CFCs is discussed within the European Union and it seems to be realistic, that these refrigerants will be prohibited until the end of this decade [2].

Carbon dioxide (CO₂, R 744) as a natural refrigerant was rediscovered and has recently demonstrated a very high potential to substitute currently used refrigerants in the area of mobile/automotive air-conditioning and cooling [3], [4]. This development is caused by the thermodynamic, transport and environmental properties of CO₂. A schematic of a transcritical CO₂ cycle is shown in Figure 1 (right figure). The cycle consists of the same components as other vapour compression cycles, but it is supplemented by an internal heat exchanger. The internal heat exchanger is an essential component in a CO₂-cycle to realise an acceptable COP.

For the automotive application the working fluid R134a is state-of-the-art. Therefore one purpose of the developed library is focused on this area. In the automotive application the AC-system is one important key for the passenger's comfort. On the other hand the AC-system influences the fuel consumption and the emission of the vehicle. For future automotives and for the aircraft application new, CO₂-based refrigeration technologies are investigated [5]. The support of this development is also one purpose of the library. In this paper the basis of the library and an outline of the library content is given. Furthermore validation results of the CO₂-models are presented and discussed with regard to the measurement uncertainty and the uncertainty of parameters.

2 Library for refrigeration systems

The aim of the modelling is to implement a library with physical based models of components of refrigeration systems. At the moment the library enables investigations with two refrigerants (CO₂, R134a). But the realised structure allows the extension of the

library by other refrigerants. Such a library can be used for investigations of components and complete refrigeration cycles. Furthermore it is of great interest to conduct dynamic simulation as well as steady state simulation of systems and single components, especially heat exchangers. This should be able with one tool and using the same models. The numerical investigation of heat exchanger components is of particular interest to find optimised heat exchangers for limited space. On the other hand, the concept of connectors in Modelica provides the opportunity of using the same heat exchanger models for single component simulation as well as for a complex cycle simulation. Finally, the library can be used for simulation and evaluation of different system designs in various applications.

2.1 ThermoFluid library

The implemented refrigeration library is based on the free Modelica library ThermoFluid [6], [7], [8]. The ThermoFluid library, especially its base classes and partial components, offers a good base for the modelling of refrigeration systems with respect to the implementation of the three balance equations and the method of discretisation. The basic design principles of the library are:

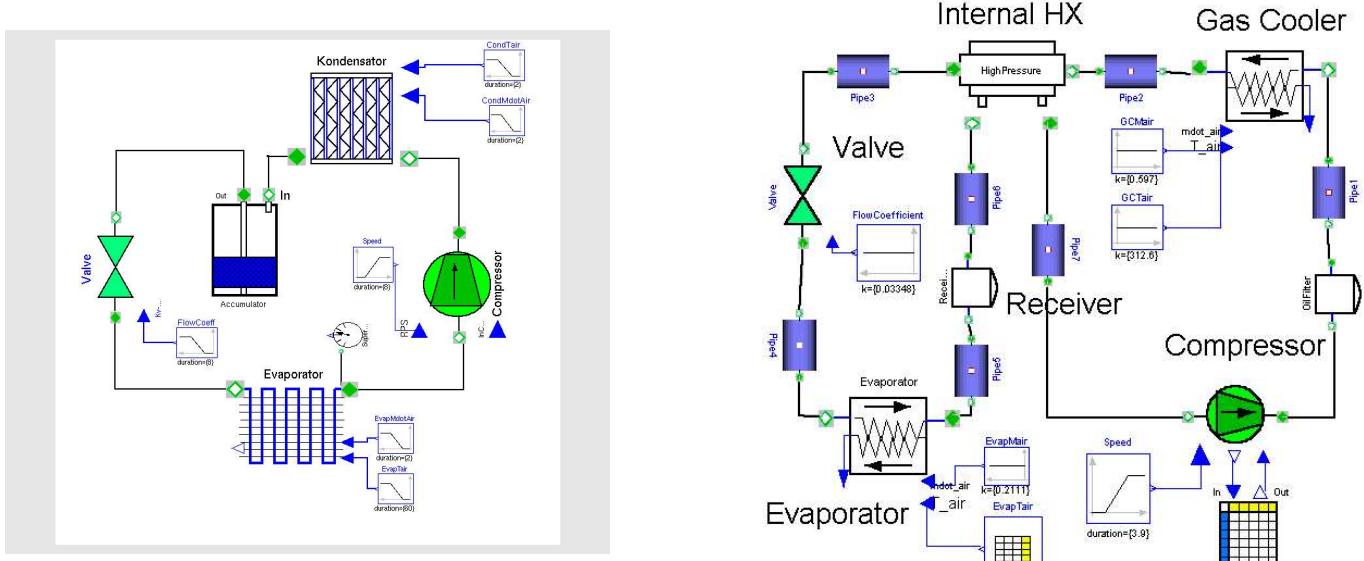
- models are designed for system level simulation,
- one-dimensional one- and two-phase flow is considered,
- one unified library for lumped and distributed parameter models,
- bi- and unidirectional flows are supported,
- conservation laws are implemented separately from the medium models, in order to improve reusability.

The use of distributed parameter models suggests the finite volume method as discretisation method. The finite volume method is very common for system modelling and one-dimensional discretisation [9]. The thermodynamic model holds the equations for total mass and internal energy for a control volume with constant volume:

$$\frac{dM}{dt} = \dot{m}_{in} - \dot{m}_{out} \quad (1)$$

$$\frac{dU}{dt} = \dot{m}_{in} \cdot h_{in} - \dot{m}_{out} \cdot h_{out} + \dot{Q} + \dot{W}_s \quad (2)$$

The fluxes on the border of the control volume are calculated by the half grid staggered flow model, which

Figure 1: Schematic of vapour compression cycle and of a transcritical CO₂ process

holds either a stationary pressure drop model or the dynamic momentum balance:

$$\Delta z \cdot \frac{dm}{dt} = \dot{I}_{in} - \dot{I}_{out} + (p_{in} - p_{out})A - \Delta p_{loss}A - M \cdot g \cdot \sin(\beta). \quad (3)$$

The state variables of $\{M, U\}$ for the thermodynamic model are numerically not efficient. Therefore, the equations (1) and (2) are transformed into a form with $\{p, h\}$ as state variables. The constitutive equations needed for the calculation of pressure drop and heat flow in the equations (2) and (3) are not implemented in the ThermoFluid library yet.

2.2 Content of the library

So far, the most important models and classes have been implemented in the model library. As already mentioned the structure of the library enables the extension by other refrigerant models. This leads to a separation into a more general part and a specialised part. The general part holds the implementation of base and partial models like heat transfer and pressure loss correlations or flow resistance. All these models are independent of a specific medium model. Nevertheless, most of the models need medium properties for their execution. Therefore, the different medium models have to be implemented in the same way using the same names of variables and records and realising the same structure. In the specialised part of the library holds ready-to-use models.

Some of the models which have been implemented in the library are:

- Heat transfer and pressure loss relations for the whole fluid region:**

These constitutive equations are used for the calculation of heat flux and pressure drop due to friction, which are terms within the balance equations of energy and momentum. Most of the implemented correlation are state-of-the-art:

The heat transfer of *single phase* flow can be calculated with Nusselt-Number based correlations for laminar and turbulent flow and smooth transition between both [10]. The heat transfer coefficient for *condensation* is computed by the assumption of film condensation using the correlation from *Shah* [11]. For the *boiling heat transfer* a superposition model introduced by *Chen* can be used, which takes forced convection and nucleate boiling into account [11]. These correlations are valid for two phase flow of refrigerants in general. A more accurate correlation has been implemented for the evaporation of R134a, which is based on the superposition model and has been adapted by *Gungor and Winterton* [12]. All these correlations calculate local heat transfer coefficients. Therefore, a discretised modelling of flow with heat transfer is required.

The pressure drop of single phase flow is calculated depending on the Reynolds-Number [13]. To avoid event iterations, a function has been implemented generating a smooth transition be-

tween the different correlations and the areas of validity [16]. The pressure drop of two-phase flow is computed by a correlation of *Friedel*, which is simple but fairly accurate [10]. The implemented correlations have been validated for R134a [12] and CO₂ [14] with experimental data and they show mostly a fair accuracy. The advantage of using the same correlations for both refrigerants is the simplicity of their implementation. The loss of accuracy by not using specialised correlations is acceptable since the most important heat transfer is outside the pipe.

- **Models for the air side of heat exchangers:**

The balance equation of energy is implemented by the finite volume method [9]; heat transfer correlations for the air side have been implemented [15] as well as medium properties of air using polynomial fitting. The condensation of humidity is taken into account in the energy balance but not in the heat transfer coefficient. The approach for modelling the condensation is quite simple: If the air temperature is equal to or below the dew point temperature of the humid air, condensation occurs. By this approach the humid air is saturated at the outlet.

- **Medium models:**

The medium properties are calculated based on the implementation in ThermoFluid. The medium models for R134a and CO₂ have been customised with regard to the implemented constitutive equations; e.g. the transport properties, the phase boundary properties and the surface tension are calculated within the medium model. To avoid event iterations at the phase boundary a crossing function has been implemented generating a smooth transition between the model of the two phase and that of the single phase area.

- **Pipes and heat exchangers:**

Based on the medium model, classes of ThermoFluid, the heat transfer and pressure drop correlations and the air side models, pipes and heat exchangers have been modelled. The pipes are modelled as one-dimensional discretised flow just like the air flow itself. By this assumption the refrigerant flow is treated as homogenous. For the air side a non-homogenous distribution of the air mass flow can be provided. Since the heat exchangers are built up from basic elements for the refrigerant flow, the wall and the air flow, different types of heat exchangers can be modelled eas-

ily. Due to the discretisation of the flow, the state of the refrigerant and the heat transfer along the heat exchanger can be predicted with the models. Up to now several types of counter flow, cross flow and cross-counter flow heat exchangers have been modelled and validated successfully. A more detailed description of the modelling of the heat exchangers is given in [16] and [17].

- **Compressor:**

The model is made for a reciprocating compressor. Therefore, the mass flow is calculated by the general equation (4) of a reciprocating compressor and the enthalpy change is calculated according to the isentropic efficiency by equation (5):

$$\dot{m}_{co} = f \cdot \lambda \cdot \rho_{in,co} \cdot V_{dv} \quad (4)$$

$$\Delta h_{co} = h_{out,co} - h_{in,co} = \frac{h_{out,co,is} - h_{in,co}}{\eta} \quad (5)$$

By using these equations the compressor is assumed to have no dynamics. The efficiencies can be provided by measured characteristic fields of a known component or are set as constant parameters if they are unknown and must be estimated.

- **Expansion valve:**

The throttling process is treated as isenthalpic and the pressure drop is calculated according to the flow coefficient of the valve using an algebraic equation [18]:

$$\dot{m}_{ev} = \frac{1}{3600} \cdot K_V \cdot Y \cdot N_6 \cdot \sqrt{x \cdot p_{in,ev} \cdot \rho_{in,ev}} \quad (6)$$

where is:

$$x = \frac{p_{in,ev} - p_{out,ev}}{p_{in,ev}} \quad (7)$$

$$Y = 1 - \frac{x}{3 \cdot F_\gamma \cdot x_T} \quad (8)$$

Since the flow coefficient K_V and the critical differential pressure ratio x_T result from specific valve data and construction, the model has to be parameterised with corresponding data.

- **Accumulator / Receiver:**

In general, the function of an accumulator or a receiver in a refrigeration system is to accumulate refrigerant, since the necessary refrigerant charge depends on the operating mode of the system. Therefore, additional charge has to be stored. In R134a systems the accumulator is placed after the

condenser and contains a desiccant for drying the refrigerant, see Fig 1. Whereas the receiver in CO₂ systems is placed at the suction line after the evaporator, see Fig 1. For modelling, the receiver is separated in a separator, a tube for the gaseous outflow, an orifice for the liquid outflow and a junction mixing the two outflows. This modelling approach is similar to [19]. The incoming two phase flow is separated into its liquid and vapour phase. The outlet condition is calculated by mixing the two mass flows through the tube and the orifice, which are modelled as flow resistances with specific friction factors. The friction factors can be estimated for steady state; then the vapour fraction at the receiver outlet is the same as at the receiver inlet. The receiver is modelled as adiabatic.

- **Flow splits and junctions:**

For these models, classes of ThermoFluid are used. The pressure drop in the momentum equation uses special correlations for splits and junctions taking the ratio of mass flow into account [20]. The change of mass flow direction is also taken into account in the implementation.

and fans. The temperature of CO₂ at inlet and exit of each component is measured with thermocouples attached to the surface. The pressure is also measured at inlet and exit of each component. The CO₂ mass flow rates are measured at different points in the system by using Coriolis type meters. Hot-wire anemometers are used to measure the air mass flow rates through the heat exchangers. The air inlet and exit temperatures are measured by thermocouple grids. The uncertainties for the measurements are listed in Table 1. Especially the uncertainties of the air temperature after the gas cooler is very high due to the inhomogeneous distribution of temperature. Due to error propagation the resulting uncertainty of the calculated capacities can be up to ± 12 % for both gas cooler and evaporator.

Table 1: Absolute, resp. relative error of measurement

Pressure at suction line	± 50 kPa
Pressure at high pressure side	± 100 kPa
CO ₂ temperature	± 0.7 K
Air temperature evaporator in/out	± 1 K
Air temperature gas cooler in	± 1 K
Air temperature gas cooler out	± 3 K
CO ₂ mass fbw rate	± 0.2 %
Air mass fbw rate	± 4 %

3 Experimental setup

The experiments were carried out at the CO₂-experimental system built at the Department of Aircraft Systems Engineering of the TUHH described in detail by Schade [21]. The test rig was constructed with prototype components of the automotive application. It realises the process of a transcritical cycle introduced by Lorentzen and Pettersen [22], which is extended in the realisation by three, parallel cooling points/evaporators. The main objective of the experimental investigations is control-oriented. Furthermore, steady state and transient data from the test rig should be used for the validation of the simulation models. The gas cooler is a cross-flow heat exchanger with three passes at the refrigerant side. The evaporators are cross-counter flow heat exchangers with eight passes in two layers. The internal heat exchanger is built as a counter-flow heat exchanger with coaxial tubes. The used compressor consists of an axial piston unit with variable or fixed displacement. The gas cooler is installed in an open channel whereas the evaporators are built in closed loop air-cycles. The temperature and mass flow rate of the air at the heat exchanger inlet is conditioned by electrical air heaters

4 Validation of air-CO₂ heat exchanger models

Simulations in a test configuration have been run with the gas cooler model discretised with $n_{CO_2} = 9$ for the CO₂ flow and $n_{air} = 4$ for the air-side flow; the evaporator was discretised with $n_{CO_2} = 8$ and $n_{air} = 4$. These discretisations were chosen for the simulation due to acceptable execution time for a simulation run of a complete refrigeration cycle.

The test configuration consists of a source providing pressure and enthalpy at the heat exchanger inlet and a mass flow sink generating a defined mass flow at the outlet. The source and sink are used to set the boundary conditions resulting from measured data at the component.

Figure 2 shows the measured and simulated capacity at the gas cooler for a wide set of operating conditions ($p_{CO_2}: 7\text{-}11.3 \text{ MPa}$, $\dot{m}_{CO_2}: 45\text{-}230 \text{ kg/h}$, $T_{CO_2,in}: 345\text{-}400 \text{ K}$). What can be seen from the comparison, is, that most of the simulated capacities are in within the error of ± 12 %. The deviation becomes higher near

the critical point which can be traced back to the chosen discretisation of the model. A higher discretisation would represent the influence of the pseudo-critical point more accurately. The discretisation also affects the exit CO₂ temperature, which the model predicts for supercritical gas cooling within 1.1 K and 2.6 K higher than the experimental data and outside the error of ± 0.7 K, see Figure 2. For operating conditions below the critical pressure the model predicts the capacity very well. The influence of discretisation with regard to consistence with experimental data is shown by Limperich [17].

In Figure 3 the results of the evaporator model are compared with experimental data. The boundary conditions were p_{CO_2} within 3.017-5.01 MPa and \dot{m}_{CO_2} within 45-140 kg/h. As Figure 3 shows, the model predicts the capacity within ± 7.4 %. The air outlet temperature is predicted within ± 0.8 K which is within the uncertainty of measurement. The humidity of the air was not taken into account since the evaporator is integrated in a closed loop air-cycle. Therefore it can be assumed that the air is dehumidified after a short time of operation.

The validation of the internal heat exchanger is shown in Figure 4. The comparison of the transferred heat fluxes shows a good agreement within an uncertainty of ± 10 %.

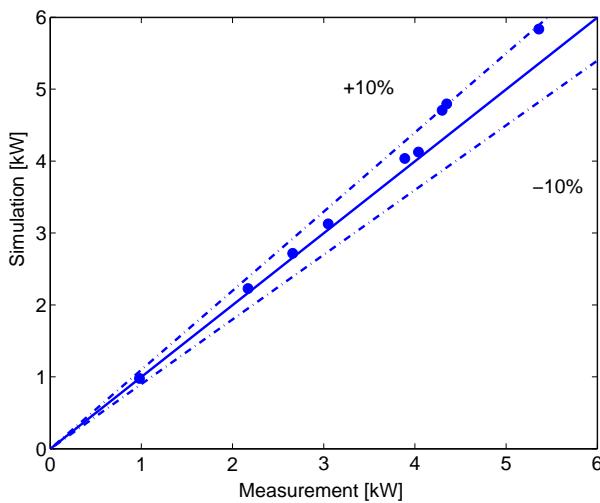


Figure 4: Measured and simulated capacity at the internal heat exchanger within ± 10 %

5 Transient simulation of a CO₂-system

In the following, results of the transient simulation of the above mentioned CO₂-system are presented. The simulated model is shown in an object diagram in Figure 1. This configuration represents the available CO₂-test rig on basic level with one evaporator. The results are compared with data of a start up of the system and following step changes in compressor speed as shown in Figure 5. The air inlet temperature of the evaporator also changed during the experiment, Figure 5. The other boundary conditions stayed constant and are listed together with the initial states in Table 2. All these data were taken from the experiment.

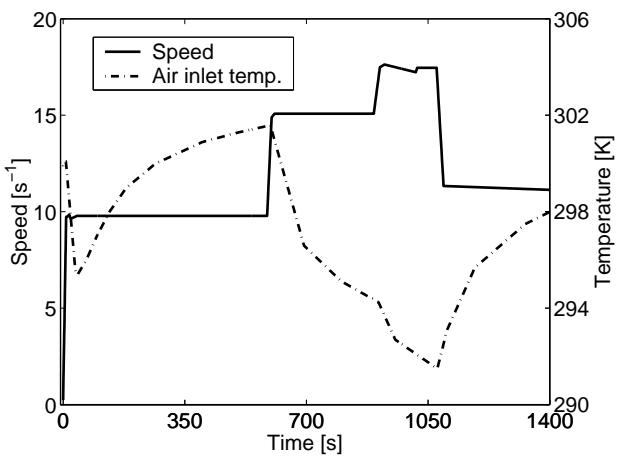


Figure 5: Step changes in compressor speed and run of air inlet temperature at the evaporator in the experiment; set as boundary condition of simulation run

Table 2: Boundary conditions and initial values of the simulation run corresponding to the experiment

Compressor	fixed displacement $V_{dv} = 33.5 \text{ ccm}$
Expansion valve	100 % open $K_v = 0.0264 \text{ m}^3/\text{h}$
Gas cooler	$\dot{m}_{air} = 2100 \text{ kg/h}$ $T_{air,in} = 312 \text{ K}$
Evaporator	$\dot{m}_{air} = 760 \text{ kg/h}$
System volume	$V_{tot} = 3.62 \text{ l}$
Specific refrigerant charge	267 kg/m ³
Initial value	$p_0 = 5.7 \text{ MPa}$ $h_0 = 425 \text{ kJ/kg}$
Initial value receiver	$p_0 = 5.7 \text{ MPa}$ $h_0 = 295 \text{ kJ/kg}$

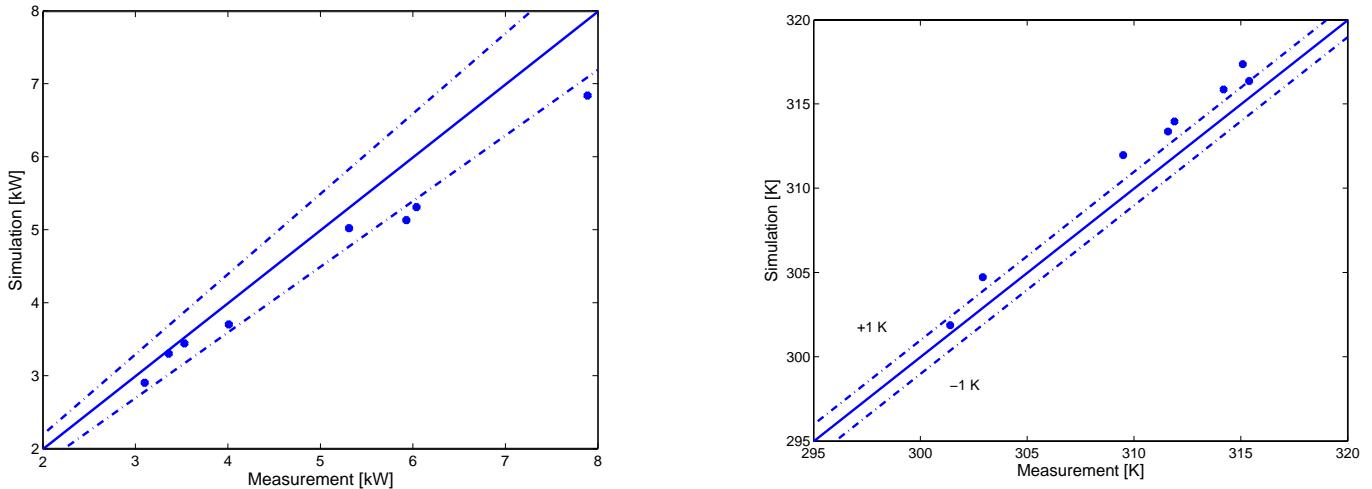


Figure 2: Measured and simulated cooling capacity at the gas cooler within $\pm 10\%$ and approach of refrigerant temperature at gas cooler exit ($\pm 1\text{ K}$)

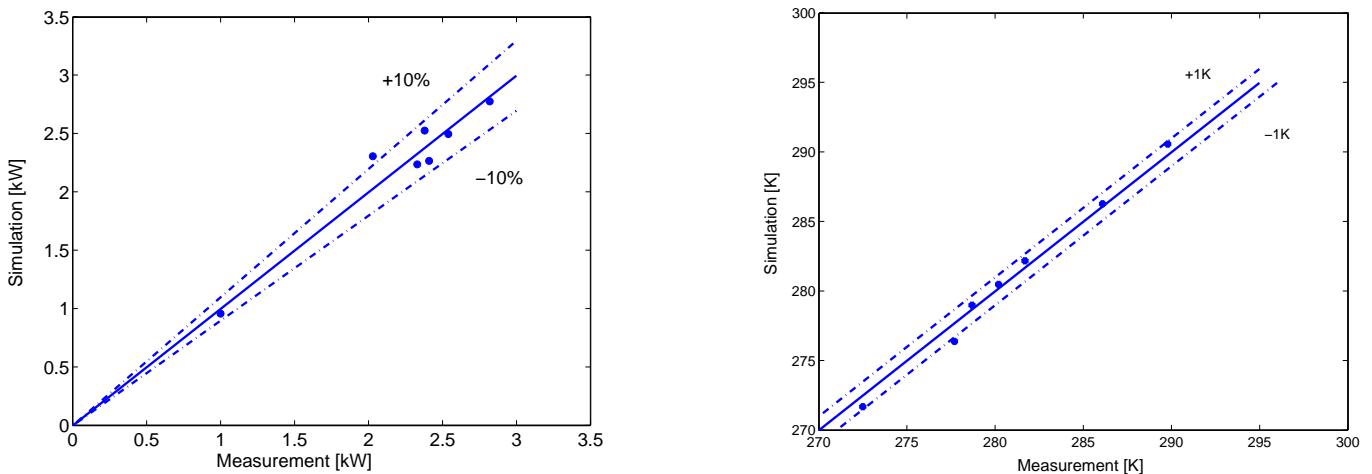


Figure 3: Measured and simulated cooling capacity at the evaporator within $\pm 10\%$ and approach of air outlet temperature ($\pm 1\text{ K}$)

In Figure 6 the simulated and measured pressure at compressor inlet and exit is plotted versus time. The plotted experimental data are filtered due to the very high measurement noise. What can be seen from the comparison is a fair agreement of the absolute values as well as the time response for the pressure at the compressor inlet. At the compressor exit there is only a partial agreement; especially at the beginning there is a clear deviation in absolute values and time response. The model predicts a pronounced undershoot whereas the experimental data show a smaller undershoot. This behaviour can also be seen in the comparison of the mass flow rate at the expansion valve in Figure 7. In general, there is a systematic underestimation of the mass flow rate by the model, which is larger than the

tolerance of the used mass flow meter. The run of pressures and mass flow rates are coupled in such systems. Therefore deviation of one value influences the other values and vice versa. Reasons for this behaviour can be seen in the modelling of the compressor using algebraic equations instead of a physical model. This leads to the use of characteristic fields for the efficiencies, which were generated by measurements at steady state. Especially at the start up of the system the used efficiencies in the model are probably different from the real behaviour of the compressor. Furthermore the available values of the flow coefficient of the expansion valve are independent from the inlet state and the pressure difference at the valve. The flow coefficient is only a function of the open ratio. From physical point

of view it seems to be obvious, that this simplified characteristic does not represent the complete operating range. So, the uncertainty of component-specific parameters like compressor efficiencies and flow coefficient of the valve influences the simulation results. This known influence can be accepted for the level of system simulation and has to be taken into account for the validation of the models.

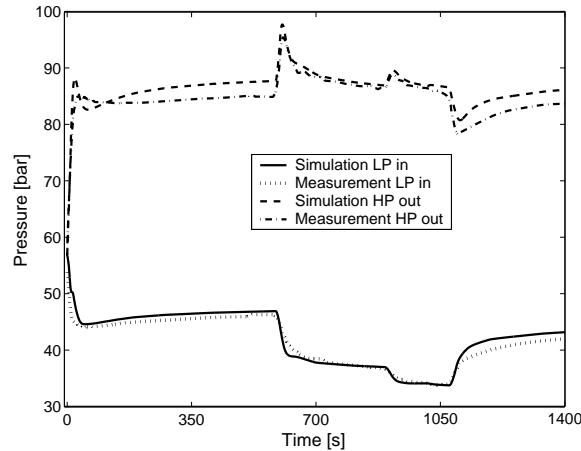


Figure 6: Transient run of the pressure at compressor inlet and exit; comparison between simulation and measurement

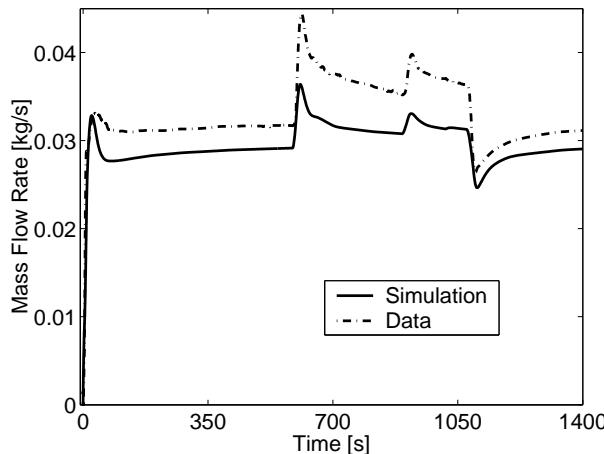


Figure 7: Transient run of the mass flow rate at the expansion valve; comparison between simulation and measurement

The object diagram of a CO₂-loop with two parallel evaporators is shown in Figure 8. This schematic represents the extension of the above mentioned CO₂ test rig. By this example it can be shown that the transient simulation of such a system, especially the split of the mass flow is predicted correctly by the models. In Figure 9 the simulated and measured mass flow rates are

plotted. Their comparison shows a good agreement. The mass flow rates in the branches differ, since the expansion valves have different flow characteristics.

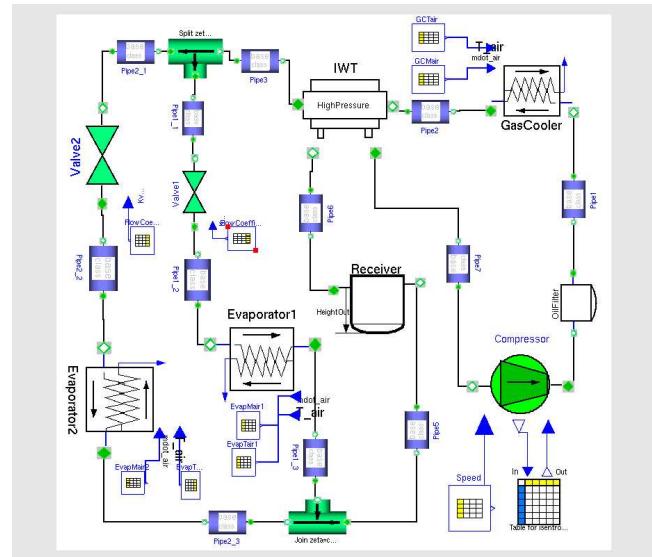


Figure 8: Object diagram of the simulated CO₂-loop with two parallel evaporators; representing the CO₂ test rig

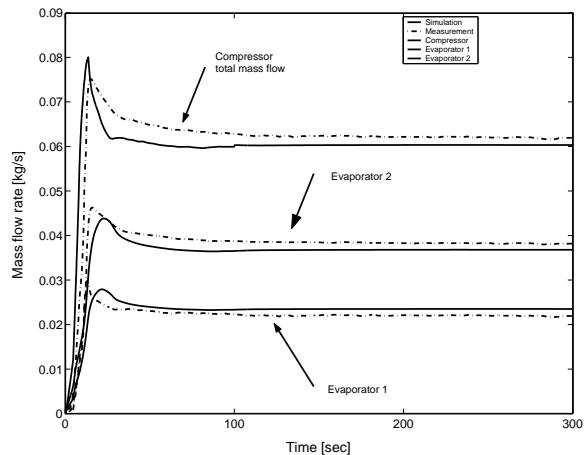


Figure 9: Simulated and measured mass flow rates

6 Conclusion

In this paper the modelling and implementation of a Modelica library for refrigeration systems was presented. The implemented library provides both base models for modelling of components and usable models of components for the automotive and aircraft application. The intention is to create a library for the simulation of single components and complete cycles.

Such a library can be used to make fundamental investigations of refrigeration systems. Furthermore, it can be used for the optimisation of specific heat exchangers, for the evaluating of an optimal system configuration and for the layout and optimisation of the system control. The presented simulation results for the steady state of different types of CO₂-heat exchangers show a fair correspondence with measured data. The results of the transient simulation show a good agreement in comparison with experimental data.

References

- [1] McMullan JT. Refrigeration and the environment - issues and strategies for the future. Int J Refrigeration 2002; 1; 89-99.
- [2] Bundesumweltministerium. Umsetzung des nationalen Klimaschutzprogramms im Bereich der fluorierten Treibhausgase. Bonn, 2002.
- [3] Lorentzen G. Revival of carbon dioxide as refrigerant. Int J Refrigeration 1994; 5.
- [4] Billard F. Editorial - Use of carbon dioxide in refrigeration and air conditioning. Int J Refrigeration 2002; (25); 1011-1013.
- [5] Pfafferott T, Schmitz G. Modelling and Simulation of Refrigeration Systems with Natural Refrigerant CO₂. In: Proceedings of the 9th Int. Refrig. and Air Conditioning conf. at purdue. Purdue university, IN, USA, 16-19. July 2002
- [6] Eborn J. On Model Libraries for Thermo-hydraulic Applications. Lund, Sweden: PhD thesis, Department of Automatic control, Lund Institute of Technology, 2001.
- [7] Tummescheit H. Design and Implementation of Object-Oriented Model Libraries using Modelica. Lund, Sweden: PhD thesis, Department of Automatic control, Lund Institute of Technology, 2002.
- [8] Tummescheit H, Eborn J. Chemical Reaction Modeling with ThermoFluid/MF and Multi-Flash. In: Proceedings of the 2th Modelica Conference 2002, Oberpfaffenhofen, Germany, Modelica Association, 18-19 March 2002.
- [9] Patankar SV. Numerical Heat Transfer and Fluid Flow. Washington: Hemisphere Publ. Corp., 1980.
- [10] VDI-Wärmeatlas - Berechnungsblätter für den Wärmeübergang (in German). Düsseldorf: VDI-Verlag, 7. Edition, 1994.
- [11] Stephan K. Wärmeübergang beim Kondensieren und beim Sieden (in German). Berlin: Springer-Verlag, 1988.
- [12] Melin P. (Editor) The behaviour of HFC-134a, HFC-152a and HCFC-22 in Evaporators. IEA Heat Pump Centre (Sittard, The Netherlands), 1994.
- [13] Bohl W. Technische Stroemungslehre. Wuerzburg: Vogel Fachbuchverlag; 11.Edition, 1998.
- [14] Pettersen J, Rieberer R, Munkejord ST. Heat transfer and pressure drop for flow of supercritical and subcritical CO₂ in microchannel tubes. Trondheim, SINTEF Energy Research, Technical Report A5127, 2000.
- [15] Kakac S, et al. Handbook of Single-Phase convective Heat Transfer. New York: John Wiley & Sons Inc., 1987.
- [16] Pfafferott T, Schmitz G. Modeling and Simulation of Refrigeration Systems with Natural Refrigerant CO₂. In: Proceedings of the 2th Modelica Conference 2002, Oberpfaffenhofen, Germany, Modelica Association, 18-19 March 2002 .
- [17] Limperich D, Pfafferott T, Schmitz G. Transient simulation of CO₂-heat pump systems using Dymola/Modelica (in German). In: DVK-Tagungsbericht 2002 Band III, volume 29, Magdeburg, Germany, 20-22 November 2002
- [18] IEC 60534-2-1. Industrial-process control valves - Part2-1: Flow capacity - Sizing equations for fluid flow under installed conditions. German Version EN 60534-2-1. Berlin: Beuth Verlag, 2000.
- [19] Schwarz T, et al. Model to investigate the performance of accumulators in vapor compression systems. In: Proceedings of 9th International Refrigeration and Air Conditioning Conference at Purdue, Purdue University, USA, 16-19 July 2002.
- [20] Idelchick IE. Handbook of Hydraulic Resistance. Florida: CRC Press, 1994.

- [21] Schade O, Carl U. Controll of direct-expansion CO₂-refrigeration plants (in German). In: DKV-Tagungsbericht 2002 Band II.2, volume 29, Magdeburg, Germany, 20-22 November 2002
- [22] Lorentzen G, Pettersen J. A new, efficient and environmentally benign system for car air-conditioning. Int J Refrigeration 1993; 16(1); 4-12.

Simulation for Operation Management : Object Oriented Approach using Modelica

Jerzy Mikler,
IIP, The Royal Institute
of technology, Sweden,
jerzy@iip.kth.se
WWW: <http://www.ida.liu.se/~vaden/or>

Vadim Engelson,
IDA, Linköping
University, Sweden,
vaden@ida.liu.se

Abstract¹

Operation management provides methods and tools for decision making in production systems. There are both mathematical and simulation-based methods for finding optimal production parameters. Simulation models are based on both continuous and discrete event simulation. These models can be reused on both component level and pattern (template) level. Modelica fits well into these requirements; one of case studies revealing related problem is discussed in this paper.

1 Introduction

Simulation is an important tool for executives in their day-to-day decision making activities. The problems occurring in sales departments, on the shop floor, and all the other departments of a company are more often difficult and complex. They cross the functional boundaries, and are dynamical in nature. Figure 2 shows a simplified causal loop diagram (CLD) of a generic assembly system. It shows all the identified factors relevant for the system flexibility, together with their relations.

The relations (arrowed lines connecting the factors) mean that the one factor affects the other². The problem is yet more complicated by the fact, that the influence is most often delayed, adding the

time dimension, what in turn makes the understanding of the behaviour (the consequences) impossible without aid of simulation. To simulate the behaviour of this system one has to build system dynamic (SD) model of the CLD above. SD approach has been initially defined by Forrester[4]. Relations between the factors in SD are described with a system of differential algebraic equations (DAE) with delays. Such a model of a part of the system in Fig. 2 (the “operations”) is shown in Fig. 1. A Modelica library for system dynamic modelling has been developed by Fabricius [3].

These kinds of models are typical for *analysis* tasks – i.e. when the real structure of the system is not known, and we examine the patterns of behaviour to identify the structure. In this case the suitable tools are system thinking and system dynamics (based on time continuous simulation). Another type of problems arises when we know the components and *synthesize* them to a system in the aim to achieve a desired behaviour of this system. In this case a suitable tool is discrete event simulation. These kinds of problems are typical for shop floor design and operations. The inventory problem evaluated later in this paper is an example of that (see even Fig. 3).

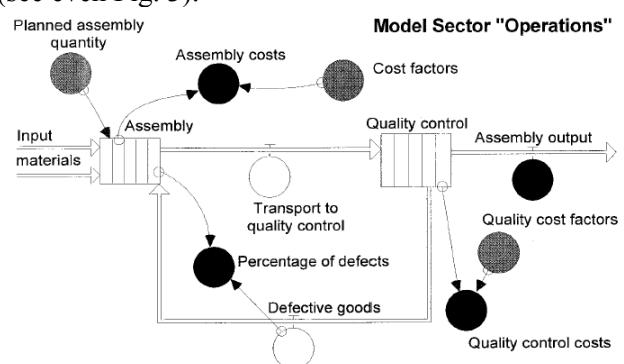


Fig. 1. The system dynamics model of the sector “Operations” on the CLD in Fig. 2, modelled in the Stella environment. System Dynamics library in Modelica uses similar graphical notation. Boxes denote levels (stocks), circles denote rates (e.g. flows between levels). Source: Zahn [1]

¹ This article describe ongoing work carried out in the VISP project supported by Vinnova foundation, Sweden. Information about the current status can be found at <http://www.ida.liu.se/~vaden/or>

² Arrows indicate the direction of causality. Signs (+ or -) imply the polarity of relationships: a '+' indicates that an increase in the independent variable causes the dependent variable to increase. A loop is identified by number and sign (-) or (+). (-) indicates restoration of balance by an action, whereas (+) implies either reinforcing vicious cycle or virtuous cycle.

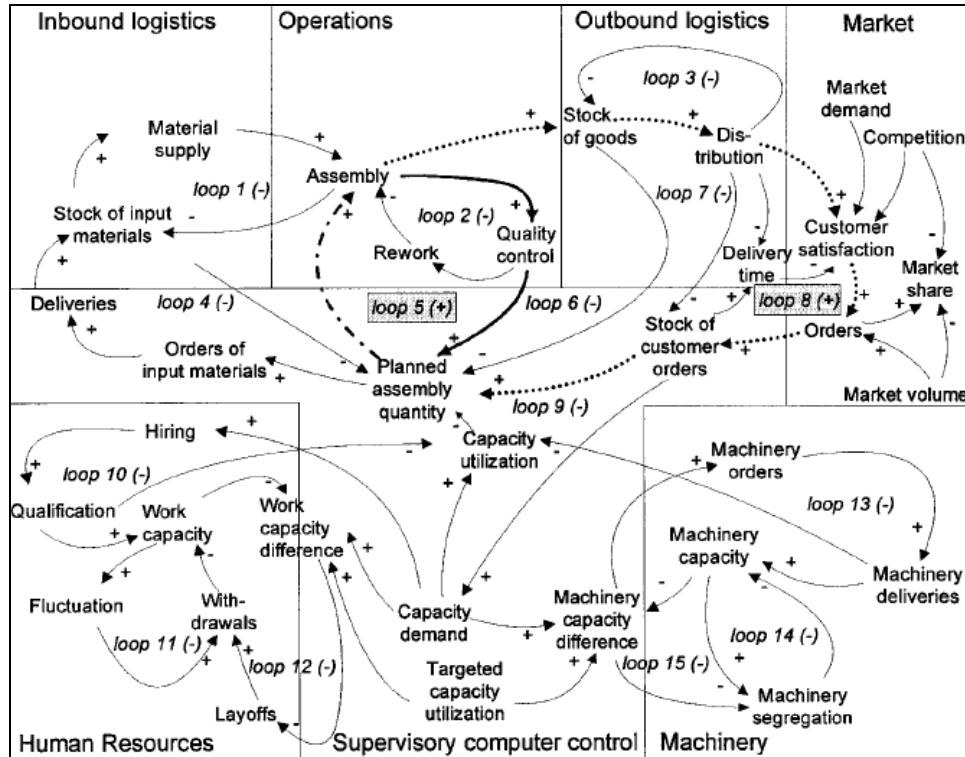


Fig. 2. Causal loop diagram of generic assembly systems. Source: Zahn [1]

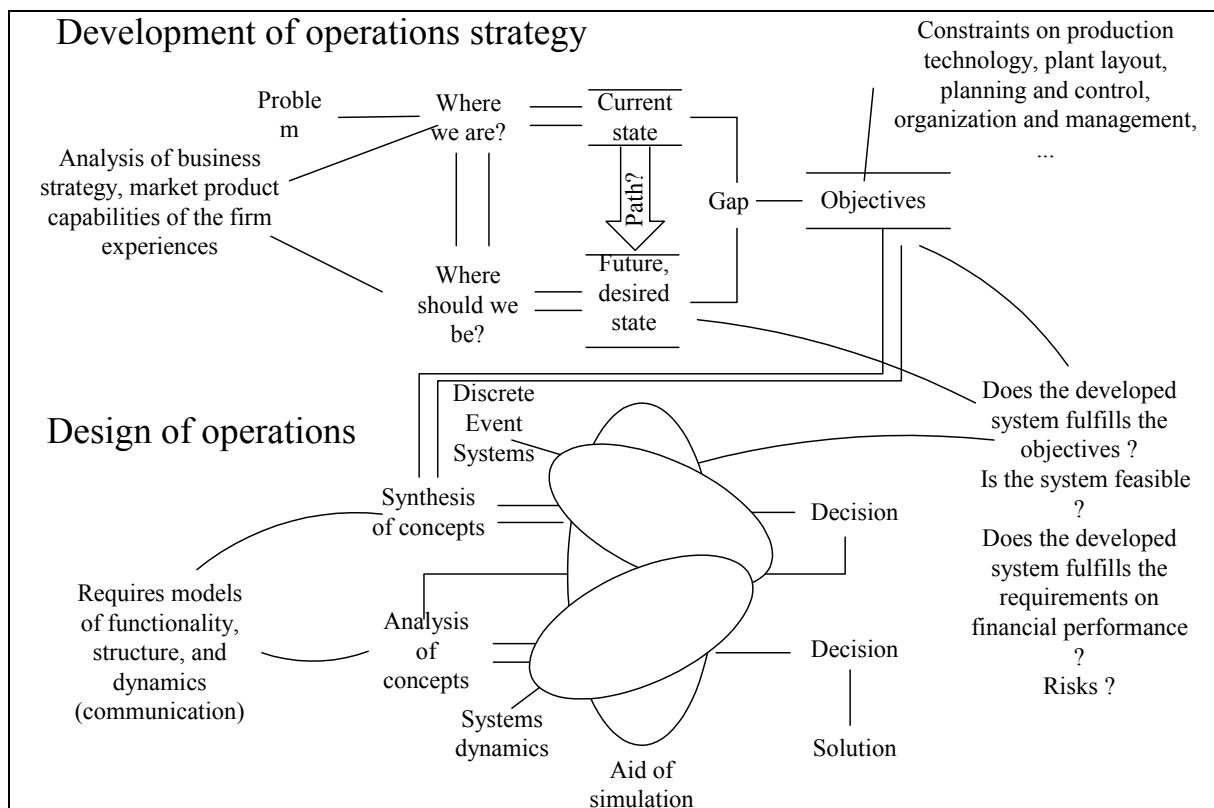


Fig. 3. Structure of a design process, based on work of B.Wu.[13]

2 . Operation Management

Operations Management is an established discipline providing insights, methods and tools facilitating decision-making in production systems. It spans the whole spectrum of managerial problems in a company – from the formation of strategy, through product and process design, to the run time operation. These decisions are usually divided into the four categories [5]:

1. *Strategic choices* – determination of competitive priorities and strategies how to best design the product and production processes that fit the priorities (operations strategy)
2. *Product and Process design* –
 - a. Product design³ (decisions concerning concept generation, screening, preliminary design, evaluation and improvement, prototyping and final design)
 - b. Process design
 - i. Network design – considers the whole network of operations for a given product (“from dirt to dust”). At this stage we have to consider three issues:
 - 1.The shape of the network, distribution of influence and responsibilities leading to make/buy decisions (vertical integration),
 - 2.Decide upon location for each node, and
 - 3.Decide upon capacity of each node (chasing strategy, level and balancing)
 - ii.Layout and flow – the physical location of all the machines, equipment, and stuff, as well as the flow of materials and information.
 - iii.Process technology – decisions upon
 - 1.what technologies to use,
 - 2.the scale of automation to use
 - 3.the degree of integration of the technology
 - iv.Job design – decisions upon the work methods to be used. The work methods define liaison between the people the used technology.
3. *Operating decisions* – production planning and control– decisions during the day-to-day operation after the production system was build; coordination of activities in the internal and external supply chain, forecast of demand, inventory control, resource planning, scheduling (prioritising of jobs).
4. *Improvement* – methods of improving existing processes.

Until recently, much of the research work in the production system area was directed towards problem solving rather than theory building. Models

developed within the Operation Research area (OR) give us valuable insights in basic trade-offs when analysing the problems, but it is neither an explanatory nor predictive method. Also, the proposed solutions frequently fails, because models, often adapted to capability of the chosen solution techniques, cover only certain aspects of the analysed problems, neglect the effect of numerous factors and most of all lacks an ability to generate understanding (and this requires synthetic thinking [6]).

Recently the OM research community noticed the increasing importance of theory and theory driven empirical research and we observe an explosive transformation of operations management to a science discipline. This implies development of explanatory and predictive models of the operational processes – that is models, that could be used to explain or predict the output or performance of the process as a function of process characteristics, process states, and inputs to the process, as a main object of study in OM [7].

Joining these efforts, we positioned our research as a model-based approach to knowledge generation by building models explaining dynamic behaviour of real-life operational processes (an axiomatic descriptive research) and decision-making problems (an axiomatic normative research) for design and operation of manufacturing systems.

Our research is based on computer simulation and experimental design. We expect that much more complex problems may be studied than in the case of using mathematical models. The aim of our work is to create library with standard components and patterns that can easily be embodied and configured for solving particular OM problems.

As the OM decision types listed above are typical for most businesses, well demarcated, and may be defined on high level of abstraction, a tempting idea is to represent them in form of parameterised patterns consisting of primitive components. Use of object-oriented paradigm would considerably simplify the modelling and simulation process during evaluation of alternatives due to specialisation, class replacement and override mechanisms. Also the possibility of merging both continuous (system dynamics) and discreet time simulation in one and the same environment is crucial as well. The reason why we choose Modelica for the VISp project is, that it provides both of the features.

³ Not discussed in this paper

3 Simulation support for decision-making.

Benefits of implementing simulation support should include increased quality of decision-making, knowledge preservation and thorough understanding of the structure of the problem as well as the decision-making process. Figure 3 shows a formalized model of the OM problem solving process [7,8]. Usually, such a process is an iterative process of incremental refinement aimed to guide the designer throughout the design process and ensure that all the necessary aspects were considered. Decision-making is associated with choice between a finite numbers of feasible alternatives generated in the synthesis and the analysis process respectively.

There are three facts indicating, that our efforts should end with promising results – firstly, *decision criteria* (cost, quality, dependability, flexibility, speed [5]), and inference mechanisms are to great extend recognised, and available in the OM literature, secondly, the evaluation of the design concepts may be categorised – typically we have to evaluate *feasibility* (is the concept feasible?), *acceptability* (how much does it cost?), and *risks* (what risks are connected with a particular solution?), and thirdly, recent development in knowledge engineering give us a solid ground in structuring of the decision making process [12].

4 The case study – the case specification

As an example of the method proposed above, in this paper we will evaluate an inventory system design problem called “one-machine-multi-storage-point” shown in Fig. 4. This is one of very typical problems in operation management. The inventory system has a number of products (that should be manufactured), each with associated variable daily demand, with different and variable lead times⁴, and required service level⁵ on the demand side. Manufacturing is organized in batches of identical products. To meet the required demand levels we have to determine safety stocks for each product, as well as a suitable prioritising strategy for the

production process (the sequencing of jobs, i.e. production orders).

The *traditional approach* to modelling an inventory with continuous review (the well-known Wilson formula) has a number of presumptions: stock outs are prohibited; demand is known with certainty and is constant over time; lead-time is known and constant; the cost of the products is fixed; adequate capacity and capital exist to implement the suggested strategy; the strategy does not affect other products the organisation handles. Unfortunately, as we can realise no one of these presumptions holds in our case, and solving it mathematically is very hard if not impossible. Also Operation Research cannot suggest any solvable mathematical model of this problem (queuing networks are not applicable here). Our approach is to build suitable simulation model to gain the understanding of the problem, generate possible solution alternatives, perform stochastic simulation with different input parameters and then choose the optimal alternative and optimal parameters.

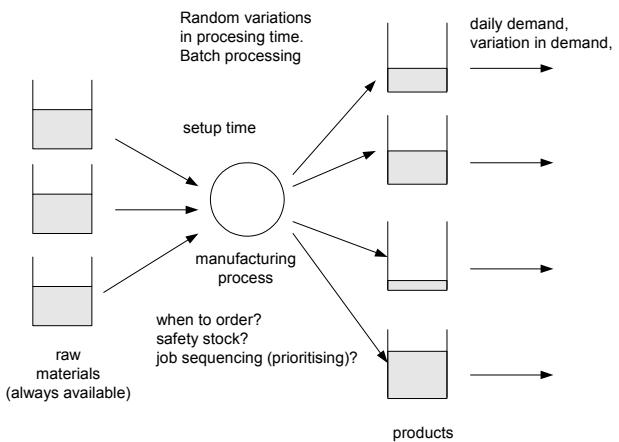


Fig. 4. The “one-machine-multi-storage-point” model.

With this example we assess the benefits of using Modelica as a simulation platform in operations management decisions.

The simulation model is shown in Fig. 5, and the object interaction diagram in Fig. 6 respectively.

⁴ Lead time: the time required between placing an order and receiving the ordered product

⁵ Service level: the desired probability of not running out of stock in any one ordering cycle, which begins at the time an order is placed and ends when it arrives in stock

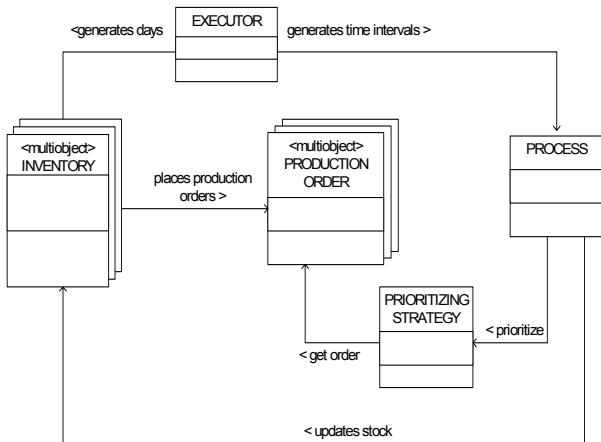


Fig. 5. The simulation model for “one-machine-multi-storage-point” problem, in UML notation. Note that in Modelica notation a specific prioritising strategy (such as “FIFO” or “least processing time – first”) should replace the placeholder for “Prioritising Strategy”.

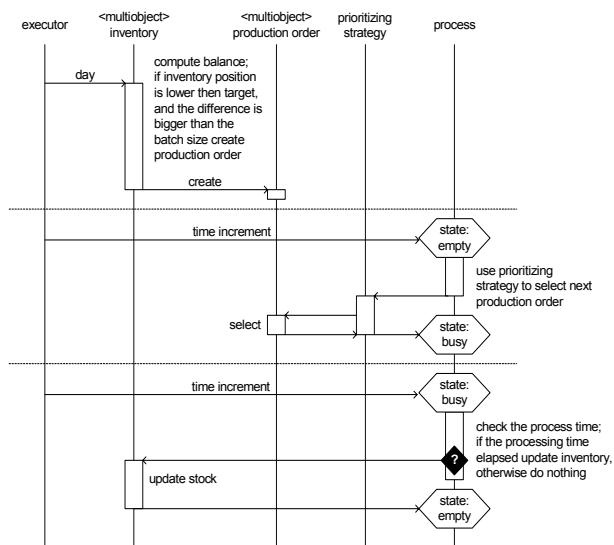


Fig. 6. The object interaction diagram for the simulation model in UML notation⁶

To find the dependency between specific prioritising strategy choice, size of safety stocks and the demand characteristics for each product we use tree-variable experimental design model [2], running each combination of values a sufficient number of times to achieve required confidence interval.

5 Using Modelica for the case study

A Modelica library for discrete event simulation targeted for operating decision support is under construction. The library contains components of business or manufacturing process, such as scheduled and random event generators, queues, delays, triggering actions, sequence schedulers, gathering statistics, etc. Therefore declarative style of modelling, high level of abstraction and ease of combining components into a working system is achieved. Some components, e.g. sequence schedulers are modelled using Modelica algorithms and functions. Signals should be used for synchronisation between the components. Since there is no explicit signal send/receive in Modelica, event-triggering mechanism will be used.

The DEVS Modelica library has been previously designed by Bunus [8]. The DEVS formalism is defined by Zeigler [9]. It describes basic discrete event entities from which complex models can be built. Atomic DEVS components are state machines; state depends on internal (timeout) and external (change in input port) components.

In addition to basic primitives, the library will contain *patterns*, i.e. collection of ready-made complex models, traditionally used in operation research. In order to simulate such patterns the user has to modify them by replacing necessary *replaceable* classes, as well as by parameterisation. Since just triggering signals is not sufficient, in the library some non-trivial connectors should be used for communicating information about incoming orders and product movement.

Possible difficulties we expect with Modelica use for the operating decision simulation is connected with data handling. As noted by Remelhe [10], an event list of a scheduler has to be realized by a fixed length vector. If just queue length is simulated, an integer variable is sufficient to describe a queue. However if queue elements contain some essential information (product identification, time stamps), a dynamic data structure would be very convenient. These should be unnaturally modelled by globally accessible vectors, indexing mechanism and predetermined maximal length.

Adding full scale dynamism to data structures would certainly break nice declarative properties of Modelica, complicate the type system and would make implementation more difficult. As a first step to limited dynamism we would suggest introduction of arrays with dynamically changing size. Such array size should be a discrete integer variable. For

⁶ The pictures 4 and 5 do not show data collection functionality that is “hidden” to keep simplicity

DES systems it is sufficient if all variables within dynamic arrays are discrete. Number of equations and variable is not known when system simulation starts but it is always a linear combination of all dynamic array lengths. For continuous systems things become more complex.

6 Test model

In order to test our ideas about future library a test package has been constructed in Modelica in order to simulate the “one-machine-multi-storage-point” model. The UML notation above has been taken as model specification. First a rapid prototype has been created in Mathematica in completely functional programming style. It took just few hours to write such model and test it for typical situations.

It has been more difficult task to do it with Modelica. We attempted to use declarative Modelica style for this package. Unfortunately most of computations here take place in conjunction with an inventory database. Such databases can be modelled in Modelica as arrays of records and records of arrays. However the visual structure of UML notation cannot be represented here directly. Our Modelica model contains two databases and one “sampler”. The databases are used as input (**ddata** and **dbase**) and output (**dbase**) arguments of many functions. Separation of these databases to smaller components would lead to more inconvenient design. The **sampler** can be divided into two samplers – one for production time and one for day start. The sampler is based on an algorithm in with **when**-section.

Some functions (**normal**, **makeCriteria**) can be replaced by user-defined functions if necessary. Simulation results are shown in Figure 8.

```
package ormodel13
final constant Integer nr=2 "Number of products";
```

```
record ddata "data which are not returned from update
functions"
parameter Real[nr] demand "average product demand";
parameter Real[nr] demandDistr
"normal distribution of product demand";
parameter Real[nr] processingTime
"average processing time for a batch";
parameter Real[nr] processingTimeDistr
"its normal distribution";
parameter Integer[nr] batchSize "required batch size";
parameter Real[nr] wantedLevel "safety stock level";
Real thetime "time of last event";
end ddata;
```

```
record dbase "data which returned from update functions"
discrete Real[nr] level "current stock level";
discrete Real[nr] backorder "current missed deliverables";
discrete Real[nr] todayCustomerOrder "today's demand";
discrete Integer[nr] orderedBatches "computed nr of batches";
discrete Real[nr] criteria "computed criteria for sorting";
discrete Integer[nr] ordering "ordering of producion orders after
sorting";
discrete Integer orderindex "currently processed production
order";
end dbase;
```

```
function busyFun "actual (random) proceessing time for each
batch"
input dbase db;
input ddata dd;
output Real busyVar;
algorithm
busyVar := normal ( dd.processingTime [ db.ordering [ db.orderindex ] ], dd.thetime );
end busyFun;
```

```
function normal
"random number, distributed in some way; should be replaced"
"by some more meaningful distribution law"
input Real mean;
input Real thetime;
output Real randval;
algorithm
randval := mean + mod(thetime, 0.5) - 0.25;
end normal;
```

```
function updateinventory
"updates inventory when production order has been fullfilled"
input dbase dbin;
input ddata dd;
output dbase dbout "returns updated database here";
protected
Real production;
Integer productidx;
algorithm
dbout := dbin;
productidx := dbout.ordering[dbout.orderindex];
production := dbout.OrderedBatches [ dbout . orderindex ] * dd
batchSize[ productidx ];
(dbout.level[productidx],dbout.backorder [productidx ] ) := update ( dbout , productidx, production);
end updateinventory;
```

```
function update "updates inventory information for a single
product"
input dbase db;
input Integer pindex "product index";
input Real production "amount of produced product";
output Real newlevel;
output Real newbackorder;
algorithm
newbackorder := db.backorder[pindex] + db
todayCustomerOrder[pindex];
if (production + db.level[pindex] >= newbackorder) then
newlevel := db.level[pindex] + production - newbackorder;
newbackorder := 0;
else
newbackorder:=newbackorder-(production+db.level[pindex] );
newlevel := 0;
end if;
end update;
```

```
function endday "statistics can be gathered here"
input dbase dbin;
output dbase dbout;
algorithm
dbout := dbin;
```

```

end endday;

function makeCriteria "computes criteria; can be replaceable
function"
  input dbase dbin;
  input ddata dd;
  input Integer i;
  output dbase dbout;
  algorithm
    dabout := dbin;
    dabout.criteria[i] := dd.ProcessingTime[i] * dabout . .
  orderedBatches[i];
  "criteria is shortest processing time first. Here the prioritising
  strategy is defined";
end makeCriteria;

function sort "bubble-sorting"
  input Real[:] qinp "sequence of criteria values";
  output Integer[nr] ordering "permutation";
  protected
    Real qttmp; Integer itmp; Real[nr] q;
  algorithm
    q := qinp;
    for i in 1:nr loop ordering[i] := i; end for;
    for i in 1:nr loop for j in 1:nr - 1 loop
      if (q[j] > q[j + 1]) then
        qttmp := q[j]; q[j] := q[j + 1]; q[j + 1] := qttmp;
        itmp := ordering[j]; ordering[j] := ordering[j + 1];
        ordering[j + 1] := itmp;
      end if; end for; end for; end sort;

function startday "preparations at start of the day"
  input dbase dbin;
  input ddata dd;
  output dbase dabout;
  algorithm
    dabout := dbin;
    for i in 1:nr loop
      dabout.todayCustomerOrder[i] := normal(dd.demand[i], dd.
    thetime );
    end for "today's demand randomly chosen";
    for i in 1:nr loop
      dabout.orderedBatches[i] := integer((dd.wantedLevel[i] +
    dabout.backorder[i]+dabout.todayCustomerOrder[i] - dabout.level [i] ) /
    dd.batchSize[i]);
    end for "necessary number of batches";
    for i in 1:nr loop
      dabout := makeCriteria(dabout, dd, i);
    end for;
    dabout.ordering := sort(dabout.criteria);
    dabout.orderindex := 0
    " production orders are performed according to this ordering
    now ";
  end startday;

model sampler "production unit and sampler"
  outer dbase db;
  outer ddata dd;
  discrete Real nextSampling(start=0.1) "when unit will become
  free";
  discrete Real busyTime(start=1) "duration of production time";
  discrete Real daystart(start=0) "when day started";
  Boolean signal "the signal could be sent over to other
  components,
  but this is not needed";
  algorithm
    signal := time >= nextSampling;
    when (pre(signal)) then
      if db.orderindex < nr then
        db.orderindex := db.orderindex + 1 "start next order";
        dd.thetime := time "used for randomizer";
        daystart := pre(daystart);
      end if;
    end when;
  end sampler;

  db := updateinventory(db, dd) "we assume that order is
done";
  busyTime := busyFun(db, dd) "production time for a new
order";
  else
    busyTime := 1 "stay idle one more hour";
  end if;
  nextSampling := pre(nextSampling) + busyTime;

  if (time - daystart > 24) then
    daystart := time;
    db := endday(db);
    db := startday(db, dd) "new day started";
  end if;
end when;
end sampler;

```

model factory

```

inner dbase db(
  backorder(start={0,0}), level(start={3,4}),
  todayCustomerOrder(start={10,12}),
  orderedBatches(start={0,0}), criteria(start={0,0}),
  ordering(start={1,2}), orderindex(start=1));

inner ddata dd ( demand={5,6}, demandDistr={2,2},
processingTime={2,3} , processingTimeDistr = {2,2}, batchSize
= {3,3} , wantedLevel={5,5});

sampler S;
end factory;

```

```

end ormodel13;

```

Fig. 8. Result of simulation of the “factory” model for 1000 hours. Production stock levels do not exceed the safety stock ({5,5}), and no backorder occurs.

An alternative sampler based on an **when**-equations would allow to build a structure consisting of models and blocks instead (or in addition to) functions. This approach is the next step in our ongoing work.

7 Future research

The plans for research include design of multiple modelling patterns for typical operation management problems. Also it is necessary to create user-friendly modelling and simulation tools that can easily be configured for solving particular problems of business operation management.

8 The VISP project

The VISP project [11] is a cooperation between the departments of Machine Design and Production Engineering at KTH, The Dept. of Computer Science at Linköping University, IVF, The Institute for Engineering Science at Skövde University College, and a number of Swedish companies.

The expected output is the development of an information platform for industry-adapted product realization based on a common, integrated map over workflows and data access during concurrent development of a product program and a manufacturing system. The work includes development of methodology, modularisation and configuration of simulation models of products and production systems, a pilot installation of the methodology with a commercial software, and evaluation of the achieved results in several real industrial cases.

References

- [1] Zahn, E., Dillerup, R., Schmid, U.: "Strategic evaluation of flexible assembly systems on the basis of hard and soft decision criteria". System Dynamics Review Vol.14, Winter 1998.
- [2] Bergman, B. "Industriell försöksplanering och robust konstruktion", Studentlitteratur 1992
- [3] Fabricius, S. "SystemDynamics Modelica Library", available from www.modelica.org, 2002
- [4] Forrester J. "Principles of Systems", Wright-Allen Press, Cambridge, U.S.A, 1969
- [5] Slack, N., et al. "Operations Management". Prentice Hall 2001
- [6] Ackoff. Russell, L. Operation Research: „after the post mortem“. "System Dynamics Review" Vol. 17. No4 Winter 2001
- [7] J Will M Bertrand; Jan C Fransoo. "Operations management research methodologies using quantitative modeling", International Journal of Operations & Production Management, Year: 2002 Volume: 22 Number: 2 Page: 241 – 264
- [8] Bunus P., Fritzson, P., DEVS-Based Multi-Formalism Modeling and Simulation in Modelica. Proc. of the 2000 Summer Computer Simulation Conf. (Vancouver, Canada, Jul. 16-20).
- [9] Zeigler, B.P., Multifaceted Modeling and Discrete Event Simulation, Academic Press. London, 1984
- [10] Remelhe, M.A.P., Combining discrete event models and Modelica – General thoughts and a Special Modelling environment. Proc. of Modelica 2002 conf. (Oberpfaffenhofen, Germany)
- [11] VISP Project site, <http://extra.ivf.se/VISP/>
- [12] Schreiber, G., at al, "Knowledge engineering and management", MIT Press, 2000.
- [13]. Wu, B., "Manufacturing systems design and analysis", Chapman&Hall, 1992.

BioChem

– A Biological and Chemical Library for Modelica

Emma Larsdotter Nilsson and Peter Fritzson

Linköpings universitet

PELAB – Programming Environment Laboratory

Department of Computer and Information Science

SE-581 83 Linköping, Sweden

{emmni, petfr}@ida.liu.se

Abstract

Many biological, biochemical and chemical systems have been mathematically defined for decades. As laboratory techniques are becoming increasingly sophisticated, more systems can be mathematically defined. But sophisticated techniques usually means more expensive and time-consuming. Simulation and modeling tools have today therefore become a very important part of biological and chemical research. In this paper the advancement on developing a library for simulation of cellular pathways in the Modelica language will be presented.

Keywords: Modelica, biological systems, chemical reactions, cellular pathways, SBML.

1 Biological Systems

All living things can be seen as systems. The prey-predator relation between foxes and rabbits, the cycle of energy-forest, the life of a bacterium or the human body are all more or less complex examples of biological systems surrounding us. Many of these systems are easy to model or simulate, their mathematical definitions have been known for years, if not decades. Other systems such as the human body or bacteria's intercellular processes, we don't know so much about, yet.

2 Cells and Cellular Pathways

One type of systems that we are just about to learn more about is the cell. Cells are the basic building blocks of all living organisms. No matter if the cells are part of a multicellular organism or constitute unicellular organisms the processes inside them do not differ that much. A cell's metabolism involves the uptake, decomposition, and rebuilding of different compounds and can be seen as a complex web or graph. The nodes are the different substances and the edges are the reactions that transform one substance to another. These complex

webs, consisting of up to a couple of hundred substances and more than twice as many reactions are referred to as cellular pathways. Some of the reactions in these pathways are already well defined, while some are not even discovered yet.

3 Why Modelica

Many biological, biochemical, and chemical systems have been mathematically defined for decades. As laboratory techniques are becoming more sophisticated, even more systems are defined and sometimes redefined. Better laboratory techniques also make it possible to analyze larger and more complex systems than before. However better techniques can also mean more expensive and sometimes more time-consuming analyses. A good modeling/simulation tool can in many cases extensively cut the cost and time.

Due to being equation-based Modelica is very suitable for modeling of biological, biochemical, and chemical systems. One major benefit is that the classes are acausal and thereby adapt to more than one data flow context [1]. The complexity of these types of systems will not be a problem, Modelica's strength as a modeling language for complex technical systems is well proven [2]. Another benefit of Modelica is that it is possible to model both discrete and continuous systems as well as hybrids thereof. Especially hybrid systems are quite common in the biological/chemical area. Modelica's strong software component model makes the language ideally suited as an architectural description language for complex systems, such as complex pathway models. Finally, the use of Mathematica [6] notebooks and commands for the simulations makes Modelica easy, even for non-computer science user. This is very important since most biologists and chemists have none or very little experience in computer science.

4 BioChem Package

The work of building a Modelica library for cellular systems has only just started. So far the main effort has been to develop classes for nodes and chemical reactions frequently occurring within cellular pathways.

4.1 Package structure

The structure of the package is shown in Figure 1.

```
package BioChem
  package BioChemUnits "Types and their units"
    end BioChemUnits;
  package Icons "Definitions of icons"
    end Icons;
  package Interfaces
    "Definitions of interfaces"
  end Interfaces;
  package Compartments
    "Definitions of compartments"
  end Compartments;
  package NodeElements
    package Nodes "Substance nodes"
    end Nodes;
    package SpecialNodes "Sources and Sinks"
    end SpecialNodes;
  end NodeElements;
  package Reactions
    package BasicReactions
      "Stoichiometric Reactions"
    end BasicReactions;
    package SBMLReactions "SBML Reactions"
      package GenericSBML "Generic reactions"
      end GenericSBML;
      package IrreversibleSBML
        "Irreversible reactions"
      end IrreversibleSBML;
      package ReversibleSBML
        "Reversible reactions"
      end ReversibleSBML;
      package MultiSBML
        "Multi reactant SBML reactions"
      end MultiSBML;
    end SBMLReactions;
  end Reactions;
end BioChem;
```

Figure 1. Structure of the BioChem package.

4.2 Package Icons

The package BioChem(Icons) contains icons used in the drag-and-drop interface in MathModelica. Icons for substance nodes are represented by circles, reactions are represented by uni and bidirectional arrows, and all other chemical signs and operators are represented by their standard symbols.

4.3 Package Interfaces

The package BioChem.Interfaces contains basic objects such as connectors and partial models used for most components in the BioChem package. The ReactionConnection (Figure 2) is the connector used for connecting the different components in the model.

```
connector ReactionConnection
  extends Icons.BlueX;
  BioChemUnits.Concentration c;
  flow BioChemUnits.MolarFlowRate r;
end ReactionConnection;
```

Figure 2. The connector ReactionConnection.

The connector is used on all connectable ends of reaction arrows, connectable parts of special nodes and signs, and for all normal nodes. All normal nodes are represented by the partial model NodeConnections (Figure 3), which contains eight connectors in order to make it easier to connect more than one reaction to a node.

```
partial model NodeConnections
  ReactionConnection rc_1;
  ReactionConnection rc_2;
  ReactionConnection rc_3;
  ReactionConnection rc_4;
  ReactionConnection rc_5;
  ReactionConnection rc_6;
  ReactionConnection rc_7;
  ReactionConnection rc_8;
end NodeConnections;
```

Figure 3. The partial model NodeConnections used for all normal nodes in the pathway web.

4.4 Package Compartments

In order to be able to control the environment of the reaction during a simulation a chemical reaction must take place in a restricted screened-off volume. The Compartments package contains models for all the different types of compartments in a cell.

4.5 Package NodeElements

The package BioChem.NodeElements.Nodes contains the different types of nodes that can appear in a metabolic pathway. The nodes must have some attributes corresponding to the properties studied during simulation of a metabolic pathway. The name of the substance and the surrounding compartment, the electrical charge (in case of the substance being an ion) and the amount of the substance and the flow through the node are such basic attributes.

The partial model BasicNode (Figure 4) contains all these basic attributes along with some equations needed for calculating the flow and the concentration of the substance.

```

partial model BasicNode
  extends Interfaces.NodeConnections;
  parameter String substanceName("");
  outer parameter String compartmentName;
  outer parameter BioChemUnits.Volume V_0;
  parameter BioChemUnits.Charge charge = 0;
  parameter BioChemUnits.Concentration
    tolerance = -1e-6;
  outer BioChemUnits.Volume V;
  BioChemUnits.Concentration c;
  BioChemUnits.MolarFlowRate r_net;
equation
  r_net = rc_1.r + rc_2.r + rc_3.r + rc_4.r +
    rc_5.r + rc_6.r + rc_7.r + rc_8.r;
  c = rc_1.c; rc_1.c = rc_2.c;
  rc_2.c = rc_3.c; rc_3.c = rc_4.c;
  rc_4.c = rc_5.c; rc_5.c = rc_6.c;
  rc_6.c = rc_7.c; rc_7.c = rc_8.c;
end BasicNode;

```

Figure 4. The partial model for the properties of a node.

In most cases the model `Node` (Figure 5) is used to represent a substance. In this type of node the concentration of the substance is allowed to change throughout the simulation without any restrictions. The total amount of substance in the node is though conserved at any time.

```

model Node
  extends BasicNode;
  extends Icons.WhiteNode;
  parameter BioChemUnits.Concentration
    c_0 = 1.0;
  BioChemUnits.AmountOfSubstance
    n(start = c_0 * V_0);
equation
  der(n) = r_net;
  c = n/V;
  assert(c > tolerance,
    "Node: c is negative!");
end Node;

```

Figure 5. The most commonly used node model.

All nodes except the node used for static reactions have an assert statement that checks that the concentration never falls lower than the tolerance below zero. If the concentration goes more than the tolerance below zero during simulation an error will be generated.

```

model FixedConcentrationNode
  extends NonStaticSubstanceNode;
  extends Icons.YellowNode;
  parameter BioChemUnits.Concentration
    c_fixed = 1;
  BioChemUnits.AmountOfSubstance n;
equation
  c = c_fixed;
  c = n/V;
  assert(c > tolerance,
    "FixedConcentrationNode: c is negative!");
end FixedConcentrationNode;

```

Figure 6. Model for nodes with fixed concentration.

Under some circumstances it is desirable to keep the concentration of a substance at a fixed value during the whole simulation. For these cases the

model `FixedConcentrationNode` (Figure 6) is used to represent the substance node. The total amount of substance in the node is still conserved at any time.

Under some circumstances it is desirable to statically pump a substance into a node from a sink or from a node into a source (Figure 7). The pump (flow) rate to or from the node is in most simulations kept at a constant level, but it is also possible to change the flow during a simulation.

```

model FixedSink
  extends Icons.YellowNode;
  extends Icons.Sink;
  parameter BioChemUnits.MolarFlowRate
    sinkFlow = 1;
  Interfaces.ReactionConnection
    sinkConnection;
equation
  sinkConnection.r = -sinkFlow;
end FixedSink;

model FixedSource
  extends Icons.YellowNode;
  extends Icons.Source;
  parameter BioChemUnits.MolarFlowRate
    sourceFlow = 1;
  Interfaces.ReactionConnection
    sourceConnection;
equation
  sourceConnection.r = sourceFlow;
end FixedSource;

```

Figure 7. Models for source and sink nodes.

4.6 Package Reactions

The package `BioChem.Reactions.BasicReactions` contains different types of elementary reactions needed in a metabolic pathway. An elementary reaction is a reaction that cannot be broken down into a simpler reaction.

For a reaction to take place there need to be at least one starting substance, the substrate, and one ending substance, the product. The substrates appear on the left side, and the products on the right side of the reaction arrow in a reaction equation. A reaction can be either irreversible, going in one direction, or reversible, going in both directions. A reaction coefficient determines the speed with which the substrate is turned into the product. The reversible reaction can be seen as two irreversible reactions and have therefore got one forward and one backward reaction coefficients.

All reactions inherit some basic attributes, such as concentration of one substrate and one product, forward reaction coefficient, and the maximum speed of the reaction (maximum volumetric reaction rate) along with some basic equations from the partial model `BasicReaction` (Figure 8).

```

partial model BasicReaction
  parameter String reactionName("");
  parameter BioChemUnits.ReactionCoef k1=1;
  parameter BioChemUnits.StoichiometricCoef
    n_S1=1;
  parameter BioChemUnits.StoichiometricCoef
    n_P1=1;
  BioChemUnits.Concentration c_S1;
  BioChemUnits.Concentration c_P1;
  BioChemUnits.VolumetricReactionRate v;
  outer BioChemUnits.Volume V;
  Interfaces.ReactionConnection rc_S1;
  Interfaces.ReactionConnection rc_P1;
equation
  c_S1 = rc_S1.c;
  c_P1 = rc_P1.c;
end BasicReaction;

```

Figure 8. The partial model for elementary reactions.

The partial models for irreversible (OneWayReaction) and reversible (TwoWayReaction) reactions are shown in Figure 9.

```

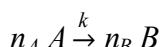
partial model OneWayReaction
  extends BasicReaction;
equation
  rc_S1.r = n_S1*v*V;
  rc_P1.r = -n_P1*v*V;
end OneWayReaction;

partial model TwoWayReaction
  extends BasicReaction;
  parameter BioChemUnits.ReactionCoef k2=1;
equation
  rc_S1.r = n_S1*v*V;
  rc_P1.r = -n_P1*v*V;
end TwoWayReaction;

```

Figure 9. The partial models for irreversible and reversible reactions.

The foundation of chemical kinetics is the so called law of mass action, which states that the rate of an elementary reaction is proportional to the amount of substance present. For the simplest elementary reaction:



k is the reaction coefficient, and n_A and n_B are the stoichiometric coefficients for the substances A and B , respectively. The reaction rate (v) for the reaction is expressed as:

$$v = k[A]^{n_A}$$

where $[A]$ is the concentration of substrate A . The Modelica code for the uni-uni irreversible reaction is shown in Figure 10.

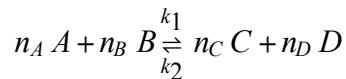
```

model OneWayReactionUniUni
  extends OneWayReaction;
  extends Icons.Irreversible1to1Arrow;
equation
  v = k1*c_S1^(n_S1);
end OneWayReactionUniUni;

```

Figure 10. The model for uni-uni irreversible reactions.

The uni-uni irreversible reaction is quite simple. A more thorny elementary reaction is the bi-bi reversible reaction:



This reaction has two substrates, A and B , which are turned into two products, C and D , under the influence of the forward reaction coefficient k_1 . The products are also reversibly turned into the substrates under the influence of the backward reaction coefficient k_2 . The reaction rate (v) for the reaction is expressed as:

$$v = k_1[A]^{n_A}[B]^{n_B} - k_2[C]^{n_C}[D]^{n_D}$$

where $[A]$, $[B]$, $[C]$, and $[D]$ are the concentrations of the substances A , B , C , and D , respectively. The Modelica code for the bi-bi reversible reaction is shown in Figure 11.

```

model TwoWayReactionBiBi
  extends TwoWayReaction;
  extends Icons.Reversible2To2Arrow;
  parameter BioChemUnits.StoichiometricCoef
    n_S2=1;
  parameter BioChemUnits.StoichiometricCoef
    n_P2=1;
  BioChemUnits.Concentration c_S2;
  BioChemUnits.Concentration c_P2;
  Interfaces.ReactionConnection rc_S2;
  Interfaces.ReactionConnection rc_P2;
equation
  c_S2 = rc_S2.c;
  c_P2 = rc_P2.c;
  rc_S2.r = n_S2*v*V;
  rc_P2.r = -n_P2*v*V;
  v = (k1*c_S1^(n_S1))*c_S2^(n_S2) -
    (k2*c_P1^(n_P1))*c_P2^(n_P2);
end TwoWayReactionBiBi;

```

Figure 11. The model for bi-bi reversible reactions.

Reactions can also be under the influence of an inhibitor or an activator. An inhibitor is a substance that through its presence slows the reaction down, but is neither consumed nor changed during the process. The basic partial model for an inhibited reaction inherits properties from the basic irreversible reaction and is thereby an irreversible reaction. The difference is the addition of the inhibitor, and some equations making sure that the inhibitor is not consumed during simulation (Figure 12).

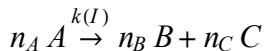
```

partial model InhibitedReaction
  extends OneWayReaction;
  extends Icons.InhibitorSign;
  extends Icons.SingleArrowModulation;
  BioChemUnits.Concentration c_I;
  Interfaces.ReactionConnection rc_I;
equation
  c_I = n_I.c;
  rc_I.r = 0;
end InhibitedReaction;

```

Figure 12. The partial model for inhibition reactions.

For the elementary inhibited reaction:



$k(I)$ is the reaction coefficient, and n_A , n_B and n_C are the stoichiometric coefficients for substance A , B , and C , respectively. The reaction rate (v) for the reaction is expressed as:

$$v = k(I)[A]^{n_A}[I]^{-1}$$

where $[A]$ and $[I]$ are the concentrations of substrate A and the inhibitor I . The Modelica code for the uni-bi irreversible inhibition reaction is shown in Figure 13.

```
model InhibitedReactionUniBi
  extends InhibitedReaction;
  extends Icons.Irreversible1To2Arrow;
  parameter StoichiometricCoef n_P2=1;
  BioChemUnits.Concentration c_P2;
  Interfaces.ReactionConnection rc_P2;
equation
  c_P2 = rc_P2.c;
  rc_P2.r = -n_P2*v*V;
  v = k1/c_I*c_S1^(n_S1);
end InhibitedReactionUniBi;
```

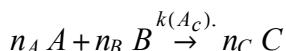
Figure 13. The model for uni-bi inhibition reactions.

An activator is a substance that through its presence speeds up the reaction, but is neither consumed nor changed during the process. The basic partial model for an activated reaction looks just like the partial model for the inhibited reaction (Figure 14).

```
partial model ActivatedReaction
  extends OneWayReaction;
  extends Icons.ActivatorSign;
  extends Icons.SingleArrowModulation;
  BioChemUnits.Concentration c_Ac;
  Interfaces.ReactionConnection rc_Ac;
equation
  c_Ac = n_Ac.c;
  rc_Ac.r = 0;
end ActivatedReaction;
```

Figure 14. The partial model for activation reactions.

For the elementary activated reaction:



$k(Ac)$ is the reaction coefficient, and n_A , n_B and n_C are the stoichiometric coefficients for substance A , B , and C , respectively. The reaction rate (v) for the reaction is expressed as:

$$v = k(Ac)[Ac][A]^{n_A}[B]^{n_B}$$

where $[A]$, $[B]$ and $[Ac]$ are the concentrations of substrate A and B , and the activator Ac . The Modelica code for the bi-uni irreversible activation reaction is shown in Figure 15.

```
model ActivatedReactionBiUni
  extends ActivatedReaction;
  extends Icons.Irreversible2To1Arrow;
  parameter BioChemUnits.StoichiometricCoef
    n_S2=1;
  BioChemUnits.Concentration c_S2;
  Interfaces.ReactionConnection rc_S1;
equation
  c_S2 = rc_S2.c;
  rc_S2.r = -n_S2*v*V;
  v = k1*c_Ac*c_S1^(n_S1)*c_S2^(n_S2);
end ActivatedReactionBiUni;
```

Figure 15. The model for bi-uni activation reactions.

4.7 SBML

The Systems Biology Markup Language (SBML) is a computer-readable format for representing models of biochemical reaction networks. SBML is applicable to metabolic networks, cell-signaling pathways, genomic regulatory networks, and many other areas in systems biology [8].

In order to make models created in Modelica interchangeable with other biological or chemical simulation and modeling tools a two-way translator between Modelica and SBML is under development. The package BioChem.Reactions.SBMLReactions contains reactions specified in the second release (Level 2) of SBML [3, 5].

5 Current work

To make it even easier for biologist/chemist to use Modelica for modeling and simulation a drag-and-drop graphical interface is currently being developed for MathModelica [4, 7]. Current work also focuses on building pathway models using the classes in the BioChem package.

Acknowledgments

The authors wish to thank Eva-Lena Lengquist Sandelin, Linköpings universitet for valuable feedback, and MathCore Engineering AB for supplying the MathModelica tool. Emma Larsdotter Nilsson was in part funded by the Swedish National Graduate School in Computer Science (CUGS).

References

- [1] Fritzson, P. Principles of Object-Oriented Modeling and Simulation with Modelica. IEEE Press and Wiley. 2003.
- [2] Fritzson P and P. Bunus. Modelica – A General Object-Oriented Language for Continuous and Discrete-Event System Modeling and Simulation. (In Proceedings of the 35th Annual Simulation Symposium. San Diego, California, April 14-18, 2002. IEEE Press. 2002.
- [3] Finney, A. and M. Hucka. Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions. <http://www.cds.caltech.edu/erato/sbml/docs/>. 2003.
- [4] Fritzson P, J. Gunnarsson, and M. Jirstrand. MathModelica – An Extensible Modeling and Simulation Environment with Integrated Graphics and Literate Programming, In Proceedings of 2nd International Modelica Conference Munich, March 2002 (www.modelica.org). 2002.
- [5] Hucka, M. et al. Systems Biology Markup Language (SBML) Level 1: Structures and Facilities for Model Definitions. <http://www.cds.caltech.edu/erato/sbml/docs/>. 2003.
- [6] Wolfram, S. The Mathematica Book. Wolfram Media. 1997.
- [7] MathModelica web site. www.mathcore.com.
- [8] SBML web site. www.sbml.org.

Session 7A

Mechatronic Systems – I

A Remote User Interface to Modelica Robot Models

G. Ferretti, M. Grittì, G. Magnani, and P. Rocco
 Politecnico di Milano - Dipartimento di Elettronica e Informazione
 Via Ponzio 34/5, 20133 Milano (Italy)
 phone: +39 02 2399 3673, fax: +39 02 2399 3412
 e-mail: {ferretti, gritti, magnani, rocco}@elet.polimi.it

Abstract

A remote interface for simulating robots via the Internet is presented. This environment is dedicated to simulations of robotic arms whose models are written in some descriptive language like Modelica. The actual application runs on robot models compiled by Dymola. This work has been developed within a project whose purpose is the Modelica/Dymola implementation of models of space robots and servomechanisms, and their dissemination to partner firms and to the scientific community. The remote simulation environment has been tested with a model of the Spider robotic arm.

1 Introduction

The purpose of the SIMECS¹ (Integrated MEChatronic Simulation for Space systems) project is to build up a library of models of mechatronic components used in space systems. Such components are building blocks for virtual prototyping robots and systems which will be exploited in future space missions.

Typical simulation environments like Dymola [1] or Simulink [2] embed powerful modeling toolboxes for building models of any kind of dynamic system. This gives the user the maximum flexibility and applicability. But once a model has already been built and tested by an expert user of such tools, only a user friendly simulation software is needed, while modeling functionalities become superfluous. That could be happening during the fine tuning of the model parameters, or while inspecting the behavior of the system for programming the sequence of working actions.

Moreover, there are scenarios in which the model-

ing expert and the ultimate simulation user not only are distinct persons, but also work in different places for distinct organizations that cooperate on the same project. This happens in SIMECS. Within this project, extremely complex models of robotic components were developed as academic research topic; then component models were assembled into models of robot² prototypes used as case studies. Working robot models will be finally put at disposal of partner firms which have to build the various parts of one real robot.

The presented scenario is not isolated, because many little establishments exist that do not have the expertise for developing complex models using complex modeling environments, and cannot afford to invest in it, but could improve the quality of their production by means of studying on pre-packed models with *easy-to-use* software tools.

The approach of running simulations remotely, instead of deploying the compiled models on site, is supported by the assumption that who builds the models should also be capable of dimensioning them to the computational power available for simulation. On the contrary, deploying models to the unknown computers of the end users would entail the risk that their computational power could be inadequate for running the simulations in a reasonable time, which would mean to make the deployed models practically unusable.

2 Application Requirements

The SIMECS-RI (SIMECS-Remote Interface) is a visual simulation environment dedicated to simulations of executions of commands assigned to robotic arms which operate in space. Requirements of such an application have been traced from different perspectives.

¹The project website is online at <http://www.elet.polimi.it/res/simecs/>. The latest version of the client of the application which is described in this paper can also be downloaded there.

²As it will be said in section 5, the robot taken into account is the Spider arm, for which different control system solutions were tested by means of a number of simulations.

2.1 Functional Requirements

The application should be flexible enough to deal with any kind of manipulator. This means it should be able to present information about any kind of mechanical chain and motion control system, and it should be also able to initialize and run simulations of robots which accept commands at different abstraction levels.

Information to be presented about the robot model can be divided into three categories:

- Robot model documentation;
- Model parameters and main variables;
- Virtual model of the robot.

The model documentation could only be read and no change to the model should be allowed. This means, for example, that neither links could be attached to or removed from the mechanical chain, nor dynamic or algebraic blocks could be attached to or removed from the control system. The simulation environment should instead allow the user to change the values of the model parameters, such as all the gains in the motion control system, or the masses and lengths of the physical links. The application should possibly remember parameter value changes after the simulation, for further reutilization. Also the main variables of the model should be accessible, and the user should be allowed to plot variables, for transient analysis. Possibly, variable plots should be visible during the simulation also and updated while the simulation is in progress. Virtual prototype animation should be allowed after the simulation, and possibly also while simulation is in progress.

Depending on the model inputs, robot commands that one could assign for simulation are:

Direct input	$\left\{ \begin{array}{l} \text{Open loop: current amplitudes,} \\ \text{Closed loop: canonical set points;} \end{array} \right.$
Movement	$\left\{ \begin{array}{l} \text{Joint space,} \\ \text{Cartesian space;} \end{array} \right.$
High level	$\left\{ \begin{array}{l} \text{Action,} \\ \text{Task,} \\ \text{Mission.} \end{array} \right.$

For each of these categories, a user friendly way for selecting the actual command should be provided.

Finally, the application should also feature an easy way for choosing the initial robot configuration.

2.2 Operational Requirements

Basically, an application like SIMECS-RI should allow the user to choose a robot model from a library of available models, to assign model parameters, and arbitrary initial state, to define assign and simulate the execution of a robot command, and finally to analyze the simulation results.

The precise steps of a typical use case are:

1. choice and loading of the robot model;
2. *model inspection and parameters configuration*;
3. choice of the variables which values should be updated during the simulation;
4. *choice of the initial state*;
5. *robot command definition*;
6. simulation parameters setup;
7. *robot command assignment*;
8. simulation, with online result presentation;
9. loading of the transients of the variables which trend has to be analyzed;
10. variables transient analysis;
11. result saving.

Items that have been emphasized constitute functionalities which should be given special attention during the design phase. These features should be designed in a way that enhances the usability of the robot simulation interface. On the contrary all other functionalities are not affected by the requirement of building an application dedicated to robotic arms.

Documentation could be loaded together with the robot model, and presented simply as a collection of data sheets of the robot mechanical chain and control system. Hyper textual format could be adequate, so the user would navigate inside and outside of robot components and subcomponents, seeing the component connections and relationships, with all model equations explained, and all constant values listed.

Parameters should be well separated from the read-only documentation. They should be presented in a clear way: possibly a list, or maybe a tree in which those belonging to the same component are grouped together under the same branch. Parameters should be also coupled with their descriptions.

The robot virtual prototype should be displayed in a separate window; possibly, the 3D robot model should be surrounded by a model of its real environment. Solid shapes are preferable to wire frames. Changes in parameter values which affect the robot appearance should be reflected immediately in the virtual prototype.

Before performing the simulation, robot initial position should be assigned by choosing the positions of its joints. It could be assumed that the robot is always initially still. This restriction is consistent with the fact that when a command is assigned to a real robot, the robot is always still.

Concerning command definition, if the restriction of dealing only with robotic arms is adopted, robot commands belong to well known categories. So commands and trajectory parameters can be separated from other parameters, and the way a command is issued can be differentiated from a simple change in some input parameter. Robot commands should be edited in a comfortable manner. A visual tool emulating one distinct teach pendant for each level of abstraction is preferable with respect to a textual prompt where each command is issued by entering a line of text.

Finally, robot command assignment could simply coincide with simulation startup.

to run on a server computer, while simulation results should be presented on a client computer; second, simulations have to be performed by an application which is external to the SIMECS-RI server.

The choice of exploiting an external software tool for dynamic simulation is motivated by the fact that several tools [1][2][3] already exist which are capable of that, and so it would be anachronistic to start by now the development of a new one. On the other hand, since exploiting an already existing simulation tool means to exchange with it information about robot models and model inputs, most important would be now to compensate the lack of standards in the way robotic systems are defined. This would simplify the interaction with such generic simulation tools.

3 Application Structure

The SIMECS-RI overall structure is shown in figure 1 through a UML [4][5] deployment diagram. This is a pure client/server structure. The diagram emphasizes the fact that the application core is by now only a proxy to a distinct process which performs the simulations. The server can handle multiple connections, which means that many users at a time can perform their own simulation. The maximum number of contemporary users is limited by the computational power of the server, which should be capable of running one simulation per user.

3.1 Server Structure

The structure of the server is shown in figure 2 through an UML component diagram. As it can be seen, a component based architecture has been adopted. The three main components of the application are the Simulation Server, the Low Level Proxy, and the Hot Feedback Manager. The Low Level Simulator Machine, which is the process that actually simulates the robot motion and motion control, does not belong to the application itself. Both the Low Level Simulator Machine and SIMECS-RI have their own configuration files. The last component, i.e. the Action Level Simulator is not currently implemented, and will be useful for future extensions which will be illustrated in section 6.

The Simulation Server waits for requests incoming from the client, and parses and executes them once they arrive. To serve the requests, the Simulation Server relies on the interfaces that the Low Level Proxy exports. The client of the Simulation Server

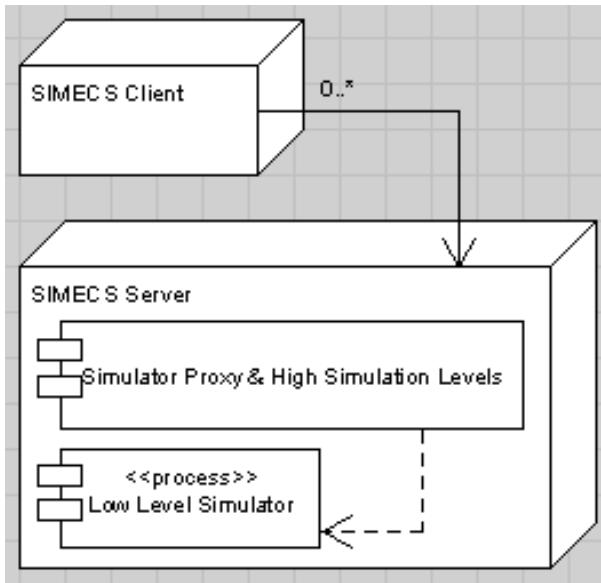


Figure 1: Application overall structure.

2.3 Structural Requirements

The structural requirements of the application are mainly two: first, as said in section 1, simulations have

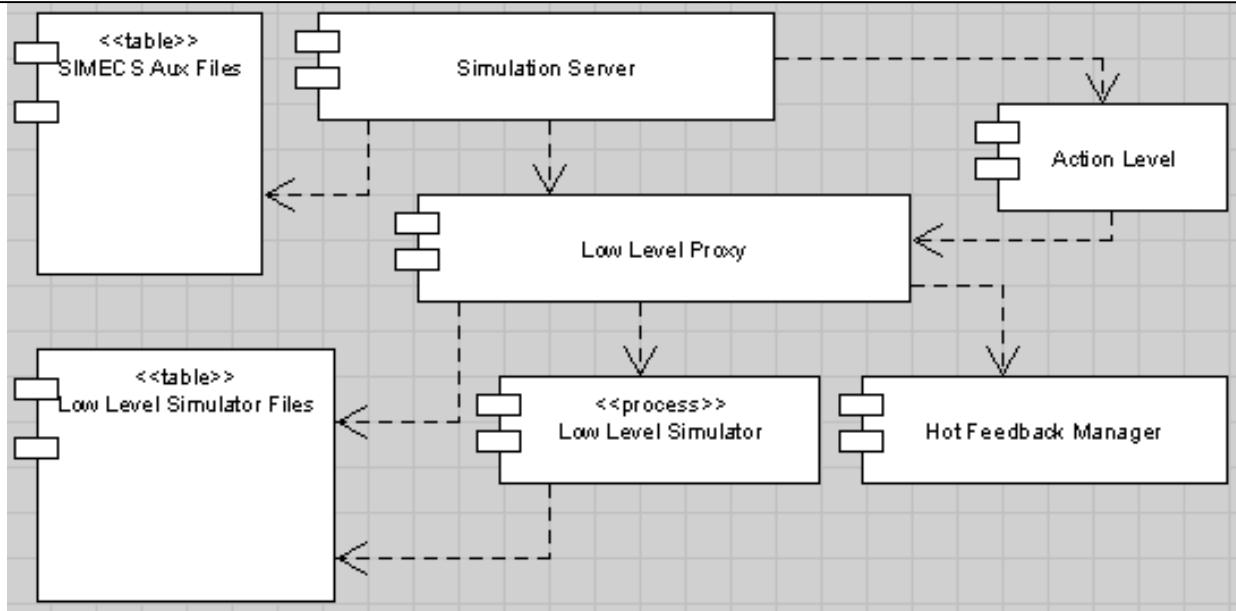


Figure 2: Server structure.

is a component of the SIMECS-RI client. These two physical components together constitute a conceptual *Communication* component.

This design has been adopted because it is general, and does not obstruct any kind of possible future development. A client/server mechanism has been adopted in spite of a remote procedure call mechanism for decoupling the two main parts of the application. Indeed, a communication layer based on an application-level protocol allows the client and the server to be implemented by means of different technologies. This also allows any one to implement a client for the SIMECS-RI, provided that he respects the communication protocol. On the other hand, abstracting a conceptual communication component allows at any step of the development process to change the way communication is performed, without affecting any other part of the application. This simplifies the way both the communication protocol and the communication mechanism can be changed, if this is considered useful.

The Low Level Proxy exports some interfaces used for gathering the robot model and the simulation results from the files of the external simulator, and some other interfaces used for controlling the external simulator itself. Building a component dedicated to the translation of simulator independent commands into simulator dependant ones has been useful during the design phase because it allowed to abstract from their implementation in the exploited simulator. A separate Low Level Proxy component is useful for the maintenance of the application also, since it allows to more easily

change the external simulator by means of changing the proxy, and letting the rest of the application unchanged.

Finally, the Hot Feedback Manager watches for the results (i.e. the transient of the variables) incoming from the simulator while a simulation is in progress, and outputs these results to the client, so they can be immediately presented to the user. Results are gathered by the Low Level Proxy from the simulator through some mechanism of interprocess communication, and fired as events to the Hot Feedback Manager. Event paradigm has been adopted since result arriving is unpredictable. The Hot Feedback Manager can reduce the sampling rate of the results that are forwarded. Not exceeding a prefixed forward rate prevents the communication channel from saturations. If a user wants to analyze the entire transient of a variable he can in any case reload it when the simulation is terminated.

3.2 Client Structure

The structure of the client is shown in figure 4 through a UML component diagram. The two main components of the client are the User Interface and the Simulation Client. The event paradigm has been adopted for exchanging information between them. This means that as soon as the User Interface receives a command from the user as a signal coming from the computer hardware, whenever this command implies a request to the SIMECS server, the User Interface itself propagates the signal as an event to the Simulation Client.

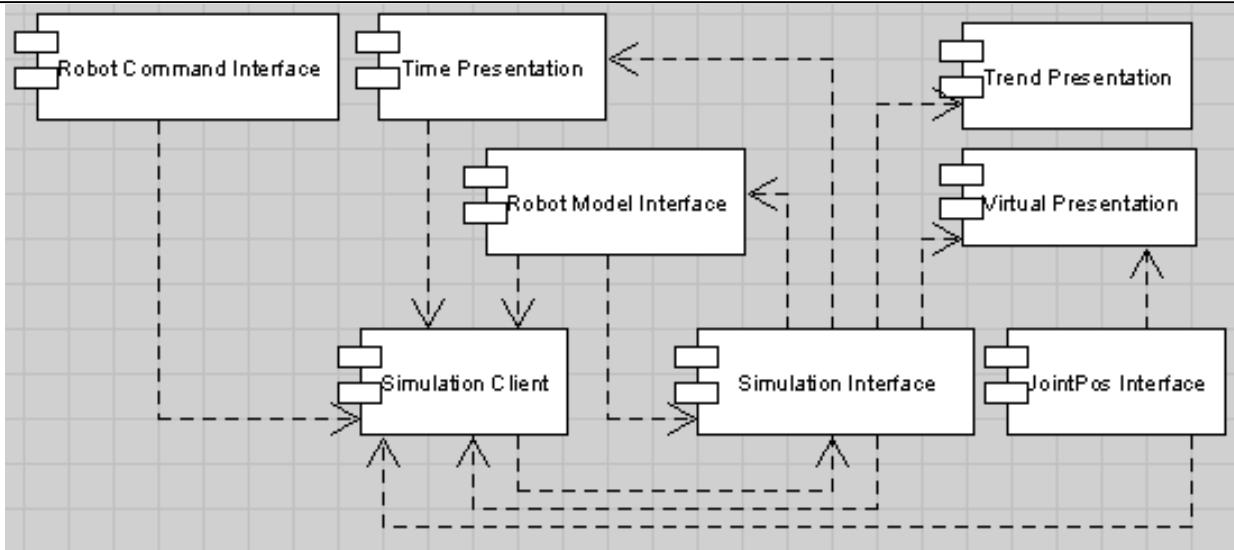


Figure 3: User interface structure.

Also when the Simulation Client reads results coming from the server, it fires events for propagating data to the User Interface.

The Simulation Client component is the counterpart of the Simulation Server component of the SIMECS-RI server. Its purposes are to translate the requests of the user into strings of the communication protocol and to dispatch the results coming from the SIMECS-RI server.

all other visual sub-components and of assigning them a place on screen. Visual components are dedicated to a specific type of interaction with the user. The Joint Position Interface allows the user to drive the joints of the robot into a desired position. The Robot Command Interface allows the user to build and assign a path in the Cartesian or joint space. The Robot Model Interface allows the user to inspect the robot parameters and variables, to change the parameters values, and to select the variables which transient he wants to see. The Time Presentation indicates the progress of the simulation, with respect to its total time; it also allows the user to pause or abort the simulation. The Trend Presentation displays the transient of a variable, through an interactive plot³. Finally, the Virtual Presentation displays the robot virtual prototype within its workspace.

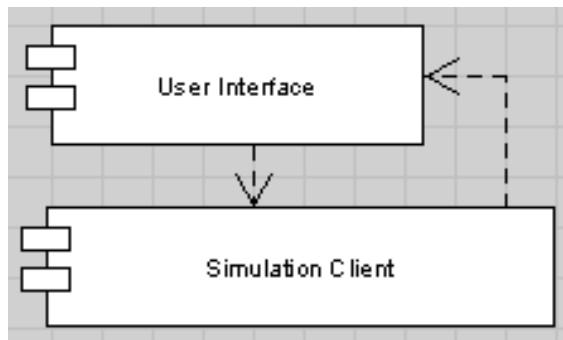


Figure 4: Client structure.

The User Interface is a super-component. Figure 3 shows the User Interface decomposed into its sub-components. For assuring the maximum decoupling, their interconnections also are asynchronous. The main sub-component is the Simulation Interface; other sub-components are the Joint Position Interface, the Robot Command Interface, the Robot Model Interface, the Time Presentation, the Trend Presentation, and the Virtual Presentation.

The Simulation Interface is responsible of instantiating

4 Application Technology

The SIMECS-RI has been implemented in Java [6]. Java is a pure object-oriented programming language, and exploiting of interfaces to the classes it allows the realization of well decoupled software components. Moreover, Java is a general purpose language, and it is provided with libraries that can be exploited for building applications which span over many different programming fields.

A language which is pure object-oriented makes it harder to obtain inconsistencies between the UML

³User can see more than one plot at a time, by creating multiple instances of the Trend Presentation component.

structural design and its implementation. This improves the software quality, and makes it easier to maintain and evolve the application. Improvements are important also in the design in-the-large, were Java interfaces have been exploited to export the methods of the various components.

JavaBeans technology has been adopted for the implementation of the SIMECS-RI client. Beans are the Java proposal for component-oriented software architectures. Beans interact exchanging events, and are well suited for light-weight visual components⁴ like those of the SIMECS-RI client.

The SIMECS-RI graphical user interface has been implemented using the *Swing* library for windowing, the *Java2D* library for graph plots, and the *Java3D* [7] library for the robot virtual presentation. Thanks to Java portability, the SIMECS-RI client can be run on any platform⁵ for which such libraries exist.

The SIMECS-RI server, on the other hand does not need an interface to the user, but one for interacting with the simulator process, and it is constrained to run on the same platform for which the simulator has been compiled. In the actual version of the SIMECS-RI server, Dymosim has been exploited as external simulator. Dymosim is a Windows executable file automatically generated by Dymola [1], by means of translating the Modelica code into C code, and then compiling the C code. Since Dymola compiles a new Dymosim executable for every Modelica model, a library of Dymosim executables is stored on the server, each of which corresponding to a single compiled model.

The Dymosim executable reads the simulation settings and the actual values of the parameters of its own model from an input file, and stores into an output file the variables transient evaluated during the simulation. So, the Proxy Simulator can actually exchange information with the simulator by writing its input file, and by reading its output file.

Dymosim allows another way also for exchanging data: the DDE (Dynamic Data Exchange) [8] inter-process communication technology. DDE is Windows native, and is based on shared memory areas. Communication through DDE has been made possible in Java by exploiting the JNI (Java to Native Interface) API. It is so by means of DDE that the Proxy Simulator gathers from Dymosim the data that are forwarded to the SIMECS-RI client while simulation is in progress.

⁴For example, all the *javax.swing* library of windowing components is implemented through JavaBeans.

⁵SIMECS-RI has been tested successfully, by now, on Linux and Windows.

5 A Case Study: The Spider Arm

The SIMECS-RI has been tested on compiled Modelica models of the Spider robotic arm, which is shown in figure 5. By now, three different types of motion control systems have been modeled and applied to the same blocks of the Spider mechanical chain and actuators array [10]. These are: joint independent control, operational space motion control, and operational space hybrid control. At current stage of development, SIMECS-RI is able to work with the first of them.



Figure 5: The Spider arm.

5.1 The Robot Model

The Spider model with joint independent control system has been provided with a joint space trapezoid speed trajectory generator which accepts as inputs the initial and final positions of each joint, and the percentages of the maximum values of joint speed and acceleration. The model [10] features *P/PI* cascade controllers with motor and joint position sensors, dynamics of brushless two-phase motors, current controllers, elastic transmissions with backlash and friction, and a seven degrees of freedom multi-body chain with optional payload. A total of more than 12,000 equations are listed at compile time.

The Modelica model of the motor (see⁶ figure 6) describes the electrical dynamics of the two phases, the electro-mechanical conversion (block *EMF_2*), the

⁶All figures referred within this section are Dymola [1] schemes.

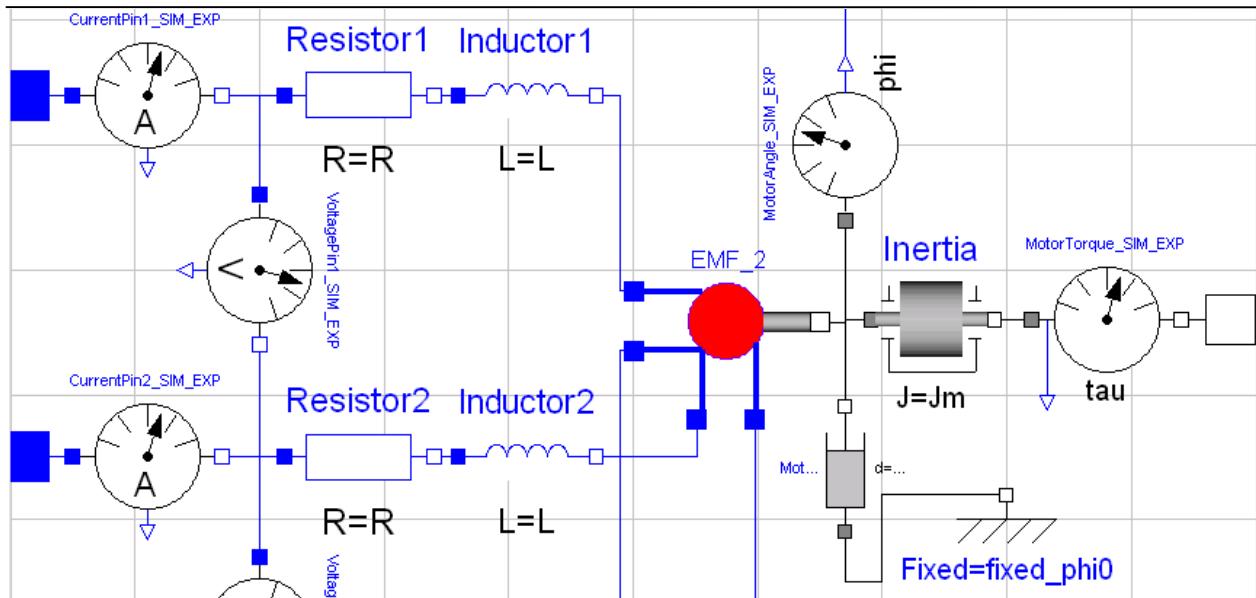


Figure 6: Scheme of the Modelica brushless two-phase motor model.

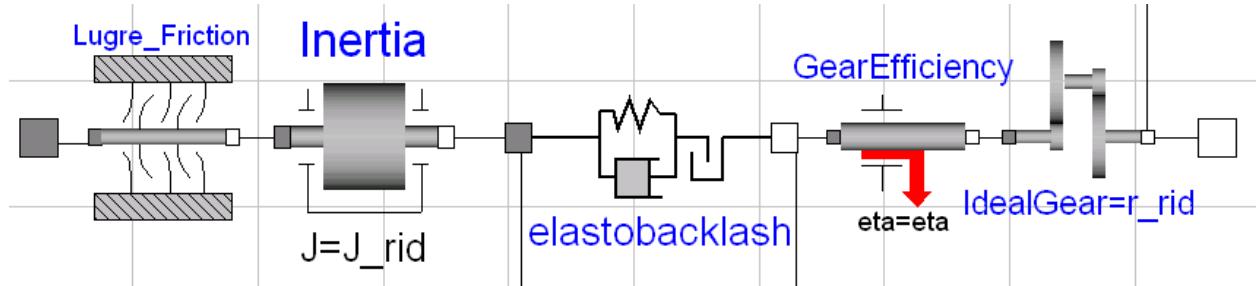


Figure 7: Scheme of the Modelica elastic gear model.

equivalent rotor inertia, the viscous friction of the motor, and it also includes the encoder. The electro-mechanical conversion block offers the interesting possibility of simulating the effect of the most important torque disturbances due to the motor dynamics, like the ripple caused by torque phase unbalanced, the ripple due to shape functions imperfections, and the detent torque, which is present also when current is null.

The Modelica model of the analog current controller includes two analog *PI* regulators with anti-windup compensation. This component allows to set the value of the current offset, which is useful to simulate sensor polarization. This way, this component can reproduce a torque disturbance on the motor.

A realistic transmission model has been built using the Rotational objects taken from Modelica Standard Library. The Gear_Box Modelica component (see figure 7) includes:

- a continuous-nonlinear friction model (LuGre);

- a mechanical efficiency model;
- an equivalent gear train inertia model;
- torsion flexibility, damping and backlash models;
- an ideal reducer model.

The analog⁷ joint control system is equipped with two resolvers for each joint (at motor and load sides). The control scheme is constituted by an inner loop (*PI* part), for motor speed control, and by an outer loop (*P* part), for joint position control. Anti-windup compensation mechanism and velocity feed-forward are also present in the inner loop.

The Spider mechanical chain model has been implemented exploiting the Modelica Multi-body library. In order to make the SIMECS-RI application able

⁷A digital control system has been implemented and tested as well, but in order to maximize the speed of the remote simulations, its equivalent analog version has been finally preferred.

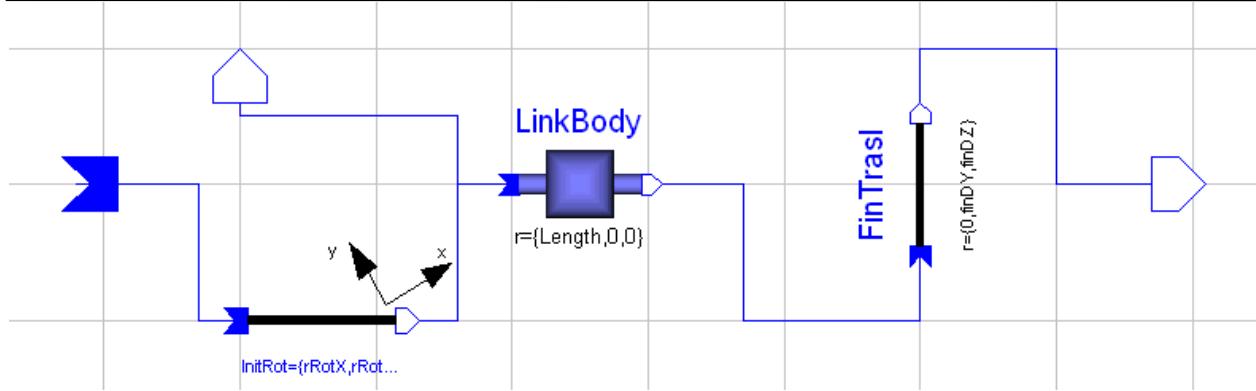


Figure 8: Scheme of the Modelica link model.

to graphically represent and animate robots, a general way of specifying robot kinematical chains with graphical appearance has been conceived. This allows to change link lengths at simulation time without loosing the link appearances.

In figure 8 the Modelica scheme of the Link component in the Spider mechanical chain is shown. As it can be seen, a central translation with associated body is preceded by an initial rotation and is followed by a final translation. The initial rotation has a null component around the axis of the joint preceding the link, and the final translation has a null component along the axis of the link. The central translation can have a non-zero component only along the x axis, and its value equals the length of the link.

Shapes that can be attached to the robot links are currently modeled in separate Modelica components. Many elementary solid shapes can be attached to the same robot link (i.e. mass plus roto-translation), but only one shape object can change its dimension along one of its axes according to the link length. The resizable shape can be virtually of any type, but it is more appropriate if it has a constant orthogonal section with respect to its resizable axis. This in order to avoid misshaping the robot link appearance in the 3D model, when the length is changed. Modelica shapes that are appropriate in this sense are Cylinder, Pipe, Box, and Beam.

5.2 The Application

The main window of SIMECS-RI is shown in figure 9, with the joint controlled Spider arm model loaded. As it can be seen, the window is divided in three main sections: at the left side there is the parameters tree, in the middle is the robot virtual presentation, and at the bottom is the joint command panel. Additionally,

a simple toolbar is placed at the top of the window. The toolbar is used to access the simulation parameters setup window, the model documentation window, and the models list window. The initialization panel is also accessible by means of this toolbar.

The initialization panel (which is not shown in figure 9) is a popup menu, and is identical to the one that is shown in figure, which is used to assign destinations in a joint space path. The initialization panel is constituted by an array of sliders, one per joint. When the user changes the robot joints positions, the 3D robot model is immediately updated.

In practice, the user can move the robot by acting directly on its joints. This is also known as kinematical simulation. By means of this feature, the user can immediately perceive the position of the robot moving within its environment, and so he/she can easily place the robot in the desired initial position.

Notice that, from the user point of view, initialization consists *only* in choosing the joint positions of an initially motionless robot. All internal states of the model components⁸ which have to be updated for maintaining consistency with the initial joint positions are automatically computed by the SIMECS-RI server on the basis of some algebraic expressions stored in an auxiliary file. These expressions state the relations between the joint positions and the unknown quantities when the robot is in an equilibrium state⁹, and should be supplied with the robot model.

Robot model parameters and variables trees share the same space in the window, and it is possible to switch

⁸In the actual model these are motor initial positions, and the initial states of the pseudo-derivatives blocks of the control systems.

⁹For example, the algebraic relation between each motor angle and the correspondent joint angle is simply $q_m = nq_l$, where n is the gear ratio. The effect of an elastic transmission is neglected since the robot is supposed to operate in zero-gravity conditions.

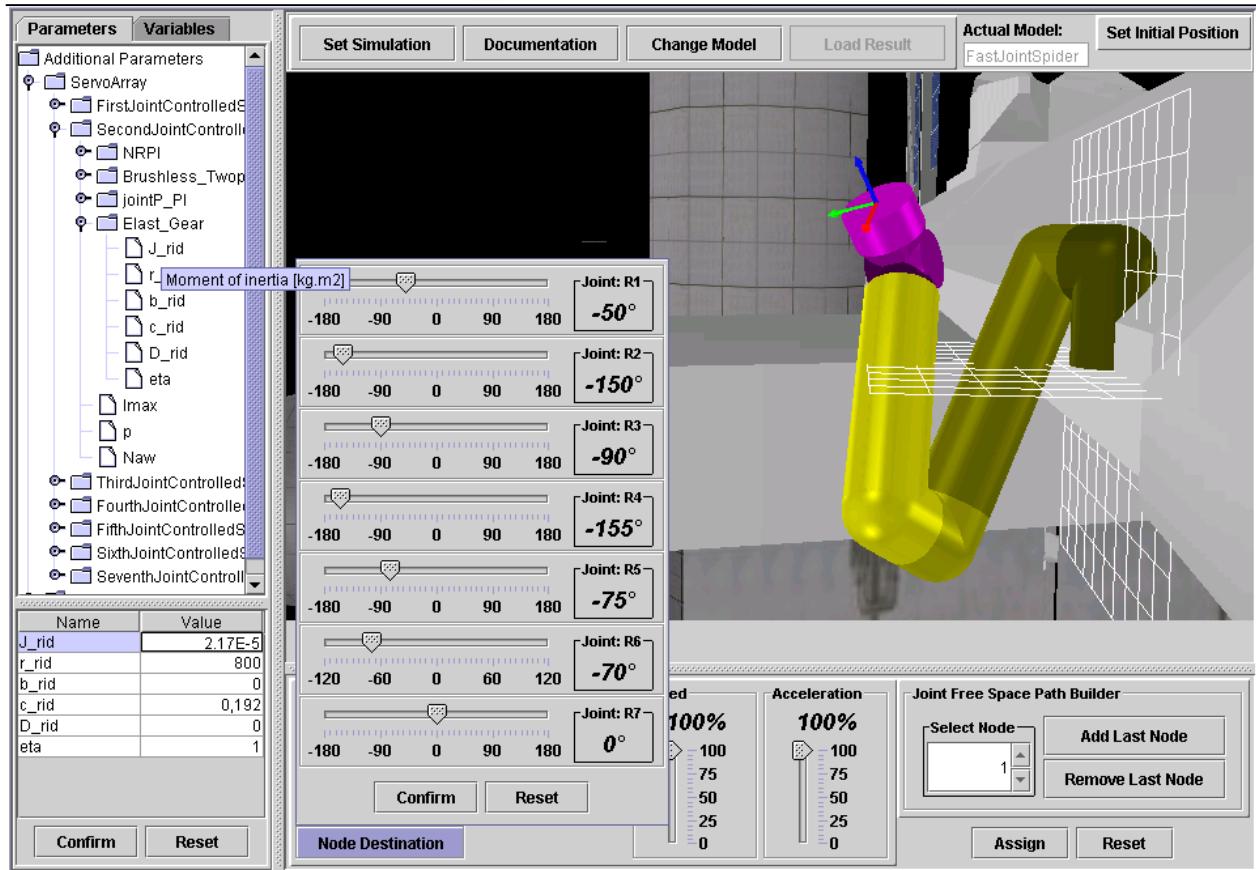


Figure 9: The SIMECS-RI graphical user interface.

from one to the other by choosing the corresponding tab at the top left corner. Parameter values can be changed by means of the table at the bottom left corner, while parameter descriptions appear as tool tip texts. Variables plots are accessed by double-clicking the variable names in the variables tree.

The 3D graphics model is built by interpreting information stored in the simulator input file. This can be done provided that the guidelines sketched in section 5.1 are followed while the modeler builds the kinematical chain of the robot and its graphical appearance

The joint command panel is used for building and assigning joint free space paths, which are sequences of path nodes. Every node has a destination (i.e. a via point of the overall path), and two parameters which state speed and acceleration reduction to be applied in the corresponding path segment. Paths that can be assigned are a subset of the ones that can be defined by means of the PDL2 [9] move instruction. Actually, only the first node of any path is really issued to the simulator. This limitation is applied for compliance with the trajectory generator of the model.

6 Future Work

SIMECS-RI is a complete simulation environment for robotic arms moving in free space. By now, robot commands can be given at joint level. The most natural way of extending such an application is to make it capable of dealing with more complex simulations, always within the field of articulated robotics.

First, it is planned to handle the case of simulations of contact situations between the robot and its surrounding environment: this includes both the case of a manipulator grasping objects and the case of a robot whose end effector slides onto a surface; second it is planned to allow to perform simulations of movements in an environment with obstacles. Extensions to such cases not only imply to design and implement novel command interfaces for robotic arms, but also to design the corresponding command interpreters, and, last but no least, to build models that can handle such new complexities.

It is planned also to modify the SIMECS-RI server in order to make it able to interface itself with hardware-in-the-loop simulators, where only the robot electro-

mechanical parts are simulated, while the motion controller is a real one. Finally, the application user interface can be extended to allow telemanipulation of a real robot.

- [10] L. Viganò, *Modellistica del Braccio Robotico Europa con Analisi del Controllo nello Spazio Operativo* [in Italian], Master's Thesis, Politecnico di Milano; June 2003.

Acknowledgments

The authors would like to thank Gianpaolo Cugola for its important contribution to the design of the SIMECS-RI application, Paolo Donadeo for the implementation of the Virtual Presentation component, and Luca Viganò for having provided all Modelica models of the SIMECS library.

References

- [1] *Dymola Multi-Engineering Modeling and Simulation* [Online]. Available:
<http://www.dynasim.se/>
- [2] *Simulink: Design and Simulate Continuous and Discrete-Time Systems* [Online]. Available:
<http://www.mathworks.com/products/simulink/>
- [3] *MBDyn - MultiBody Dynamics Software*, [Online]. Available:
<http://www.aero.polimi.it/~mbdyn/>
- [4] *OMG UML Specification* [Online]. Available:
<http://www.omg.org/uml/>
- [5] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modelling Language User Guide*, Addison-Wesley Pub Co; 1st ed. September 1998.
- [6] B. Eckel, *Thinking in Java*, Prentice Hall PTR; 3rd ed. December 2002.
- [7] D. J Bouvier, *Java3D API Tutorial*, Sun Microsystems Inc. [Online]. Available:
<http://developer.java.sun.com/developer/onlineTraining/java3d/>
- [8] C. Petzold, *Programming Windows*, Microsoft Press; 1st ed. 1988.
- [9] *Linguaggio di Programmazione PDL2 - Versione 3.0* [in Italian], COMAU S.p.A. Robotics Division; 1992.

Physical-based Friction Identification of an Electro-Mechanical Actuator with Dymola/Modelica and MOPS

Angelika Peer

Naim Bajcinca

Christian Schweiger

German Aerospace Center (DLR)
 Institute of Robotics and Mechatronics
 Oberpfaffenhofen, 82234 Weßling, Germany
<http://www.robotic.dlr.de/control/>

Abstract

An identification procedure consisting of iterative parameter optimization and model validation tasks using the optimization tool MOPS and Dymola/Modelica simulation environment is presented. This method is used for modelling of a force-feedback electro-mechanical actuator with Harmonic-Drive gear. A modelling approach for speed and torque dependent gear losses introduced in a prior work is validated.

1 Introduction

Several objectives, such as model-based control, simulation and design of complex systems require accurate system models. Especially, mechanical systems exhibit complex nonlinear phenomena, e.g. stick-slip effects, whose modelling may play an essential role in the dynamics of the whole system. Such complex modelling tasks require tools, which should provide a clear hierarchical model structure, efficient equation solvers and fast component parametrization. These requirements are e.g. fulfilled by Dymola/Modelica simulation environment. Modelica is a physical object-oriented modelling language suitable for modelling and simulation of heterogeneous multi-physical systems. It is designed in such a way, that the user can build a physical model in a natural way, as he would build it in real-world. Additionally, due to symbolical code preprocessing, Dymola/Modelica enables real-time simulation of complex physical systems, [OE00]. While the structure of a model is physically defined by Modelica, yet for modelling completion, its parameters need to be computed or identified via measurements. A convenient environment for parameter

identification is the optimization tool MOPS (Multi-Objective Parameter Synthesis), [JBL⁺02]. Multi-objective optimization is enhanced by providing robust gradient-free direct-search solvers and an intuitive user interface. Parameter optimization with MOPS is especially convenient since different measurement data can be handled simultaneously in the context of a multi-objective optimization task with respect to different criteria types (typically least-squares).

The main aim of this paper is to present an identification procedure for accurate modelling in the example of an electro-mechanical actuator. Therefore an identification feedback-loop consisting of iterative parameter optimization and model validation tasks. While the latter is performed in a Dymola/Modelica simulation environment, the parameter optimization is done in MOPS.

A natural way of a parameter identification task is to split it in subtasks by discriminating between different physical conditions, which primarily excite a certain parameter subset. This paper uses this strategy for separate identification of linear stiffness, damping and inertia, as well as, non-linear bearing- and mesh-friction parameters. Thereby, a modelling formalism for gear friction as proposed in [PSO02] has been used. The latter work introduces a tabular description of friction (loss table), which includes speed- and torque-dependent gear losses terms, i.e. bearing- and mesh-friction parameters for braking and driving gear conditions. While carrying out of physical conditions needed to measure the loss table sets great demands on technical equipment, in this paper it is shown that identification is an effective alternative.

The paper is organized as follows. In the next section the electromechanical force-feedback actuator is

introduced. Section 3 describes the identification loop with MOPS and Dymola/Modelica. Section 4 provides the identification of a linear actuator model, including the Dymola/Modelica actuator scheme and linear parameter identification with MOPS. Section 5 recalls the modelling approach of gear losses as proposed in [PSO02], which has been further used to extend the linear model by inclusion of nonlinear gear losses. Finally, concluding remarks and future related work complete the paper.

2 Actuator physical description

This chapter provides the physical description of an electro-mechanical actuator, see Fig. 1 and Fig. 2, which has been used in a steer-by-wire control structure for force-feedback. In order to avoid rotating wiring a strain-gauge torque sensor is placed between the motor housing and a fixed console, as shown in the figure. Thus, in addition to the torque at the output shaft, a dynamical component resulting from housing rotation is measured, as well.

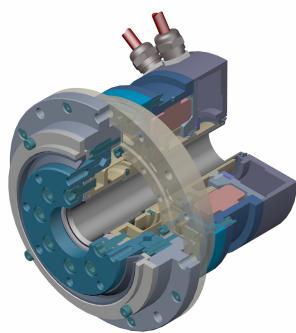


Figure 1: The force-feedback actuator

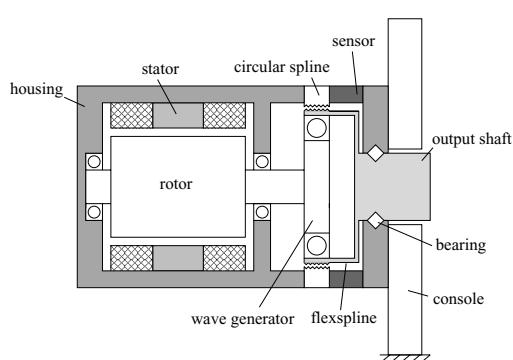


Figure 2: Force-feedback actuator components

Besides the torque sensor, the main component in the force feedback actuator is a Harmonic Drive series hollow-shaft gear. In Fig. 3 its main components, Wave Generator, Flexspline and Circular Spline are shown. The teeth on the nonrigid Flexspline and the rigid Circular Spline are in continuous engagement. Since the Flexspline has two teeth fewer than the Circular Spline, one revolution of the input causes relative motion between the Flexspline and the Circular Spline equal to two teeth. With the Circular Spline rotationally fixed, the Flexspline rotates in the opposite direction to the input at a reduction ratio equal to one-half the number of teeth on the Flexspline. Typical characteristics of a Harmonic-Drive gear are high positioning accuracy, virtually no backlash, periodic torque ripples and a high gear ratio. One of the main topics of this paper is modelling of friction losses of this gear using Modelica.

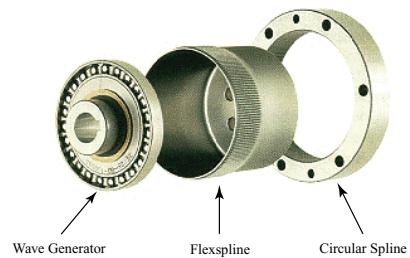


Figure 3: Harmonic Drive gear components

3 Parameter identification with Dymola/Modelica and MOPS

Modelica is an object-oriented language for modelling of large, complex and heterogeneous multi-physical systems involving mechanical, electrical and hydraulic subsystems. The engineer can build its model in a fraction-by-fraction manner, as he would build it in real-world, that is link components like motors, pumps and valves using their physical interfaces. Such a simulation framework is very convenient to use in an identification feedback-loop consisting of parameter optimization and simulation tasks, as shown in Fig. 4. Thereby, one can perform parameter identification of specific components or/and of specific physical conditions independently and integrate them easily in the next identification setup. The insight into physical system is important for decoupling of different physical conditions which primarily excite a known set of parameters. Note that using Modelica for physical sim-

ulation may be indispensable for complex systems, since a signal-flow simulation model may be essentially influenced by additional physical fractions.

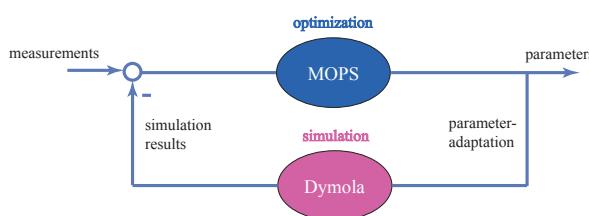


Figure 4: Identification loop

The parameter optimization in the fraction-by-fraction identification procedure is done using the optimization tool MOPS (Multi-Objective Parameter Synthesis), [JBL⁺02]. Basically, MOPS provides a multi-objective optimization environment for the design of systems with a large amount of parameters and criteria, but it may be used equally well for parameter estimation in identification problems. The multi-criteria optimization problem in MOPS is handled by reformulating it as a standard Nonlinear Programming Problem (NLP) with equality, inequality and bound constraints. MOPS uses several available gradient-free direct-search solvers, which are more robust compared to though more efficient gradient-based solvers. These include algorithms such as sequel quadratic programming (SQP), Quasi-Newton, pattern search, simplex method and genetic algorithms. To overcome the problem of local minima to some extent, solvers based on statistical methods or genetic algorithms can be alternatively used. An identification problem may be formulated as a multi-objective optimization problem, whereby measured data corresponding to different physical conditions or/and inputs define a set of optimization objectives. Different scalar or/and vector criteria may be defined, e.g least-square-error, error-vector, etc.

4 Linear model

4.1 Actuator Modelica Model

Fig. 5 represents a Modelica modelling setup of the electro-mechanical actuator. Since a Harmonic-Drive gear can be classified as a typical sun-carrier-ring planetary gear (Wave Generator corresponding to the sun, Circular Spline to the carrier and Flexspline to the ring), a planetary gear component from the Modelica

rotational mechanics library has been used for its modelling. The torque balance and angular equations of Harmonic-Drive are modelled as follows,

$$\begin{aligned}\tau_C &= (n - 1)\tau_W \\ \tau_F &= -n\tau_W \\ \varphi_W &= (1 - n)\varphi_C + n\varphi_F,\end{aligned}\quad (1)$$

with

τ_C : torque at the Circular Spline

τ_F : torque at the Flexspline

n : gear ratio

φ_C : Circular Spline angle

φ_F : Flexspline angle.

Note that in the linear model the losses of this component are neglected.

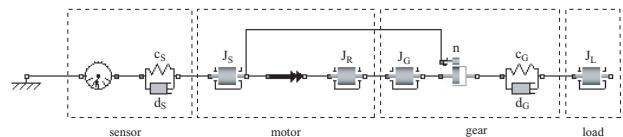


Figure 5: Linear physical model of the actuator

The following listing introduces the physical description of the setup parameters.

n : gear transmission ratio

J_L : gear output inertia

d_S : sensor damping

c_S : sensor stiffness

d_G : gear damping

c_G : gear stiffness

J_R : rotor inertia

J_S : stator (housing) inertia.

4.2 Parameter identification

Two different physical conditions are respectively discriminated for parameter identification of the linear and nonlinear actuator model. In the linear model the friction losses in Harmonic-Drive gear are neglected. In order to match the physical model as close as possible to such a linear one, the non-linear effects excited on Harmonic-Drive gear are minimized by fixing the output shaft.

With the output shaft keeping fixed, load inertia, J_L in Dymola/Modelica model in Fig. 5 has no dynamical effect. While several parameters, such as Harmonic-Drive gear ratio ($n = 50$), the *emf* motor constant ($K_m = 0.7 \text{ Nm/rad}$), torque sensor stiffness ($c_s = 130000 \text{ Nm/rad}$) and Flexspline stiffness ($c_G = 25500 \text{ Nm/rad}$) are given by the manufacturer, the rest

of parameters, i.e. sensor damping (d_S), motor housing inertia (J_S) and Flexspline damping (d_G) need to be identified. Their initial values for optimization are set to reasonable values estimated by some simple experiments,

$$\begin{aligned} d_S &= 0.6 \text{ Nm s/rad}, \\ J_S &= 0.003 \text{ kgm}^2, \\ d_G &= 50 \text{ Nm s/rad}. \end{aligned}$$

Thereby, as input data in Fig. 4 are used measurements corresponding to a set of current inputs (step, sinusoidal and PRBS) of different amplitudes and frequencies and torque response is measured by the sensor. After 18 successive iterations of the identification feedback-loop in Fig. 4, the parameter values listed below result,

$$\begin{aligned} d_S &= 2.66 \text{ Nm s/rad}, \\ J_S &= 0.003039 \text{ kgm}^2, \\ d_G &= 70.625 \text{ Nm s/rad}. \end{aligned}$$

The respective optimization history is shown in Fig. 6. Further in Fig. 7 several validation results for different input and measurement data are collected.

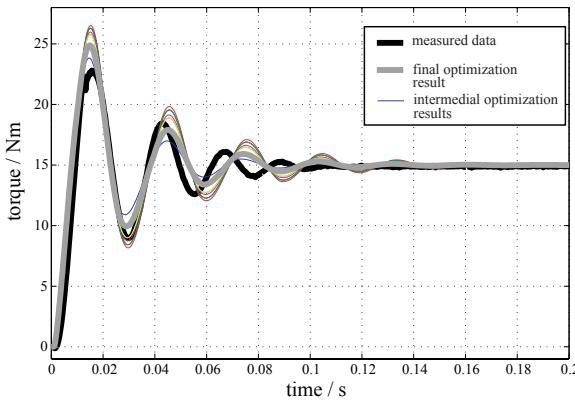


Figure 6: Optimization step response history of linear model

5 Nonlinear model

5.1 Gear losses

The nonlinear model of Harmonic-Drive Gear used in this paper is based on the friction modelling approach proposed in [PSO02]. Therefore, the gear model implemented in component `Lossy Planetary` of

the Modelica Power Train library includes a torque-dependent (due to mesh friction in the gear teeth contact) and speed-dependent friction (due to bearing friction). Similar to the standard Modelica friction model, the three modes *forward sliding*, *stuck* and *backward sliding* are available. The friction torque $\Delta\tau$ for the sliding modes is given by Table 1, whereby τ_W denotes the driving torque, τ_{bf} the bearing friction and η_{lmf} the mesh friction coefficient.

ω_W	τ_W	$\Delta\tau =$
> 0	≥ 0	$(1 - \eta_{lmf1}) \tau_W + \tau_{bf1} $ ($= \Delta\tau_{max1} \geq 0$)
> 0	< 0	$(1 - 1/\eta_{lmf2}) \tau_W + \tau_{bf2} $ ($= \Delta\tau_{max2} \geq 0$)
< 0	≥ 0	$(1 - 1/\eta_{lmf2}) \tau_W - \tau_{bf2} $ ($= \Delta\tau_{min1} \leq 0$)
< 0	< 0	$(1 - \eta_{lmf1}) \tau_W - \tau_{bf1} $ ($= \Delta\tau_{min2} \leq 0$)

Table 1: $\Delta\tau = \Delta\tau(\omega_W, \tau_W)$ in sliding mode

It can be shown, that the linear torque equations in (1) are extended by the friction component, $\Delta\tau$ as follows,

$$\begin{aligned} \tau_F &= -n(\tau_W - \Delta\tau) \\ \tau_C &= (n-1)\tau_W - n\Delta\tau. \end{aligned} \quad (2)$$

The typical relationship between τ_W and $\Delta\tau$ is illustrated in Fig. 8 for both the sliding and the stuck mode and in combination in Fig. 9.

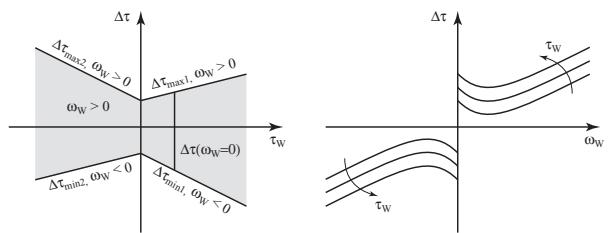


Figure 8: friction torque in sliding and stuck mode

The parameters to be provided are the stationary gear ratio n and table `lossTable` to define the gear losses, see Table 2.

Whenever η_{lmf1} , η_{lmf2} , τ_{bf1} or τ_{bf2} are needed, they are determined by interpolation in `lossTable`. The interface of this Modelica model is therefore defined as

```
parameter Real i = 1;
parameter Real lossTable[: ,5]
    = [0, 1, 1, 0, 0];
```

using the unit gear ratio and no losses as a default.

5.2 Parameter Identification

This section deals with identification of the `lossTable` in Table 1. For mesh friction, it is

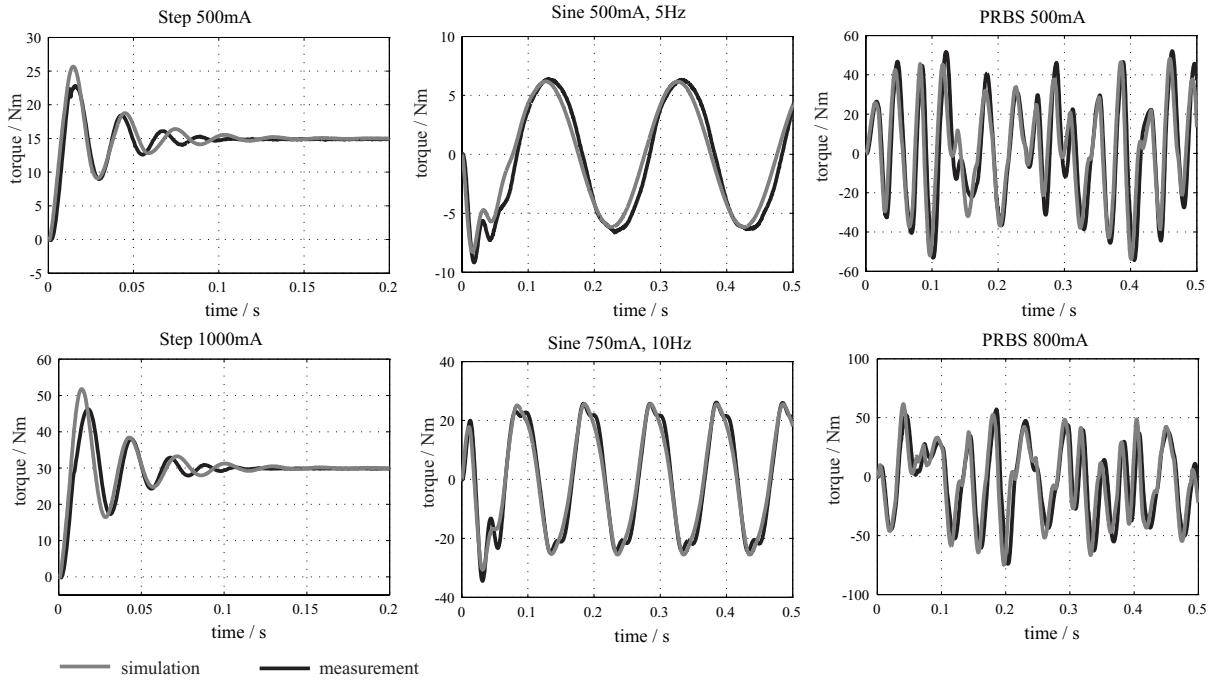


Figure 7: Validation results linear model

$ \omega_W $	η_{mf1}	η_{mf2}	$ \tau_{bf1} $	$ \tau_{bf2} $
\vdots	\vdots	\vdots	\vdots	\vdots

Table 2: Format of table `lossTable`

natural to assume no-loss (ideal gear) conditions as initial values. Besides, it is relatively difficult to set an experimental setup for its measurement, since additional drives have to be installed on the output shaft for covering the whole set of conditions as described in Table 2. While measurement of bearing friction is not essentially simpler, it may be roughly assumed that,

$$\tau_{bf1} \approx \tau_{bf2}.$$

However, assuming ideal conditions as initial ones may cause difficulties in optimization of bearing-friction, since the solvers are required to change the initial structure by including additional damping into the model. Fortunately, using the above assumption initial values are relatively easily estimated in a setup with free rotation of the output shaft (no external load) at different constant velocities. Given that torque sensor sits between the input and output bearing friction, it can see just the output bearing friction. Thus, assuming that the torque generated on the motor shaft balances the net (both input and output) bearing friction

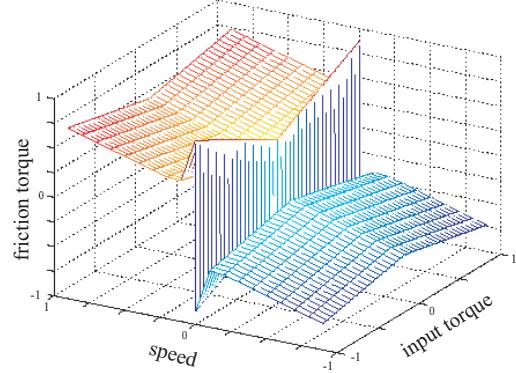


Figure 9: Friction model

(output load effects due to the Flexspline inertia are neglected), motor current can be used for its estimation. Fig. 10 presents the estimation results corresponding to rotation in both directions. From this curve the τ_{bf1} , i.e. τ_{bf2} are read as initial values for the optimization. Note that the above figure indicates clearly the appearance of the Stribeck effect when switching from stuck to sliding mode.

For completion of the `lossTable` the identification procedure is repeated for different constant velocities. Each identification step corresponds to a row in the `lossTable`.

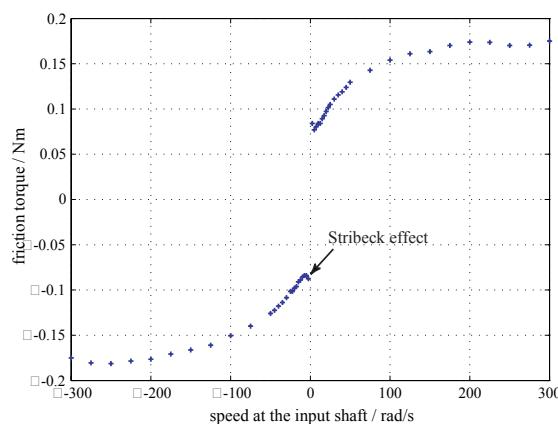


Figure 10: Bearing friction measurement

5.3 Model validation

For illustration purposes the row corresponding to the rotor speed of 10 rad/s will be discussed. Based on the previous discussion the initial values are chosen to be,

$$\begin{aligned}\eta_{mf1} &= 1, \\ \eta_{mf2} &= 1 \\ \tau_{bf1} &= 0.09 \\ \tau_{bf2} &= 0.09.\end{aligned}$$

After 103 optimization/simulation iterations in Fig. 4 these parameters converge to the values,

$$\begin{aligned}\eta_{mf1} &= 0.923, \\ \eta_{mf2} &= 0.864 \\ \tau_{bf1} &= 0.058 \\ \tau_{bf2} &= 0.058.\end{aligned}$$

For model validation the authors have set the setup shown in Fig. 12, whereby a defined torque at the output shaft has been applied by an excentric load. Different load conditions may be realized by varying the load radius. The Dymola/Modelica actuator model corre-

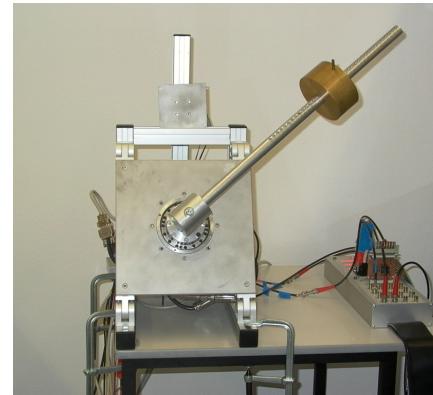


Figure 12: Excentric load experiment

$ \omega_W $	η_{mf1}	η_{mf2}	$ \tau_{bf1} $	$ \tau_{bf2} $
10	0.979	0.945	0.086781	0.086781
15	0.9625	0.92125	0.090313	0.088438
20	0.854	0.847	0.0565	0.049

Table 3: Format of table lossTable

sponding to the physical situation in Fig. 11 is augmented as shown in the above figure, by making use of the new Multi-body Modelica Library, [OEM03].

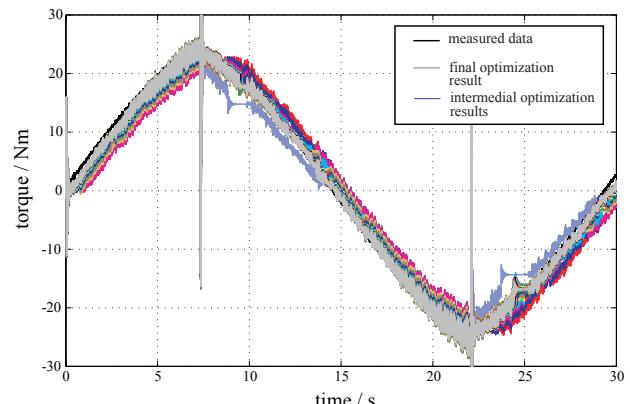


Figure 13: Optimization history of nonlinear model

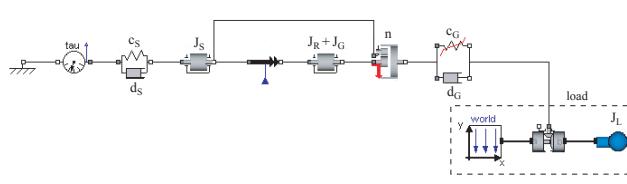


Figure 11: Modelica model with excentric load

The identification history of the loop in Fig. 4 for the row 10 rad/s assuming $\tau_{bf1} = \tau_{bf2}$ is shown in Fig. 13. In a next identification step the assumption $\tau_{bf1} = \tau_{bf2}$ is removed. Table 3 shows three rows of lossTable corresponding to the rotor speeds of 10 , 15 and 20 rad/s . Notice that $\tau_{bf1} \approx \tau_{bf2}$.

Finally, Fig. 14 collects the validation results for different input current signals.

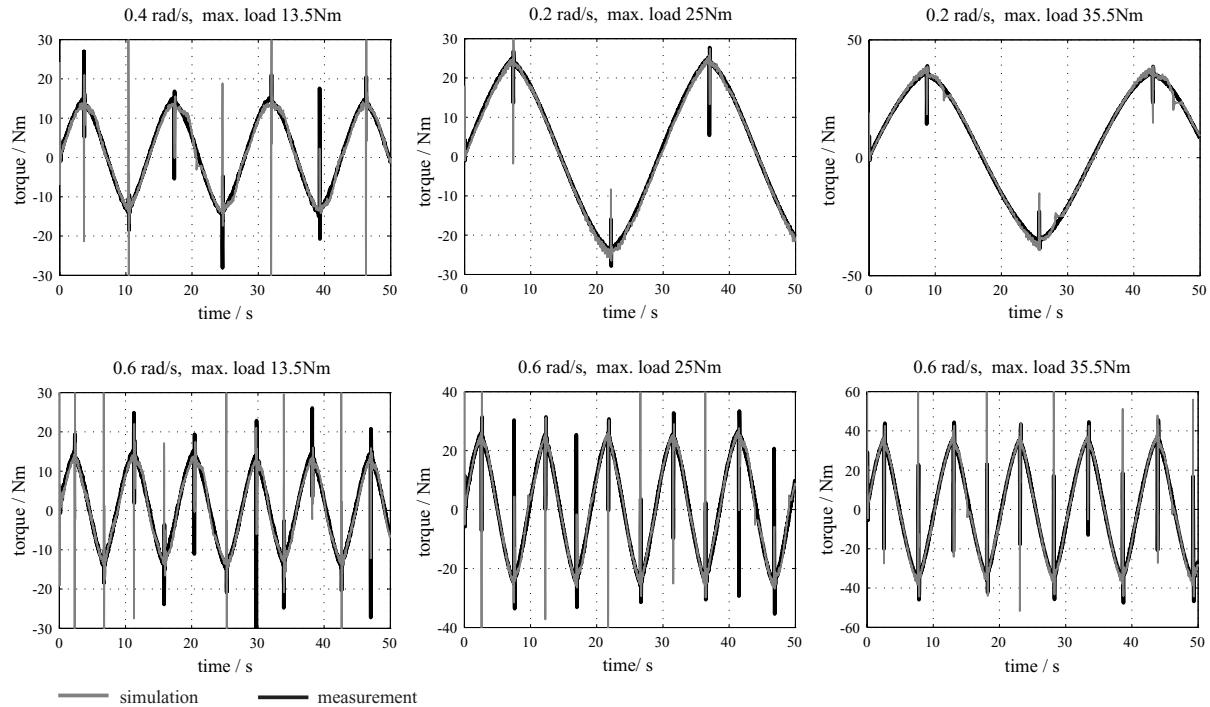


Figure 14: Validation results for nonlinear model

6 Conclusions

It is shown that iterative parameter optimization with MOPS and model validation using Dymola/Modelica is a powerful identification environment. This method is used for modelling of a force-feedback electro-mechanical actuator with Harmonic-Drive gear. A modelling approach for speed and torque dependent gear losses introduced in a prior work is validated. Future work might include identification of dynamical friction models. The procedure presented in this paper may be applied for dynamics identification of other gear technologies.

[PSO02] PELCHEN, Christoph ; SCHWEIGER, Christian ; OTTER, Martin: Modeling and Simulating the Efficiency of Gearboxes and of Planetary Gearboxes. In: *Proceedings of the 2nd International Modelica Conference*. Oberpfaffenhofen, March 2002, S. 257–266

References

- [JBL⁺02] JOOS, Hans-Dieter ; BALS, Johann ; LOOYE, Gertjan ; SCHNEPPER, Klaus ; VARGA, Andras: A multi-objective optimisation-based software environment for control systems design. In: *IEEE International Symposium on Computer Aided Control System Design Proceedings*. Glasgow, September 2002
- [OE00] OTTER, Martin ; ELMQVIST, Hilding: Modelica — Language, Libraries, Tools, and Conferences. In: *Simulation News Europe* (2000), December
- [OEM03] OTTER, Martin ; ELMQVIST, Hilding ; MATTSSON, Sven E.: The New Modelica MultiBody Library. In: *Proceedings of the 3rd International Modelica Conference*. Linköping, November 2003

VehProLib – Vehicle Propulsion Library Library development issues

Lars Eriksson

Vehicular Systems, Department of Electrical Engineering
Linköping University, SE-581 83 Linköping, Sweden

larer@isy.liu.se, <http://www.fs.isy.liu.se/~larer>

Abstract

A Modelica library called *Vehicle Propulsion Library* VehProLib is under development. Its structure and important design issues are described and the current status is shown. The vehicle propulsion library aims at providing functionality for studying and analyzing the performance of different powertrain configurations. The included components cover the range from zero dimensional in-cylinder models to longitudinal models for complete vehicles.

1 Introduction

The performance of vehicles and their powertrains are continuously being improved and computer based models and simulation tools are used routinely in investigations. Models and libraries that can be reused also provide leverage to the investigations. Vehicle powertrains are truly multi domain and Modelica is therefore a well suited modeling language for building a library upon.

Intended users are both engineers in the automotive industry and less experienced students. The aim is to provide the engineers with a basic structure that provide a platform for collaboration and exchange of component models. Students should be provided with a set of basic components that can be used to learn the functionality of powertrains and to investigate different structures.

The initial development of the package is focused on the engine components since these form the foundation for the torque production in vehicles. Important phenomena in the engine components such as in-cylinder heat transfer and combustion propagation where well established models are used in the combustion engine models.

2 Library Structure

The library is under development and the structure will change with the acquired knowledge and from feedback from the users. Currently, the library has the following structure, only the package names are written and the indentation show the hierarchy.

```

VehProLib
  Types
  Functions
  Interfaces
  Partial
  GasProp
  Engine
    Functions
    Partial
    Examples
  Chassis
  Driveline
    Components
  HEV
  DrivingCycles
  Examples
  Tests

```

These packages contain models for the different components as well as full example models. Some of the components that are included in the library will be covered in the upcoming sections.

3 Development Guidelines

One aim of the package is that it shall be possible to use it jointly with the powertrain library, which provides more comprehensive component models for powertrains. Therefore the interfaces will be designed to agree with those of the powertrain library, currently the interfaces are implemented using the Modelica rotational library and there is no control bus implemented.

Furthermore it is important that it is easy to exchange different component models e.g. to exchange a sim-

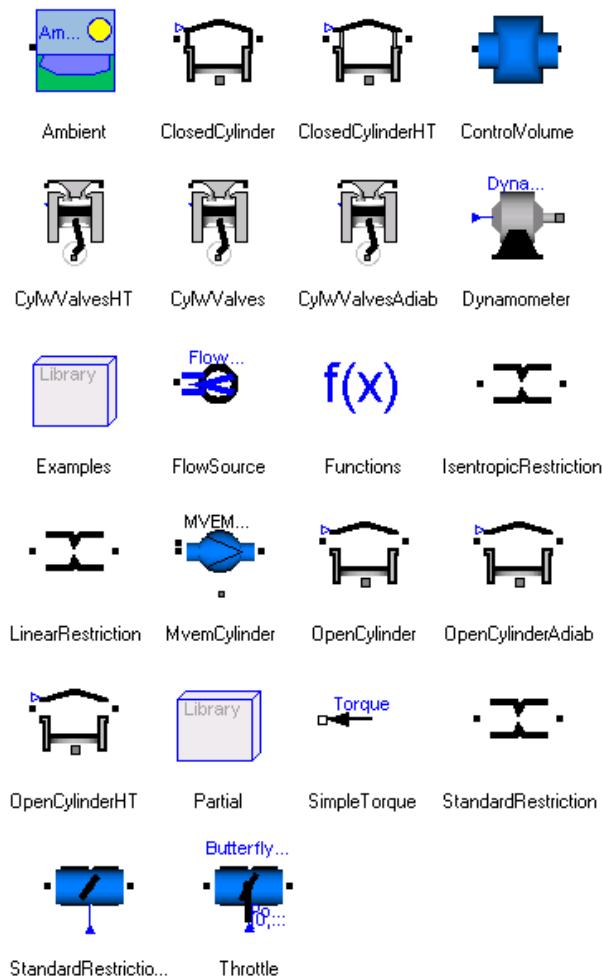


Figure 1: Components in the engine library that are building blocks in engines and that can be used to develop new engines.

ple model for an engine with one that is more complex when more detailed knowledge is needed. On the engine level it should also be possible to exchange the models for the working fluid in a simple manner and the fluid models are therefore separated from the engine components.

Finally the library should also facilitate the study of advanced driveline topologies like electric and hybrid vehicles.

4 Engine Components

Several components are included in the Engine package, see Figure 1. The library contains basic components for flow restrictions and control volumes.

- compressible and isentropic restrictions (fixed and variable area)

- incompressible restrictions (fixed area)
- control volumes
- exchangeable gas properties
- mean value engine models
- single zone, zero dimensional in cylinder models with and without heat transfer.

Many of the components in the model library are partial components that provide a basis for users to develop new components at a suitable level of refinement for their usage. The basic structure for the components are to use flow components in series with control volumes. An assumption in all these engine models is that the influence of the potential energy (due to gravity) on the gas flow is so small that it can be neglected.

4.1 Flow Connector

The simplest approach for flow connectors are used in the current implementation. There are two intensity variables pressure and temperature, and two flow variables enthalpy flow rate and mass flow rate.

```
connector FlowCut_i "Standard connector"
  package SI = Modelica.SIunits;
    SI.Pressure p(nominal=100000, start=100000)
      "Pressure sensed by the connector";
    SI.Temperature T(nominal=500, start=300)
      "Temperature sensed by the connector";
    flow SI.EnthalpyFlowRate H
      "Enthalpy flow through the connector";
    flow SI.MassFlowRate W
      "Mass flow through the connector";
  end FlowCut_i;
```

Inertia effects that rely on the momentum balance are neglected when using this connector.

4.2 Control Volume Design Issue

Control volumes are described using mass- and energy-balance equations (positive directions are inward)

$$\frac{dm}{dt} = \sum_i W_i$$

$$\frac{dU}{dt} = \frac{d}{dt} Heat + \frac{d}{dt} Work + \sum_i H_i$$

Using these formulation directly in the code results in that mass and energy will be selected as state variables. This in turn results in trouble when specifying initial conditions since an engineer working within this area can specify initial values on temperature and pressure.

The energy equation is therefore rewritten by assuming an ideal gas, which gives

$$mc_v \frac{dT}{dt} = \frac{d}{dt} Heat + \frac{d}{dt} Work + \sum_i (H_i - m_i u_i)$$

The result when implementing this equation is that the temperature is selected as state variable. A differential equation for the pressure could also be determined by differentiating the ideal gas law, but this is a special case for the pressure, therefore the mass balance is selected as the second balance equation. The initial pressure, which is a parameter in the control volume, is then used together with the temperature and the ideal gas law to give an initial value for the mass. It is important to note that the equation above and the initial conditions for the mass will be revised when non ideal gases will be included in the library. It is worth to note that the base class for the gas model, shown in Appendix A, does not contain any assumptions about ideal gases. So it is general.

4.3 In-Cylinder Models

The in-cylinder models considered here are zero dimensional and have a single zone, see e.g. [2]. In-cylinder models are control volumes and the discussion about initial conditions also apply here. Several different versions of the in-cylinder models are implemented there are those that are adiabatic and other that have heat transfer and these are implemented for comparing the effects, these are shown in Figure 1.

There are two heat transfer functions implemented, one comes from Woschni [6] and the other comes from Hohenberg [3]. To describe the combustion two different choices are available one is the standard Sigmoid function and the other is the well known Vibe-function [5].

Currently the equations for mechanics and fluid are collected in only one component but these will be separated in the future, so that the cylinder is modeled in a truly multi-domain manner.

4.3.1 MVEM

There are also engine components implemented that fall in the category of Mean Value Engine Models (MVEM). These have lower complexity and are faster to simulate compared to the in-cylinder pressure models. Since they are less complex and faster they are used for studying control design and for complete vehicle simulation. Both the in-cylinder models and the MVEMs have inherited the same interfaces so they are

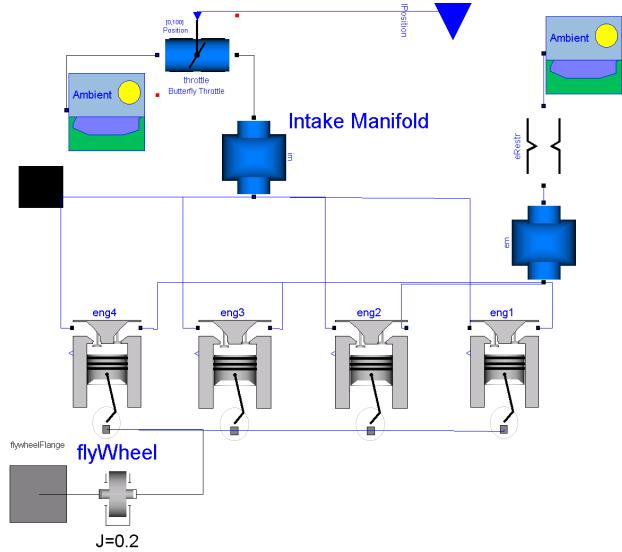


Figure 2: The components in the four cylinder engine used in the example. This model shows the components that are included in the MultiCylEngine1 in Figure 3.

easily exchangeable with the models. A common scenario is to set up a simulation problem using a MVEM to see that all components work together as expected. Then when more detailed knowledge is required and the more advanced engine model is inserted instead of the MVEM.

4.4 A Four Cylinder Engine

A four cylinder engine on a dynamometer, that is included as an example in the package, is used as an demonstration to show the simulation results from one of the models. Figures 2 shows the components in the four cylinder engine, where the throttle input goes to the butterfly throttle. Figure 3 shows how the four cylinder engine is arranged with the dynamometer tank and the step change in the accelerator pedal at t=0.2 s.

The resulting cylinder pressure traces are shown in Figures 4 and 5. Figure 4 shows the cylinder pressures for the four cylinders and the result from the step change in throttle angle is clearly visible. Note that there is a delay from the step to when the maximum cylinder pressure is affected by the change, this is due to the delay caused by the intake stroke and compression stroke.

Figure 5 shows the simulation result presented in a pV-diagram. Two groups of loops can be seen, the lowest comes from the period before the step change and the

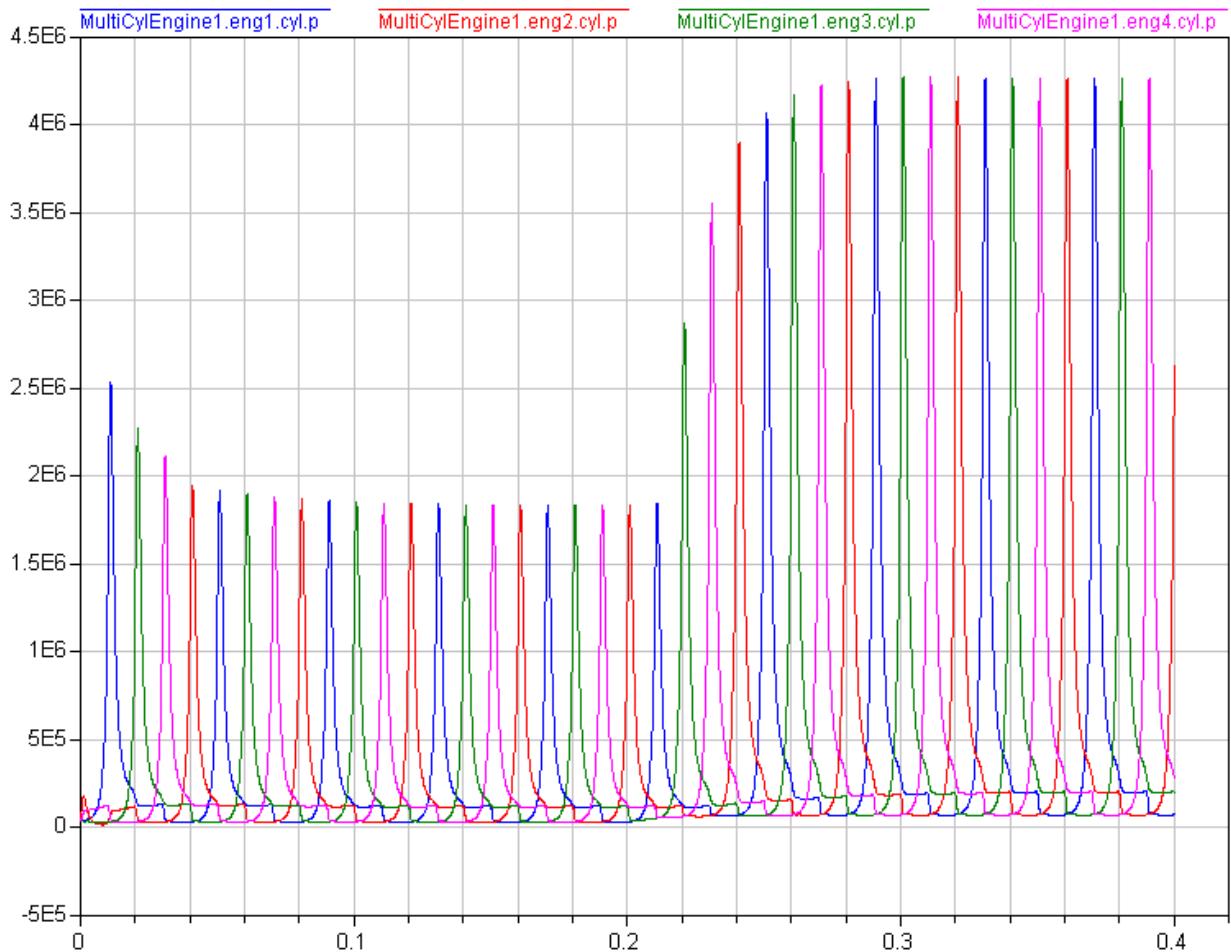


Figure 4: Figure showing the cylinder pressures from a four cylinder engine included in the examples of the library. At $t = 0.2\text{s}$ a step change is made in the throttle angle, and the response is clearly seen in the model.

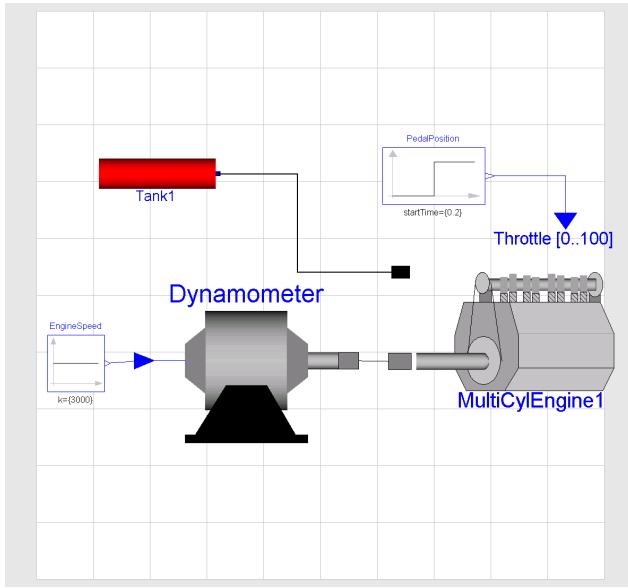


Figure 3: The multi-cylinder engine on the dyno. The dyno maintains the engine at a constant speed. A step input is applied to the throttle input at $t = 2$ s.

higher comes from the period after the step change. This model family was proposed by Gatowski et.al. [1] and has been widely studied and is well known that it is can give a good description of measured in-cylinder pressures during an engine cycle.

5 A Complete Vehicle

To show some of the components that are available in the Driveline and Vehicle packages an example is used. Figure 6 shows an example of a model for a vehicle in longitudinal motion, with engine, driveline, and vehicle components. This example shows some of the components that are included in the library. For example there are models for vehicle body with air drag and rolling resistance, tires. The basis for the driveline modeling was presented in [4]. The driveline consists of clutch, five step gearbox, final drive, flexible drive shafts, brakes and wheels. Also included in the library is a driver which uses a driving cycle, implemented as a speed and gear table. Finally the engine is selected from the mean value engine, since it is less complex and gives much shorter simulation time compared to the multi cylinder engine model that was shown previously.

The results from a simulation running the longitudinal vehicle model with the driver following the New European Driving Cycle is shown in Figure 7. The top plot shows the vehicle speed as a function of time. Both

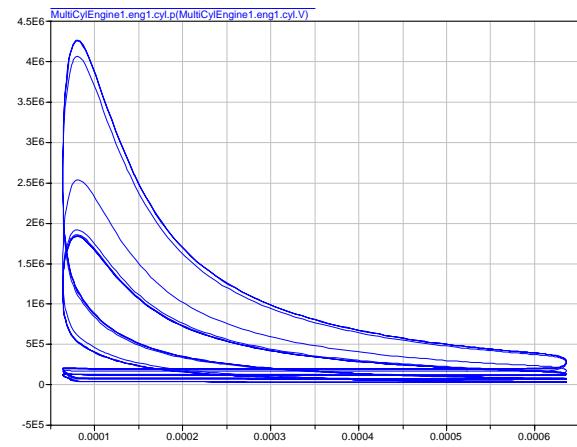


Figure 5: Figure showing the pv diagram for cylinder one shown in Figure 4 The step change in throttle at $t = 0.2$ s gives a change between the lower pressure traces to the higher in the pv-diagram.

the desired and actual vehicle speeds are shown and it is seen that the driver is well tuned and manages to follow the desired speed well. The middle plot shows the engine speed and the bottom plot shows the gear number and clutch position.

6 Future Work

The library is continuously being developed and some of the areas with highest priority are:

- Decoupling of the mechanics and thermodynamics in the engine.
- A more general gas model and an extended connector that includes multi-component flow. Follow the work by the Modelica standardization group on thermofluid library and decide if the full library should be implemented.
- Implement and incorporate more engine components, foremost turbocharger models.
- Implement and incorporate more driveline and vehicle components, for example hybrid components.
- Continuously build up test models for the components that are added to the library.

The model library development is an ongoing task and everybody that are interested in contributing to the library are encouraged to contact the author by e-mail.

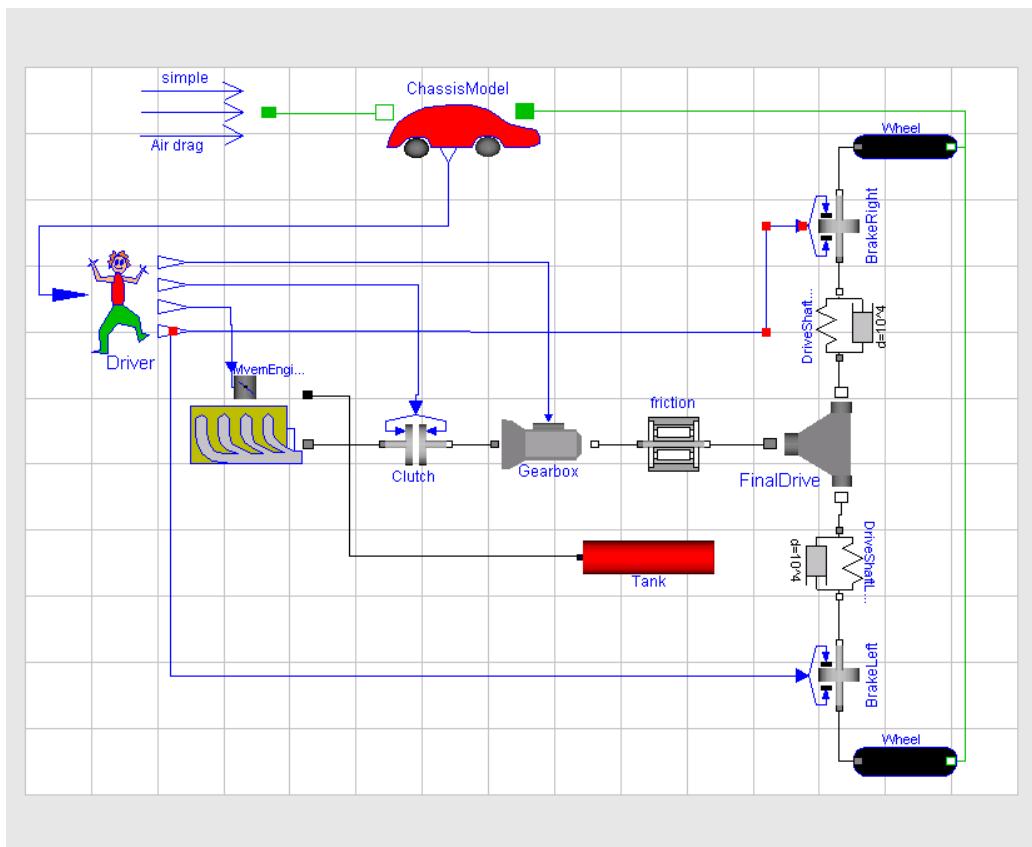


Figure 6: A complete vehicle modeled using components from VehProLib and standard Modelica components. The example shows some of the components included in the library. Here the mean value engine model is used instead of the multi cylinder model since the vehicle follows a longer driving cycle.

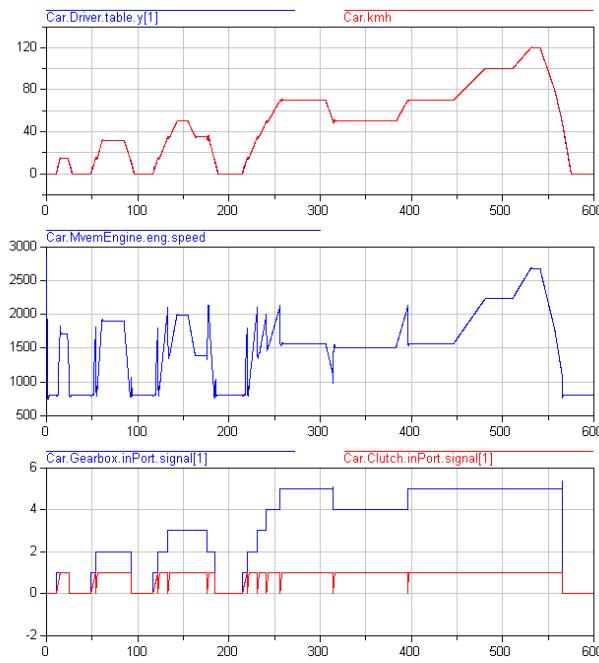


Figure 7: Simulation results from the model shown in Figure 6. The plots show: top—vehicle speed, middle—Engine speed, and bottom—gear and clutch position. The top plot shows both desired and actual vehicle speed.

7 Summary

A library VehProLib for vehicle propulsion modeling is being developed. Design issues related to the engine components have been addressed. The components in the Vehicle and Driveline packages have been illustrated using a model for a complete vehicle in longitudinal motion.

Acknowledgments

The library is currently being extended with the help of the MSc students Otto Montell and Johanna Wallin. Some icons for the models have been provided by Michael Tiller at Ford and he is greatly acknowledged.

References

- [1] J. A. Gatowski, E. N. Balles, K. M. Chun, F. E. Nelson, J. A. Ekchian, and J. B. Heywood. Heat release analysis of engine pressure data. *SAE Technical Paper 841359*, 1984.
- [2] J. B. Heywood. *Internal Combustion Engine Fundamentals*. McGraw-Hill series in mechanical engineering. McGraw-Hill, 1988.

- [3] Günter F. Hohenberg. Advanced approaches for heat transfer calculations. *SAE Technical Paper 790825*, 1979.
- [4] Per Nobrant. Driveline modelling using mathmod- elica. Master's thesis, Linköping University, SE-581 83 Linköping, 2001.
- [5] I.I. Vibe. *Brennverlauf und Kreisprocess von Verbrennungsmotoren*. VEB Verlag Technik Berlin, 1970. German translation of the russian original.
- [6] G. Woschni. A universally applicable equation for the instantaneous heat transfer coefficient in the internal combustion engine. *SAE Technical Paper 670931*, 1967.

A Gas property base

The gas property base, shown below, defines what functionality the gas model must have. This partial model is then extended by the gas models in the library.

```
partial class GasPropBase
  "Base class for gas properties"
  package SI = Modelica.SIunits;
  SI.Pressure p "Pressure";
  SI.SpecificVolume v "Specific volume";
  SI.Temperature T "Temperature";
  SI.Density rho "Gas density";
  SI.SpecificEnthalpy h "Mass specific enthalpy";
  SI.SpecificEnergy u "Mass specific internal energy";
  Real R(final unit="J/(kg.K)") "Gas constant";
  SI.MolarMass M "Molar mass";
  SI.SpecificHeatCapacityAtConstantPressure c_p
    "Specific heat capacity at constant pressure";
  SI.SpecificHeatCapacityAtConstantVolume c_v
    "Specific heat capacity at constant volume";
  SI.RatioOfSpecificHeatCapacities gamma
    "Ratio of specific heats";
equation
  gamma = c_p/c_v;
  h = u + p*v;
  rho*v = 1;
end GasPropBase;
```

Session 7B

Thermodynamic Systems – II

Economical Analysis of Complex Heating and Cooling Systems with the Simulation Tool HKSim

Dipl.-Ing. St. Wischhusen₁* Dr.-Ing. B. Lüdemann₂† Prof. Dr.-Ing. G. Schmitz₁

₁ Technical University Hamburg–Harburg, Department of Technical Thermodynamics,
Denickestr. 17, D–21073 Hamburg.

₂ Imtech Deutschland GmbH & Co. KG, Zentrale Ingenieurtechnik,
Tilsiter Str. 162, D–22047 Hamburg.

Abstract

Dynamic simulations of energy systems are essential when it comes to transient analysis and design of complex plants. Besides the choice of efficient subcomponents, like boilers, pumps or chillers, the control strategies have a large impact on the running costs of a cooling, heating or combined heating and cooling plant. This paper describes an applied simulation tool for heating and cooling systems. The economical benefits are discussed by means of a typical application: the simulation and optimisation of a complex industrial energy system.

1 Introduction

In cooperation with Imtech Deutschland GmbH & Co. KG (formerly known as Rudolf Otto Meyer GmbH & Co. KG and Rheinelektra Technik) a research project was conducted. The aim of the project was to develop a simulation tool, called HKSim [1, 2], for heating (Fig. 1) and cooling systems in building applications. This tool enables configuration studies and dynamic system simulations with time scales from a few seconds up to one year. For this purpose the simulation environment of Dymola [3], containing the object-oriented modelling language Modelica, is used to model complex heterogeneous systems. The graphical user interface, including the integration of Dymola and a data base for project management, was created by the department “Zentrale Ingenieurtechnik” of Imtech Deutschland while the model libraries [4] were developed at the Department of Technical Thermodynamics at the Technical University Hamburg–Harburg.

*wischhusen@tuhh.de, tel.: +49–40–42878–3267

†bruno.luedemann@imtech.de, tel.: +49–40–6949–2546

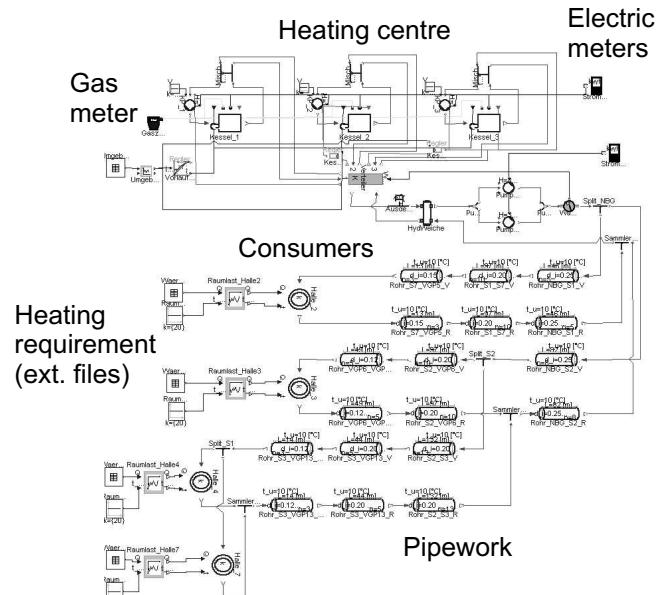


Figure 1: System schematic of a heating centre with distributed consumers and an earth–laid pipework

The component models are focused on the simulation of entire years. Therefore, the model equations have to be formulated as efficient as possible. The model design philosophy, which results from this important requirement, will be discussed in detail with respect to typical system components. The components can be mostly parameterised using manufacturer information or values resulting from own measurements. The handling of the models is primarily focused on users who want to use the models as they are provided or with alternative parameter settings. Expert–users are able to exchange model equations (e.g. models for

medium properties, pressure losses and heat transfer) by means of replaceable models and develop their own components from the existing base classes. The boundary conditions of the system simulation can be supplied by data files from a building simulation or even measurement data.

Due to the separation of the building from the system simulation an efficient calculation for performance studies is realised when simulating complex plants.

The simulation tool HKSim was used successfully in several projects for the economical analysis of energy systems. In this article a typical project will be described, beginning with the selection of component models, followed by the consideration of individual control elements and determination of necessary boundary conditions. The last item usually consists of local weather data and calculated or measured heating and / or cooling requirement.

2 Current Library Content

So far, the most important components of cooling and heating systems have been supplied by the model libraries. All components are compatible by using identically defined hydraulic interfaces. Some elements which have been modelled and integrated into the libraries are [4]:

- normal, low-temperature and condensing boilers,
- cogeneration plants,
- consumers for heating, cooling and domestic water,
- pipes and storage tanks,
- splits and joints, mixing valves,
- controlled and uncontrolled pumps,
- heat exchangers [1],
- mechanical driven chillers,
- absorption chillers,
- cooling towers and dry coolers,
- special controllers (beyond: utilisation of Modelica's standard libraries [5]),
- electric and gas meter, oil supply.

Nevertheless, the development of new parts is continuing. In future, a model for different types of fuel cells will also be offered.

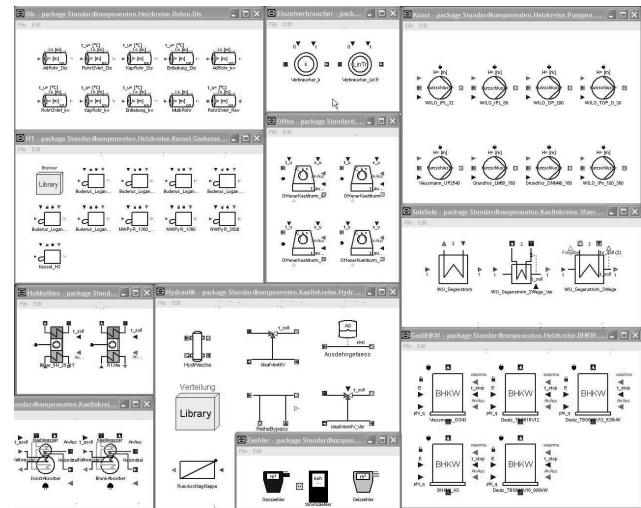


Figure 2: Screenshot of selected packages of typical heating and cooling components

In practice, it is important to have models with different levels of control and efficiency descriptions. Usually, a new project starts with the modelling of a system base layout. This first approach takes a lot of time because a correct understanding of the system's thermal and hydraulic behaviour is required. Generally, every single component which is part of the whole system should be modelled as accurate as possible. Unfortunately, this requirement contradicts most component manufacturers' information policies. Additionally, information of existing plants is often incomplete or not available. For this reason, models with a different depth of physical description are supplied by the libraries. For example, one can start with a boiler model which has a constant efficiency with the opportunity to replace it with a refined model later on.

For the user it is important, that the models use parameters, which are easily available or which can be determined without model knowledge.

3 Applied Thermal and Hydraulic Model Concept

The models are designed to enable a quick synthesis of plant models. This requirement is already considered in the modelling process in such a way that emphasis is placed on the calculation of the thermal behaviour. The hydraulic behaviour of a plant is not neglected but

simplified in that direction, that the mass flow can be directly influenced at split and joint elements. The display of a transient pressure is not possible in favour of a fast and stable simulation. In combination with the concept of a load dependent simulation, designed to satisfy thermal loads, the calculation of the mass flow rate \dot{m} is implemented in the model of the thermal consumers. Applying the first law of thermodynamics, the heat (or cooling) requirement \dot{Q}_{build} of the building (or consumer, resp.) equals:

$$\dot{Q}_{build} = \dot{m} \cdot c \cdot (\vartheta_f - \vartheta_r) . \quad (1)$$

If the feed temperature ϑ_f of the liquid and its heat capacity c is given two unknowns remain in this algebraic equation. By a second equation the dependency of the mass flow and the return temperature ϑ_r can be modelled applying heat transfer laws or measurement data. A simple but efficient approach is a proportional gain k of that mass flow rate, which is theoretically necessary to deliver the needed heat under the assumption of a perfect heat transfer ($\vartheta_r = \vartheta_{build}$)

$$\dot{m} = k \cdot \frac{\dot{Q}_{build}}{c(\vartheta_f - \vartheta_{build})} . \quad (2)$$

The constant gain factor k may vary between 1 ... 3. Another problem is that the mass flow rate should never be higher than the rated capacity of the installed pumps. In order to take this important limitation into account the rated mass flux of the pump \dot{m}_{max} is added to the hydraulic interfaces (see Fig. 3). Furthermore, this value is divided at splitting elements with regard to the actual load $q = \dot{Q}_{build}$ of parallel consumers.

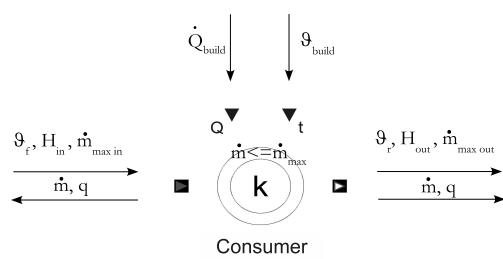


Figure 3: Input and output signals of the consumer model

In addition to this, the pumps head H must overcome the pressure losses of the plant. This is checked during an initial calculation assuming worst case conditions. The pressure check functionality is implemented in the model of an expansion vessel, which is also used as a sink for the algebraic signals (e.g. mass flow) in closed loops.

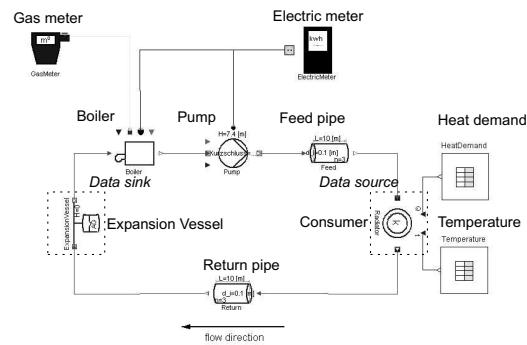


Figure 4: Simple heating plant

The base configuration for a simple heating plant is shown in Fig. 4. A minimum composition for a determined equation system must at least consist of the following three components:

- a consumer model for the calculation of a mass flow signal,
- a pump model for limiting the maximum mass flow rate and
- an expansion vessel which checks for a sufficient pump head (under worst case conditions) and avoids algebraic signal loops (e.g., mass flow rate).

Since the hydraulic simplifications lead to a heterogeneous data flow the user has to follow a few rules during the plant model generation. Those rules are visually supported by coloured interfaces which identify sources (triangle with black background), sinks (grey background) and neutral components (no background). Following the rules even complex plant models can be composed by the user without causing over- or underdetermined system models. Furthermore, some models must allow a flexible data flow due to numerical reasons. For example, the pump model can be switched into a mass flow source, when it is used in independent circuits without consumers. This is a common situation in complex applications where hydraulic bypasses are needed to realise a fail safe control strategy. By introducing a structural parameter it is possible to switch the model equations on demand. With regard to this feature on one hand and the connection rules on the other hand it would

be very convenient for the model developer and the model user to have graphical annotations which can be controlled by structural parameters. Additionally, an update of graphical icons, when replaceable objects like interfaces are exchanged, would be very helpful.

To verify the used hydraulic concept, it was validated with measurement data from a complex cooling system displayed in Fig. 5. The chillers are shown on the left hand side of the picture including one conventional water chiller and two heat exchangers which are part of a chiller / ice storage circuit. The total capacity is 1,600 kW at a system temperature of 7°C/ 15°C. The cold water pump of chiller #3 operates with a constant speed in contrast to all other pumps, which are controlled. Therefore, a hydraulic bypass between the feed and return duct is necessary (see Fig. 5).

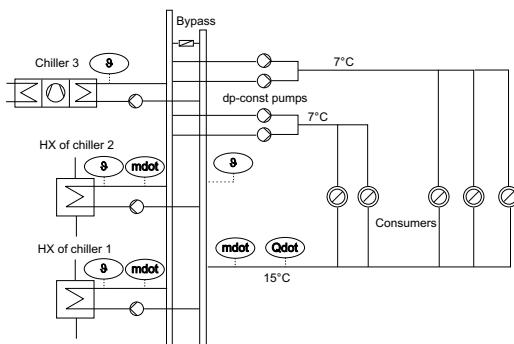


Figure 5: Schematic of the original plant and location of measuring points

As an input to the simulation the output temperatures of the chiller and heat exchangers were provided as well as the mass flow rates of the controlled pumps. Furthermore, the cold water mass flow through the buildings and the cooling requirement is known. Since the returning water's temperature is an important system variable it can be compared with the measurement to determine the quality of the consumer model description.

The corresponding system model to the description above is shown in Fig. 6. In this model the cold water temperature and flow is already merged to a single flux in the source model on the left hand side. A potential overshoot of cold water can be passed to the return side through the bypass. Moreover, the four feed pumps and five buildings are modelled as a single feed model with the same capacity since a local

deviation cannot be resolved by the measurement data. The consumer model refers to the equations 1 and 2. The model can be adjusted rather easily by two parameters: the constant gain, which was set to $k = \frac{\vartheta_r - \vartheta_f}{\vartheta_{build} - \vartheta_f} = 1.75$ and the building reference temperature $\vartheta_{build} = 21^\circ\text{C}$. The simulation was carried out with measurement data of one week during November with a peak load of approximately 530 kW and base load of 300 kW at the weekend (16th and 17th of Nov.).

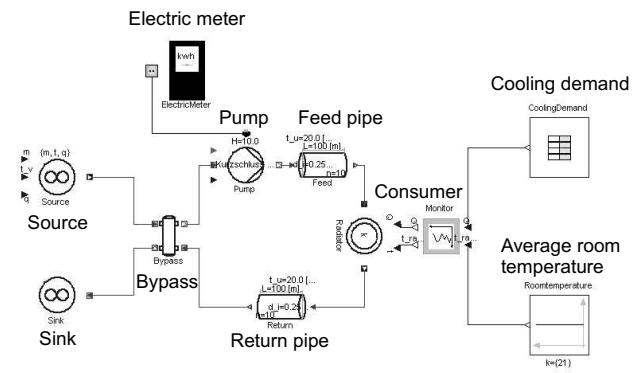


Figure 6: Modelica model of the cooling plant

Comparing the temperature of the returning water after the bypass in Fig. 7, a very good agreement with the measured values can be found. Some very few exceptions are due to the lack of information about the exact switch off times of chiller #3. Here only the output temperature was available. Instead, it was assumed that a cold water temperature of more than 7°C indicates a turned off chiller. Knowing that the water temperature during operation usually varies between 6.0 and 6.8°C and the outlet temperature due to a good insulation increases from 6°C to 20°C in more than 2 days, the temperature deviation shown in Fig. 7 can be explained.

The validation of the mass flow rate shown in Fig. 8 reveals that this value depends strongly on the cooling load. Obviously, there are periods where the agreement of measurement and simulation is very good but on some days there is a higher deviation. In a sensitivity analysis it was found that the building reference temperature of $\vartheta_{build} = 21^\circ\text{C}$ may have a large influence on the mass flow rate. Especially, Nov.

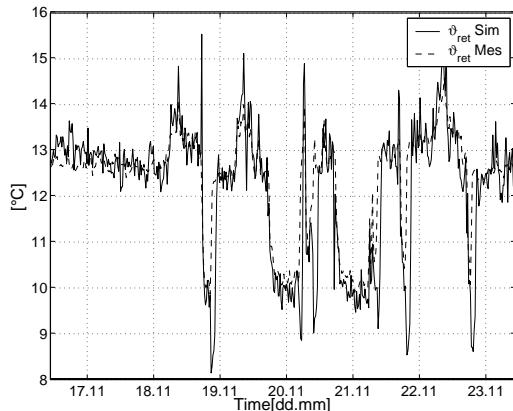


Figure 7: Comparison of the return temperature

19th and 20th were colder days ($<0^{\circ}\text{C}$) in contrast to the 22nd which had a higher ambient temperature (4°C). Hence, it can be suggested that the unknown average room temperature was not constant during the measurement. Changing the temperature by 1 or 2°C gives the right mass flow rate.

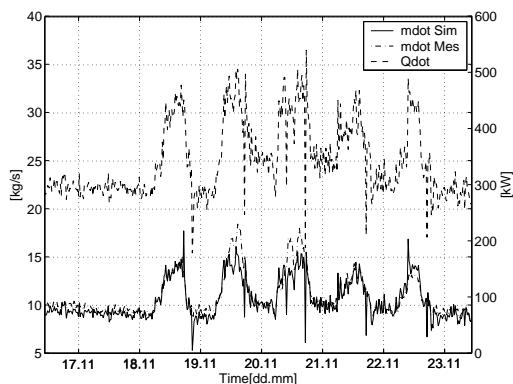


Figure 8: Cooling requirement and comparison of the water mass flow rate

It should be pointed out that the consumer model presented in this paper though it is based on a very simple approach can predict the mass flow and the return temperature with a sufficient accuracy. In addition to this, it enables a very fast model generation and simulation with computation times of a few seconds for a whole year. The model is also equipped with an overload routine which enables stable simulations when the heating and cooling capacity is too small for peak loads. In such an event the lack of energy will be compensated when the demand decreases again. A flag variable indicates that the plant's capacity is not sufficient.

4 Simulation of a Complex Industrial Energy Supply System

Within the development of an innovative energy supply system for a production facility the simulation tool HKSim was used to analyse the efficiency of the design concept and the running costs. After the simulation of the original layout optimisation measures were developed to increase the economical and technical efficiency.

The plant produces heat for the production lines, the heating system and the domestic hot water supply. Cold water is needed for industrial cooling processes and for air conditioning. The total cooling requirement reaches a maximum level of 2.6 MW in summer. The main idea behind the given plant schematic in Fig. 9 is to save primary energy by reusing as much waste heat as possible. Therefore, the heat needed for the periodic production processes is recovered and the superheat of the refrigerant after the compressor and the superheat of the compressed air is also transferred to the heating system. To ensure a minimum feed temperature a steam heat exchanger is implemented as a backup heat source. For cooling purposes, the continuous fresh water supply for steam production and domestic use is treated as a heat sink.

Since the continuous operation of the production lines has the highest priority it has to be ensured that the production lines are supplied with a sufficient amount of cooling water independent of the actual heat demand of the associated heating system. Hence, cooling towers with a large capacity were installed as a backup. Two of the three existing cooling towers can be switched between cooling of production lines, air compressors and chillers or free cooling of returning cold water to increase the average utilisation. The water of the cooling system is stored in two parallel tanks with a capacity of 700 m^3 .

The following questions and tasks were identified and should be clarified by means of the system simulation:

1. How much heat can be recovered to decrease the additional heat input for the facilities heating? A coverage of 70% by waste heat would allow a cheaper building insulation with regard to German regulations.
2. Which inexpensive optimisation measures could help to realise a further reduction of running costs?

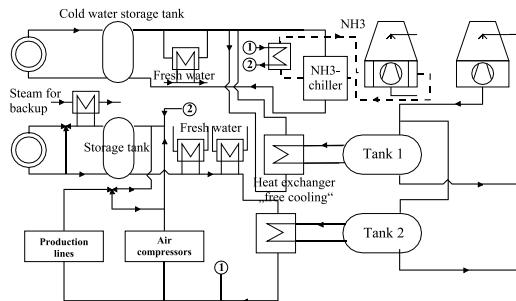


Figure 9: Simplified schematic of the original plant concept

Starting with the modelling of the original schematic displayed in Fig. 9 the plant model is divided into three supermodels (modules): the “heating module” (shown in Fig. 10), the “cooling module” integrating the cooling towers and the “cold water module”. The model of the heating and the production heat recovery system consists of eight different consumers (1), a boiler model representing the backup steam heat exchanger (2), four heat sources standing for the production heat recovery (8) and a number of hydraulic interfaces to the two other modules (4, 6, 7). The modules are connected in a supermodel. The reason for dividing the plant into subsystems is that the schematic is much clearer and the graphical update of Dymola works quicker, too.

After the model generation all necessary boundary conditions have to be determined. This data set includes the ambient temperature and humidity which is provided by a test reference year of the corresponding region in Germany. This fundamental input is also used in combination with the known rated loads to derive the thermal demand profiles for the air conditioning and heating with a simple linear approach. By means of measurements carried out on the existing production processes the possible heat recovery was determined and implemented using table interpolation models. The production processes can be described by a characteristic, transient heat output (Fig. 11). As a result from that fact, the outlet temperature of the heat exchanger’s cold side varies between 45 and 95°C during a period of a few minutes.

Based on a simulation of the original plant layout, potential modifications for an improved performance are determined, regarding the hydraulic circuit and governing control of the plant and its components:

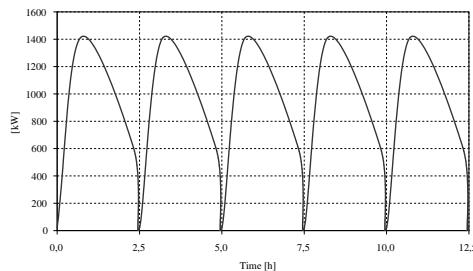
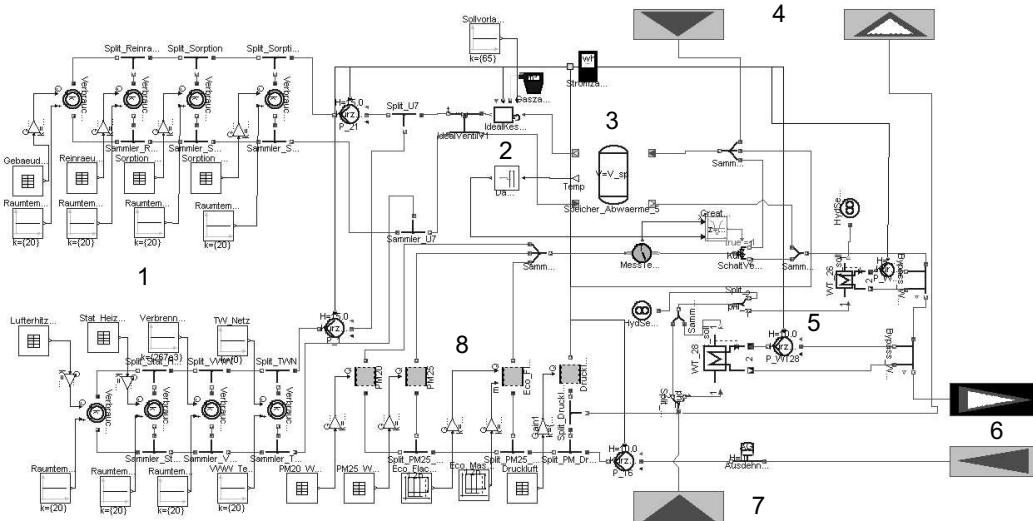


Figure 11: Periodic heat output of one production process

- **Modification 1:** All heat exchangers connected to the return side of the cold water storage tank are connected to the return side of the cold water consumers, whereby the heat exchanger’s input temperature is increased since the mass flow of the chiller pumps is always higher than the mass flow of the main pumps.
- **Modification 2:** A fixed periodical day / night switch of the cooling towers is replaced by a temperature dependent control to account for a changing level of primary cooling demand of the production lines. This measure shall increase the free cooling capacity, when more heat is transferred to the heating system.
- **Modification 3:** Cooling water with a temperature level below the desired value of the feed temperature is hydraulically connected to the cooling heat exchangers.
- **Modification 4:** The set point of the heating feed temperature is dependent on the actual heat demand and can be decreased if high temperatures are not necessary.

The simulations of the actual and the modified layout are carried out with identical boundary conditions and except from the modified parts the models are identical. The reference system for the economical analysis is a non-coupled heating and cooling system without heat recovery. All modifications are investigated separately as well as combined, since their impact may show a compensation effect in the coupled system. The investigation is focused on the possible savings of natural gas, which result from the demand of heating and domestic hot water with variable heat recovery.

**Legend:**

- | | |
|--|---|
| 1 Heat consumers | 6 Hyd. interface to module "cooling water" |
| 2 Backup steam heat exchanger (boiler model) | 7 Hyd. interface for service water from module "cold water" |
| 3 Hot water storage tank | 8 Heat recovery from production processes and air compressors |
| 4 Hyd. interface to module "cold water" | |
| 5 Service water heat exchangers | |

Figure 10: Modelica model of the heating module

Fig. 12 reveals the development of the primary energy consumption for the conventional system (the steam exchanger covers the total load), the original layout and the optimised system. Since the total heat demand is 9,546 MWh and the original concept is only able to cover it with not more than 38% waste heat the first requirement is not satisfied with this approach. The situation changes when the plant layout is modified in the way described above. Now, the heat recovery contributes more than 80% to the heat load which is sufficient enough to reduce the compulsory thermal insulation of the associated buildings. The total savings of gas needed for backup heating by the steam heat exchanger sum up to 148,000 EUR in one year regarding the difference between the original and final layout. This corresponds to 4,000 MWh (-68%) less heat supply at a heat price of 37 EUR/MWh. This moderate price results from experiences of the plant owner and also considers costs for maintenance and amortisation of the devices.

Comparing the remaining cooling load of the chillers before and after the optimisation one can determine an increased coverage of the cooling demand (15,445 kWh) by the cooling towers from 3.1% to 15.8%

compared to a separated conventional heating and cooling plant. This result can be explained by the enhanced free cooling, especially during winter, spring and autumn (Fig. 13 and 14) since the return temperature is higher and the operation time of the cooling towers in free cooling mode is prolonged.

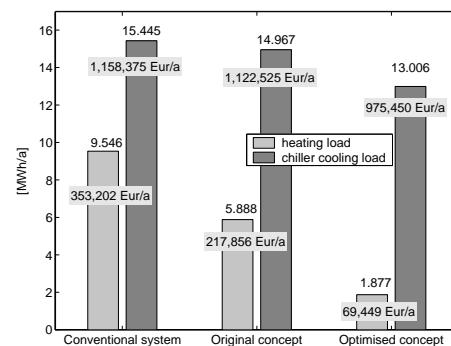


Figure 12: Development of the primary energy consumption for heating and cooling and reduction of running costs for the conventional separated system without heat recovery, the original concept with heat recovery and the optimised plant

It has to be emphasised that this study in this depth was only possible by means of transient simulation which enables the display of the temperatures throughout all components under consideration of the thermal capacities. This unique feature of a dynamic simulation is one advantage when the temperatures have a large impact on the efficiency of the process (i. e. the COP of chillers, denoting the quotient of the cooling capacity to the electric power consumption) and dominating capacities are characterising the energy system (i. e. storage tanks). A static simulation is not able to consider these important effects.

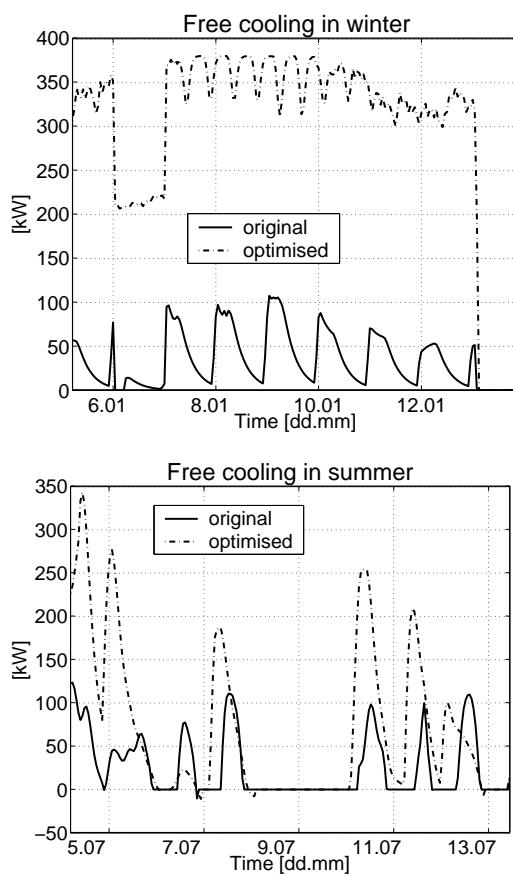


Figure 13: Cooling impact of the heat exchanger for free cooling in winter (a.) and summer (b.) - optimised plant including all modifications

The remaining cooling load of each calculation is multiplied with a fixed cooling price of 75 EUR/MWh resulting from experiences of the plant owner, again. The original concept could already save 36,000 EUR/a. Taking all modifications of the optimisation into account the costs could be drastically decreased by 183,000 EUR/a. These cost reductions are mainly due to the optimisation of the free cooling heat

exchanger position. The plot of the heat, transferred by the free cooling heat exchanger, is shown in Fig. 14.

Regarding the economical effect of the simulation, it is evident, that the optimised plant layout can save 295,000 EUR/a in comparison to the original concept and more than 465,000 EUR/a if the plant would have been built in a conventional way without heat recovery. Apart from the costs for the changed piping the modifications of the optimisation do not require expensive components.

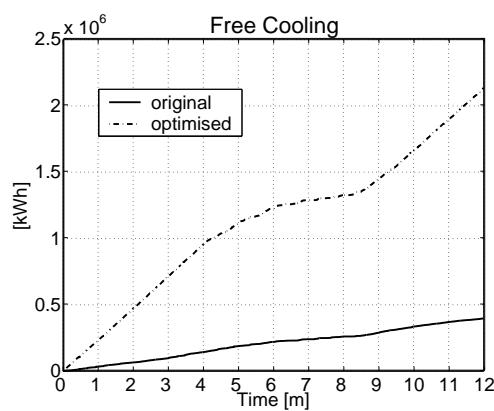


Figure 14: Amount of free cooling - optimised plant including all modifications

The computing time for a year time simulation of the original model and the optimised model are in the range of 10 to 24 hours depending on the installed processor.

5 Conclusions

This article is dedicated to the transient simulation of complex energy systems like they appear in large buildings and industrial plants. For this purpose a simulation tool, called HKSim [1, 4], was developed by Imtech Deutschland GmbH & Co. KG and the Department of Technical Thermodynamics of the Technical University of Hamburg-Harburg. It was pointed out, that such a tool is capable to simulate even complex systems. The computational effort can be justified by the prevention of possible failures in system layouts and estimation of possible savings, which are of economic interest as could be shown in the described optimisation. In the carried out simulation of the industrial plant the savings will pay back the investment in a short period of time.

In addition, a model of a thermal consumer was presented, which is necessary to integrate the heating or cooling demand from an external file into the system simulation. The model is designed for fast model generation and simulations of whole years and it predicts the mass flow rate and the return temperature with a good agreement to measurement data.

References

- [1] B. Lüdemann, St. Wischhusen, O. Engel, G. Schmitz: Optimierte Energiesysteme, BWK, Bd. 55, Nr. 9, Springer VDI-Verlag, Düsseldorf, Germany, 2003.
- [2] St. Wischhusen, G. Schmitz: Numerical Simulation of Complex Cooling and Heating Systems, Proceedings of the 2nd Int. Modelica-Conference, Modelica Association, Oberpfaffenhofen, Germany, 2002.
<http://www.modelica.org/conference2002/papers.shtml>
- [3] Dynasim AB, Dymola 5.1, Lund, Sweden, 2003.
- [4] St. Wischhusen, G. Schmitz: Abschlussbericht zum Projekt "Entwicklung eines Simulationswerkzeuges zur wirtschaftlichen und energetischen Planung von Prozessen zur Wärme- und Kälteerzeugung – Entwicklung der Komponentenmodelle", Department of Technical Thermodynamics, Technical University of Hamburg-Harburg, Hamburg, Germany, 2003.
- [5] Modelica-Association: Modelica Standard Library, Linköping, Sweden, 2002.
<http://www.modelica.org/libraries.shtml>
- [6] B. Lüdemann: Auslegung, Energiebedarf und Komfort von Anlagen zur Heizung und Warmwasserbereitung im Niedrigenergiehaus bei Berücksichtigung des Nutzerverhaltens, PhD thesis at the Department of Technical Thermodynamics, Technical University Hamburg-Harburg, Books on Demand, 2002.
<http://www.bod.de>

Object-Oriented Modeling of Thermo-Fluid Systems

Hilding Elmquist¹, Hubertus Tummescheit², and Martin Otter³

¹Dynasim AB, Lund, Sweden, www.dynasim.se, Elmquist@dynasim.se

²UTRC, Hartford, U.S.A., Hubertus@control.lth.se

³DLR, Germany, www.robotic.dlr.de/Martin.Otter, Martin.Otter@dlr.de

Abstract

Modelica is used since 1998 to model thermo-fluid systems. At least eight different libraries in this field have been developed and are utilized in applications. In the last year the Modelica Association has made an attempt to standardize the most important interfaces, provide good solutions for the basic problems every library in this field have and supply sophisticated base elements, especially media descriptions. This paper summarizes the design, new Modelica language elements, new symbolic transformation algorithms and describes two new libraries – for media description and for fluid base components – that will be included in the Modelica standard library.

1 Introduction

Careful decomposition of a thermodynamic system is essential to achieve reusable components. This paper discusses appropriate Modelica interfaces to handle thermodynamic properties, empirical closure relations like pressure drop correlations, mass balances and energy balances. Special attention has been placed on allowing flows with changing directions and allowing ideal splitting and merging of flows by connecting several components at one junction as well as parallel flow paths having zero (neglected) volume. A purely declarative approach solves the problem of splitting and merging flows in a physically based way. For mixing, the resulting specific enthalpy or temperature is *implicitly* defined and is obtained by solving a system of equations.

All balance equations are provided in their natural form. Necessary differentiations are carried out by a tool through index reduction. Due to newly developed symbolic transformation algorithms, the described approach leads to the same simulation efficiency as previously developed thermo-fluid libraries, but without having their restrictions.

The discussed method is implemented in two new Modelica libraries, “Modelica_Fluid” and “Modelica_Media” that will become part of the free Modelica standard library as Modelica.Fluid and

Modelica.Media. “Media” contains a generic interface to media property calculations with required and optional media variables. A large amount of pre-defined media models are provided based on media models of the ThermoFluid library Tummescheit and Eborn (2001). Especially, about 1200 gases and mixtures of these gases, as well as a high precision water model based on the IF97 standard are included. The “Fluid” library provides the generic fluid connectors and the most important basic devices, such as sources, sensors, and pipes for quasi 1-dimensional flow of media with single or multiple phases and single or multiple substances. The same device model is used for incompressible and compressible flow. A tool will perform the necessary equation changes by index reduction when, e.g., an incompressible medium model is replaced by a compressible one in a device model.

The “Fluid” and “Media” libraries are a good starting point for application specific libraries, such as for steam power plants, refrigeration systems, machine cooling, or thermo-hydraulic systems.

2 Devices, medium models, balance volumes and ports

We will consider thermodynamic properties of fluids in coupled devices, such as tanks, reactors, valves as well as pipes, Figure 1. Control volumes (or balance volumes) will be considered for all devices.

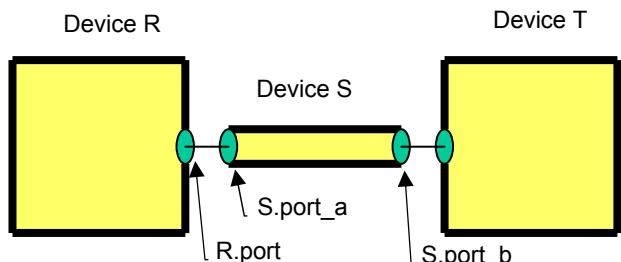


Figure 1. Connected devices

2.1 Medium models

The thermodynamic state of the fluid at any point is represented by two variables, e.g., pressure p and

specific enthalpy h . Other thermodynamic quantities may be calculated from the chosen thermodynamic state variables. It is important that a model for a device can be written in such a way that it can be used in combination with different media models. This property is achieved by representing the media as a replaceable package. The details are given in Section 5. Such a media package contains, among other definitions, a model with three equations as shown in the following partial example for a simple model of air based on the ideal gas law:

```
package SimpleAir
  ...
  constant Integer nX = 0;
  model BaseProperties
    AbsolutePressure      p;
    Temperature          T;
    Density               d;
    SpecificInternalEnergy u;
    SpecificEnthalpy      h;
    MassFraction          X[nX];
    constant Real R_air = 287.0506;
    constant Real h0     = 274648.7;
  equation
    p = d*R_air*T;
    h = 1005.45*T + h0;
    u = h - p/d;
  end BaseProperties;
  ...
end SimpleAir;
```

How such a media package can be utilized in a model is shown in the following heated device model without incoming or leaving mass flows.

```
model ClosedDevice
  import M = Modelica.Media;
  replaceable package Medium=
    M.Interfaces.PartialMedium;
  Medium.BaseProperties medium
  parameter ...
equation
  // Mass balance
  der(m) = 0;
  m = V*medium.d;

  // Energy balance
  der(U) = Q;
  U = m*medium.u;
end ClosedDevice;
```

When using this device model, a specific medium has to be defined:

```
ClosedDevice device(redeclare
  package Medium = SimpleAir);
```

The device model is not influenced by the fact that the medium model is compressible or incompressible.

2.2 Ports

Figure 2 shows a detailed view of a connection between two devices. An important design decision

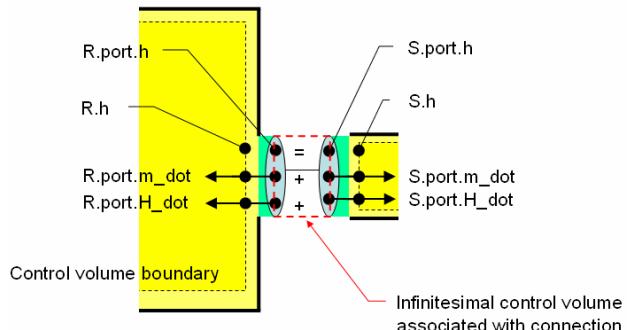


Figure 2. Details of device connection

is the selection of the Modelica connector that describes a device port. For the Modelica_Fluid library the connector is defined for quasi one-dimensional fluid flow in a piping network, with incompressible or compressible media models, one or more phases, and one or more substances. The connector variables are selected such that the equations of the connect(...) statements of connected components fulfill the following balance equations:

- mass balance
- substance mass balance (of a medium with several substances).
- energy balance in the form of the “internal energy balance” (see Section 3).

Additionally, a non-redundant set of variables is used in the connector in order to not have any restrictions how components can be connected together (restrictions would be present, if an overdetermined set of describing variables would be used in the connector). These design requirements lead to a unique selection of variables in the connector:

Pressure p , specific (mixing) enthalpy h , independent (mixing) mass fractions X , mass flow rate m_{dot} , enthalpy flow rate H_{dot} , and the independent substance mass flow rates mX_{dot}

```
connector FluidPort
  replaceable package Medium =
    Modelica_Media.Interfaces.PartialMedium;

  Medium.AbsolutePressure p;
  flow Medium.MassFlowRate m_dot;

  Medium.SpecificEnthalpy      h;
  flow Medium.EnthalpyFlowRate H_dot;

  Medium.MassFraction          X [Medium.nX]
  flow Medium.MassFlowRate mX_dot [Medium.nX]
end FluidPort;
```

Due to the design of the connectors, the mass and energy balance is fulfilled in connection points (see also discussion of perfect mixing in the next Section). Since the momentum balance is not taken into account, device couplings with a considerable amount of losses (e.g., if pipes with different diameters are connected) have to be modeled with a dedicated loss model.

2.3 Splitting, Joining and Reverse Flow

Figure 2 also shows the control volumes associated with the devices and the boundary conditions. The flow through the port of a device is equal to the flow through the corresponding boundary of the control volume. Note that the specific enthalpy might have a discontinuity.

The connector variable FluidPort.h represents the specific enthalpy outside the control volume of the device. In fact, for two connected devices R and S, with FluidPort instances named “port”, R.port.h = S.port.h represent the specific enthalpy of an infinitesimally small control volume associated with the connection. The relation between the boundary and the port specific enthalpy depends on the flow direction. It is established indirectly by considering the enthalpy flow. We will introduce the notation $h_{port} = R.port.h = S.port.h$ and will for simplicity of notation neglect spatial variation of the specific enthalpy, h_R and h_S , within each control volume. The enthalpy flow rate into device R, \dot{H}_R is then dependent on the mass flow rate, \dot{m}_R as follows.

$$\dot{H}_R = \begin{cases} \dot{m}_R h_{port} & \dot{m}_R > 0 \\ \dot{m}_R h_R & \text{otherwise} \end{cases}$$

This equation has to be present within the model of device R. Such conditional expressions could be written as if-then-else expressions, but to facilitate a recently identified set of powerful symbolic simplifications a new function, semiLinear(...), has been proposed for inclusion in the Modelica language (see also Figure 3), that can be used as follows in model R:

```
port.H_dot =
    semiLinear(port.m_dot, port.h, h);
```

The corresponding equation for a device S is

$$\dot{H}_S = \begin{cases} \dot{m}_S h_{port} & \dot{m}_S > 0 \\ \dot{m}_S h_S & \text{otherwise} \end{cases}$$

Devices R and S, see Figure 2, are connected together with a connect(...) statement of the form:

```
connect(R.port, S.port);
```

leading to the following *zero sum equations* that are equivalent to the mass and energy balance of the infinitesimal small control volume at the connection point:

$$\begin{aligned} 0 &= \dot{m}_R + \dot{m}_S \\ 0 &= \dot{H}_R + \dot{H}_S \end{aligned}$$

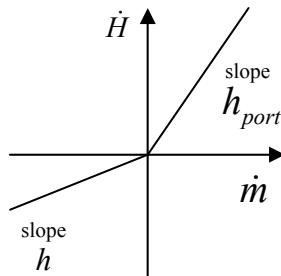


Figure 3. The semiLinear(...) function

From these four equations, h_{port} can be solved

$$h_{port} = \begin{cases} h_S & \dot{m}_R > 0 \\ h_R & \dot{m}_R < 0 \\ \text{undefined} & \dot{m}_R = 0 \end{cases}$$

According to Modelica flow semantics, $\dot{m}_R > 0$ corresponds to flow *into* component R and therefore the specific enthalpy flowing across the boundary is h_S at the device boundary, h_{port} . It should be noted that although h_{port} is undefined for zero mass flow rate, \dot{H}_R and \dot{H}_S are well-defined as zero, i.e., the dynamics of the system are independent of what value is chosen for h_{port} .

We will now consider the connection of three ports R.port, S.port and T.port. A symbolic solution of the common specific enthalpy,

$h = R.port.h = S.port.h = T.port.h$ is given by

$$\begin{aligned} h = & - (\\ & (\text{if } R.port.m_dot > 0 \text{ then } 0 \text{ else } \\ & \quad R.port.m_dot * R.h) + \\ & (\text{if } S.port.m_dot > 0 \text{ then } 0 \text{ else } \\ & \quad S.port.m_dot * S.h) + \\ & (\text{if } T.port.m_dot > 0 \text{ then } 0 \text{ else } \\ & \quad T.port.m_dot * T.h)) \\ / & (\\ & (\text{if } R.port.m_dot > 0 \text{ then } \\ & \quad R.port.m_dot \text{ else } 0) + \\ & (\text{if } S.port.m_dot > 0 \text{ then } \\ & \quad S.port.m_dot \text{ else } 0) + \\ & (\text{if } T.port.m_dot > 0 \text{ then } \\ & \quad T.port.m_dot \text{ else } 0)) \end{aligned}$$

For a *splitting* flow, for example from R to S and T, i.e., $R.port.m_dot < 0$, $S.port.m_dot > 0$ and $T.port.m_dot > 0$, we get

$$h = -R.\text{port.m_dot} * R.h / (S.\text{port.m_dot} + T.\text{port.m_dot})$$

Since

$$0 = R.\text{port.m_dot} + S.\text{port.m_dot} + T.\text{port.m_dot}$$

the specific enthalpy h in the port is computed as

$$h = -R.\text{port.m_dot} * R.h / (-R.\text{port.m_dot})$$

or

$$h = R.h$$

For a *merging* flow, for example, from R and T into S (i.e., $R.\text{port.m_dot} < 0$, $S.\text{port.m_dot} < 0$ and $T.\text{port.m_dot} > 0$) we get

$$h = -(R.\text{port.m_dot} * R.h + S.\text{port.m_dot} * S.h) / T.\text{port.m_dot}$$

or

$$h = (R.\text{port.m_dot} * R.h + S.\text{port.m_dot} * S.h) / (R.\text{port.m_dot} + S.\text{port.m_dot})$$

i.e., the perfect mixing condition.

The degenerate case that all mass flows are zero can be handled symbolically by the tool, as it does not influence the dynamics: For two connected devices R and S, the division with $R.\text{port.m_dot}$ can be performed symbolically leading to

$$h = \text{if } R.\text{port.m_dot} > 0 \text{ then } R.h \text{ else } S.h$$

As a result, for zero mass flow rate $h = S.h$. For three and more connected devices, the equation system is underdetermined. From the infinitely many solutions the one can be picked that is closest to the solution in the previous integrator step.

It should be noted that a similar approach could be used to handle flow composition for flows with several substances.

Earlier attempts tried to solve a restricted problem of changing flow direction in a programming style, i.e., by *explicitly* defining the temperature depending on the flow direction. Such a method cannot be generalized to mixing flows, because the temperature is not given by equations in just one volume. The presented solution for splitting and joining flows is derived by considering the equations of a small connection volume. By setting its mass to zero, the usual sum-to-zero equations for mass flow rate and energy flow rate are obtained. This means that the usual flow semantics is appropriate for modeling of splitting and merging flows.

3 Mass-, momentum- and energy-balances

We will show a general implementation of the governing equations, which might serve as a

template for specialized models. Consider the equations (mass, momentum and energy balances) for quasi-one-dimensional flow in a device with flow ports in the ends such as a pipe, Thomas (1999) [16], Anderson (1995) [1].

$$\begin{aligned} \frac{\partial(\rho A)}{\partial t} + \frac{\partial(\rho Av)}{\partial x} &= 0 \\ \frac{\partial(\rho vA)}{\partial t} + \frac{\partial(\rho v^2 A)}{\partial x} &= -A \frac{\partial p}{\partial x} - F_F - A \rho g \frac{\partial z}{\partial x} \\ \frac{\partial(\rho(u + \frac{v^2}{2})A)}{\partial t} + \frac{\partial(\rho v(u + \frac{p}{\rho} + \frac{v^2}{2})A)}{\partial x} &= \\ -F_F v - A \rho v g \frac{\partial z}{\partial x} + \frac{\partial}{\partial x}(kA \frac{\partial T}{\partial x}) \\ F_F &= \frac{1}{2} \rho v |v| f S \end{aligned}$$

where t represents time, x is the spatial coordinate along device, ρ is the density, v is the velocity, A is the area, p is the pressure, F_F represents the friction force per length, f is the Fanning friction factor, S is the circumference, g is the gravity constant, z is the vertical displacement, k is the thermal conductivity and medium properties:

$$p = p(\rho, T)$$

$$u = u(\rho, T)$$

$$h = u + p/\rho$$

where h is the specific enthalpy and u is the specific internal energy.

The energy equation can be considerably simplified by subtracting the momentum balance multiplied by v . Simplifications that are shown in the appendix, give the result.

$$\frac{\partial(\rho u A)}{\partial t} + \frac{\partial(\rho v(u + \frac{p}{\rho})A)}{\partial x} = v A \frac{\partial p}{\partial x} + \frac{\partial}{\partial x}(k A \frac{\partial T}{\partial x})$$

Finite volume method

Such partial differential equations can be solved by various methods like finite difference, finite element or finite volume methods. The finite volume method is chosen because it has good properties with regards to maintaining the conserved quantities. The device is split into segments, for which the PDEs are integrated and approximated by ODEs. Let $x=a$ and $x=b$ be the coordinates for the ends of any such segment. Integrating the mass balance equation over the spatial coordinate, x , gives

$$\int_a^b \frac{\partial(\rho A)}{\partial t} dx + \rho A v|_{x=b} - \rho A v|_{x=a} = 0$$

Assuming the segment boundaries (a, b) to be constant, we can interchange the integral and derivative:

$$\frac{d\left(\int_a^b \rho A dx\right)}{dt} + \rho A v|_{x=b} - \rho A v|_{x=a} = 0$$

In order to handle the general case of changing volumes for, e.g., displacement pumps, tanks, or moving boundary models of two phase flows, this formula needs to be extended by use of the Leibnitz formula.

Introducing appropriate mean values for density and area and introducing incoming mass flow rates \dot{m} , i.e. $\dot{m}_b = -\rho A v|_{x=b}$ and $\dot{m}_a = \rho A v|_{x=a}$, we can rewrite the mass balance as:

$$\frac{d(\rho_m A_m (b-a))}{dt} = \dot{m}_a + \dot{m}_b$$

Introducing $m = \rho_m A_m L$ and $L = b - a$ gives the desired form of the mass balance

$$\frac{dm}{dt} = \dot{m}_a + \dot{m}_b$$

We proceed in a similar way with the momentum balance:

$$\begin{aligned} & \int_a^b \frac{\partial(\rho v A)}{\partial t} dx + \rho v^2 A|_{x=b} - \rho v^2 A|_{x=a} \\ &= -Ap|_{x=b} + Ap|_{x=a} + \int_a^b \frac{\partial A}{\partial x} pdx - \\ & \int_a^b \frac{1}{2} \rho v |v| f S dx - \int_a^b A \rho g \frac{\partial z}{\partial x} dx \end{aligned}$$

and introducing appropriate mean values gives:

$$\begin{aligned} & \frac{d(\rho_m v_m A_m)}{dt} L + \rho v^2 A|_{x=b} - \rho v^2 A|_{x=a} \\ &= A_m (p_a - p_b) \\ & - \frac{1}{2} \rho_m v_m |v_m| f_m S_m L - A_m \rho_m g \Delta z \end{aligned}$$

with $A_m = (A_a + A_b)/2$. Substitution by \dot{m} and the values at the respective boundaries and introducing the approximation $\dot{m}_m = \frac{\dot{m}_a + \dot{m}_b}{2}$ gives

$$\begin{aligned} \frac{d\dot{m}_m}{dt} L &= \frac{\dot{m}_a^2}{A_a p_a} - \frac{\dot{m}_b^2}{A_b p_b} + A_m (p_a - p_b) \\ & - \frac{1}{2} \frac{1}{\rho_m A_m^2} \dot{m}_m |f_m| f_m S_m L - A_m \rho_m g \Delta z \end{aligned}$$

We will make the approximation that $\rho_a = \rho_b = \rho_m$ evaluated at mean pressure $p_m = \frac{p_a + p_b}{2}$.

Integrating the energy balance for internal energy gives:

$$\begin{aligned} & \int_a^b \frac{\partial(\rho u A)}{\partial t} dx + \rho h v A|_{x=b} - \rho h v A|_{x=a} = \\ & \int_a^b v A \frac{\partial p}{\partial x} dx + k A \frac{\partial T}{\partial x} \Big|_{x=b} - k A \frac{\partial T}{\partial x} \Big|_{x=a} \end{aligned}$$

Substitution and approximation gives

$$\begin{aligned} & \frac{d(\rho_m u_m A_m L)}{dt} - \dot{m}_b h_b - \dot{m}_a h_a = \\ & v_m A_m (p_b - p_a) + k \frac{\partial T}{\partial x} \Big|_{x=b} - k \frac{\partial T}{\partial x} \Big|_{x=a} \end{aligned}$$

Introducing $U = \rho_m u_m A_m L = m u_m$, the inner energy and $\dot{H} = \dot{m} \cdot h$, the enthalpy flow rate give

$$\begin{aligned} \frac{dU}{dt} &= \dot{H}_a + \dot{H}_b + v_m A_m (p_b - p_a) \\ & + k A \frac{\partial T}{\partial x} \Big|_{x=b} - k A \frac{\partial T}{\partial x} \Big|_{x=a} \end{aligned}$$

The diffusion term contains the temperature gradients at the segment boundaries. A first order approximation of the gradient is

$$\frac{\partial T}{\partial x} \Big|_{x=a} = \frac{T(a + \frac{\Delta x}{2}) - T(a - \frac{\Delta x}{2})}{\Delta x}$$

It should be noticed that $T(a - \frac{\Delta x}{2})$ is a property of an adjacent segment, i.e. not directly accessible. However, such diffusion terms are already available in the model ThermalConductor of the Modelica.Thermal.HeatTransfer library. This means that we can introduce a heat flow port with T_m and \dot{Q} and write the energy equation as

$$\frac{dU}{dt} = \dot{H}_a + \dot{H}_b + v_m A_m (p_b - p_a) + \dot{Q}$$

The flow variable \dot{Q} will be the sum of the diffusion from neighboring segments at $x=a$ and $x=b$

and external heat transfer (for example in a heat exchanger).

Modelica model

The Modelica model equations corresponding to the mass-, momentum and energy balances derived above are given below. In addition, a medium component is used for the mean quantities. The semiLinear function is used to handle the interfacing of the balance volume boundary quantities with the quantities of the device ports as discussed earlier.

```

model DeviceSegment
  replaceable package Medium =
    Modelica_Media.Interfaces.PartialMedium;
  FluidPort port_a (redeclare package
    Medium=Medium);
  FluidPort port_b (redeclare package
    Medium=Medium);
  Medium.BaseProperties medium;
  // Variable and parameter declarations
  equation
    // Mean values
    medium.p = (port_a.p + port_b.p)/2;
    m_dot_m = (port_a.m_dot - port_b.m_dot)/2;
    d_m = medium.d;

    // Mass balance
    der(m) = port_a.m_dot + port_b.m_dot;
    m = medium.d*A_m*L;

    // Substance balances
    port_a.mX_dot = semiLinear(port_a.m_dot,
      port_a.X, medium.X);
    port_b.mX_dot = semiLinear(port_b.m_dot,
      port_b.X, medium.X);
    der(mX) = port_a.mX_dot + port_b.mX_dot;
    mX = m*medium.X;

    // Momentum balance
    L*der(m_dot_m) =
      A_m*(port_a.p - port_b.p)
      + port_a.m_dot*port_a.m_dot/(A_a*d_m)
      - port_b.m_dot*port_b.m_dot/(A_b*d_m)
      - m_dot_m*abs(m_dot_m) /
      (2*d_m*A_m^2)*f*S*L
      - A_m*d_m*g*(z_b - z_a);

    // Energy balance
    port_a.H_dot = semiLinear(port_a.m_dot,
      port_a.h, medium.h);
    port_b.H_dot = semiLinear(port_b.m_dot,
      port_b.h, medium.h);
    der(U) = port_a.H_dot + port_b.H_dot +
      m_dot_m/d_m*(port_b.p - port_a.p) +
      heatPort.Q_dot;
    U = m*medium.u;
    heatPort.T = medium.T;
  end DeviceSegment;

```

The model derivation given above is generic. It can be generalized and extended in many ways. For example, to allow changing volume of the segment, the integrations can be carried out with variable

boundaries, using the Leibnitz rule. In the above derivations, simple definitions of the mean values were used. It is possible to get better accuracy, for example, by using an upwind scheme taking into account the flow direction when calculating the mean values.

A staggered grid is sometimes used for solving such PDEs. It is claimed to give better convergence properties in certain cases by a better approximation of the pressure gradient. It is possible to make such an implementation in Modelica. In fact, the ThermoFluid library uses the staggered grid approach. In this case, the equation for momentum

is integrated over another interval $\left[a - \frac{L}{2}, a + \frac{L}{2} \right]$.

This momentum can be included in a flow element model. The mass and energy balances are included in a finite volume model. There are special problems of communicating, for example, the momentum term $\rho v^2 A \Big|_{x=a+L/2} - \rho v^2 A \Big|_{x=a-L/2}$ since the flow element is assumed to have the same mass flow rate at both its connectors. Additional, non-physical, connectors or additional connector variables need to be introduced in order to communicate these variables to neighboring flow elements.

4 Pressure Loss due to Friction

The momentum balance contains a term for the friction force

$$F_{fric} = \frac{1}{2} \frac{1}{\rho_m A_m^2} \dot{m}_m |\dot{m}_m| f_m S_m L$$

Often, the pressure loss is used instead of the friction force ($p_{Loss} = F_{fric}/A_m$) and different equations are in use to compute the pressure loss from the mass flow rate. In the Modelica_Fluid library a component to model this pressure loss is available that provides two versions of a generic pressure loss equation:

```

if from_dp then
   $\dot{m}_m = f_1(p_{Loss}, \dots)$ 
else
   $p_{Loss} = f_2(\dot{m}_m, \dots)$ 
end if

```

Using the parameter “from_dp” in the “Advanced”-menu, users can select whether the mass flow rate is computed from the pressure loss (this is the default) or whether the pressure loss is computed from the mass flow rate. Depending on how the device is connected in a network, there might be fewer non-linear equations if parameter “from_dp” is selected

correspondingly. In a future version, this selection might be performed automatically by a tool.

The user can currently choose between three variants of the pressure loss model:

1. **Constant Laminar:** $p_{Loss} = k \cdot \dot{m}$

It is assumed that the flow is only laminar. The constant k is defined by providing p_{Loss} and \dot{m} for nominal flow conditions that, for example, are determined by measurements.

2. **Constant Turbulent:** $p_{Loss} = k \cdot \dot{m} \cdot |\dot{m}|$.

It is assumed that the flow is only turbulent. Again, the constant k is defined by providing p_{Loss} and \dot{m} for nominal flow conditions. For small mass flow rates, the quadratic, or in the inverse case the square root, characteristic is replaced by a cubic polynomial. This avoids the usual problems at small mass flow rates.

3. **Detailed Friction:** provides a detailed model of frictional losses for commercial pipes with non-uniform roughness (including the smooth pipe as a special case) according to.:

$$\begin{aligned} p_{Loss} &= \lambda(Re, \Delta) \cdot \frac{L}{2D} \cdot \rho \cdot v \cdot |v| \\ &= \lambda_2(Re, \Delta) \cdot \frac{L \eta^2}{2D^3 \rho^3} = \lambda_2(Re, \Delta) \cdot k_2 \\ Re &= \frac{v \cdot D \cdot \rho}{\eta} = \frac{D}{A \cdot \eta} \cdot \dot{m} \end{aligned}$$

with

λ : friction coefficient ($= 4 \cdot f_m$)

λ_2 : used friction coefficient ($= \lambda \cdot Re \cdot |Re|$)

Re : Reynolds number.

L : length of pipe

A : cross-sectional area of pipe

D : hydraulic diameter of pipe

$= 4 * A / \text{wetted perimeter}$

(circular cross Section: $D = \text{diameter}$)

δ : Absolute roughness of inner pipe wall
(= averaged height of asperities)

Δ : Relative roughness ($= \delta/D$)

ρ : density

η : dynamic viscosity

v : Mean velocity

k_2 abbreviation for $L \eta^2 / (2D^3 \rho^3)$

Note that the Reynolds number might be negative if the velocity or the mass flow rate is negative. The "Detailed Friction" variant will be discussed in more detail, since several implementation choices are non-standard: The first equation above to compute the pressure loss as a function of the friction coefficient λ and the mean velocity v is usually used and presented in textbooks, see **Figure 4**. This form

is **not** suited for a simulation program since $\lambda = 64/|Re|$ if $|Re| < 2000$, i.e., a division by zero occurs for zero mass flow rate because $Re = 0$ in this case. More useful for a simulation model is the friction coefficient $\lambda_2 = \lambda \cdot Re \cdot |Re|$ introduced for the pipe loss component, because $\lambda_2 = 64 \cdot Re$ if $Re < 2000$ and therefore no problems for zero mass flow rate occur. The characteristic of λ_2 is shown in **Figure 5** and is implemented in the pipe loss model. The absolute roughness δ of the pipe is a parameter of this model.

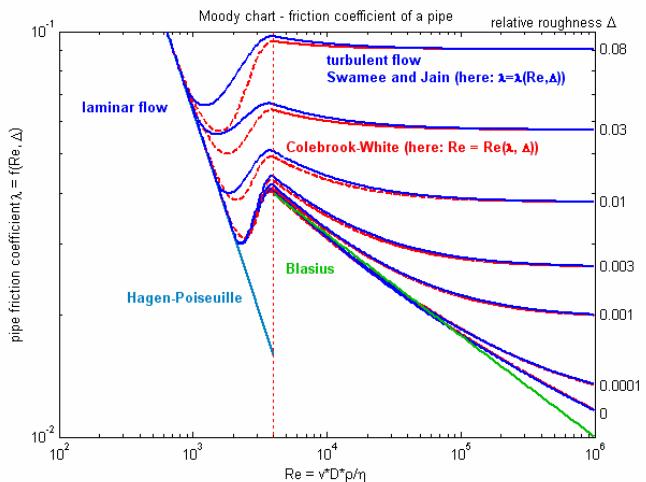


Figure 4. Moody Chart: $\lg(\lambda) = f(\lg(Re), \Delta)$

The pressure loss characteristic is divided into three regions:

Region 1: For $Re \leq 2000$, the flow is **laminar** and the exact solution of the 3-dim. Navier-Stokes equations (momentum and mass balance) is used under the assumptions of steady flow, constant pressure gradient and constant density and viscosity (= Hagen-Poiseuille flow):

$$\lambda_2 = 64 \cdot Re \quad \text{or} \quad p_{Loss} = \frac{64 \cdot k_2 \cdot D}{A \cdot \eta} \cdot \dot{m}$$

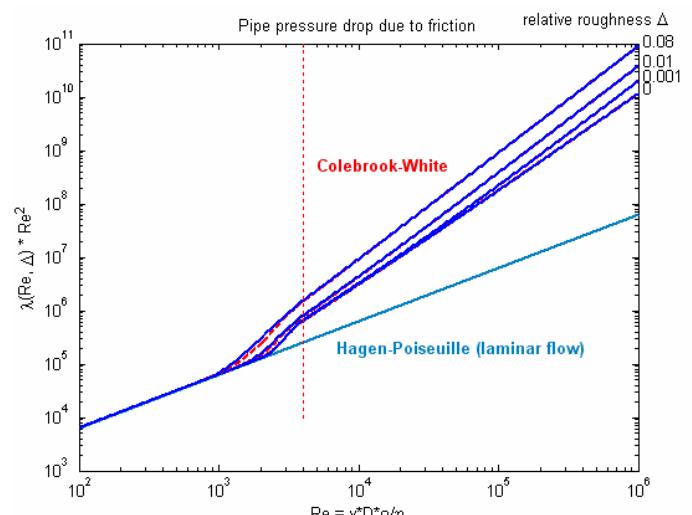


Figure 5. $\lambda_2 = \lambda_2(Re, \Delta) = \lambda \cdot Re \cdot |Re|$.
(x-axis: $\lg(Re)$, y-axis: $\lg(\lambda_2)$)

Region 3: For $\text{Re} \geq 4000$, the flow is **turbulent**. Depending on parameter “`from_dp`” either of two explicit equations are used: If `from_dp = true` ($\dot{m} = f_1(p_{Loss})$), λ_2 is computed directly from p_{Loss} using $\lambda_2 = p_{Loss}/k_2$. The Colebrook-White equation (Colebrook (1939); Idelchik (1994) p. 83, eq. (2-9))

$$\frac{1}{\sqrt{\lambda}} = -2 \cdot \lg \left(\frac{2.51}{\text{Re} \sqrt{\lambda}} + 0.27 \cdot \Delta \right)$$

gives an implicit relationship between Re and λ . Inserting $\lambda_2 = \lambda \cdot \text{Re} \cdot |\text{Re}|$ allows to solve this equation analytically for Re :

$$\text{Re} = -2 \cdot \sqrt{|\lambda_2|} \lg \left(\frac{2.51}{\sqrt{|\lambda_2|}} + 0.27 \cdot \Delta \right) \cdot \text{sign}(\lambda_2)$$

These are the full-line curves in **Figure 4** and **Figure 5**. If `from_dp = false` ($p_{Loss} = f_2(\dot{m})$), λ_2 is computed by an approximation of the inverse of the Colebrook-White equation (Swamee and Jain (1976); Miller (1990) p. 191, eq. (8.4)) adapted to λ_2 :

$$\lambda_2 = 0.25 \cdot \left[\text{Re} / \lg \left(\frac{\Delta}{3.7} + \frac{5.74}{|\text{Re}|^{0.9}} \right) \right]^2 \cdot \text{sign}(\text{Re})$$

These are the dotted-line curves in **Figure 4** and **Figure 5**.

Region 2: For $2000 \leq \text{Re} \leq 4000$ there is a **transition** region between laminar and turbulent flow. The value of λ_2 depends on more factors than just the Reynolds number and the relative roughness, therefore only crude approximations are possible in this area. A laminar flow up to $\text{Re} = 2000$ is only reached for smooth pipes. The deviation Reynolds number Re_1 at which the transition region starts is computed according to (Idelchik (1994), p. 81, sect. 2.1.21):

$$\text{Re}_1 = 745 \cdot e^{k_3}$$

$$k_3 = \text{if } \Delta \leq 0.0065 \text{ then } 1 \text{ else } 0.0065 / \Delta$$

Between $\text{Re}_1 = \text{Re}_1(\Delta)$ and $\text{Re}_2 = 4000$, λ_2 is approximated by a cubic polynomial in the “ $\lg(\lambda_2) = f(\lg(\text{Re}))$ ” chart (see **Figure 5**) such that the first derivative is continuous at these two points. In order to avoid the solution of non-linear equations, two different cubic polynomials are used for the direct and the inverse formulation (`from_dp = false/true`). This leads to some discrepancies in λ and λ_2 if $\Delta > 0.003$ (= differences between the full and the dotted curves in the above Figures). This is acceptable, because the transition region is not precisely known since the actual friction coefficient depends on

additional factors and since the operating points are usually not in this region.

The pressure loss equations above are valid for incompressible flow. According to (Idelchik (1994) p. 97, sect. 2.1.81) they can also be applied for compressible flow up to a Mach number of about $\text{Ma} = 0.6$ with a maximum error in λ of about 3 %. In a wide region the effect of gas compressibility can be taken into account by:

$$\lambda_{comp} = \lambda \cdot \left(1 + \frac{\kappa - 1}{2} \cdot \text{Ma}^2 \right)^{-0.47}$$

where κ is the isentropic coefficient (for ideal gases, κ is the ratio of specific heat capacities c_p/c_v). This effect is not yet included in the pipe friction model. Another restriction is that the pressure loss model is valid only for steady state or slowly changing mass flow rate. For large fluid acceleration, the pressure drop depends additionally on the frequency of the changing mass flow rate.

To summarize, the pipe friction component provides a detailed pressure loss model in pipes in the form $\dot{m} = f_1(p_{Loss}, \Delta)$ or $p_{Loss} = f_2(\dot{m}, \Delta)$. These functions are continuous and differentiable, are provided in an explicit form without solving non-linear equations, and do behave well also at small mass flow rates. This pressure loss model can be used stand-alone in a static momentum balance and in a dynamic momentum balance as the friction pressure drop term. It is valid for incompressible and compressible flow up to a Mach number of 0.6.

5 Standard Medium Interface

The main properties of a single substance medium are described by 3 algebraic equations between the 5 thermodynamic variables pressure (p), temperature (T), density (d), specific internal energy (u) and specific enthalpy (h). In a medium model, three of these variables are given as function of the remaining two. For multiple substance media, additionally nX independent mass fractions $X[nX]$ are present. For example, if p and T are selected as independent variables besides X, a medium model provides the algebraic equations

$$d = d(p, T, X)$$

$$u = u(p, T, X)$$

$$h = h(p, T, X)$$

The mass and energy balance equations in a device structurally have the following form for a single substance medium (see Section 3):

$$m = d \cdot V$$

$$U = m \cdot u$$

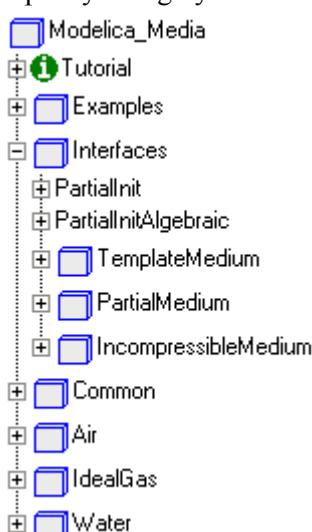
$$\frac{dm}{dt} = \sum_i \dot{m}_i \text{ // mass balance}$$

$$\frac{dU}{dt} = \sum_i \dot{m}_i h_i + \sum Q \text{ // energy balance}$$

where m is the mass and U is the internal energy in the control volume. Since the time derivatives of m and U appear, the derivatives of density d and internal energy u are implicitly needed which in turn means that the partial derivatives of $d(p,T)$ and $u(p,T)$ with respect to the independent variables p and T have to be calculated. As a result, the balance equations are reformulated in the variables p , T and this requires differentiation and formula manipulation.

Depending on the modeled device, additional fluid properties are needed, e.g., the dynamic viscosity if friction is modeled directly or the thermal conductivity for heat transfer coefficients or if diffusion is taken into account. Finally, a fluid may undergo phase changes and/or multiple substances may be involved.

Obtaining and computing the discussed fluid properties often takes the most effort in the modeling process. The availability of measurement data or correlations defines the level of accuracy that can be obtained with a thermo-fluid model. The needs of applications vary broadly from very simple properties with constant density and constant heat capacity to highly accurate non-linear models.



this package in the Modelica standard library as `Modelica.Media` after an evaluation phase.

The `Modelica_Media` library is designed such that it can be used in different thermo-fluid libraries that may, e.g., have completely different connector

definitions and design philosophies. In particular, the `Modelica_Fluid` library discussed in previous sections is based on this library, but it might also be useful for other thermo-fluid libraries. The `Modelica_Media` library has the following fundamental properties:

- Different independent medium variables may be used for media description, e.g., p,T or p,h or d,T or p,d .
- The definition of the medium is decoupled from the formulation of the balance equations in order that the balance equations can be formulated in their most natural form. There is enough information available for a tool to transform the medium equations into the form needed by the balance equations. This is achieved with the same efficiency as a usually used balance equation dedicated to a particular set of independent medium variables.
- Device models can be implemented independently of the choice of medium model. For example, exchanging an incompressible by a compressible medium model or a single by a multiple substance medium model is usually possible and has no major influence on the design of the device model.

5.1 Structure of Medium Interface

A medium model of `Modelica_Media` is essentially a **package** that contains the following definitions (the basic idea for this approach is from Newman et al (2002)):

- Definition of **constants**, such as the medium name or the number of substances.
- A **model** in the package that contains the 3 basic thermodynamic equations that relate the $5+nX$ primary medium variables.
- **Optional functions** to compute medium properties that are only needed in certain circumstances, such as dynamic viscosity. These optional functions need not be provided by every medium model.
- **Type definitions**, which are adapted to the particular medium. For example, a type "Temperature" is defined where the attributes "min" and "max" define the validity region of the medium. In a device model, it is advisable to use these type definitions, e.g., for parameters, in order that medium limits are checked as early as possible.

Note, although we use the term "medium model", this is actually a Modelica "package" that contains all the constants and definitions required for a complete "medium model". The basic interface to a

medium is defined by Modelica_Media.Interfaces.PartialMedium that has the following structure:

```

partial package PartialMedium
  import SI = Modelica.SIunits;

  constant String mediumName;
  constant String substanceNames;
  constant Boolean incompressible;
  constant Boolean reducedX;
  constant Integer nX = size(
    substanceNames, 1);

  record BasePropertiesRecord
    AbsolutePressure      p;
    Temperature          T;
    Density               d;
    SpecificInternalEnergy u;
    SpecificEnthalpy      h;
    MassFraction          X[nX];
  end BasePropertiesRecord;

  replaceable model BaseProperties
    extends BasePropertiesRecord;
    // parameter declarations
  end BaseProperties;

  // optional medium properties
  replaceable partial function
    dynamicViscosity
    input BasePropertiesRecord
           medium;
    output DynamicViscosity eta;
  end dynamicViscosity;
  // other optional functions

  // medium specific types
  type AbsolutePressure =
    SI.AbsolutePressure (
      min      = 0,
      max      = 1.e8,
      nominal = 1.e5,
      start   = 1.e5);
  type DynamicViscosity = ...;
  // other type definitions
end PartialMedium;

```

We will discuss all parts of this package in the following paragraphs. An actual medium model should extend from PartialMedium and has to provide implementations of the various parts.

The constants at the beginning of the package (with exception of nX) do not have a value yet (this is valid in Modelica), but a value has to be provided when extending from package PartialMedium. Once a value is given, it cannot be changed any more. The reason to use constants instead of parameters in the model BaseProperties is that some of these constants have to be used in connector definitions

(such as the number of mass fractions nX). When defining the connector, only *constants* in packages can be accessed, but not *parameters* in a model, because a connector cannot contain an instance of BaseProperties.

The record BasePropertiesRecord contains the variables primarily used in balance equations. Three equations for these variables have to be provided by every medium in model BaseProperties. Optional medium properties are defined by functions, such as the function "dynamicViscosity" (see code Section above) to compute the dynamic viscosity. Model BaseProperties extends from the record and the optional functions have an instance of this record as an input argument. This construction simplifies the usage considerably as demonstrated in the following code fragment:

```

replaceable package
  Medium = PartialMedium;
  Medium.BaseProperties medium;
  Medium.DynamicViscosity eta;
  ...
  U = m*medium.u; //Internal energy
  eta=Medium.dynamicViscosity(medium);

```

"Medium" is the medium package that satisfies the requirements of a "PartialMedium" (when using the model above, a value for Medium has to be provided by a redeclaration). The "medium" component is an instance of the model "Medium.BaseProperties" and contains the core medium equations. Variables in this model can be accessed just by dot-notation, such as medium.u or medium.T. If an optional medium variable has to be computed, the corresponding function from the actual Medium package is called, such as "Medium.dynamicViscosity". The medium instance can be given as input argument to this function, because model Medium.BaseProperties is a subclass of BasePropertiesRecord – the argument required from the function.

If a medium model does not provide implementations of all optional functions and one of these functions is called in a model, an error occurs during translation since the not redeclared optional functions have the "partial" attribute. For example, if function dynamicViscosity is not provided in the medium model when it is used, only simple pressure drop loss models without a reference to the viscosity can be used and not the sophisticated ones.

At the bottom of the PartialMedium package type declarations are present that are used in all other parts of the PartialMedium package and that should be used in all models and connectors where a medium model is accessed. The reason is that minimum, maximum, nominal and sometimes also

start values are defined and these values can be adapted to the particular medium at hand. For example, the nominal value of AbsolutePressure is $1.0e^5$ Pa. If a simple model of water steam is used that is only valid above 100 °C, then the minimum value in the Temperature type should be set to this value. The minimum and maximum values are also important for parameters in order to get an early message if data outside of the validity region is given. The “nominal” attribute is important as a scaling value if the variable is used as a state in a differential equation or as an iteration variable in a non-linear system of equations. The “start” attribute is useful to provide a meaningful default start or guess value if the variable is used, e.g., as iteration variable in a non-linear system of equations. Note, all these attributes can be set specifically for a medium in the following way:

```
package MyMedium
  extends PartialMedium(
    ...
    Temperature(min=373);
  );
...
end MyMedium;
```

The type PartialMedium.MassFlowRate is defined as

```
type MassFlowRate = SI.MassFlowRate
  (quantity =
   "MassFlowRate." + mediumName);
```

Note that the constant “mediumName”, that has to be defined in every medium model, is used in the quantity attribute. For example, if mediumName = “SimpleLiquidWater”, then the quantity attribute has the value “MassFlowRate.SimpleLiquidWater”. This type should be used in a connector definition of a fluid library:

```
connector FluidPort
  replaceable package Medium =
    PartialMedium;
  flow Medium.MassFlowRate m_dot;
  ...
end FluidPort;
```

In the model where this connector is used, the actual Medium has to be defined. Connectors can only be connected together, if the corresponding attributes are either not defined or have identical values. Since mediumName is part of the quantity attribute of MassFlowRate, it is not possible to connect connectors with different media models together. In Dymola this is already checked when models are connected together in the diagram layer of the graphical user interface.

5.2 Defining Medium Models

The definition of a new medium model based on the PartialMedium interface is demonstrated using a simple model for air. First, the template package “Modelica_Media.Interfaces.TemplateMedium” should be copied and renamed. Afterwards, all parts of this template should be adjusted to the actual medium model. In particular:

```
package SimpleAir
  extends Modelica_Media.Interfaces.PartialMedium(
    mediumName = "SimpleAir";
    substanceNames = fill("", 0);
    incompressible = false;
    reducedX = true;
  );
...
end SimpleAir;
```

The new medium package is extended from PartialMedium and all constants that do not have a value in PartialMedium are defined now. If the medium consists of only one substance, set the dimension of the substanceNames vector to zero with the `fill(..)` operator. If the medium defines the density to be a constant, set “incompressible” to `true`. If there is only one substance, set reducedX also to `true` (the meaning of this flag will be explained below).

In a next step, implementations of model BaseProperties and of all supported functions have to be provided. With the current Modelica language, this is cumbersome, since new classes with different names have to be introduced and then the PartialMedium classes have to be redeclared to the new names. A more convenient Modelica definition could be:

```
redeclare model BaseProperties
  extends;
  ...
end BaseProperties;
```

This just means that model BaseProperties, which is available due to “extends PartialMedium” is replaced by a model with the same name and all properties defined in PartialMedium.BaseProperties are included via the “extends” statement. This proposed language construct is available as a test implementation in Dymola. At the next Modelica design meeting, a formal decision will be made whether this or something similar will be included into the Modelica language. For the simple air model the redeclaration takes the form:

```

package SimpleAir
...
redeclare model BaseProperties
  import Modelica.SIunits.
    conversions.*;
  extends (
    p(stateSelect = ...),
    T(stateSelect = ...)
  );
  constant Real R_air = 287.0506;
  constant Real h0      = 274648.7;
equation
  p = d*R_air*T;
  h = 1005.45*T + h0;
  u = h - p/d;
end BaseProperties;
...
end SimpleAir;

```

The “stateSelect = ...” statements read

```

stateSelect =
  if preferedMediumStates then
    StateSelect.prefer
  else
    StateSelect.default

```

This is the essential definition to decouple balance and medium equations: “preferedMediumStates” is a Boolean parameter defined in PartialMedium. In every device that needs medium properties for balance equations in the form of differential equations, this flag has to be set to **true**. If no derivatives of any of the 5+nX basic thermodynamic variables are needed, this flag has to be set to **false**. Due to the above **if**-expression, the stateSelect attributes of the independent medium variables are set to “prefer” if preferedMediumStates = **true**. This in turn means that implicitly equations of the form “pd = **der**(p)” and „Td = **der**(T)“ are present and that p and T should be selected as states, if this is possible. This is important, if the property functions, such as u(p,T) are non-linear in the independent variables. If the independent variables would not be selected as states, this would result in non-linear systems of equations for the inversion of the property function.

The balance equations and the medium equations together with the above definition of preferred states define a DAE (= Differential Algebraic Equation system) of index 2. For example, if p and T are used as independent medium variables, this DAE consists of the following equations:

$$\begin{aligned}
 & // medium equations \\
 d &= d(p, T) \\
 u &= u(p, T) \\
 h &= h(p, T) \\
 pd &= \dot{p} \\
 Td &= \dot{T} \\
 & // balance equations \\
 m &= d \cdot V \\
 U &= m \cdot u \\
 \frac{dm}{dt} &= \dots // mass balance \\
 \frac{dU}{dt} &= \dots // energy balance
 \end{aligned}$$

Modelica models often result in higher index DAEs. Dymola solves this problem by using (a) the Pantelides algorithm (Pantelides (1988)) to determine the equations that have to be differentiated and (b) the dummy derivative method (Mattsson and Söderlind (1993), Mattsson et.al. (2000)) to select appropriate states. For the above code fragment, the Pantelides algorithm determines that the equations of m, U and therefore also of d and u need to be differentiated resulting in the following additional equations:

$$\begin{aligned}
 \dot{m} &= V \cdot \dot{d} \\
 \dot{U} &= \dot{m} \cdot u + m \cdot \dot{u} \\
 \dot{d} &= \frac{\partial d}{\partial p} \cdot \dot{p} + \frac{\partial d}{\partial T} \cdot \dot{T} \\
 \dot{u} &= \frac{\partial u}{\partial p} \cdot \dot{p} + \frac{\partial u}{\partial T} \cdot \dot{T}
 \end{aligned}$$

With the dummy derivative method it is possible to select p and T as states from the original set of potential states (p,T,m,U), especially since p and T have the “prefer” attribute. Using symbolic formula manipulation it is possible to solve the above equations efficiently for \dot{p}, \dot{T} .

Note, it is important to set the stateSelect attribute to its default value when preferedMediumStates = **false**. Otherwise, a tool would have to compute the derivative of p and T, although these derivatives are not needed. Worse, in order to compute these derivatives most likely other device equations would have to be differentiated.

After implementation of the BaseProperties model, the optional functions supported by the medium model have to be defined, e.g., a constant dynamic viscosity for the simple air model:

```

package SimpleAir
...
redeclare function dynamicViscosity
  input BasePropertiesRecord medium;
  output DynamicViscosity eta;
algorithm
  eta := 1.82e-5;
end dynamicViscosity;
...
end SimpleAir;

```

Note, instead of using the short “extends;” as in the `BaseProperties` model, it is also possible to just repeat the declaration of the function (this is possible with Modelica’s type system). For the optional functions, this is a bit longer but seems to be easier to understand for someone looking up the function definition.

The essential part of the medium model is now defined and can be utilized. However, there are additional issues that have to be taken into account, especially for non-linear medium models. This is discussed in the next subsections.

5.3 Initialization

Since variables of the medium are used as states, and the device models using the medium model do (on purpose) not know what independent variables are defined in the medium, initialization has to be defined in the medium model.

For fluid modeling, two types of standard initializations are common: steady state and prescribed initial conditions. A third alternative is additionally supported in the `Modelica_Media` library: The time scales of the energy- and mass balance related dynamics can be very different for fluid systems and are therefore treated differently in the initialization. A potential state that is determined by the mass balance dynamics (pressure or density) is initialized in steady state i.e., `der(d)=0` or `der(p)=0`. A potential state that is determined by the energy balance equation (temperature or specific enthalpy) is directly set (e.g. $T = 300.0$ or $h = 2.5e6$). This case occurs also when, e.g., initial temperatures are determined by measurements.

In package `PartialMedium`, several parameters are declared in order to define the initialization. A Dymola screen shot of the “Initialization” menu tab is shown in **Figure 6**. In the lower part, start values for p or d, T or h, and X can be defined. The meaning of a start value, e.g., whether it is a guess value or a definite start value is defined by the first parameter “`initType`”. It is defined with a selection box containing several alternatives (this is implemented as Integer with annotations to specify the content of the selection box, since Dymola does

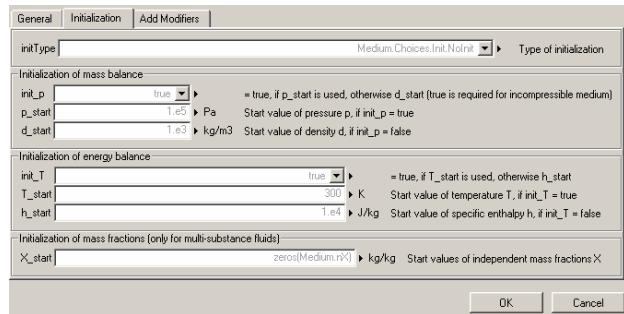


Figure 6 Initialization menu of `PartialMedium`

not yet support Modelica enumerations):

- Selection `NoInit` (the default) does nothing, to allow user-specific initialization.
- Selection `InitialStates` means that the independent variables of the medium model should be initialized with start values.
- Selection `SteadyState` sets the time derivatives of the independent medium variables to zero. The start values are interpreted as guess values for the occurring non-linear algebraic equations.
- Selection `SteadyMass` sets one of the equations $\text{der}(p) = 0.0$ or $\text{der}(d) = 0$, depending whether p or d is an independent variable of the medium model. The start value for p or d is interpreted as a guess value. The start value for T or h is used to initialize the remaining independent variable of the medium model.

In the lower part of the “Initialization” menu, start values can be defined. If the Boolean `init_p = true`, then the start value `p_start` for pressure is used, otherwise the start value `d_start` for density. Correspondingly, if `init_T = true`, the start value `T_start` for temperature is used, otherwise the start value `h_start` for specific enthalpy. Additionally, for multiple substance fluids, start values for mass fractions `X_start` can be defined. Start values that are not needed are used as initial guesses, where appropriate.

While this is not a fully exhaustive list of useful initializations for fluid models, it provides a broad range of practically important cases.

The above parameters are defined in package `PartialMedium`. An actual implementation must be provided by every medium model. For example, the simple air model, needs the following additions:

```

package SimpleAir
...
redeclare model BaseProperties
  import C = Choices.Init;
  protected
    parameter T_start2 =
      if init_T then
        T_start
      else

```

```

        (h_start - h0)/cp_air;
parameter h_start2 =
  if init_T then
    cp_air*T_start + h0
  else h_start;
parameter p_start2 =
  if init_p then
    p_start
  else R_air*d_start*T_start2;
parameter d_start2 =
  if init_p then
    p_start/(R_air*T_start2)
  else d_start)
public
extends (
  p(start = p_start2, stateSelect=...),
  T(start = T_start2, stateSelect=...),
  d(start = d_start2),
  h(start = h_start2),
  u(start = h_start2 - p_start2/
    d_start2)
);
constant Real R_air = 287.0506;
constant Real cp_air = 1005.45;
constant Real h0 = 274648.7;

```

Above is the first part of the initialization. In the extends clause of the BaseProperties model together with the new protected Section, start values for all variables are calculated from the given start values. This requires to evaluate the medium equations with the given start values. In situations with more complex equations, it is often useful to define them with functions and call the functions for start value calculation and in the equation section. The reason to provide consistent start values for all variables is that these variables are potentially iteration variables in non-linear algebraic loops and need therefore reasonable guess values. It is not known beforehand which iteration variable the symbolic translator will select. In the remaining code, the initialization equations and the medium equations are defined:

```

initial equation
  if preferredMediumStates then
    if initType == C.InitialStates then
      p = p_start2;
      T = T_start2;
    elseif initType==C.SteadyState then
      0 = der(p);
      0 = der(T);
    elseif initType == C.SteadyMass then
      0 = der(p);
      T = T_start2;
    end if;
  end if;
equation
  p = d*R_air*T;
  h = cp_air*T + h0;
  u = h - p/d;
end BaseProperties;
...
end SimpleAir;

```

Initial equations are only provided if preferredMediumStates = **true**, i.e., if medium variables should be used as states. Depending on parameter initType, the different initialization equations are defined. These equations depend on the independent variables of the medium model.

5.4 Multiple Substance Media

Media that consist of several (non-reacting) substances are both supported from the Modelica_Media and the Modelica_Fluid library. In Modelica_Media essentially the mass fractions X of the substances are used as independent variables to compute the medium properties. Two common approaches are supported by the Modelica_Media library:

- From the n substances, n-1 substances are treated as independent, i.e., n-1 mass fractions are additional independent variables. If needed, the n-th mass fraction is computed from the algebraic equation $X_n = 1 - \sum(X[1:n-1])$.
- All n substances are treated as independent during simulation, i.e., n mass fractions are used as independent variables and there are n additional substance mass balance equations. Since the constraint that the mass fractions sum up to one, is not utilized, a slight drift of the mass fractions may occur. Of course, the initial mass fractions have to be defined such that they are summed up to one (this is checked in the PartialMedium package).

In order to not have special cases, the Modelica_Media and Modelcia_Fluid libraries define the constant "nX" of PartialMedium to be the "number of independent" mass fractions. This might be n-1 or n substances of a multiple substance medium. In order to be able to make some checks, such as for initialization, the constant "reducedX" must be defined. If **true**, nX characterizes n-1 substances, if this flag is **false**, nX characterizes n substances.

Note, for single substance media, no mass fraction vector or substance mass flow rate vector is present, because nX = 0 in this case and zero sized vectors are removed in the code generation phase.

6 Medium Models in Modelica_Media

In this Section, some of the more advanced medium models available in the Modelica_Media package are discussed in more detail. All of them are based on the medium interface described in the last Section.

6.1 High Accuracy Water Model IF97

The Modelica_Media library contains a very detailed medium model of water in the liquid, gas and two phase region based on the IF97 standard [6]. It is an adapted and slightly improved version from the ThermoFluid library (Tummescheit and Eborn (2001), Tummescheit (2002)).

High accuracy thermodynamic properties of fluids are modeled with two kinds of multi-parameter, fundamental equations of state:

- An equation for the specific Helmholtz free energy $f(\rho, T)$ or $f(v=1/\rho, T)$
- An equation for the Gibbs free enthalpy $g(p, T)$

For numerical reasons the fundamental equations use dimensionless variables which are most often scaled with the critical parameters. The IF97 industrial steam tables uses both equation types and furthermore divides the overall fluid state into 5 regions in order to achieve high accuracy everywhere with a lower number of parameters. In spite of the complexities of the underlying formulation, the user interface for calling the properties is very simple. The medium interface is implemented with utility functions that have a simple interface, e.g.

```

rho = Water.IF97.rho_ph(p, h);
      //density
T    = WaterIF97.T(p, h);
      //temperature
s    = WaterIF97.s_ph(p, h);
      //specific entropy

```

Common sub-expression elimination and nested inlining of function calls ensure that the computationally expensive call to one of the fundamental equations happens only once. A record containing the fundamental derivatives of the equation of states is used by Dymola in the common sub-expression elimination and is thus only computed once. The fundamental derivatives for the free Helmholtz energy $f(\rho, T)$ are:

$$p = \rho^2 f_\rho = \rho^2 \frac{\partial f}{\partial \rho} \Big|_T$$

$$s = -f_T$$

$$p_T = \rho^2 f_{T\rho}$$

$$p_\rho = 2\rho f_\rho + \rho^2 f_{\rho\rho}$$

$$c_v = -T f_{TT}$$

Here the short subscript notation is used for partial derivatives, see explanation above. A similar set of fundamental derivatives exists for the Gibbs free enthalpy $g(p, T)$:

$$\begin{aligned}s &= -g_T \\v &= g_p \\v_T &= g_{pT} \\v_p &= g_{pp} \\c_p &= -Tg_{pp}\end{aligned}$$

From these fundamental derivatives, all other partial derivatives of thermodynamic properties with respect to other properties can be computed using thermodynamic determinants, e.g.

$$\frac{\partial \rho}{\partial p} \Big|_h = \frac{\rho(c_v \rho + p_T)}{\rho^2 p_\rho c_v + T p_T^2}, \quad \frac{\partial \rho}{\partial h} \Big|_p = \frac{\rho^2 p_T}{\rho^2 p_\rho c_v + T p_T^2}$$

When needed, e.g. for index reduction to change the states to numerically favorable ones, these partial derivatives can be computed with minimal effort from the fundamental derivatives in the property record. In order to add other Helmholtz- or Gibbs-based equations of state to Modelica_Media, only the fundamental derivatives need to be computed, the functions to compute the standard properties are part of the library.

The partial derivatives are used in two situations where the Modelica_Media properties provide unique features for efficiency and model order reduction. For all property calls that may have to be differentiated for index reduction, efficient derivative functions are provided. A very useful model order reduction for large two-phase heat exchangers is to equate the metal mass and boiling water temperatures, e.g. as in the drum Boiler model in [3]. Equating the temperatures leads to an index reduction problem. The algorithm for index reduction needs to compute the time derivative of temperature as a function of the time derivatives of the states. When pressure p and specific enthalpy h are the states, the expansion reads:

$$\frac{dT}{dt} = \frac{\partial T}{\partial p} \Big|_h + \frac{\partial T}{\partial h} \Big|_p \quad \text{if in single phase}$$

$$\frac{dT}{dt} = \frac{\partial T_{sat}}{\partial p} \quad \text{if in the two phase region}$$

These derivatives are automatically computed when needed without user interaction. This allows writing the equations in the most natural form, as demonstrated in [3]. The same algorithmic procedure is used to transform the “natural” form of the mass- and energy balances into equations using the input to the property routines as states.

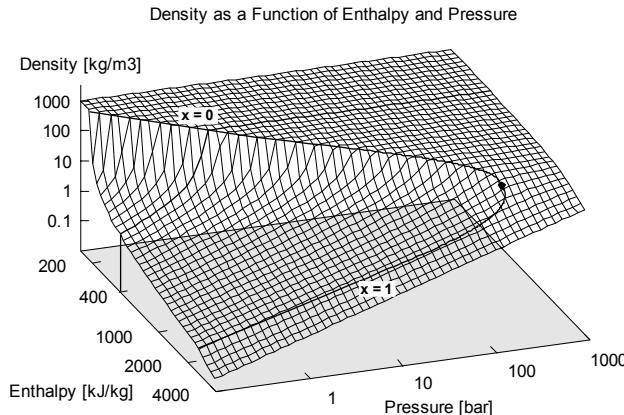


Figure 7: log.-plot of $\rho(p,h)$ for IF97 water

6.2 High Accuracy Ideal Gas Models

Ideal gas properties cover a broad range of interesting engineering applications: air conditioning and climate control, industrial and aerospace gas turbines, combustion processes, automotive engines, fuel cells and many chemical processes. Critically evaluated parameter sets are available for a large number of substances. The coefficients and data used in the Modelica_Media library are from [9]. Care has been taken to enable users to create their own gas mixtures with minimal effort. For most gases, the region of validity is from 200 K to 6000 K, sufficient for most technical applications. The equation of state consists of the well-known ideal gas law $p = \rho \cdot R \cdot T$ with R the specific gas constant, and polynomials for the specific heat capacity $cp(T)$, the specific enthalpy $h(T)$ and the specific entropy $s(T, p)$:

$$\begin{aligned} cp(T) &= R \sum_{i=1}^7 a_i T^{i-3} \\ h(T) &= RT \left(-\frac{a_1}{T^2} + a_2 \frac{\log(T)}{T} + \sum_{i=3}^7 a_i \frac{T^{i-3}}{i-2} + \frac{b_1}{T} \right) \\ s_0(T) &= R \left(-\frac{a_1}{2T^2} - \frac{a_2}{T} + a_3 \log(T) + \sum_{i=4}^7 a_i \frac{T^{i-3}}{i-3} + b_2 \right) \\ s(T, p) &= s_0(T) - R \ln \left(\frac{p}{p_0} \right) \end{aligned}$$

The polynomials for $h(T)$ and $s_0(T)$ are derived via integration from the one for $cp(T)$ and contain the integration constants b_1, b_2 that define the reference specific enthalpy and entropy. For entropy differences the reference pressure p_0 is arbitrary, but not for absolute entropies. It is chosen as 1 standard atmosphere (101325 Pa). Depending on the intended use of the properties, users can choose between different reference enthalpies:

1. The enthalpy of formation H_f of the molecule can be included or excluded.
2. The value 0 for the specific enthalpy without H_f can be defined to be at 298.15 K (25 °C) or at 0 K.

For some of the species, transport properties are also available. The form of the equations is:

$$\lg \left(\frac{\nu}{10^k} \right) = A \cdot \lg(T) + \frac{B}{T} + \frac{C}{T^2} + D$$

$$\lg(\lambda) = A_\lambda \cdot \lg(T) + \frac{B_\lambda}{T} + \frac{C_\lambda}{T^2} + D_\lambda$$

$$\eta = \nu \cdot \rho$$

with the kinematic viscosity ν , dynamic viscosity η , thermal conductivity λ and parameters A,B,C,D and k. Note, though, that the thermal conductivity is only the “frozen” thermal conductivity, i.e., not valid for fast reactions.

6.3 Ideal Gas Mixtures

For mixtures of ideal gases, the standard, ideal mixing rules apply:

$$h_{mix}(T) = \sum_{i=1}^{n_{species}} h_i(T) x_i$$

$$s_{mix}(T) = \sum_{i=1}^{n_{species}} (s_i(T) - R_i \ln(y_i)) x_i - R \ln \left(\frac{p}{p_0} \right),$$

where the x_i are the mass fractions, the R_i are the specific gas constants and the y_i are the molar fractions of the components of the mixture. Most other properties are then computed just as for single species media. Dynamic viscosity and thermal conductivity for mixtures require interaction parameters of a similar functional form as the viscosity itself and are (not yet) implemented.

For mixtures of ideal gases, three usage scenarios can be distinguished:

1. The composition is fixed and is the same throughout the system. This means that a new data record can be computed by preprocessing the component property data that can be treated as a new, single species data record (assuming ideal mixing).
2. The composition is variable, but changes in composition occur only through convection, i.e. slowly.
3. The composition is variable and may change through reactions too, i.e. composition changes are possibly very fast.

Case 1 and 2 above can be handled within a single model with a Boolean switch, case 3 needs to extend from that model because usually a number of

additional properties are needed, e.g. the parameters to compute chemical equilibrium reaction constants. Modelica_Media will initially not contain models for reactive flows, but all data is present for users who wish to define such models.

7 Conclusions

Thermodynamic fluid modeling is complex in many ways. This paper has shown a careful structuring of libraries for medium and fluid components in such a way that the same component models can be used with different easily replaceable media. To our knowledge this is the first approach that is able to treat compressible and incompressible fluids in a unified framework. A careful consideration of numerous issues concerning numerical efficiency, model structuring and user friendliness has been presented in this paper:

- Suitable device interfaces
- Principles for handling of reversing, joining and splitting flows
- The governing partial differential equations and their transformation into ODEs
- Pressure loss calculations
- Medium interface design
- Initialization
- Media available in Modelica_Media

Much design effort has been spent on considerations for robust and efficient simulation. The presented framework and libraries have the potential to serve as a powerful base for the development of application-oriented libraries.

Appendix – Energy balance

This appendix contains the derivation of the equivalent but simpler energy balance.

Multiplication of the momentum balance by v gives

$$v \left(\frac{\partial(\rho v A)}{\partial t} + \frac{\partial(\rho v^2 A)}{\partial x} \right) = -vA \frac{\partial p}{\partial x} - vF_F - vA\rho g \frac{\partial z}{\partial x}$$

Utilizing the mass balance, this equation can be rewritten as

$$\frac{\partial(\rho(v^2/2)A)}{\partial t} + \frac{\partial(\rho(v^3/2)A)}{\partial x} = -vA \frac{\partial p}{\partial x} - vF_F - vA\rho g \frac{\partial z}{\partial x}$$

To show the equivalence, consider the two left hand sides:

$$LH_1 = v \frac{\partial(\rho v A)}{\partial t} + v \frac{\partial(\rho v^2 A)}{\partial x} = \\ v \left(\frac{\partial v}{\partial t} \rho A + v \frac{\partial(\rho A)}{\partial t} + \frac{\partial v}{\partial x} \rho v A + v \frac{\partial(\rho v A)}{\partial x} \right) = \\ v \left(\frac{\partial v}{\partial t} \rho A + \frac{\partial v}{\partial x} \rho v A + v \left(\frac{\partial(\rho A)}{\partial t} + \frac{\partial(\rho v A)}{\partial x} \right) \right) = \\ v \left(\frac{\partial v}{\partial t} \rho A + \frac{\partial v}{\partial x} \rho v A \right)$$

$$LH_2 = \frac{\partial(\rho(v^2/2)A)}{\partial t} + \frac{\partial(\rho(v^3/2)A)}{\partial x} = \\ \frac{\partial(v^2/2)}{\partial t} \rho A + (v^2/2) \frac{\partial(\rho A)}{\partial t} + \\ \frac{\partial(v^2/2)}{\partial x} \rho v A + (v^2/2) \frac{\partial(\rho v A)}{\partial x} = \\ v \frac{\partial v}{\partial t} \rho A + (v^2/2) \frac{\partial(\rho A)}{\partial t} + \\ v \frac{\partial v}{\partial x} \rho v A + (v^2/2) \frac{\partial(\rho v A)}{\partial x} = \\ v \left(\frac{\partial v}{\partial t} \rho A + \frac{\partial v}{\partial x} \rho v A \right)$$

i.e. $LH_1 = LH_2$.

Subtracting the equation derived above from the energy balance gives

$$\frac{\partial(\rho u A)}{\partial t} + \frac{\partial(\rho v(u + \frac{p}{\rho})A)}{\partial x} = vA \frac{\partial p}{\partial x} + \frac{\partial}{\partial x} (kA \frac{\partial T}{\partial x})$$

Acknowledgements

The design of this library has been a collaborative effort and many have contributed. Many thanks to Mike Tiller for suggesting the package concept and useful discussions and proofreading the paper. Many thanks to Rüdiger Franke for the first realistic tests of the libraries and his feedback, many thanks to Daniel Bouskela, Andreas Idebrant, Gerhart Schmitz, John Batteh, Charles Newman, Jonas Eborn, Sven Erik Mattsson, Hans Olsson and the users of the ThermoFluid library for many useful comments and feedback.

Bibliography

- [1] Andersson J.D., Jr. (1995): **Computational Fluid Dynamics – The Basics with Applications**, McGraw-Hill International Editions, ISBN 0 07 001685 2.
- [2] Colebrook F. (1939): **Turbulent flow in pipes with particular reference to the transition region between the smooth and rough pipe laws**. J. Inst. Civ. Eng. no. 4, pp. 14-25.
- [3] Dymola (2003): **Dymola Users Guide, Version 5.1**. Dynasim AB, <http://www.dynasim.se/>
- [4] Elmquist, H. (1978): **A Structured Model Language for Large Continous Systems**. PhD-Thesis, Lund Institute of Technology, Lund, Sweden.
- [5] Idelchik I.E. (1994): **Handbook of Hydraulic Resistance**. 3rd edition, Begell House, ISBN 0-8493-9908-4
- [6] IAPWS (1997); **Release on the IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam**. The International Association for the Properties of Water and Steam.
- [7] Mattsson S.E.; Söderlind G. (1993): **Index reduction in differential-algebraic equations using dummy derivatives**. SIAM Journal of Scientific and Statistical Computing, Vol. 14 pp. 677-692, 1993.
- [8] Mattsson S.E., Olsson H., Elmquist H. (2000): **Dynamic Selection of States in Dymola**. Modelica Workshop 2000 Proceedings, pp. 61-67, <http://www.modelica.org/workshop2000/proceedings/Mattsson.pdf>
- [9] McBride B.J., Zehe M.J., and Gordon S. (2002): **NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species**. NASA report TP-2002-211556
- [10] Miller D.S. (1990): **Internal flow systems**. 2nd edition. Cranfield:BHRA(Information Services).
- [11] Newman C.E., Batteh J.J., Tiller M. (2002): **Spark-Ignited-Engine Cycle Simulation in Modelica**. 2nd International Modelica Conference, Proceedings, pp. 133-142.
- [12] Pantelides C. (1988): **The Consistent Initialization of Differential-Algebraic Systems**. SIAM Journal of Scientific and Statistical Computing, pp. 213-231.
- [13] Rüdiger Franke (2003): **On-line Optimization of Drum Boiler Startup**. Proceedings of the 3rd International Modelica Conference, Linköping, 2003
- [14] Span R. (2000): **Multiparameter Equations of State – An Accurate Source of Thermodynamic Property Data**, Springer-Verlag.
- [15] Swamee P.K., Jain A.K. (1976): **Explicit equations for pipe-flow problems**. Proc. ASCE, J.Hydraul. Div., 102 (HY5), pp. 657-664.
- [16] Thomas P. (1999): **Simulation of Industrial Processes – For Control Engineers**, Butterworth, Heinemann, ISBN 0 7506 4161 4.
- [17] Tummescheit H. (2002): **Design and Implementation of Object-Oriented Libraries using Modelica**, PhD thesis, Department of Automatic Control, Lund Institute of Technology.
- [18] Tummescheit H., Eborn J. (2001): **ThermoFluid Modelica Library**. Homepage: <http://www.control.lth.se/~hubertus/ThermoFluid/>

On-line Optimization of Drum Boiler Startup

Rüdiger Franke, Manfred Rode
 ABB Corporate Research
 Wallstädter Str. 59
 68526 Ladenburg, Germany

Klaus Krüger
 ABB Utilities GmbH
 Kallstadtter Str. 1
 68309 Mannheim, Germany

E-Mail: {Ruediger.Franke, Manfred.Rode, Klaus.Krueger}@de.abb.com

Abstract

On-line optimization of industrial processes is increasingly important to minimize cost and environmental impact of a plant during its operation. The new Modelica.Media and Modelica.Fluid base libraries allow the dynamic modeling of process systems [4]. Their application to dynamic optimization is discussed on a tutorial example.

The optimization is based on a simple non-linear drum boiler model from the literature [1]. The model is implemented in Modelica using the new Modelica.Media and Modelica.Fluid base libraries.

The model exhibits three control inputs: feed water flow rate, heat input, and position of a valve at the steam outlet. A PI control is embedded into the model for the feed water flow. The remaining two control inputs are optimized. Optimization results are compared with a straightforward control strategy.

On-line optimization based on a sophisticated boiler model is currently being applied to a 700 MW coal fired power plant.

1 Introduction

The primary aim of on-line optimization of industrial processes is to minimize cost and environmental impact of a plant during its operation. This cost includes energy and raw material consumption, losses due to off-spec production, waste and exhaust treatment as well as maintenance. Especially for complex processes with many mutually interacting decision variables and constraints, a secondary aim often is to automatically find a feasible and reproducible operation.

A dynamic model can serve as basis for on-line optimization. An optimal control problem is formulated

for the model. The problem is solved in real-time and the optimization results are applied to the process.

On-line optimization is especially interesting as simulation models are being reused. However, the optimization is computationally expensive. This is because control trajectories are being sought and as constraints have to be fulfilled for state and output trajectories. Time discretization leads to large numbers of optimization variables and constraints, as opposed to a comparable small number for design optimization problems. That is why advanced numerical optimization methods are required to solve the large-scale optimization problems in real-time.

The on-line optimization can be performed repeatedly in a control loop. The resulting Nonlinear Model based Predictive Control (NMPC) is advantageous if a process model is available or affordable and if

- multiple controlled and manipulated variables need to be considered
- state constraints have to be fulfilled
- the control problem is non-linear

The example discussed here exhibits all these features.

2 Drum boiler model

2.1 Generic drum model

The drum boiler model from [1] is used. Inside the drum there is water in two phases: liquid and vapour. The thermodynamic state of both phases is assumed to be at the phase boundary. Feed water enters the drum and saturated steam leaves the drum. A furnace supplies energy for heating up and evaporating the feed

water. The model assumes a global mass balance

$$\frac{dm}{dt} = q_{m,W} - q_{m,S} \quad (1)$$

for feed water entering and steam leaving the drum and with the mass

$$m = \rho_v V_v + \rho_l V_l + m_D. \quad (2)$$

See the code in figure 2 for a list of variables. The global energy balance is

$$\frac{dU}{dt} = q_F + q_{m,W} h_W - q_{m,S} h_S. \quad (3)$$

with the internal energy

$$U = \rho_v h_v V_v + \rho_l h_l V_l - p(V_v + V_l) + m_D c_{p,D} T_D \quad (4)$$

considering vapour phase, liquid phase, volume work, and the thermal energy in the surrounding metal, respectively. It is assumed that the specific enthalphy of steam leaving the boiler is equal to the vapour enthalpy: $h_S = h_v$. Furthermore, ideal heat transfer between the water inside the drum and the surrounding metal is assumed. Consequently the metal temperature is equal to the saturation temparature of water for the pressure inside the drum: $T_D = T_{sat}(p)$. The constant total volume inside the drum boiler is

$$V_t = V_v + V_l. \quad (5)$$

Thermal stress occures in the thick walled drum if there are spatial temperature differences, which are caused by fast temperature variations, e.g. during start-up. As this stress leads to fatigue or even destruction, it needs to be hold within given limits. Here the thermal stress is modeled proportional to the time derivative of the metal temparature

$$\sigma_D = k \frac{dT_D}{dt}. \quad (6)$$

Note that the modeling of temperature gradients is also of practical importance when no measurements of wall temperatures are available.

2.2 Mathematical model analysis and transformation

The physical oriented generic model formulation given in subsection 2.1 can directly be formulated in Modelica. Mathematical details are treated automatically by the Modelica tool Dymola. Nevertheless the model is analyzed in this section, in order to outline important mathematical details.

The model forms a system of differential and algebraic equations (DAE). It has a number of disadvantages when applied to on-line optimization. The balance equations are formulated for mass and internal energy that are not measured. Drum pressure, temperature and liquid water level are important quantities for drum boiler control. The physical oriented model defines them via an implicit non-linear relationship, being numerically disadvantageously.

Moreover, equation (6) for thermal stress causes a high-index DAE, as the time derivative of drum temperature is used.

That is why the model shall be transformed into a more appropriate form prior to its application. In [1] it is proposed to use pressure p and volume of liquid V_l as state variables. It is explained, how the model equations for mass and energy balance can be transformed accordingly. This results into

$$e_{1,1} \frac{dV_l}{dt} + e_{1,2} \frac{dp}{dt} = q_{m,W} - q_{m,S}, \quad (7)$$

$$e_{2,1} \frac{dV_l}{dt} + e_{2,2} \frac{dp}{dt} = q_F + q_{m,W} h_W - q_{m,S} h_S \quad (8)$$

with

$$e_{1,1} = \rho_l - \rho_v \quad (9)$$

$$e_{1,2} = V_v \frac{\partial \rho_v}{\partial p} + V_l \frac{\partial \rho_l}{\partial p} \quad (10)$$

$$e_{2,1} = \rho_l h_l - \rho_v h_v \quad (11)$$

$$e_{2,2} = V_v \left(h_v \frac{\partial \rho_v}{\partial p} + \rho_v \frac{\partial h_v}{\partial p} \right) \quad (12)$$

$$+ V_l \left(h_l \frac{\partial \rho_l}{\partial p} + \rho_l \frac{\partial h_l}{\partial p} \right) \quad (13)$$

$$- V_{vl} + m_D c_{p,D} \frac{\partial T_{sat}(p)}{\partial p} \quad (14)$$

Furthermore, equation (6) can be re-written as

$$\sigma_D = k \frac{\partial T_D}{\partial p} \frac{dp}{dt}. \quad (15)$$

```

package WaterPhaseBoundaryIF97
  "Physical properties for water at phase boundary at boiling and dew curves"
  extends Modelica_Media.Interfaces.PartialMedium(
    mediumName = "WaterIF97",
    substanceNames = fill("", 0),
    incompressible = false,
    reducedX = true,
    MassFlowRate(quantity="MassFlowRate.WaterIF97"));
  // basic property definitions required for each medium model
  redeclare model BaseProperties
    extends;
    parameter Integer region = 0 "specify region 1 (liquid) or 2 (vapour)";
  equation
    assert(region == 1 or region == 2,
      "WaterPhaseBoundaryIF97 medium model only valid for regions 1 and 2");
    T = Modelica_Media.Water.IF97.BaseIF97.Basic.tsat(p);
    if region == 1 then
      d = Modelica_Media.Water.IF97.BaseIF97.Regions.rhol_p(p);
      h = Modelica_Media.Water.IF97.BaseIF97.Regions.hl_p(p);
    else
      d = Modelica_Media.Water.IF97.BaseIF97.Regions.rhov_p(p);
      h = Modelica_Media.Water.IF97.BaseIF97.Regions.hv_p(p);
    end if;
    u = h - p/d;
  end BaseProperties;
end WaterPhaseBoundaryIF97;

```

Figure 1: Property model for water at phase boundary between liquid and vapour

This model can now easily be applied to study the system dynamics and to determine relevant terms for a control application. After the non-linear state-space transformation, the implicit dependency of pressure and liquid volume has become linear.

2.3 Model implementation

The model implementation consists of three steps: The selection of appropriate medium models, the implementation of the evaporator component model, and the assembling of a complete system model allowing the simulation of the drum boiler.

The Modelica.Media library contains accurate properties for water and steam according to the IAPWS/IF97 standard [5]. Here a new medium model is formulated for the phase boundaries at the boiling and dew curves, using available low level function calls for property evaluation.

Figure 1 shows the Modelica formulation. The medium model is defined as a package assembling general information, like medium name, and actual property definitions. The package is derived from

the predefined Modelica.Media.Interfaces.PartialMedium. As the medium model is for a single substance, the flag for reduced mass fraction vector X is set to true, resulting in $\dim(X) = n - 1 = 0$ for $n = 1$ substance. Substance names are not defined.

The model BaseProperties contains the actual property definitions. It defines saturation temperature, density, enthalpy and specific total energy as functions of pressure. The region parameter is used to determine at which boundary the properties are evaluated.

This medium model can now be used to formulate the evaporator component model. Note that in Modelica the physically oriented model is directly formulated. Model transformations required for efficient and robust simulation, as e.g. discussed in subsection 2.2, are left to the Modelica translator.

Figure 2 shows the Modelica formulation. Considering that just the model given in subsection 2.1 is implemented, the listing appears relatively long. This is because many parameters and variables are involved that are declared one per line.

The evaporator model first imports the re-used libraries Modelica.Fluid.Interfaces and Model-

```

model Evaporator
  import Modelica.Fluid.Interfaces.*;
  import Modelica.SIunits.Conversions.*;
  import SI = Modelica.SIunits;
  // property and interface declarations
  package Medium = WaterPhaseBoundaryIF97;
  Medium.BaseProperties medium_a(region=1, p=port_a.p) "Medium in port_a";
  Medium.BaseProperties medium_b(region=2, p=port_b.p) "Medium in port_b";
  FluidPort_a port_a(redeclare package Medium = Medium);
  FluidPort_b port_b(redeclare package Medium = Medium);
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a heatPort;
  Modelica.Blocks.Interfaces.OutPort V(redeclare type SignalType = SI.Volume)
    "liquid volume (level)";
  Modelica.Blocks.Interfaces.OutPort sigma_D "Thermal stress in metal";
  // public parameters
  parameter SI.Mass m_D=300e3 "mass of surrounding drum metal";
  parameter SI.SpecificHeatCapacity cp_D=500
    "specific heat capacity of drum metal";
  parameter SI.Volume V_t=100 "total volume inside drum";
  parameter SI.Pressure p_start=from_bar(1) "initial pressure";
  parameter SI.Volume V_start=67 "initial liquid volume";
protected
  SI.Pressure p(start=p_start, stateSelect=StateSelect.prefer)
    "pressure inside drum boiler";
  SI.Volume V_v "volume of vapour phase";
  SI.Volume V_l(start=V_start, stateSelect=StateSelect.prefer)
    "volumes of liquid phase";
  SI.SpecificEnthalpy h_v=medium_b.h "specific enthalpy of vapour";
  SI.SpecificEnthalpy h_l=medium_a.h "specific enthalpy of liquid";
  SI.Density rho_v=medium_b.d "density in vapour phase";
  SI.Density rho_l=medium_a.d "density in liquid phase";
  SI.Mass m "total mass of drum boiler";
  SI.Energy U "internal energy";
  SI.Temperature T_D=heatPort.T "temperature of drum";
  SI.HeatFlowRate q_F=heatPort.Q_dot "heat flow rate from furnace";
  SI.SpecificEnthalpy h_W=port_a.e "feed water enthalpy";
  SI.SpecificEnthalpy h_S=medium_b.h "steam enthalpy";
  SI.MassFlowRate qm_W=port_a.m_dot "feed water mass flow rate";
  SI.MassFlowRate qm_S=port_b.m_dot "steam mass flow rate";
equation
  // balance equations
  m = rho_v*V_v + rho_l*V_l + m_D;
  U = rho_v*V_v*h_v + rho_l*V_l*h_l - p*V_t + m_D*cp_D*T_D;
  der(m) = qm_W + qm_S;
  der(U) = q_F + qm_W*h_W + qm_S*h_S;
  T_D = medium_a.T;
  // ideal heat transfer between metal and water
  V_t = V_l + V_v;
  // pressure and specific total enthalpies at ports
  port_a.p = p;
  port_b.p = p;
  port_b.E_dot = semiLinear(port_b.m_dot, port_b.e, h_v);
  port_a.E_dot = semiLinear(port_a.m_dot, port_a.e, h_l);
  // thermal stress
  sigma_D.signal[1] = 60*der(T_D);
  // liquid level
  V.signal[1] = V_l;
end Evaporator;

```

Figure 2: Evaporator component model

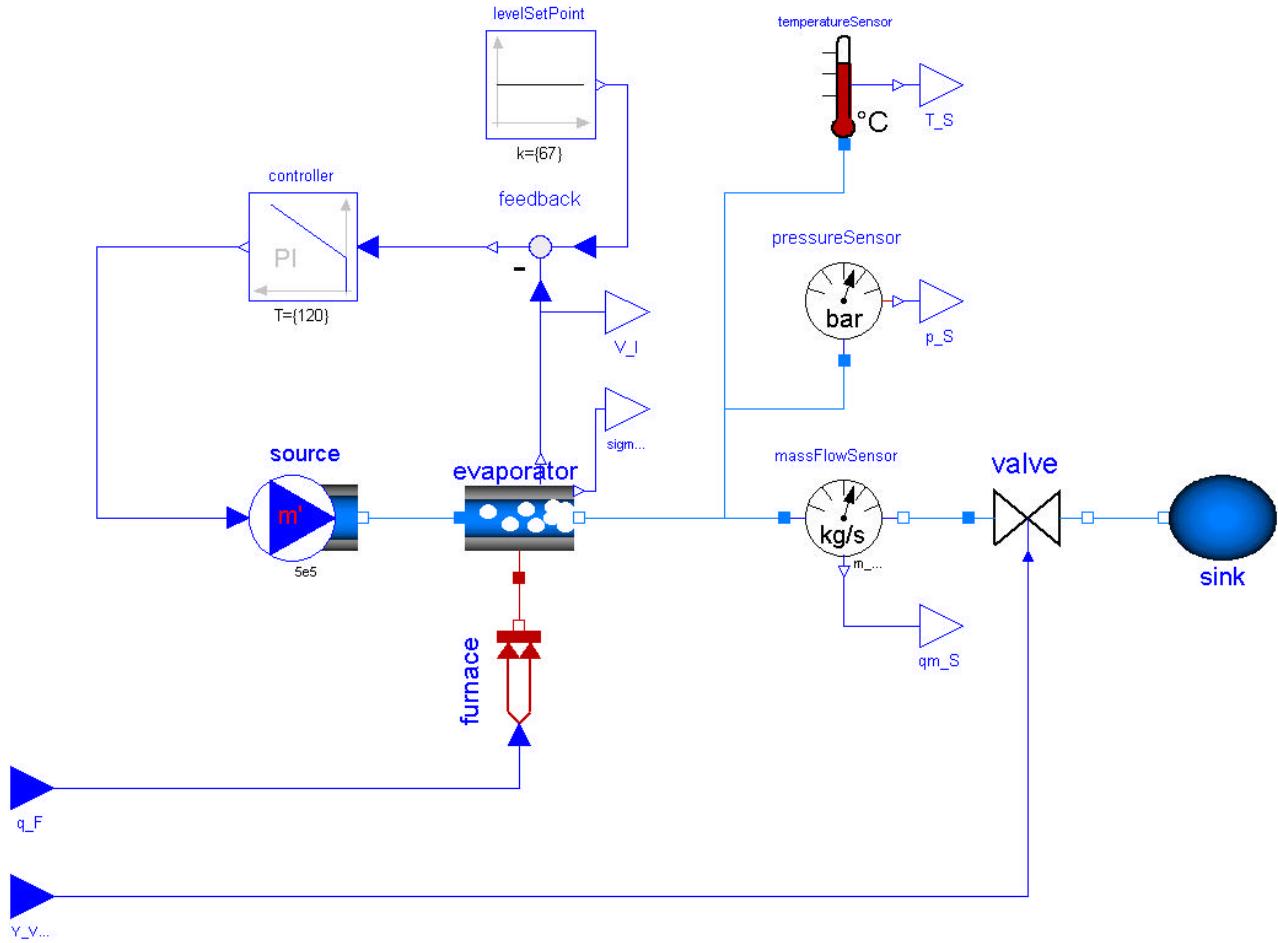


Figure 3: DrumBoiler system model in Dymola

ica.SIunits. Then the medium models and ports are declared together with parameters in a public section. The medium model defined above is instantiated twice: one for the liquid phase and one for the vapour phase. Internal model variables are declared in a protected section. Finally the model equations are stated and internal model variables are assigned to the public ports.

Note the use of semiLinear to define the energy flow through each port. This mechanism enables the treatment of reversible flow, see [4]. For example at port_a, either water with given total enthalpy port_a.e may enter, or liquid with enthalpy h_l may leave the evaporator.

A further important detail is the stateSelect attribute defined for pressure p and liquid volume V_l . This tells the translator to do the model transformation outlined in subsection 2.2.

Having the medium model and the evaporator component model ready, a complete system model is assembled in the third step, adding a feed water pump, a

valve at the steam outlet, sensors, and an ambient reference point. The composition of a system model is easiest done graphically. Figure 3 shows the complete drum boiler model assembled with Dymola.

The feed water flow needs to be controlled so that the water level inside the drum is kept at its set point. A PI control loop is added to the system model for this purpose. These additional component models are taken from the standard Modelica.Blocks library.

2.4 Model translation

Prior to numerical calculations, the Modelica model is translated to a mathematical system of differential-algebraic equations (DAE) and further transformed to a system of ordinary differential equations (ODE)

$$\dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t], \quad (16)$$

$$\mathbf{f} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \mapsto \mathbb{R}^{n_x},$$

$$\mathbf{y}(t) = \mathbf{g}[\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t], \quad (17)$$

$$\mathbf{g} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \mapsto \mathbb{R}^{n_y}.$$

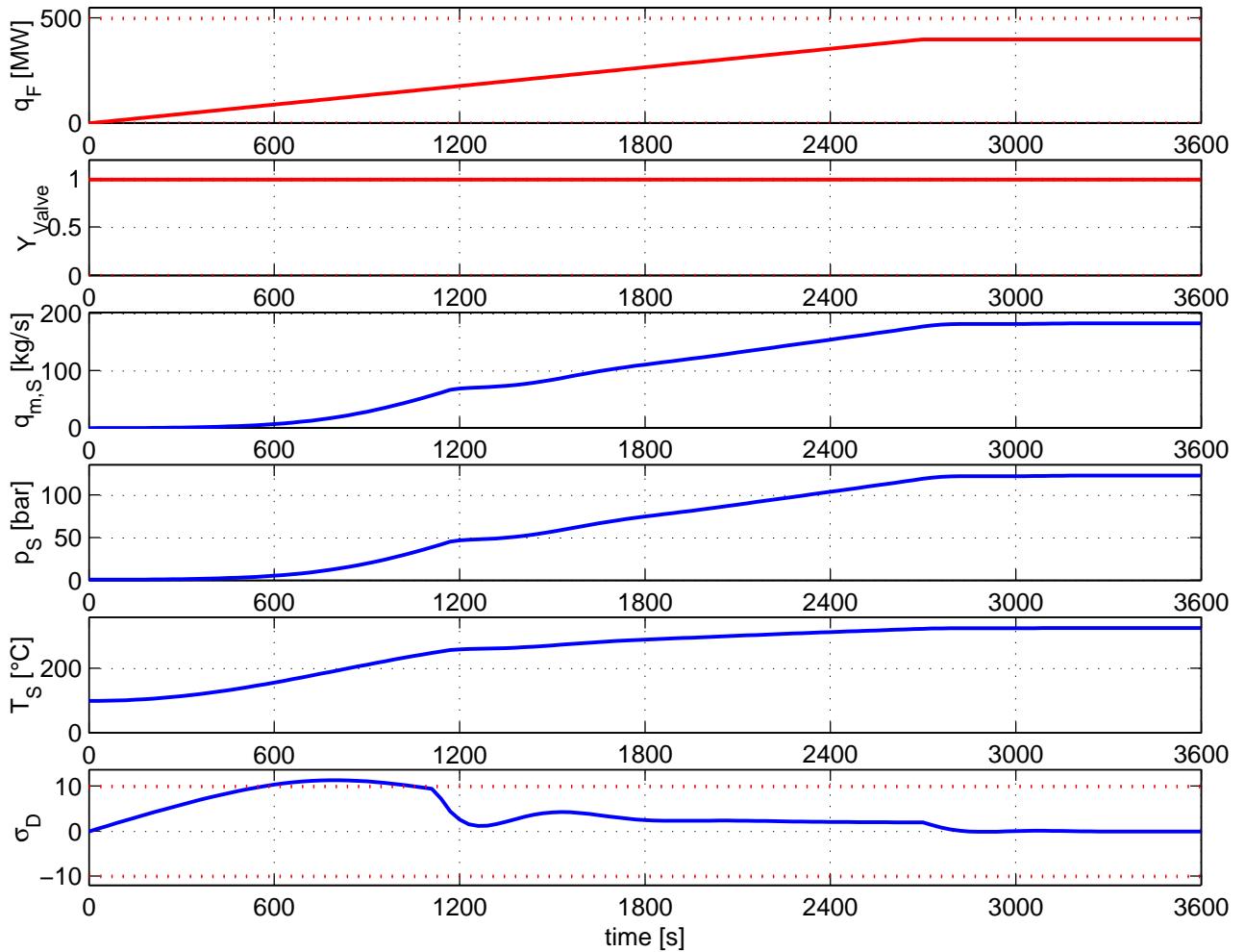


Figure 4: Simulation results applying a ramp to the heat flow and holding the valve open.

Model variables are internal continuous-time states $\mathbf{x} \in \mathbb{R}^{n_x}$, controlled inputs $\mathbf{u} \in \mathbb{R}^{n_u}$, constant parameters $\mathbf{p} \in \mathbb{R}^{n_p}$, and model outputs $\mathbf{y} \in \mathbb{R}^{n_y}$.

In the drum boiler example there are

$$\mathbf{u} = (q_F, Y_{\text{Valve}}) \quad (18)$$

$$\mathbf{x} = (V_l, p_S, x_{PI}) \quad (19)$$

$$\mathbf{y} = (T_S, p_S, q_m, S, V_l, \sigma_D) \quad (20)$$

with x_{PI} coming from the PI controller.

The model exhibits a nonlinear dynamics caused by material properties of water and steam, the large range of operation passed during startup, and the embedded control of the water level.

3 Boiler startup problem

During startup, a specified set point for steam temperature, pressure and mass flow rate shall be reached as

fast and efficient as possible, considering constraints on process variables. The most important constraints arise from thermal stress on thick-walled parts that are heated up, σ_D in the example treated here, cf. (6).

The boiler startup can be simulated for a given control strategy using e.g. Dymola [2] or Simulink [12]. Here the HQP optimization solver is applied, see next section, which provides initial-value simulation as a sub-functionality. Figure 4 shows simulation results when increasing the heat flow in the form of a ramp during 45 minutes and holding the valve at the steam outlet open. With this strategy, the startup takes about 45 minutes. The constraint on thermal stress is violated.

4 Optimal boiler startup

The boiler startup problem can be treated as optimal control problem minimizing an objective function subject to constraints.

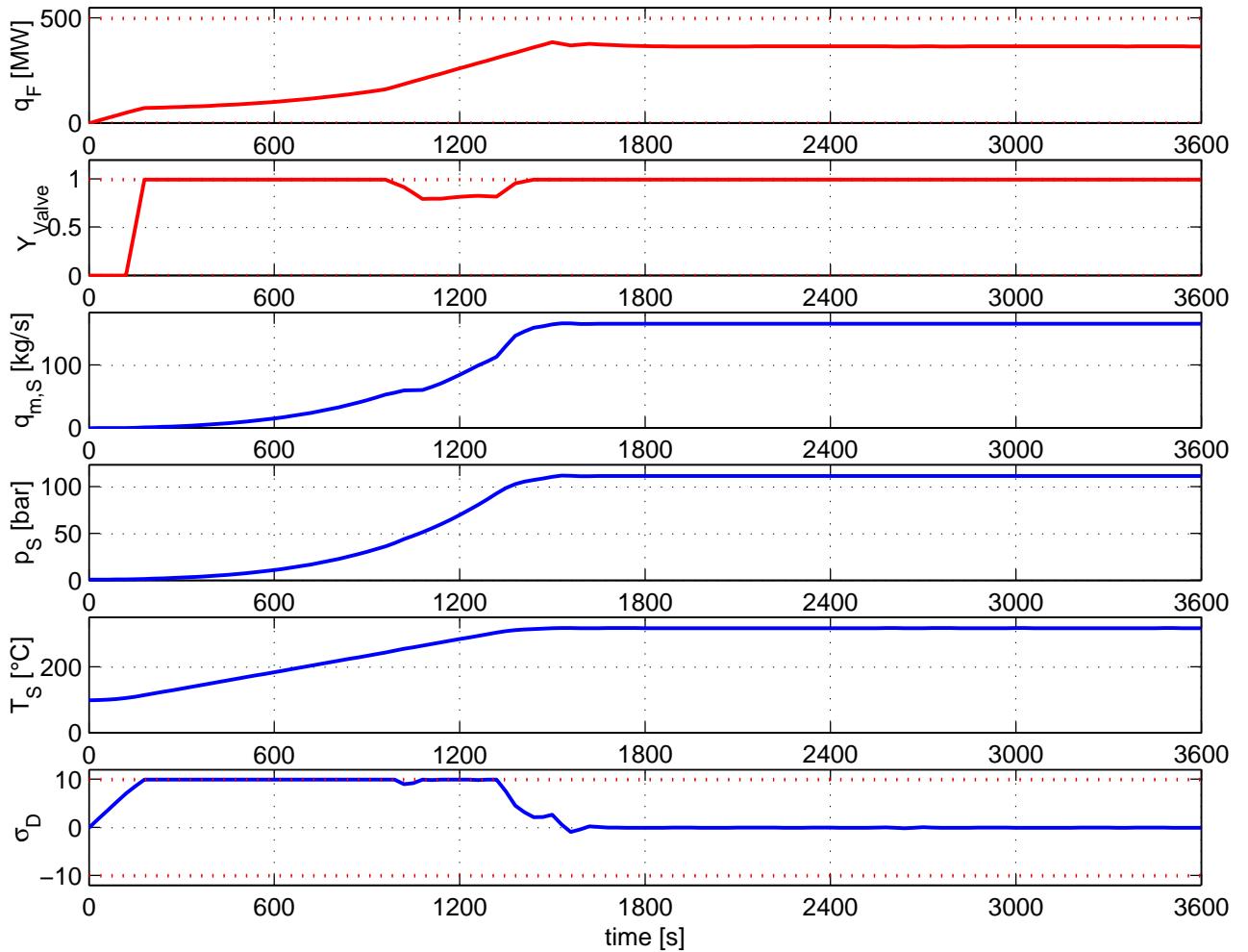


Figure 5: Optimal startup control minimizing the optimization objective subject to constraints.

In the example treated here, the objective is to minimize the deviation of generated steam pressure and mass flow rate from given reference points over the time horizon $[t_0, t_f]$

$$J = \int_{t=t_0}^{t_f} \mathbf{w}^T \left\{ \begin{bmatrix} p_S(t) - p_{ref} \\ [q_{m,S}(t) - q_{m,ref}]^2 \end{bmatrix}^2 \right\} dt \rightarrow \min_{\mathbf{u}(t)} \quad (21)$$

with the reference point being $p_{ref} = 110$ bar, $q_{m,ref} = 180$ kg/s and the vector of weighting terms $\mathbf{w} = (10^{-3}, 10^{-4})^T$. The objective shall be minimized subject to the system model (16).

Constraints that have to be fulfilled over the entire optimization horizon $t \in [t_0, t_f]$ can be divided into control input constraints and state or output constraints. Input constraints are the control bounds

$$0 \leq q_F \leq 500 \text{ MW} \quad (22)$$

$$0 \leq Y_{valve} \leq 1 \quad (23)$$

and the rate-of-change bound on the heat flow

$$-25 \text{ MW/min} \leq dq_F/dt \leq 25 \text{ MW/min} \quad (24)$$

The thermal stress is formulated as output constraint

$$-10 \leq \sigma_D \leq 10. \quad (25)$$

Figure 5 shows the solution of the optimal startup control problem. As a result of the dynamic optimization, the startup time can be reduced to less than 30 minutes, fulfilling all constraints. This is mainly due to better exploitation of allowed limits and the mutual dependence of multiple process variables. Especially the thermal stress limit is exploited during almost the complete startup. The valve position is used to fine control the startup, especially at times when the heat flow is limited by the rate-of-change bound (between 1000 and 1500 seconds).

5 Solver issues

The dynamic optimization problem was solved using the HQP code [7]. The optimizer accesses the same compiled model (16) as a simulator. In fact the model is loaded dynamically as Simulink S-function. During each optimization iteration, HQP solves the model differential equations and internally derived sensitivity equations, in order to evaluate optimized control input trajectories and to find directions for further improvement, respectively. The dynamic optimization problem is transformed to a non-linear optimization problem with the method of control vector parameterization. In total 121 parameters (free optimization variables) are introduced to describe the two control trajectories. 183 additional optimization variables are introduced for discrete-time states. The state constraint is evaluated at 120 sampling time points. The solution of the optimization problem takes 34 seconds on a PC Pentium IV, 1.8 GHz. It can be reduced down to 5 seconds applying a fixed step-size implicit integration rule to differential and sensitivity equations (cf. inline integration for real-time simulation [3]).

While accessing the same model as a simulation solver, an optimization solver generally has stronger requirements on a model. HQP implements a sparse Sequential Quadratic Programming algorithm that is of quasi Newton type and considered state-of-the art for large-scale non-linear optimization. The model needs to be smooth with respect to the optimization variables, allowing the determination of model sensitivities. This strongly limits the use of features like discrete events, reversible flows, and the like, that can be treated by simulation solvers without any problem.

Model sensitivities are obtained by integrating sensitivity equations together with model equations. The sensitivity equations are based on model Jacobians, see [6] for more details. That is why a model being translated for optimization should contain Jacobians, besides the compiled model equations. Note that the used Simulink S-function format supports Jacobians and that Dymola can generate them.

The Modelica.Fluid and Modelica.Media libraries were designed such that model Jacobians can be derived automatically by a Modelica tool. This is especially important for medium models accessing external functions that can not be differentiated automatically by a Modelica tool. An annotation syntax exists to refer to Jacobian information as available.

The automatic generation of model Jacobians does

not work for high-index DAE's and medium models providing first order derivatives only, as the example treated here. This is because the first order derivatives are already used for the transformed model, cf. subsection 2.2. Second order derivatives of the state dependent medium properties would be needed for the model Jacobian. Even higher order derivatives would be needed for a DAE index > 2. If a medium model is formulated using internal Modelica functions, like Water.IF97 used here, a Modelia tool might apply algorithmic differentiation to automatically obtain required derivatives [9].

6 On-line application

The method discussed in this paper is being applied to a 700 MW coal fired power plant. The model used there is significantly more rigorous, esp. with respect to the thermal stress models, see also [10]. It considers additional important components like the furnace, economizer, superheaters, headers, spray water injection and long pipes.

A number of new challenges arise in an on-line application, ranging from repeated update of the solution in a control loop, on-line identification of transient initial states and numerical robustness, through issues of the integration with the control system, supervision of the optimization and fall-back strategy, up to user interface design and acceptance by operating staff. It is not in the scope of this paper to discuss these issues. More details about implications on the optimization finds in [8]. More application specific information is given in [11].

7 Conclusions

This paper discusses the application of a drum boiler simulation model taken from the literature to dynamic optimization. The model is formulated in Modelica exploiting the new Modelica.Media and Modelica.Fluid base libraries. The mathematical model transformation performed automatically by the Dymola tool is outlined.

Based on the model, the optimal control problem is specified with an objective function and constraints. The optimal control problem is solved as large-scale non-linear optimization problem. For the example, the startup time can be reduced from 45 minutes to less

than 30 minutes, while constraint on thermal stress is fulfilled better.

The new Modelica.Fluid and Modelica.Media libraries allow the formulation of thermo-fluid models from a physical point of view. The object-oriented design supports a flexible design. Many mathematical details that traditionally had to be treated by a human modeler have been automated, making the modeling more efficient and allowing better re-use. Such details include the treatment of high-index DAEs, non-linear state-space transformations and the automatic detection of flow directions for multiple inter-connected fluid ports, including support for flow reversal.

The new libraries are applicable to on-line optimization. While making the job for human modelers easier, the libraries pose high requirements on a Modelica translator for the generation of efficient simulation code. Most important are the analytical treatment of connection equations and the elimination of common sub-expressions for multiple property evaluations at the same point, e.g. in inter-connected components. Dymola offers these features.

Reversible flows must be used carefully in optimization models as they are treated discontinuously, causing problems for the sensitivity analysis. A model allowing for reversible flow does not cause problems if flow does not revert, however, switching may also occur due to small numerical errors if flows are zero. For models with known unidirectional flows, one would like to be able to fix the direction in the model and to enforce correct results with optimization constraints instead. An other point that might be improved by tool vendors is the automatic differentiation of medium functions, as required for DAE index reduction and for the calcualtion of model sensitivities. Currently derivatives have to be provided explicitly together with medium functions in a model library.

References

- [1] K.J. Åström and R.D. Bell. Drum-boiler dynamics. *Automatica*, 36:363–378, 2000.
- [2] Dynasim AB. Dymola: Dynamic Modeling Laboratory. <http://www.dynasim.se>.
- [3] H. Elmquist, S.E. Mattsson, and H.Olsson. New methods for hardware-in-the-loop simulation of stiff models. In *Proceedings of the 2nd International Modelica Conference*. Oberpfaffenhofen, Germany, March 2002.
- [4] H. Elmquist, H. Tummescheit, and M. Otter. Modeling of thermo-fluid systems – Modelica.Media and Modelica.Fluid. In *Proceedings of the 3rd International Modelica Conference*. Linköping, Sweden, November 2003.
- [5] W. Wagner et. al. The IAPWS Industrial Formulation 1997 for the thermodynamic properties of water and steam. *Transactions of the ASME*, 122:150–182, 2000.
- [6] R. Franke. Formulation of dynamic optimization problems using Modelica and their efficient solution. In *Proceedings of the 2nd International Modelica Conference*. Oberpfaffenhofen, Germany, March 2002.
- [7] R. Franke, E. Arnold, and H. Linke. HQP: a solver for nonlinearly constrained large-scale optimization. <http://hqp.sourceforge.net>.
- [8] R. Franke, K. Krüger, and M. Rode. Nonlinear model predictive control for optimized startup of steam boilers. In *Proceedings of the GMA-Kongress 2003*. Baden-Baden, Germany, June 2003.
- [9] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, volume 19 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 2000.
- [10] K. Krüger, R. Franke, and M. Rode. Optimization of boiler startup using a nonlinear boiler model and hard constraints. In *Proceedings of the 15th International Conference on Energy, Costs, Optimization, Simulation and Environmental Impact of Energy Systems (ECOS 2002)*, volume II, pages 1310–1318. Berlin, Germany, July 2002.
- [11] M. Rode, R. Franke, and K. Krüger. Optimize IT model predicteve control for boiler start-up (BoilerMax). *ABB Review*, 3:30–36, 2003.
- [12] The MathWorks, Inc. Simulink: for model-based and system level design. <http://www.mathworks.com>.

Session 8A

Mechatronic Systems – II

How One Can Simulate Dynamics of Rolling Bodies via Dymola: Approach to Model Multibody System Dynamics Using Modelica

Ivan I. Kossenko and Maia S. Stavrovskaya

MSUS, Moscow Region, Russia, kosenko@ccas.ru

Abstract

An attempt to build more accurately the method to describe dynamics of multibody system (MBS) by means of Modelica is undertaken. In frame of the method under consideration can be simulated constraints of different types: holonomic/nonholonomic, scleronomic/rheonomic.

The model of a constraint allows to isolate mutually behavioral descriptions based on differential and algebraic equations correspondingly.

To illustrate an approach being applied the implementation of a constraint for bodies, rolling one relative to another is described. As an example the model of rattleback rolling on horizontal surface is investigated.

1 Preliminaries

A lot of methods to describe the structure of a multi-body system using different graph approaches are known. See for instance [1, 2, 3], and further references one can find there. Usually MBS is assumed to consist of rigid bodies. Note that in the frame of a bondgraph approach the background of an energy interchanges is used [4].

When implementing the MBS structural analysis based on a force interactions either oriented or nonoriented graphs are used in dependence of the problems to be resolved. Newton's laws [5] allow to describe dynamics within the so called Newton's viewpoint. In such a way the translational–rotational motion of each body is described by the system of Newton–Euler's ODEs. The graph structure is derived from an analysis of mutual interactions for bodies the system composed from. Such an interactions is caused mainly by constraints. But there are cases of physical fields also occurred. In general, Newton's third law of dynamics implies a dual nature of interactions between the

bodies.

Thus in a natural way from Newton's viewpoint the graph of an MBS structure is to be considered as a nonoriented one. In some particular cases the graph possesses special structure, and constraints are holonomic (i. e. integrable). Such situation occurs for instance in robotics where the structure is a tree. This fact used to reduce the source Newton–Euler system of ODEs with an attached subsystem of algebraic equations to some special kind of purely differential equations, for example of Lagrange ones. In this case natural approach assumes association of each dynamical ODE of the second order with the object comprising usually joint corresponding to a generalized coordinate, and an appropriate generalized force. Both usual linear force and torque of a couple can be in use. This force mainly is control one. It arises due to drive located at a joint. The solution of a kind previously described has been used in current Modelica MBS library of classes. For instance one can find such an approach in Revolute model where application of d'Alembert's principle relative to the revolution axis in behavioral section is equivalent to use of one second order ODE from Lagrange's equations for the whole holonomic mechanical system.

In general case the situation is more complicated, especially if nonholonomic (i. e. nonintegrable) constraints are used. Modeler has to take into account equations of constraints of algebraic, or even transcendental type. Fortunately today there exists background to build the models mentioned above, in particular: algorithms [6], modeling languages [7], and compilers [8]. To describe the models of an MBS we start from: (a) object-oriented paradigm [9] on one hand, and (b) so called physical principles of modeling [10] on the other one. Note that in our case of MBS dynamics one can consider in a natural way the rigid body as a main physical entity of the problems to be simulated.

1.1 Formal Description

Consider an MBS consisting of $m + 1$ bodies B_0, \dots, B_m . Represent it as a set $\mathcal{B} = \{B_0, \dots, B_m\}$. Here B_0 is assumed to be a base body. We suppose B_0 to be connected with an inertial frame of reference, or to have a known motion with respect to the inertial frame of reference. For example one can imagine the base body as a rotating platform, or as a vehicle performing its motion according to a given law.

Some bodies are considered as connected by mechanical constraints. But in general it is not necessarily. Suppose all constraints compose the set $\mathcal{C} = \{C_1, \dots, C_n\}$. We include in our considerations constraints of the following types: holonomic/nonholonomic, scleronomous/rheonomic. The latter properties mean the constraints having stationary/nonstationary parameters. For example one could consider the surface moving according to a prescribed law as a rheonomic constraint.

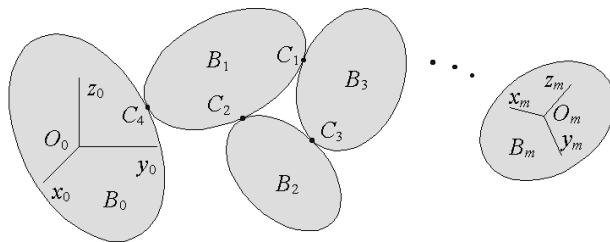


Figure 1.1: Multibody System

Thus one can uniquely represent a structure of the MBS via a nonoriented graph $G = (\mathcal{B}, \mathcal{C}, \mathcal{I})$. Here $\mathcal{I} \subset \mathcal{C} \times \mathcal{B}$ is an incidence relation setting in a correspondence the vertex incident to every edge $C_i \in \mathcal{C}$ of the graph. According to physical reasons it is easy to see that for any mechanical constraint C_i there exist exactly two bodies $B_k, B_l \in \mathcal{B}$ connected by this constraint.

The incidence relation generates an adjacency relation $\mathcal{S} \subset \mathcal{B}^2$ on the set of vertices. In our case this relation has the properties: (a) antireflexiveness: a body isn't be connected with itself; (b) symmetry: because of the graph is nonoriented if $(B_k, B_l) \in \mathcal{S}$, then $(B_l, B_k) \in \mathcal{S}$.

1.2 Architecture of Bodies Mutual Interactions in MBS

It is clear that consideration of the graph G does not provide a structural information sufficient for the MBS

dynamics description. Indeed, in addition to the force interaction represented usually by wrenches between bodies B_k, B_l through the constraint C_i there exist kinematical conditions specific for different kinds of constraints. In turn wrenches themselves can be represented by constraint forces and constraint torques couples. These forces and couples are connected by virtue of Newton's third law of dynamics.

Thus if the system of ODEs for translational–rotational motion can be associated with the object of a model corresponding to rigid body, then the system of the algebraic equations can be naturally associated with the object of a model corresponding to constraint. Note that according to consideration fulfilled above the set of algebraic equations comprises relations for constraint forces, torques of couples, and kinematical relations depending on kind of constraints. For such approach the differential and algebraic equations are said encapsulated in behavioral sections of the mentioned objects.

Thus all the “population” of any MBS model is reduced to objects of two classes: “Body” (objects B_0, \dots, B_m), “Constraint” (objects C_1, \dots, C_n). According to this approach simulating of the whole system behavior is reduced to permanent informational interaction between the objects of two considered types. Within the frame of Newton's laws of dynamics one can construct the MBS as a communicative network for this interaction. In this case the objects of bodies “feel” the action of other ones through corresponding objects of constraints.

Physical interactions are conducted in models due to objects splitted also in two classes of ports: “Wrench Port”, “Kinematical Port”. The first one is to be used to transfer vectors of force, and torque of couple. In addition, “Wrench Port” has to be used for transferring the information about a current location of the point constraint force acts upon.

Remark 1.1 *In our idealized model the force interaction between bodies is realized at a geometric point. Its coordinates are fed outside constraint object through “Wrench Port” permanently in time.*

“Kinematical Port” is to be used to transfer the data of rigid body kinematics: configuration (position of center of mass, orientation), velocity (velocity of the center of mass, angular rate), and acceleration (acceleration of the center of mass, angular acceleration). Objects of classes “Body” and “Constraint” work as

it is represented in Figure 1.2. The certain duality in a behavior of these objects can be easily observed.

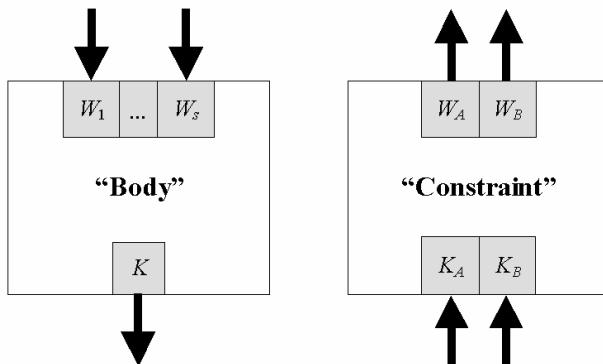


Figure 1.2: Typical Objects of MBS

Indeed, when getting force information through the ports W_1, \dots, W_s from the incident objects of a class “Constraint” the object of a class “Body” simultaneously generates, due to an integrator, kinematical information feeding outside through the port K . On the other hand every object of a class “Constraint” gets kinematical data from the objects corresponding to bodies connected by the constraint under consideration through its two “input” ports K_A, K_B . Simultaneously using the system of algebraic equations this object generates information concerning forces and torques of couples, and transmits the data to “output” ports W_A, W_B for the further transfer to objects of bodies under constraint.

According to classification of communicative ports performed above one can similarly classify the connectors used as “cables” for two purposes: (a) to transfer information about forces and torques; (b) to transfer the kinematical information.

Now it is possible to describe the architecture of information interactions within the particular constraint C_i corresponding to an individual edge of graph G , see Figure 1.3.

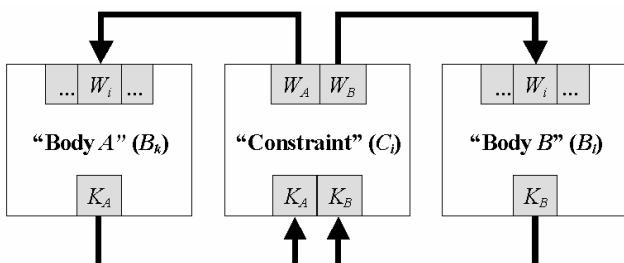


Figure 1.3: Architecture of Constraint

One should consider all connectors used above as bidirected ones. Arrows in Figure 1.3 are used to show the semantics of interactions. It’s clear that the whole construction considered above is a virtual one. Constructing the model the compiler extracts all equations from the objects and assembles them composing the DAE system optimized for a numeric integrator.

As usual physical fields one can implement by applying of **inner** and **outer** specifications.

2 Rolling of Rigid Bodies

It turned out that the attempts to treat problems of nonholonomic mechanics within existing MultiBody classes library are not effective. Indeed, this library has been developed mainly for modeling of controlled motion in mechatronics and theory of machines and mechanisms. The case of rolling bodies, typical for nonholonomic mechanics can’t be inserted in the formalism of joints and cuts in bodies. Here the position of a point of interaction between the bodies depends on dynamics of MBS.

Moreover, the situation turns being even more complicated if the friction of different kinds is taken into account, because relative sliding of bodies, unilateral motions, and impacts are assumed being allowed. To describe the dynamics of phenomena enumerated above one can apply well-known methods of classical mechanics staying simultaneously on positions of physical objects modeling. We mean differential equations of translational–rotational motion for interacting bodies, known as Newton–Euler’s equations.

2.1 General Description

As an example for formal approach discussed above let us consider the problem on description of one body contiguous to another one. Such approach can be used by designer in order to avoid derivation of dynamic equations both for holonomic and nonholonomic mechanical systems. In the second case problem itself may be complicated enough. Note that traditional cuts, flanges, or joints as constraint interfaces seem to be impossible for use in the situation under consideration. In addition, one should take into account a useful property of mutual isolation of differential and algebraic equations encapsulated in the classes of types “Body” and “Constraint” correspondingly.

Consider a local fragment of a mechanical system, and suppose that this fragment consists of a pair of rigid bodies rolling one upon another. Then a general schema in Figure 1.3 is satisfied. Suppose that all force and kinematical data to be transferred through ports are represented in a unified way: relative to the base frame $O_0x_0y_0z_0$ of a reference connected with the body B_0 . Wrench port consists of three arrays $\mathbf{P}, \mathbf{F}, \mathbf{M} \in \mathbf{R}^3$. Here \mathbf{P} is an array of coordinates for the point of a contact between two bodies under constraint, \mathbf{F} is a constraint force vector, \mathbf{M} is a constraint couple torque vector. In general case components of the array \mathbf{P} are computed in the object of a constraint. The vectors \mathbf{F} , \mathbf{M} are assumed expressing the “action” of constraint object to body object, of course in a virtual sense. The class to transfer force information can read:

```
connector WrenchPort
  SI.Position P[3];
  SI.Force F[3];
  SI.Torque M[3];
end WrenchPort;
```

The kinematical port consists of six arrays: $\mathbf{r}, \mathbf{v}, \mathbf{a}, \omega, \epsilon \in \mathbf{R}^3$, $T \in SO(3)$. The array \mathbf{r} corresponds to the radius–vector of the mass center of the body, \mathbf{v} corresponds to the velocity of this point, \mathbf{a} corresponds to its acceleration. T is an orthogonal matrix of a current body orientation. The columns of the matrix T consist of projections of unit vectors of the orthonormal base connected with a moving body into the axes of the base body frame. The class “Kinematical Port” in Modelica can be defined as:

```
connector KinematicPort
  SI.Position r[3];
  SI.Velocity v[3];
  SI.Acceleration a[3];
  SI.Real T[3,3];
  SI.AngularVelocity omega[3];
  SI.AngularAcceleration epsilon[3];
end KinematicPort;
```

All the objects of a class “Constraint” must have classes–inheritors as subtypes of a corresponding superclass. According to Newton’s third law this superclass must contain the equations of the form

$$\mathbf{F}_A + \mathbf{F}_B = \mathbf{0}, \quad \mathbf{M}_A + \mathbf{M}_B = \mathbf{0}. \quad (2.1)$$

in its behavioral section. Here arrays \mathbf{F}_A , \mathbf{M}_A and \mathbf{F}_B , \mathbf{M}_B represent constraint forces and torques “acting in

directions” of bodies A and B correspondingly. Kinematical equations for different types of constraints are to be added to equations (2.1) in different classes–inheritors corresponding to these particular types of constraints.

Properties (2.1) usually conducted through **flow**–variables are implemented here in a natural way in the behavioral section of the base superclass for mechanical constraints. They read:

```
partial model Constraint
  KinematicPort InPortA;
  WrenchPort OutPortA;
  KinematicPort InPortB;
  WrenchPort OutPortB;
equation
  OutPortA.F + OutPortB.F = {0,0,0};
  OutPortA.M + OutPortB.M = {0,0,0};
end Constraint;
```

Remark 2.1 *Model developer can create classes of complicated types of constraints such that equations (2.1) are not satisfied. For example such a constraint one can imagine as a thread thrown over the pulley, see Figure 2.1. It is clear that this constraint can be decomposed to components in such a way that the equations (2.1) are satisfied for each elementary constraint. However in applications it is often suitable to deal with constraints of a complex, combined type directly.*

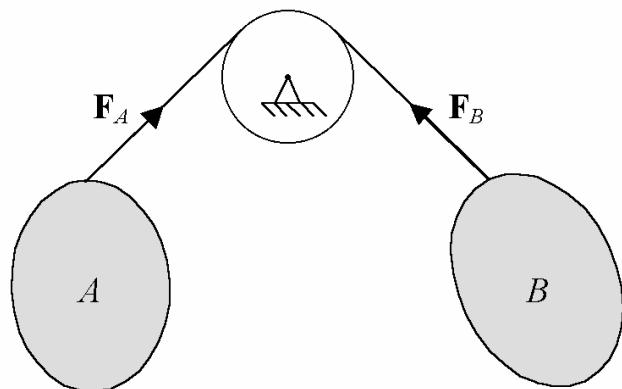


Figure 2.1: Example to Remark 2.1

Now start to construct behavioral equations for the object simulating of a constraint of the rolling type, see Figure 2.2. First of all let us describe the system of equations defining the position of a contact point. A constraint object has to “know”, i. e. to encapsulate

inside itself the equations for contiguous surfaces

$$f_A(x_k, y_k, z_k) = 0, \quad f_B(x_l, y_l, z_l) = 0.$$

Here all equations are defined with respect to the frame fixed in a corresponding body. Suppose that for the instant t these surfaces are described by equations

$$g_A(x_0, y_0, z_0) = 0, \quad g_B(x_0, y_0, z_0) = 0$$

written in the base frame. Here the expressions for the functions g_A and g_B can be easily obtained as

$$\begin{aligned} g_A(\mathbf{r}_0) &= f_A \left[T_k^{-1} (\mathbf{r}_0 - \mathbf{r}_{O_k}) \right], \\ g_B(\mathbf{r}_0) &= f_B \left[T_l^{-1} (\mathbf{r}_0 - \mathbf{r}_{O_l}) \right]. \end{aligned}$$

Here for sake of brevity we denote $\mathbf{r}_0 = (x_0, y_0, z_0)^T$. The vectors $\mathbf{r}_{O_k}, \mathbf{r}_{O_l}$ determine mass center positions for the connected bodies. All radius vectors $\mathbf{r}_0, \mathbf{r}_{O_k}, \mathbf{r}_{O_l}$ are assumed being represented in the base frame. Note that computation of matrices inverse to orthogonal ones is reduced simply to matrix transposition.

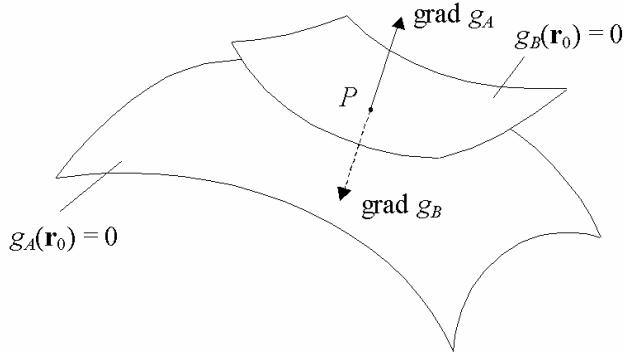


Figure 2.2: Vicinity of Contact Point

When rolling, the surfaces touch each other at the point P which is to be found, see Figure 2.2. The necessary condition of tangency reads

$$\text{grad } g_A = \lambda \cdot \text{grad } g_B. \quad (2.2)$$

Here λ is a scalar factor playing a role of additional auxiliary variable. In general position the system (2.2) defines uniquely a curve consisting of points in which the surfaces

$$g_A(\mathbf{r}_0) = \alpha, \quad g_B(\mathbf{r}_0) = \beta$$

have common tangent planes. One can consider λ as a coordinate on the curve. In general position such a curve intersects the surface

$$g_A(\mathbf{r}_0) = 0, \quad \text{or} \quad g_B(\mathbf{r}_0) = 0 \quad (2.3)$$

transversally. Thus the system of equations to find a contact point can be composed using (2.2) and (2.3). Here one has four scalar equations and four unknown variables: x_0, y_0, z_0, λ .

To complete the process of constructing the class “Roll”, an inheritor of a superclass “Constraint” one should append the condition of absence of sliding at the point of a contact

$$\mathbf{v}_k + [\boldsymbol{\omega}_k, \mathbf{r}_0 - \mathbf{r}_{O_k}] = \mathbf{v}_l + [\boldsymbol{\omega}_l, \mathbf{r}_0 - \mathbf{r}_{O_l}] \quad (2.4)$$

to a system of equations (2.2), (2.3). Here vectors $\mathbf{v}_k, \boldsymbol{\omega}_k, \mathbf{v}_l, \boldsymbol{\omega}_l$ denote linear velocities of mass centers, and angular velocities for the bodies subjected to constraint. Both sides of (2.4) are obtained from the rigid body kinematics [5]. Corresponding inheritor has the following Modelica code:

```
partial model Roll
  extends Constraint;
  SI.Position r[3];
equation
  InPortA.v +
  cross(InPortA.omega,r - InPortA.r) =
  InPortB.v +
  cross(InPortB.omega,r - InPortB.r);
  OutPortA.P = r;
  OutPortB.P = r;
  OutPortB.M = {0,0,0};
end Roll;
```

2.2 Dynamics of Rattleback

Further consider the simplified model of a rigid bodies rolling, namely dynamics of the rattleback on an immovable horizontal surface [11]. In this case the base body is supposed being fixed. Its boundary is a fixed horizontal plane which is considered as a surface for rolling. This body plays a role of the “Body A”, see Figure 2.3. Thus it is the same body as above denoted by B_0 . To describe its dynamics one need not the differential equations. All kinematical variables are zero-valued vectors. Matrix of orientation T_0 is an identity one.

Note objects of base bodies play a special role to describe the motion according to a predefined law. Corresponding classes have no any differential equations in their behavior. One can write down superclass of base body in the form:

partial model BaseBody

```
KinematicPort OutPort;
end BaseBody;
```

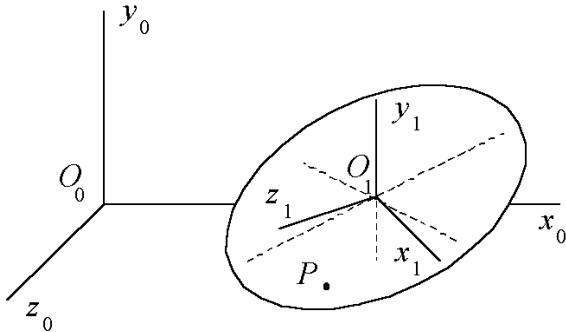


Figure 2.3: Rattleback on Horizontal Surface

Since body B_0 is supposed being fixed in the inertial frame then the class-inheritor can be described as:

```
model Base
  extends BaseBody;
  VisualShape Plane(
    r0={0,0,0},
    Shape="box",
    LengthDirection={0,-1,0},
    WidthDirection={1,0,0},
    Length=0.1,
    Width=10,
    Height=10,
    Material={0,0,1,0});
  WrenchPort InPortRoll;
equation
  OutPort.r = {0,0,0};
  OutPort.v = {0,0,0};
  OutPort.a = {0,0,0};
  OutPort.T = [1, 0, 0;
               0, 1, 0;
               0, 0, 1];
  OutPort.omega = {0,0,0};
  OutPort.epsilon = {0,0,0};
end Base;
```

The rattleback B_1 plays a role of the “Body B”. Number of moving bodies in the MBS is $m = 1$. In superclass “Body”, named in our package as `RigidBody` dynamics of rigid body is described here by means of Newton’s differential equations for the body mass center, and by Euler’s differential equations for rotational motion. The Euler equations are constructed using quaternion algebra [12] in a following way

$$\frac{d\mathbf{q}}{dt} = \frac{1}{2}\mathbf{q} \circ \begin{bmatrix} 0 \\ \Omega_1 \\ \Omega_2 \\ \Omega_3 \end{bmatrix}, \quad I \frac{d\Omega}{dt} + [\Omega, I\Omega] = \mathbf{N}.$$

Here first equation is kinematical one, and the second equation is one for dynamics of rigid body. Quaternion $\mathbf{q} = (q_1, q_2, q_3, q_4)^T \in \mathbf{H} \simeq \mathbf{R}^4$ uniquely defines rotational matrix R ; the quaternion algebra \mathbf{H} is considered as a linear space \mathbf{R}^4 ; the binary operation \circ denotes quaternion multiplication. The matrix of inertia tensor I , the vector of angular velocity $\Omega = (\Omega_1, \Omega_2, \Omega_3)^T \in \mathbf{R}^3$, and vector of total torque \mathbf{N} are considered with respect to central principal axes of inertia of the body. Hence before exporting of kinematical data from the object of class `RigidBody` one must perform the conversion $\omega = R\Omega$. Similarly total torque after importing and before using inside behavioral section also is to be transformed $\mathbf{M} \mapsto \mathbf{N}$ according to the formula $\mathbf{N} = R^T \mathbf{M}$.

Usually the rattleback, or wobblestone, or Celtic stone is assumed being rigid body bounded by paraboloidal or ellipsoidal surface. This body is assumed possessing a central principal axes of inertia which are not collinear to body’s axes of symmetry. Consider the case of an ellipsoidal surface.

Suppose that the central principal moments of inertia for the moving body read

$$I_{x_1 x_1} = 2, \quad I_{y_1 y_1} = 3, \quad I_{z_1 z_1} = 1.$$

Fix also its mass value $\mu = 1$. To be definite one can suppose all physical measures based for instance on SI units. Then the rattleback model can be represented in the form:

```
model RollingBody
  extends RigidBody;
  outer Real[3] Gravity;
  // Ellipsoid semi-diameters
  outer SI.Length a1;
  outer SI.Length b1;
  outer SI.Length c1;
  VisualShape Body(
    r0=0,0,-c1,
    Shape="sphere",
    LengthDirection=0,0,1,
    WidthDirection=1,0,0,
    Length=2*c1,
    Width=2*a1,
    Height=2*b1,
    Material=1,0,0,1);
  SI.Energy E; // Full energy
  SI.Energy K1;// Kinetic energy
  // of translations
  SI.Energy K2;// Kinetic energy
  // of rotations
  SI.Energy P; // Potential energy
  WrenchPort InPortRoll;
```

```

equation
  F = m*Gravity + InPortRoll.F;
  M = InPortRoll.M +
    cross(InPortRoll.P - r,
           InPortRoll.F);
  Body.S = T;
  Body.r = r;
  K1 = 0.5*m*v*v;
  K2 = 0.5*omega*I*omega;
  P = -m*r*Gravity;
  E = K1 + K2 + P;
end RollingBody;

```

Now consider the building of a constraint. Since $f_A(x_0, y_0, z_0) \equiv y_0$ then $\text{grad } f_A = (0, 1, 0)^T$. A bounding surface for the body B is assumed to be of the ellipsoidal shape having the following semi-diameters $a_1 = 2$, $b_1 = 1$, $c_1 = 3$. The matrix of the corresponding quadratic form relative to principal axes of the ellipsoid reads

$$B_1 = \begin{pmatrix} a_1^{-2} & 0 & 0 \\ 0 & a_2^{-2} & 0 \\ 0 & 0 & a_3^{-2} \end{pmatrix}.$$

Suppose that the second principal axis directed along the axis O_1y_1 of the ellipsoid of inertia is coincident to the axis of the body surface. outer shape directed identically. Further, let us turn the second ellipsoid relative to the first one about O_1y_1 by an angle $\delta = \pi/10$. Then the matrix of a quadratic form for rolling ellipsoid relative to central principal-axis system has the form

$$B = RB_1R^T, \quad R = \begin{pmatrix} \cos\delta & 0 & \sin\delta \\ 0 & 1 & 0 \\ -\sin\delta & 0 & \cos\delta \end{pmatrix}.$$

The equations defining the position $\mathbf{r}_0 = \mathbf{r}_P$ of a contact point P read

$$\begin{aligned} (\text{grad } f_A, \mathbf{r}_P) &= 0, \\ \text{grad } f_A &= \lambda \cdot (T_1 \cdot B \cdot T_1^{-1}) (\mathbf{r}_0 - \mathbf{r}_{O_1}). \end{aligned} \quad (2.5)$$

According to previous considerations the system (2.5) comprises four scalar equations and four unknown values: x_P, y_P, z_P, λ . First equation can be easily reduced to the simple form $y_P = 0$. Finally, class-inheritor for the constraint under consideration takes the form:

```

model Ellipsoid_on_Plane
  extends Roll;
  outer SI.Length a1;

```

```

  outer SI.Length b1;
  outer SI.Length c1;
  outer SI.Angle delta;
  parameter Real R[3,3]=
    [cos(delta), 0, sin(delta);
     0, 1, 0;
     -sin(delta), 0, cos(delta)];
  parameter Real B1[3, 3]=
    [1/a1^2, 0, 0;
     0, 1/b1^2, 0;
     0, 0, 1/c1^2];
  parameter Real B[3,3]=
    R*B1*transpose(R);
  parameter Real n[3]=0,1,0;
  parameter SI.Length d=0;
  Real lambda;
equation
  n*r = d;
  n = lambda*
    InPortB.T*B*
    transpose(InPortB.T)*
    (r - InPortB.r);
end Ellipsoid_on_Plane;

```

Now we can compose the testbench model for simulation of dynamics of the rattleback as:

```

model Test
  parameter SI.Acceleration g=9.81;
  inner parameter SI.Acceleration[3]
    Gravity={0,-g,0};
  inner parameter SI.Length a1=2;
  inner parameter SI.Length b1=1;
  inner parameter SI.Length c1=3;
  inner parameter SI.Angle delta=
    Modelica.Constants.pi/10;
  Base Basel1;
  RollingBody RollingBody1(
    q(start={1,0,0,0}), // Initial
    // quaternion
    r(start={0,1,0}),
    I=[2, 0, 0; 0, 3, 0; 0, 0, 1],
    v(start={0.05,0,0}),
    omega(start={0,-1,-0.05}));
  Ellipsoid_on_Plane
  Ellipsoid_on_Plane1;
equation
  connect(Basel1.InPortRoll,
            Ellipsoid_on_Plane1.OutPortA);
  connect(Basel1.OutPort,
            Ellipsoid_on_Plane1.InPortA);
  connect(Ellipsoid_on_Plane1.InPortB,
            RollingBody1.OutPort);
  connect(Ellipsoid_on_Plane1.OutPortB,
            RollingBody1.InPortRoll);
end Test;

```

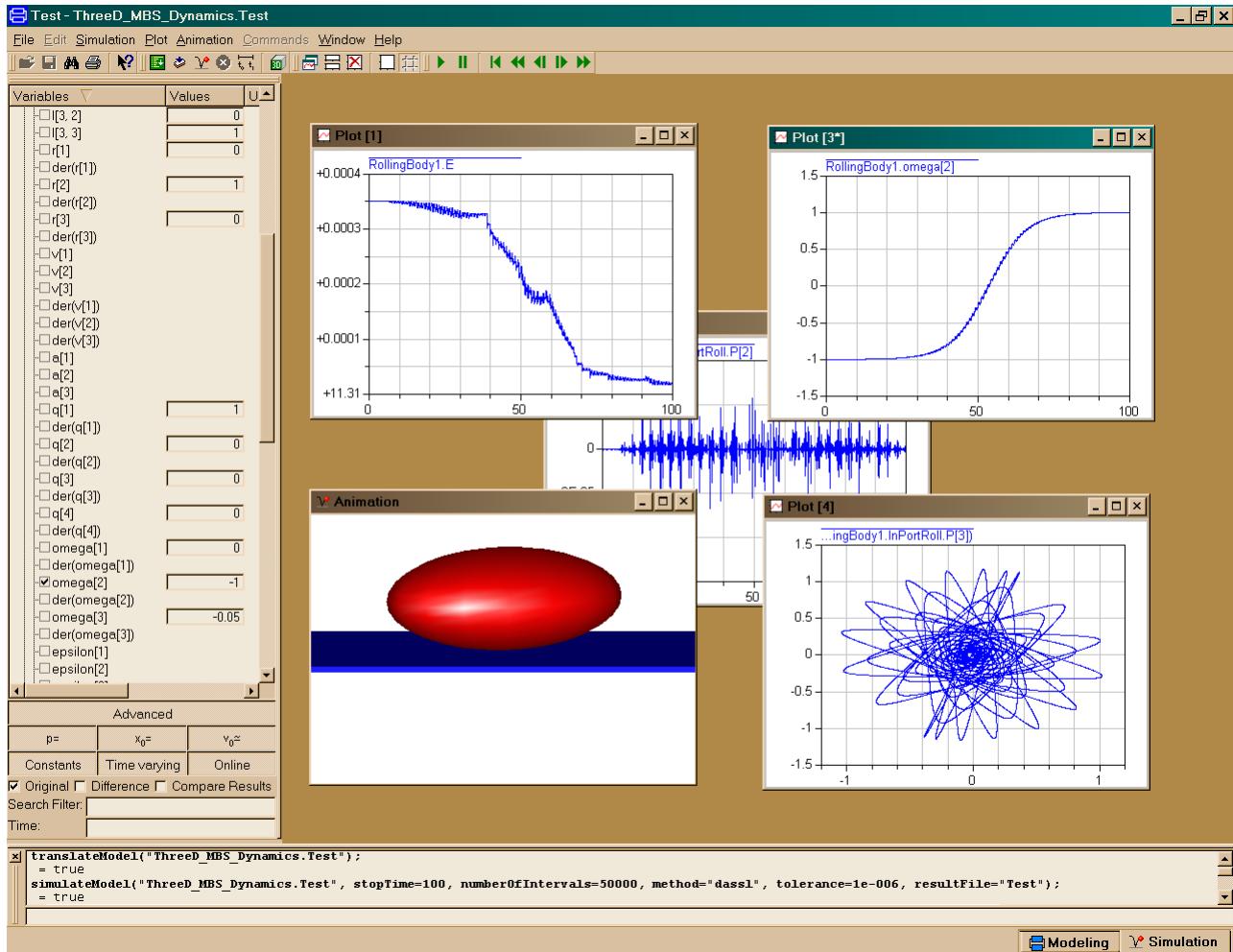


Figure 2.4: General View of Simulation Results

The model described above has been developed using Modelica language as a package. The high quality of an approximation for the rattleback motions has been verified through different simulations performed. For one of the model runs general view of simulation results is shown in Figure 2.4. Initial conditions are defined in a following way

$$\begin{aligned} \mathbf{r}_1(0) &= \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, & \mathbf{v}_1(0) &= \begin{pmatrix} 0.05 \\ 0 \\ 0 \end{pmatrix}, \\ T_1(0) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, & \omega_1(0) &= \begin{pmatrix} 0 \\ -1 \\ -0.05 \end{pmatrix}. \end{aligned}$$

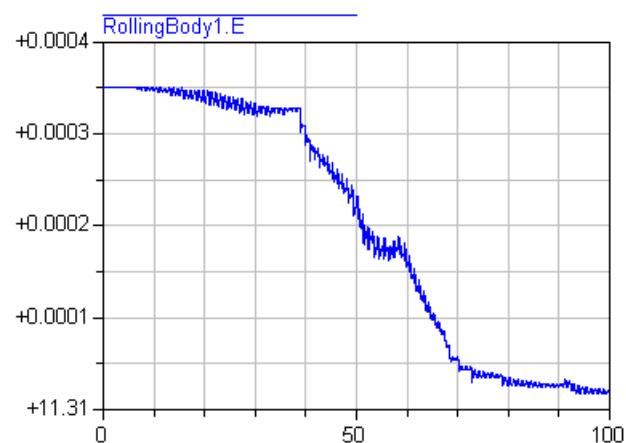


Figure 2.5: Preservation of Energy

Integral of energy value was under monitoring see Figure 2.5. One can observe for this value an extremely slow drift: height of the whole plot equals to 0.0004 while base value is equal to 11.31 units of energy. Obviously such a drift is caused by computational errors.

Trajectory of a contact point in plane z_0x_0 , see Figure 2.6 was also under monitoring. The constraint is satisfied with high accuracy permanently for all instants of simulation time. Indeed, such an accuracy can be investigated using variable y_P from the equa-

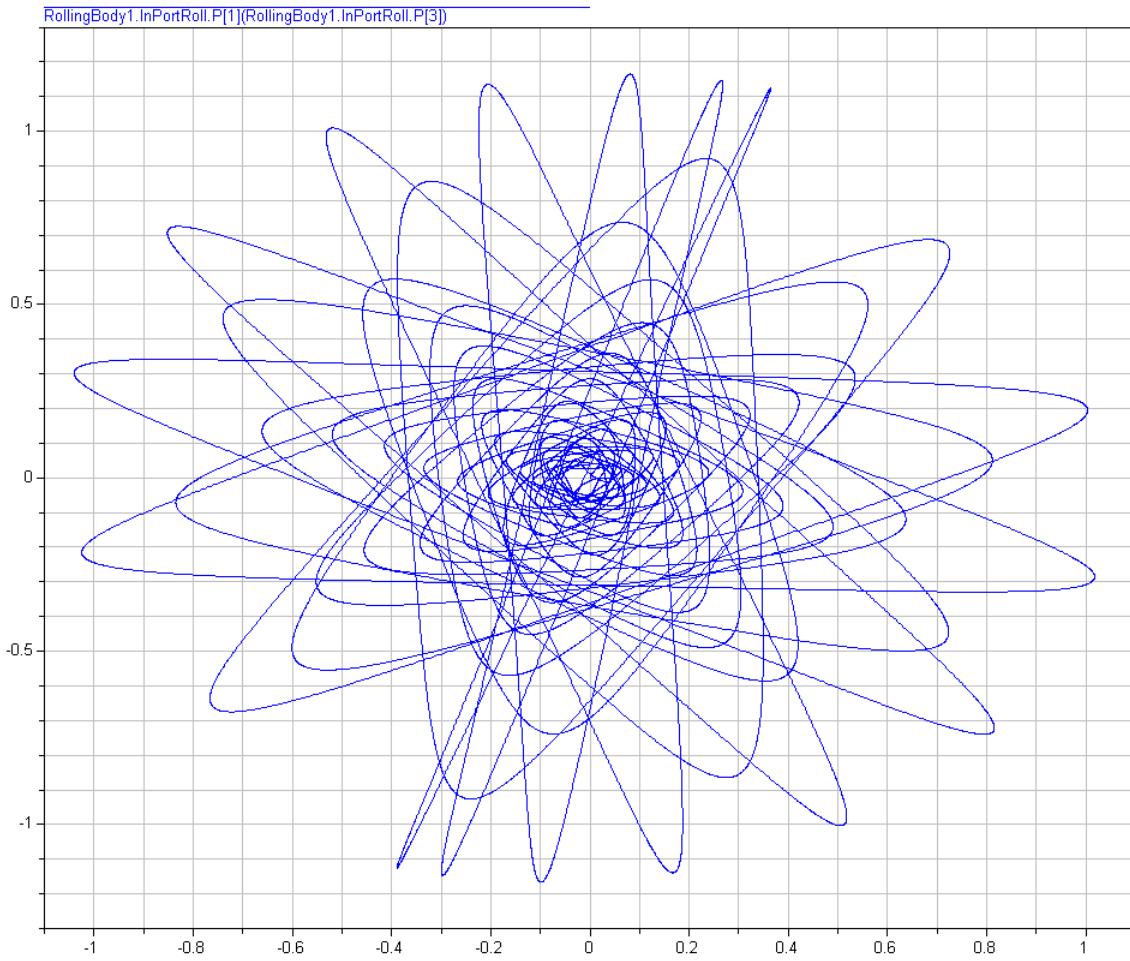


Figure 2.6: Contact Point Trajectory

tions (2.5). In Figure 2.7 we see that the function $y_P(t)$ performs only noisy oscillations almost vanishing near its zero value. Here height of the plot equals to 10^{-24} of length unit.

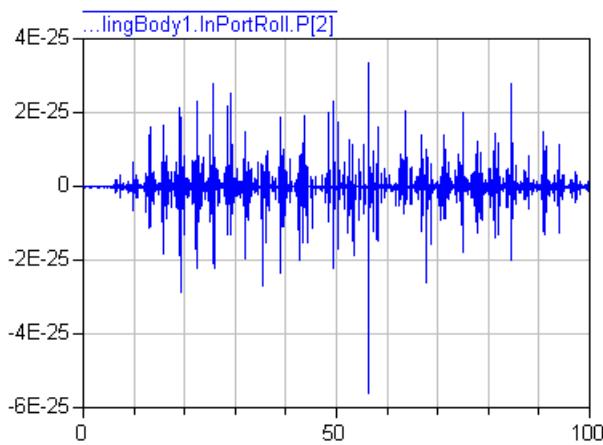


Figure 2.7: Preservation of Constraint Accuracy

Due to high quality of a numeric model one can easily

observe the known dynamical properties of the rattleback. These latter cause in particular change of direction of angular velocity vector corresponding to rotation about central principal axis O_1y_1 of Celtic stone, see Figure 2.8. Initially axis O_1y_1 is directed vertically downwards, and rattleback rotates clockwise. Initial direction of the angular velocity slightly deviates from the local vertical. Then when time passes value of $t = 50$ units vertical component of angular velocity passes through its zero value, and one can observe wobbling motions of the rattleback. One instant of such wobbling is fixed in Figure 2.9. Note that the total energy is a constant because the mechanical system under consideration is conservative one. Then rotation becomes almost permanent but now counterclockwise. It is easy to see (in Figure 2.8) that the angular velocity projection onto inertial axis y_0 is scaled from -1 to 1 during time of simulation. But its value undergoes slight oscillations of several frequencies.

Using visual environment of Dymola one can also easily build 3D-animation of the rattleback rolling on

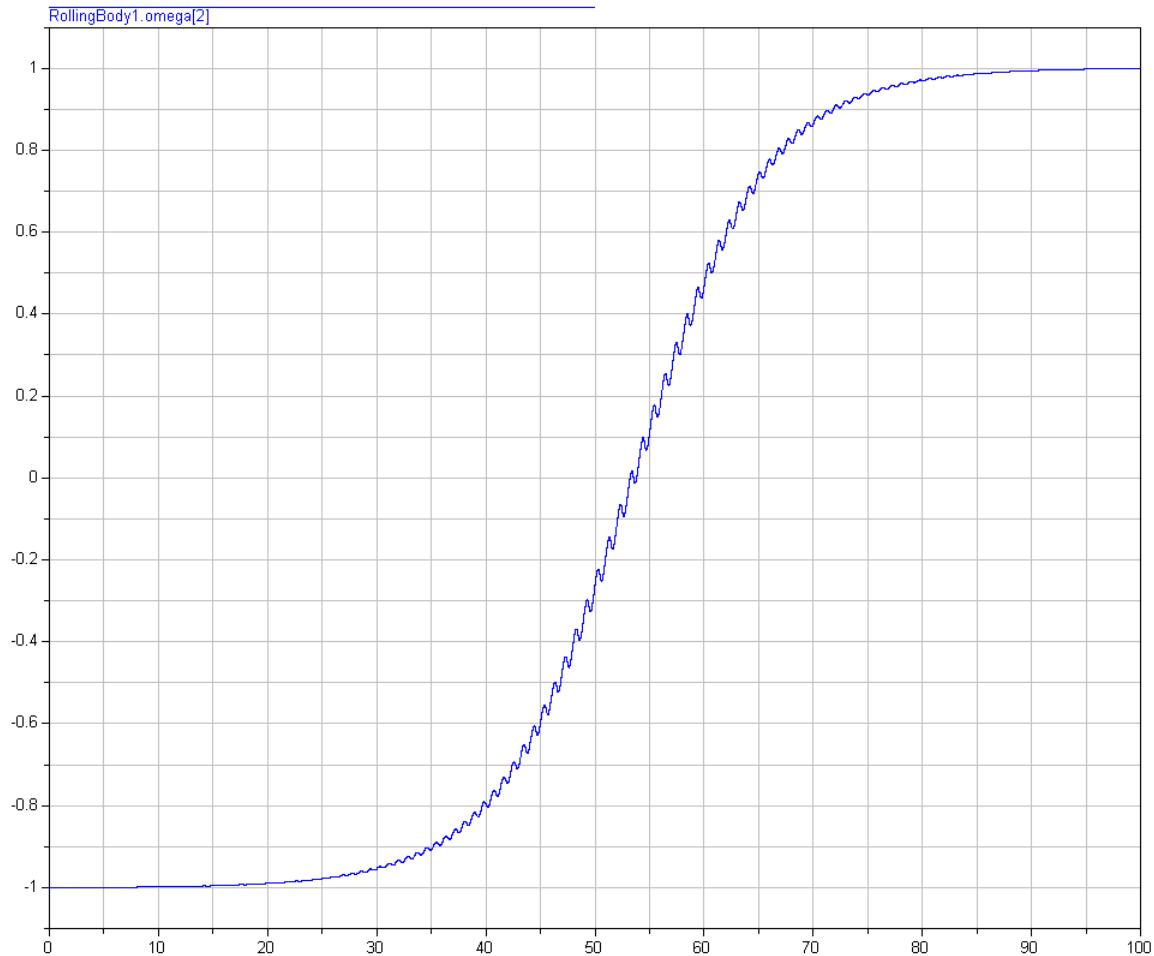


Figure 2.8: Behavior of Vertical Component of Angular Rate

a plane, see for example Figure 2.9.

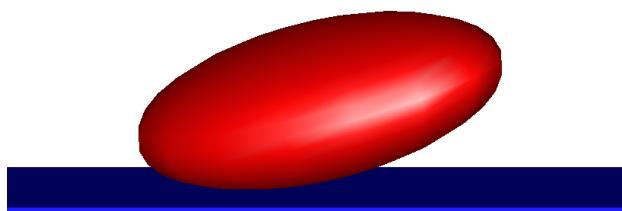


Figure 2.9: Instant Shot at a Moment When Stone Is Prepared to Change Orientation of Its Rotation

model easily complicated problems of MBS dynamics. Among them: (a) dynamics of systems with sliding subjected to friction of various kinds; (b) dynamics of systems subjected to unilateral constraints with impacts [13]. In both cases to realize models one can apply Modelica's `when` clause in behavioral section. Then different cases of sliding and friction correspond to different cases of equations for forces and torques. For instance in simplest cases numerical models of tops rolling/sliding on surfaces can be investigated as exercises. Note that modeling of dynamics with unilateral constraints is also convenient for Modelica use because of its facilities for events processing. In all cases we deal with dynamics arranged as a piecewise smooth motion.

It should be also interesting to construct realistic model of dynamics for the truck rolling on a road of different surface quality in different weather conditions, and a model of dynamics for a heap consisting of contacting bodies, etc.

Returning to the problem considered above note that

3 Directions of Further Development

Development of the Modelica code similar to one presented above opens a wide range of possibilities to

the model also simplifies qualitative dynamical analysis for long time simulations. An existence of such structures in phase space like attractors in dynamics of the rattleback [14] can be demonstrated.

4 Conclusion

Computations corresponding to case of Kane and Levinson have been performed. Results of simulations are identical in all details. Moreover, no special differential equations for dynamics of nonholonomic systems have been used! This is a real way to achieve a unified approach to modeling of both holonomic and nonholonomic MBS. Compiler itself incorporates implicitly the use of equations of motion for nonholonomic mechanical system in its algorithm. Thus the known problem of ODEs derivation for such systems sometimes nontrivial and difficult seems to be overcome in automatic mode, at least for problems with algebraic (nontrancendental) constraints, and for cases of so called general position.

5 Acknowledgement

The paper was prepared with partial support of Russian Foundation for Basic Research, grants 02–01–00196, SS–2000.2003.1.

References

- [1] Wittenburg, J., Dynamics of Systems of Rigid Bodies. — Stuttgart: B. G. Teubner, 1977.
- [2] von Schwerin, R., Multi Body System SIMulation. Numerical Methods, Algorithms, and Software. Lecture Notes in Computational Science and Engineering, Vol. 7. — Berlin Heidelberg: Springer, 1999.
- [3] Sinha, R., Paredis, C. J. J., Liang, V-C., and Khosla, P. K., Modeling and Simulation Methods for Design of Engineering Systems. // Journal of Computing and Information Science in Engineering, 2001, Vol. 1, Iss. 1, pp. 84–91.
- [4] Mukherjee, A., Karmakar, R., Modelling and Simulation of Engineering Systems through Bondgraphs. — Alpha Science International Ltd., 2000.
- [5] Routh, E. J., A Treatise on the Dynamics of a System of Rigid Bodies. — London: Vol. 1, 1897; Vol. 2, 1905.
- [6] Ascher, U. M., Petzold, L. R., Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. — SIAM, 1998.
- [7] Modelica — A Unified Object-Oriented Language for Physical Systems Modeling. Tutorial. — Modelica Association, 2000.
- [8] Dymola. Dynamic Modeling Laboratory. User's Manual. Version 5.0a — Lund: Dynasim AB, Research Park Ideon, 2002.
- [9] Booch, G., Object-Oriented Analysis and Design with Applications. — Addison-Wesley Longman Inc. 1994.
- [10] Cellier, F. E., Elmquist, H., Otter, M., Modeling from Physical Principles. // in: Levine, W. S. (Ed.), The Control Handbook. — Boca Raton, FL: CRC Press, 1996. — pp. 99–107.
- [11] Kane, T. R., Levinson, D. A., Realistic Mathematical Modeling of the Rattleback. // International Journal of Non-Linear Mechanics, 1982, Vol. 17, Iss. 3, pp. 175–186.
- [12] Kosenko, I. I., Integration of the Equations of the Rotational Motion of a Rigid Body in the Quaternion Algebra. The Euler Case. // Journal of Applied Mathematics and Mechanics, 1998, Vol. 62, Iss. 2, pp. 193–200.
- [13] Pfeiffer, F., Unilateral Multibody Dynamics. // Meccanica, 1999, Vol. 34, No. 6, pp. 437–451.
- [14] Borisov, A. V., and Mamaev, I. S., Strange Attractors in Rattleback Dynamics // Physics–Uspekhi, 2003, Vol. 46, No. 4, pp. 393–403.

The New Modelica MultiBody Library

Martin Otter¹, Hilding Elmquist², and Sven Erik Mattsson³

¹DLR, Oberpfaffenhofen, Germany, Martin.Otter@dlr.de

²Dynasim AB, Lund, Sweden, Elmquist@dynasim.se

³Dynasim AB, Lund, Sweden, SvenErik@dynasim.se

<http://www.robotic.dlr.de/Martin.Otter> and <http://www.dynasim.se>

Abstract

A new Modelica library for the modeling and simulation of 3-dimensional mechanical systems has been developed. It will be freely available in the Modelica standard library. Furthermore, the Dymola simulation environment has been considerably enhanced to support the needed features. The MultiBody library is first presented from a user's point of view. Furthermore, all essential details of the implementation are described. The library includes features that are usually not available in other multi-body software, such as analytic handling of a large class of kinematical loops, or the arbitrary connection feature of objects. For example, series connection of 3D line force components is possible.

1 Introduction

The *MultiBody* library is a free Modelica package providing 3-dimensional mechanical components to conveniently model mechanical systems, such as robots, mechanisms, or vehicles. It will be accessible as Modelica.Mechanics.MultiBody and is a replacement of the Modelica library ModelicaAdditions.MultiBody which has been used for a long time. The main design goal of the library and of the supporting features in Dymola [7] was that standard applications can be carried out in a convenient way without knowledge of the Modelica language. The MultiBody library has the following important features:

- Components can be connected together in a nearly arbitrary fashion. If kinematical loop structures occur, they are automatically handled in an efficient way by a new technique explained in section 5. Also force components can be connected directly together, a feature that is usually not available in other multi-body software.
- The non-linear equations occurring in kinematical loops are solved *analytically*, i.e., in a robust and efficient way, for a large class of mechanisms, such as a 4 bar and slider-crank mechanism, or a MacPherson suspension by

constructing such loops with elements from the MultiBody.Joints.Assemblies sub package.

- Most joints and all bodies have potential states. A Modelica translator, such as Dymola, will use the generalized coordinates of joints as states if possible. If this is not possible, e.g., because bodies are moving freely in space, states are selected from body coordinates. An advanced user may select states manually from the "Advanced" menu of the corresponding components.
- Whenever a multi-body system model is constructed, all defined components are automatically visualized in an animation using appropriate default sizes and colors. This allows an easy visual check of the constructed model, without extra work of the modeler. Both, the complete animation as well as individual component animation can be switched off. In this case the equations defining animation are removed from the generated code.
- Annotations and assert statements have been introduced that provide in many cases warning or error messages that are related to the library components and not to specific equations as it is usual in Modelica libraries.

2 A First Example

In a first example it shall be demonstrated how to build up, simulate and animate a simple pendulum, consisting of a body and a revolute joint with linear damping in the joint. In Figure 1 the composition diagram of this model is shown. It uses components from the MultiBody library, see figure on next page. Every model utilizing the MultiBody library must

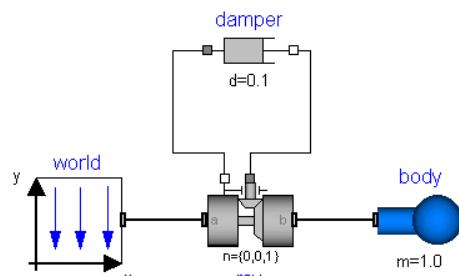


Figure 1. Composition diagram of pendulum



have an instance of the *MultiBody.World* model on top level. The reason is that in the world object the gravity field is defined (no gravity, uniform gravity or point gravity), as well as the default sizes of animation shapes and this information is reported to all used components. Joint "rev" is dragged from Joints.*ActuatedRevolute*, "body" from Parts.*Body* and the "damper" as 1-dimensional force element from "Modelica.Mechanics.Rotational.Damper". All components are connected together according to the physical connection structure. After translation, automatically the animation from Figure 2 is shown:

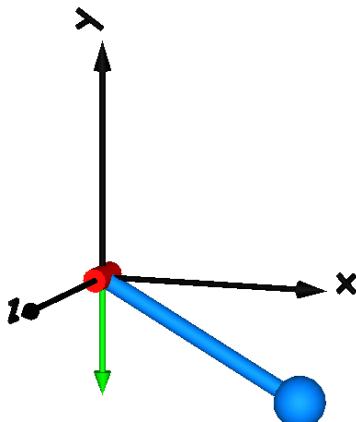


Figure 2. Automatic animation of pendulum

The coordinate system represents the world frame, the green arrow pointing in negative y-axis characterizes the direction of the gravity acceleration, the red cylinder in the world origin is directed along the axis of rotation of the revolute joint, and the light blue cylinder and sphere characterize the body (the center of the sphere is located in the center of mass of the body).

Before translation, the parameters of the dragged components need to be defined. Some parameters are vectors that have to be defined with respect to a local coordinate system of the corresponding component. A convenient way is often a definition of the multi-body model in a configuration where all local frames are parallel to the world frame. This is usually the case when all joint variables, such as the angle of a revolute joint, are zero. Since in such a reference configuration only one coordinate system is essential, the definition is easier as if n frames of n components would have to be taken into account. The reference configuration for the simple pendulum shall be defined in the following way: The y-axis of the world frame is directed upwards,

i.e., the opposite direction of the gravity acceleration. The revolute joint is placed in the origin of the world frame. The rotation axis of the revolute joint is directed along the z-axis of the world frame. The body is placed on the x-axis of the world frame (i.e., the rotation angle of the revolute joint is zero, when the body is on the x-axis). In the following figure, the Dymola menu to define the revolute joint according to this definition is shown:

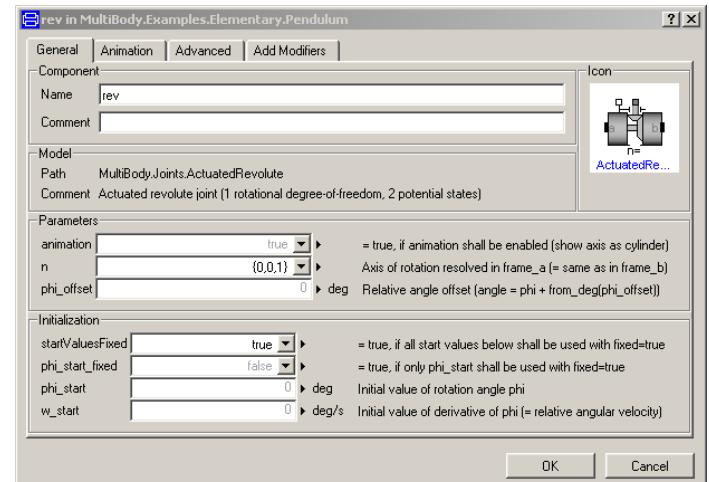


Figure 3. Dymola menu to define a revolute joint

The axis of rotation is defined as " $n = \{0, 0, 1\}$ " meaning that it is directed into the direction of the z-axis of the World coordinate system in the reference configuration. Accordingly, the body component is defined in Figure 4.

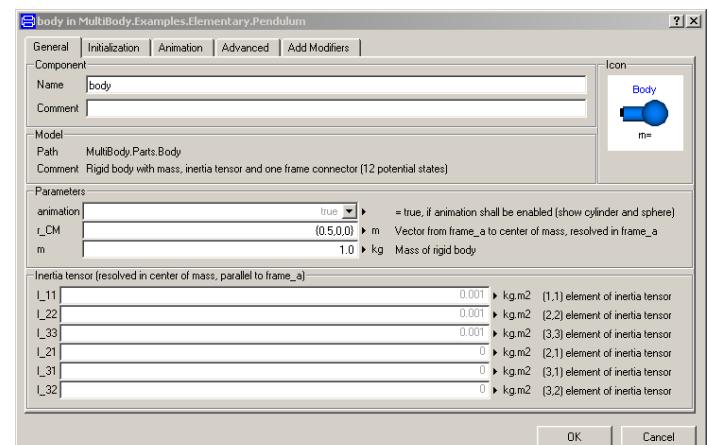


Figure 4. Dymola menu to define a body

The vector " r_{CM} " from the origin of the "left" coordinate system of the body called "frame_a" to the center of mass of the body is defined as " $r_{CM} = \{0.5, 0, 0\}$ ", meaning that it is directed 0.5 m along the x-axis of the world frame in the reference configuration. Note, for subsystems in a hierarchical model, e.g., a MacPherson suspension, it is also often convenient to use a local reference configuration for the vector definitions.

3 Describing Orientation

In mechanical systems many variables have to be described with respect to coordinate systems. The notation used in the MultiBody library for this purpose is discussed at hand of Figure 5.

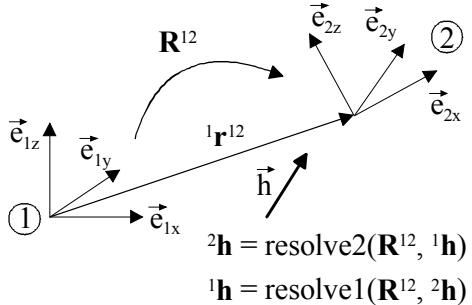


Figure 5. Notation for coordinate systems

For notational convenience the word “frame” is used in the sequel as a synonym for “coordinate system”. Frame 1 in Figure 5 is described by 3 unit vectors $\vec{e}_{1x}, \vec{e}_{1y}, \vec{e}_{1z}$ that are orthogonal to each other and Frame 2 is described in a similar fashion by unit vectors $\vec{e}_{2x}, \vec{e}_{2y}, \vec{e}_{2z}$. Frame 2 is defined relatively to frame 1 by the position vector ${}^1\mathbf{r}^{12}$ that is directed from the origin of frame 1 to the origin of frame 2 and is resolved in frame 1, i.e.,

$$\vec{r}^{12} = {}^1\mathbf{r}^{12} \cdot \vec{e}_1 = \{{}^1r_x^{12}, {}^1r_y^{12}, {}^1r_z^{12}\} \cdot \{\vec{e}_{1x}, \vec{e}_{1y}, \vec{e}_{1z}\}$$

Note, that ${}^1\mathbf{r}^{12}$ is a one-dimensional (Modelica) array that holds the 3 coordinates of vector \vec{r}^{12} with respect to frame 1. In the sequel, (Modelica) arrays with one or two dimensions are always characterized by bold face characters if the complete array is referenced.

The relative orientation of frame 2 with respect to frame 1 is defined by the “orientation object” \mathbf{R}^{12} (also called “rotation object”). There are different ways to mathematically describe orientation. To ease usage, the MultiBody library is designed such that knowledge about the actual description form of orientation is not necessary. This is achieved by providing a pre-defined type

`MultiBody.Frames.Orientation`

and utility functions in `MultiBody.Frames` operating on instances of this type. The two most important functions are shown in Figure 5: An arbitrary vector \vec{h} might be represented by its coordinates with respect to frame 1 (${}^1\mathbf{h}$) or with its coordinates with respect to frame 2 (${}^2\mathbf{h}$), respectively. If either of the two representations is given, the other one can be computed in the following way:

```
import MultiBody.Frames;
Frames.Orientation R12;
Real h1[3] "h resolved in frame 1"
Real h2[3] "h resolved in frame 2"
equation
  h2 = Frames.resolve2(R12, h1); //or
  h1 = Frames.resolve1(R12, h2);
```

There are about 30 of these utility functions in sub library `MultiBody.Frames`. We will explain some more of them when needed. Note, that with every orientation object a direction is associated. E.g., the inverse orientation R_{21} of R_{12} is computed by $"R_{21} = Frames.inverseRotation(R_{12})"$.

During the development of the MultiBody library, 3 different representation forms of the orientation object have been implemented:

1. Transformation matrix \mathbf{T} (${}^2\mathbf{h} = \mathbf{T}^{12} \cdot {}^1\mathbf{h}$).
2. Two rows of the transformation matrix.
3. Quaternions (see, e.g., [16]).

Benchmark tests revealed that the transformation matrix leads usually to the most efficient code and therefore this representation form was selected. Since in some situations quaternions are useful, the implemented functions operating on quaternions are provided in the MultiBody library under `MultiBody.Frames.Quaternions`. Also some quite involved functions are present, e.g., to compute quaternions from a transformation matrix in a numerically robust way (`Quaternions.from_T`).

Dymola has the built-in rule that functions with one statement are always “inlined” before they are used. Most of the utility functions in `MultiBody.Frames` are therefore defined just with one statement to enforce inlining, in order (a) to **not** have any function call overhead, (b) to allow symbolic rearrangement of terms and (c) that symbolic differentiation is possible. Other tools using the MultiBody library should also have support for inlining in order to get efficient code.

4 MultiBody Frame Connector

We are now in the position to present the design of the “Frame” connector that is used to connect multi-body components together. All variables used in this connector are displayed in Figure 6: A coordinate system “frame a” is rigidly fixed at an attachment point of a mechanical part. This Frame is described with respect to the world frame by the

- position vector ${}^0\mathbf{r}^{0a}$ that is directed from the origin of the world frame to the origin of frame a and is resolved in the world frame and by the
- orientation object \mathbf{R}^{0a} describing the relative orientation between the world frame and frame a.

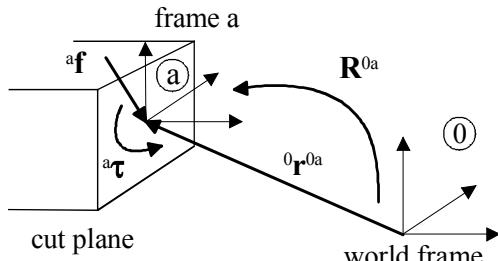


Figure 6. MultiBody “Frame” connector

It is assumed that a free body diagram is constructed, i.e. that a cut is performed between mechanical parts that shall be connected together at frame a. In the cut plane a resultant cut force ${}^a\mathbf{f}$ and a resultant cut torque ${}^a\tau$ act on frame a. Both vectors are resolved in this frame.

```
connector Frame
  import SI = Modelica.SIunits;
  SI.Position      r_0[3]    " =  {}^0\mathbf{r}^a ";
  Frames.Orientation R          " =  R^0a ";
  flow SI.Force     f[3]      " =  {}^a\mathbf{f} ";
  flow SI.Torque    t[3]      " =  {}^a\tau ";
end Frame;

connector Frame_a = Frame;
connector Frame_b = Frame;
```

The four previously defined variables are used in the connector. The additional connectors Frame_a and Frame_b have the identical definition as connector Frame. The only difference is that Frame_a and Frame_b have different icons in order to be able to distinguish Frame connectors more easily in a composition diagram.

The cut force and cut torque are **flow** variables in order that the force and torque balance at a point where several components are connected together is fulfilled. Note, that two connected frames (a and b) coincide, since $a.r_0 = b.r_0$ and $a.R = b.R$ due to the connection rules of Modelica.

The orientation between two frames can be described by 3 independent variables, see, e.g., [16][18]. Unfortunately, every such description form has a singularity and therefore cannot be used in a connector. For this reason, an orientation object has to be described by a set of **redundant** variables that are related to each other with constraint equations. In the MultiBody library the orientation object is described by a transformation matrix that has 9 entries, i.e., a highly redundant description form. This property leads to significant difficulties and is one of the reasons why it needed so long time to come up with a “truly” object-oriented multi-body library (E.g. the first Dymola multi-body library was developed in 1994 [17]).

In several components, such as a body or a sensor, velocities or accelerations of connector variables are needed. These derivatives can be easily obtained in the following way:

```
import SI = MultiBody.SIunits;
import MultiBody.Interfaces;
import MultiBody.Frames;
Interfaces.Frame_a   frame_a;
SI.Velocity           v_0[3];
SI.Acceleration        a_0[3];
SI.AngularVelocity    w_a[3];
SI.AngularAcceleration z_a[3];
equation
  v_0 = der(frame_a.r_0);
  a_0 = der(v);
  w_a = Frames.angularVelocity2(
    frame_a.R, der(frame_a.R));
  z_a = der(w_a);
```

As can be seen, the velocity v_0 and the acceleration a_0 of the origin of frame_a (resolved in the world frame) are simply computed by applying the derivative operator **der(..)**. The angular velocity of frame_a is computed with a function that requires as input the orientation object R and its derivative dR/dt and returns the angular velocity ${}^a\omega$ resolved in frame_a according to Poisson’s equation. With $R^T = [\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z]$, ${}^a\omega$ is computed as:

$${}^a\omega^a = \{\mathbf{e}_z^T \cdot \dot{\mathbf{e}}_y, -\mathbf{e}_z^T \cdot \dot{\mathbf{e}}_x, \mathbf{e}_y^T \cdot \dot{\mathbf{e}}_x\}$$

Applying the derivative operator **der(..)** on w_a results in the angular acceleration, resolved in frame_a, since according to Euler’s differentiation rule (${}^i d\vec{h} / dt = {}^k d\vec{h} / dt + \vec{\omega}^a \times \vec{h}$):

$$\begin{aligned} {}^0 d\vec{\omega}^a / dt &= {}^0 d\vec{\omega}^a / dt + \vec{\omega}^a \times \vec{\omega}^a \\ &= {}^0 d\vec{\omega}^a / dt \end{aligned}$$

where ${}^i d\vec{h} / dt$ is the derivative of vector \vec{h} with respect to coordinate system i and $\vec{\omega}^a$ is the absolute angular velocity of frame_a.

In books about multi-body systems it is usually recommended to compute the angular velocity by recursive calculations and it is claimed that this is much more efficient as using the direct application of Poisson’s equation as it is performed with function “angularVelocity2” above. For a “truly” object-oriented library it is difficult or not possible to apply a recursive calculation directly since in an object only relations between connector variables can be formulated. It turns out that the generated code of the MultiBody library is nearly as efficient as from the ModelicaAdditions.MultiBody library where the angular velocity is computed recursively. This is due to the particular implementation of Poisson’s equation and Dymola’s symbolic capabilities.

5 Overdetermined DAEs

By collecting together all explicit equations in a Modelica model and its submodels and all equations due to “connect” statements, a Modelica model is mapped to a DAE (= Differential Algebraic Equation system) of the following form:

$$\mathbf{0} = \mathbf{f}(dx/dt, \mathbf{x}, \mathbf{y}, t)$$

where \mathbf{x} contains all variables appearing differentiated and \mathbf{y} contains all pure algebraic variables. To get efficient code, this DAE has to be symbolically processed and transformed to state space form (at least numerically) with a subset of \mathbf{x} as states. This is performed by BLT partitioning [8] to get a sequential model evaluation and to identify algebraic loops, the Pantelides algorithm [19] to determine equations to be differentiated and the dummy derivative method [13] to select independent states (this method can be interpreted as a variant of the currently popular “projection methods” of higher index DAEs). All these algorithms require that $\dim(\mathbf{f}) = \dim(\mathbf{x}) + \dim(\mathbf{y})$, i.e., the number of equations has to be identical to the number of unknown variables.

Whenever the variables in a connector are **not** independent from each other, connection structures that have loops may result in a DAE where there are more equations as unknowns, i.e., $\dim(\mathbf{f}) > \dim(\mathbf{x}) + \dim(\mathbf{y})$. Usually, this overdetermined set of equations is still consistent, so that a unique mathematical solution exists. Since the Frame connector has an overdetermined set of variables due to the orientation object, also models of the MultiBody library may result in an overdetermined DAE.

It seems unlikely that the symbolic algorithms from above can be generalized to directly handle such DAEs, because it is not possible to distinguish consistently overdetermined DAEs from erroneous DAEs (that are a result of modeling errors), by pure structural information. For this reason, the only practical way seems to be to mark the overdetermined equation subset in the model and transform this set of equations before the standard algorithms from above are applied. One such way of marking and transforming an overdetermined set of equations has been designed for the next version 2.1 of the Modelica language and has been implemented in Dymola version 5.1. This approach is sketched in the rest of the section.

It is assumed that overdetermined DAEs are due to overdetermined sets of (non **flow**) variables \mathbf{v} in connectors. Such connectors will be called “overdetermined connectors” in the sequel. When

connecting two or more overdetermined connectors together, equality equations for corresponding overdetermined variable sets are generated, such as “ $\mathbf{v}_1 = \mathbf{v}_2$ ”. Whenever, say, \mathbf{v}_1 is computed in one component and then passed to the next component via a “connect” statement, everything is fine, because \mathbf{v}_2 is uniquely computed from \mathbf{v}_1 by “ $\mathbf{v}_2 := \mathbf{v}_1$ ”. Difficulties arise, if both $\mathbf{v}_1 = \mathbf{v}_1(\mathbf{x})$ and $\mathbf{v}_2 = \mathbf{v}_2(\mathbf{x})$ are computed from potential state variables \mathbf{x} , since a connection equation $\mathbf{v}_1 = \mathbf{v}_2$ imposes an overdetermined (but consistent) set of constraints on the variables \mathbf{x} .

The basic requirement is that the developer of an overdetermined connector provides a function called “`equalityConstraint(v1,v2)`” that returns a **non-redundant** set of residues that should be zero if the equality constraint $\mathbf{v}_1 = \mathbf{v}_2$ is fulfilled. In a pre-processing step of the model equations, a translator has then to decide for every connection set whether an equation of the form “ $\mathbf{v}_1 = \mathbf{v}_2$ ” or an equation of the form “ $0 = \text{equalityConstraint}(\mathbf{v}_1,\mathbf{v}_2)$ ” has to be added to the DAE. Let us demonstrate this by considering the Frame connector.

Modelica is enhanced such that a type or record declaration may optionally contain a definition of function “`equalityConstraint(...)`”:

```
type Orientation
  extends Real[3,3];

function equalityConstraint
  input Orientation R1;
  input Orientation R2;
  output Real residue[3];
protected
  Orientation R_rel;
algorithm
  R_rel = R2*transpose(R1);
  residue := {R_rel[2,3],
              R_rel[3,1],
              R_rel[1,2]};
end equalityConstraint;
end Orientation;
```

An orientation object is defined by a transformation matrix of dimension [3,3]. Two orientation objects, i.e., transformation matrices, \mathbf{R}_1 and \mathbf{R}_2 are identical ($\mathbf{R}_1 = \mathbf{R}_2$) if the relative transformation matrix between \mathbf{R}_1 and \mathbf{R}_2 , i.e., $\mathbf{R}_{\text{rel}} = \mathbf{R}_2 \cdot \mathbf{R}_1^T$ is the unit matrix. A transformation matrix describing a small rotation can be approximated by (see, e.g., [18])

$$\mathbf{R}_{\text{rel}} \approx \begin{bmatrix} 1 & \varphi_3 & -\varphi_2 \\ -\varphi_3 & 1 & \varphi_1 \\ \varphi_2 & -\varphi_1 & 1 \end{bmatrix}$$

where $\varphi_1, \varphi_2, \varphi_3$ are a set of 3 **independent** variables describing the deviation from the unit matrix. As a result, if the outer diagonal elements [2,3], [3,1] and

[1,2] of \mathbf{R}_{rel} vanish, then $\mathbf{R}_1 = \mathbf{R}_2$. Therefore, these 3 outer diagonal elements are returned as residues by function `equalityConstraint(...)`. To summarize, a connection between two Frame connectors will either result in 9 equations $\mathbf{R}_1 = \mathbf{R}_2$ to define the equality between two orientation objects or in 3 equations by calling function `equalityConstraint(...)`. If appropriately selected, the result is a regular DAE where the number of equations is identical to the number of unknowns. A call to function `equalityConstraint(...)` will usually result in a nonlinear system of equations that has only the desired solution $\mathbf{R}_1 = \mathbf{R}_2$, if the initial guess values of the iteration variables are close enough to this solution.

The remaining open question is how a tool can decide which connection equations to use? An informal description is given below. Details of the algorithm are sketched in the appendix.

A new package called “Connections” is introduced in Modelica, containing a set of built-in operators to mark overdetermined equations. Let us sketch these operators using the orientation object R as an example:

- **`root(A.R)`** defines that the orientation object R in connector A is computed in a consistent way. The world object has such a definition because R is defined as identity matrix.
- **`branch(A.R, B.R)`** defines that there is an algebraic relationship between the orientation object A.R in connector A and the orientation object B.R in connector B. Joint objects have such a definition, if there is an algebraic constraint between frame_a.R and frame_b.R.

These two operators are already sufficient, since a tool can determine whether the graph constructed with `root(...)`, `connect(...)` and `branch(...)` statements contains loops. These loops have to be cut and for every cut the `equalityConstraint(...)` function has to be used to state the equality of orientation objects.

If there is a free flying body, coordinates of the body should be used as states from which the orientation object in the body connector can be computed. This in turn means that a free flying body is also a root in the graph. Formally, this situation is defined by operators:

- **`potentialRoot(A.R)`** defines that the orientation object R in connector A might be computed in a consistent way, if this is necessary. Body objects have such a definition.
- **`isRoot(A.R)`** returns true if the orientation object A.R has been selected as a root. This means that different equations have to be provided.

The sketched method to handle overdetermined DAEs with symbolic transformation techniques is not specific to multi-body systems. For example,

efficient implementations of electric power systems use the Park transformation to define currents and voltages in the connector **relatively** to the harmonic, high-frequency signal of a power source that is described by the angle of the rotor of the source. This allows much faster simulations, since the basic high frequency signal of the power source is not part of the differential equations. On the other hand, the source angle has to be included into the connector leading to an overdetermined description that can be handled with the method presented in this section.

6 Elementary Components

Using the “Frame” connector and the utility functions in `MultiBody.Frames`, it is straightforward to implement the elementary components that are usually available in multi-body programs.

The `MultiBody` library has about 40 components. The most important ones are shown in Table 1. Contrary to approaches described in text books about this topic, equations are **only** defined on “position” level. A tool has enough information to figure out via the Pantelides algorithm [19] which equations have to be differentiated in order to transform the DAE to state space form with the dynamic dummy derivative method [13][14]. This feature simplifies the implementation and the understanding of the `MultiBody` library considerably.

In the left column of Table 1, the icon of the respective model is shown whereas in the right column the essential equations are given that are mapped directly to Modelica equations in the library. Abbreviations which are used for variable and function names in the right column (to save space) are stated at the top row of Table 1. The new built-in operators “`root`”, “`isRoot`”, “`branch`”, “`potentialRoot`” from Table 1 are actually within a package “Connections” (the correct name would therefore be, e.g., `Connections.root`). All other used functions are from subpackage `MultiBody.Frames`. Let us discuss the components in a bit more detail, see Table 1.

6.1 MultiBody.World

In the `World` model essentially the position vector of its frame connectors is set to zero and the orientation object of the frame is set to a null rotation (e.g., the transformation matrix is the identity matrix). When dragging `MultiBody.World` into a model, the following declaration is generated (this behavior is defined via an annotation):

```
inner MultiBody.World world;
```

This is necessary since nearly all components have a corresponding “outer” declaration to access the definitions in the world object, such as defaults for animation and the gravity function. In components that have a mass, the function `world.gravityAcceleration(r)` is called to inquire the gravity acceleration at position \mathbf{r} . Depending on user input, different gravity fields can be used. Currently, no gravity field, parallel and point gravity field is supported. This allows, e.g., to easily simulate a satellite in the gravity field of the earth. An example is given in Figure 7.

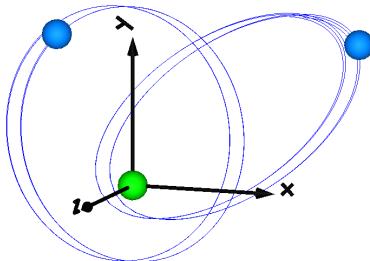


Figure 7. Two point masses in a point gravity field

If the World object is missing in a model, a warning message is printed and an instance of the World object with default settings is automatically utilized. This feature is again defined via an annotation (this is useful for any type of inner declaration).

6.2 MultiBody.Parts.FixedTranslation

This component defines a fixed translation of a frame. It is, e.g., used to define frames for several attachment points on a body. The equations state that the position vector of frame_b is defined from the position vector of frame_a and the relative position vector ${}^a\mathbf{r}^{ab}$ from frame_a to frame_b (${}^a\mathbf{r}^{ab}$ is defined as parameter “ r ”). Since frames are translated, the orientation objects in the two frames are set equal. This in turn requires a “Connections.branch(...)”, see section 5. Finally, a force and torque balance of this massless part is present in the Modelica model.

6.3 MultiBody.Joints.Revolute

This component defines a rotation along an axis vector $\mathbf{n} = {}^a\mathbf{n} = {}^b\mathbf{n}$ via angle φ . When $\varphi = 0$, frame_a and frame_b coincide. As with most other joints, the generalized coordinates (here: φ and $\omega = \dot{\varphi}$) have the attribute `stateSelect = StateSelect.prefer` in order that they are selected as states if possible. Since the origins of both frames are located at the same point on the axis of rotation, the position vectors in the two frames are identical. The relative orientation object \mathbf{R}^{rel} is computed with \mathbf{n} and φ . It is used to define the relationship between the orientation objects from frame_a and frame_b. It is also stated

Abbreviations:	
$\mathbf{r}^a, \mathbf{R}^a, \mathbf{f}^a, \boldsymbol{\tau}^a := \text{frame_a.R_0}, \mathbf{R}, \mathbf{f}, \boldsymbol{\tau}$	$\mathbf{r}^a, \mathbf{R}^a, \mathbf{f}^a, \boldsymbol{\tau}^a := \text{frame_a.R_0}, \mathbf{R}, \mathbf{f}, \boldsymbol{\tau}$
$\mathbf{r}^b, \mathbf{R}^b, \mathbf{f}^b, \boldsymbol{\tau}^b := \text{frame_b.R_0}, \mathbf{R}, \mathbf{f}, \boldsymbol{\tau}$	$\mathbf{r}^b, \mathbf{R}^b, \mathbf{f}^b, \boldsymbol{\tau}^b := \text{frame_b.R_0}, \mathbf{R}, \mathbf{f}, \boldsymbol{\tau}$
<code>absRotation := Frames.absoluteRotation</code>	<code>absRotation := Frames.absoluteRotation</code>
<code>relRotation := Frames.relativeRotation</code>	<code>relRotation := Frames.relativeRotation</code>
<code>angVel2 := Frames.angularVelocity2</code>	<code>angVel2 := Frames.angularVelocity2</code>
<code>Q.angVel2 := Frames.Quaternions.angularVelocity2</code>	<code>Q.angVel2 := Frames.Quaternions.angularVelocity2</code>
<code>Q.constraint := Frames.Quaternions.orientationConstraint</code>	<code>Q.constraint := Frames.Quaternions.orientationConstraint</code>
<code>grav := world.gravityAcceleration</code>	<code>grav := world.gravityAcceleration</code>
World	root(frame_b.R) $\mathbf{r}^b = \mathbf{0}$ $\mathbf{R}^b = \text{nullRotation}()$
Parts.FixedTranslation	branch(frame_a.R, frame_b.R) $\mathbf{r}^b = \mathbf{r}^a + \text{resolve1}(\mathbf{R}^a, {}^a\mathbf{r}^{ab})$ $\mathbf{R}^b = \mathbf{R}^a$ $\mathbf{0} = \mathbf{f}^a + \mathbf{f}^b$ $\mathbf{0} = \boldsymbol{\tau}^a + \boldsymbol{\tau}^b + {}^a\mathbf{r}^{ab} \times \mathbf{f}^b$
Joints.Revolute	branch(frame_a.R, frame_b.R) $\mathbf{r}^b = \mathbf{r}^a$ $\mathbf{R}^{rel} = \text{planarRotation}(\mathbf{n}, \varphi)$ $\mathbf{R}^b = \text{absRotation}(\mathbf{R}^a, \mathbf{R}^{rel})$ $\omega = \dot{\varphi}$ $\mathbf{0} = \mathbf{n}^T \cdot \boldsymbol{\tau}^b$ $\mathbf{0} = \mathbf{f}^a + \text{resolve1}(\mathbf{R}^{rel}, \mathbf{f}^b)$ $\boldsymbol{\tau}^a = \boldsymbol{\tau}^a + \text{resolve1}(\mathbf{R}^{rel}, \boldsymbol{\tau}^b)$
Joints.Spherical	//no branch(...) $\mathbf{r}^b = \mathbf{r}^a$ $\mathbf{R}^{rel} = \text{relRotation}(\mathbf{R}^a, \mathbf{R}^b)$ $\mathbf{0} = \mathbf{f}^a + \text{resolve1}(\mathbf{R}^{rel}, \mathbf{f}^b)$ $\boldsymbol{\tau}^a = \mathbf{0}$ $\boldsymbol{\tau}^b = \mathbf{0}$
Parts.Body	potentialRoot(frame_a.R) if isRoot(frame_a.R) then $\mathbf{0} = Q.\text{constraint}(\mathbf{p})$ $\omega^a = Q.\text{angVel2}(\mathbf{p}, \mathbf{p})$ $\mathbf{R}^a = \text{Frames.from_p}(\mathbf{p})$ else $\omega^a = \text{angVel2}(\mathbf{R}^a, \dot{\mathbf{R}}^a)$ $\mathbf{p} = Q.\text{nullRotation}()$ end if $\mathbf{v} = \dot{\mathbf{r}}^a$ $\mathbf{g} = \text{grav}(\mathbf{r}^a + \text{resolve1}(\mathbf{R}^a, \mathbf{r}^{CM}))$ $\mathbf{a} = \text{resolve2}(\mathbf{R}^a, \dot{\mathbf{v}} - \mathbf{g})$ $\mathbf{f}^a = m \cdot (\mathbf{a} + \dot{\boldsymbol{\omega}}^a \times \mathbf{r}^{CM} + \boldsymbol{\omega}^a \times (\boldsymbol{\omega}^a \times \mathbf{r}^{CM}))$ $\boldsymbol{\tau}^a = \mathbf{I}\dot{\boldsymbol{\omega}}^a + \boldsymbol{\omega}^a \times \mathbf{I}\boldsymbol{\omega}^a + \mathbf{r}^{CM} \times \mathbf{f}^a$

Table 1. Elementary components of MultiBody library

that the projection of the cut-torque on \mathbf{n} must vanish. Finally, the force and torque balance of this massless part is present. Besides model “Revolute” there is also a joint “ActuatedRevolute” that has an additional 1-dim. flange connector. Via this flange, a drive train can be attached driving the revolute

joint, e.g., with components from the Modelica-Mechanics.Rotational library (see Figure 1).

There is an additional utility function “rooted(...)” to inquire whether there is a path in the spanning trees of the virtual connection graphs from a selected root to the frame under consideration. This is used here and at some other places to give two equation variants depending on the actual connection structure in order to avoid small linear algebraic equations. For example, if rooted(frame_a.R) = **true** then the force and torque at frame_a are computed from the frame_b quantities. Otherwise, the force and torque at frame_b are computed from the frame_a quantities.

6.4 MultiBody.Joints.Spherical

This component defines a spherical joint, i.e., the origins of frame_a and frame_b coincide and the two frames can freely rotate relative to each other. No torques are transmitted via this joint. Since frame_a.R and frame_b.R are not related together in an algebraic equation, no “branch(...)” statement is present. No states are defined for this joint.

6.5 MultiBody.Parts.Body

This component defines the mass and inertia properties of a body. It has one frame_a that is usually used as reference coordinate system of a part which is associated with a specific geometric position on the part. Other points on the part are often defined via FrameTranslation components connected to frame_a of the body component. The mass m, the position vector $\mathbf{r}^{CM} = {}^a\mathbf{r}^{aCM}$ from the origin of frame_a to the center of mass (resolved in frame_a) and the inertia tensor $\mathbf{I} = {}^a\mathbf{I}^{CM}$ with respect to the center of mass are given as parameters and define the body properties, see also Table 1.

The body component is defined as “potentialRoot”, i.e., it may be selected as root of a spanning tree of the virtual connection graph. Whether it is selected or not can be inquired via function “isRoot(...)”. If the body frame is **not** selected as root, the orientation object in the frame is defined somewhere else. In this case the second branch of the if clause in Table 1 is used and the angular velocity of the body frame is determined by frame_a.R and its derivative which for example means that it is computed (indirectly) by the generalized position and velocity variables of joints.

If “isRoot(...) = **true**”, it is required that frame_a.R is calculated within the body object. This is only possible if variables of the body are used as states from which frame_a.R can be determined. By default, quaternions \mathbf{p} are used as potential states. Consequently frame_a.R is computed from \mathbf{p} and

the angular velocity is computed from \mathbf{p} and its derivative $\dot{\mathbf{p}}$. The 4 coordinates of the quaternion vector \mathbf{p} have to fulfill the constraint equation “ $\mathbf{p}^T \cdot \mathbf{p} = 1$ ”. This non-linear equation is added in the first if-clause. Since there is a non-linear equation relating potential states, a tool has to use the dynamic dummy derivative method to dynamically select 3 states out of 4 potential states during simulation. Whenever the selection comes close to its singularity, Dymola changes the states at a completed step of the integrator. The 4th potential state has to be computed by solving the non-linear quaternion constraint equation. Dymola performs this in an efficient and robust way, because it can detect that the special non-linear equation of quaternions is present and solves this equation analytically. E.g., if p[1:3] are selected as states, then

$$p[4] = \sqrt{1 - p[1:3] \cdot p[1:3]} * \text{signAtLastStep}(p[4]).$$

Via a parameter in the “Advanced” menu of the body object, it is possible to alternatively also use the 3 Cardan angles as states. They are defined with respect to a coordinate system “Fix” fixed in frame_a. Whenever the Cardan angles come close to their singularity, frame “Fix” is changed such that the new Cardan angles are far away from their singularity. The advantage of this approach is that no dynamic dummy derivative method is needed. The disadvantage is that every change of states results in a state event which is less efficient as the state change performed with the dynamic dummy derivative method. Furthermore, several variables are discontinuous (especially the Cardan angles) which can lead to problems if equations are further differentiated, e.g., for inverse models.

The non-standard feature to have potential states both in **joints** and in **bodies** is especially useful for inexperienced users, since they do not have to introduce a “virtual” joint with 6 degrees of freedom. For example, it is easy to just build up a system as in Figure 8, where a body is connected via a spring to the environment.

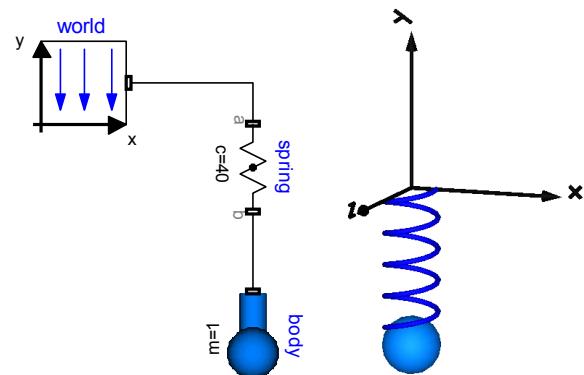


Figure 8. Free body with spring

In the left part of the figure the Modelica schematic and in the right part the default animation is shown. No “non-physical” joint has to be introduced to build up such a model, as it is usually the case in other multi-body programs.

Let us now return to the body equations in Table 1. Once the orientation object and the angular velocity of the body frame are determined, all other kinematical quantities are derived by differentiation and used in the Newton/Euler equations that are formulated with respect to frame_a of the body (and not with respect to the center of mass).

7 Loop Structures

Due to the new handling of overdetermined DAEs, the modeler does not have to take special actions if loop structures occur (contrary to the ModelicaAdditions.MultiBody library). An example is presented in Figure 9. It is available as MultiBody.Examples.Loops.Fourbar1. In the upper

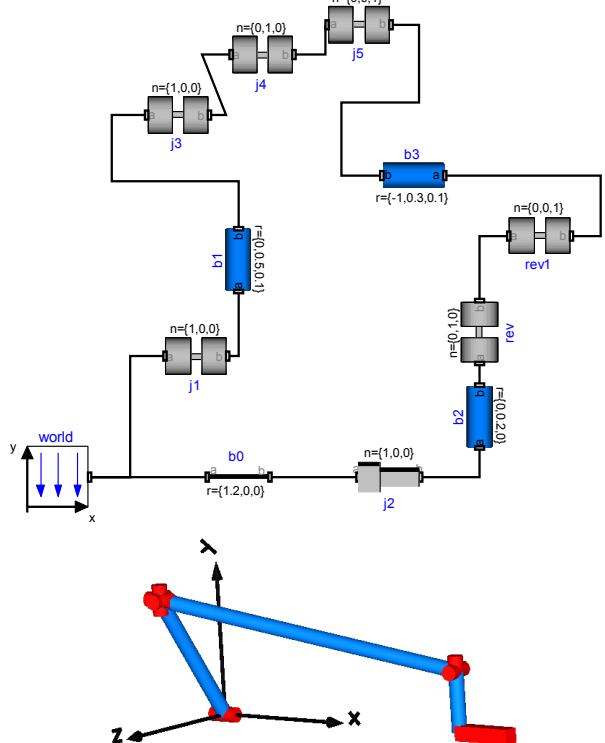


Figure 9. Four bar mechanism with 7 joints and 1 dof

part of the figure the Modelica schematic of a four bar mechanism is shown constructed with the MultiBody library. It consists of 6 revolute, 1 prismatic joint and forms a kinematical loop. This mechanism has one degree of freedom.

In the lower part of the figure the default animation is shown. Note, that the axes of the revolute joints are represented by the red cylinders

and that the axis of the prismatic joint is represented by the red box on the lower right side.

Whenever loop structures occur, non-linear algebraic equations are present on “position level”. It is then usually not possible by structural analysis to select states during translation (which is possible for non-loop structures). In the example above, Dymola detects a non-linear algebraic loop of 57 equations and reduces this to a system of 7 coupled algebraic equations. Note, that this is performed without using any “cut-joints” as it is usually done in multi-body programs, but by just appropriate symbolic equation manipulation. Via the dynamic dummy derivative method the generalized coordinates on position and velocity level from one of the 7 joints are dynamically selected as states during simulation. Whenever, these two states are no longer appropriate, states from one of the other joints are selected.

The efficiency of loop structures can usually be enhanced, if states are statically fixed at translation time. For this mechanism, the generalized coordinates of joint j1 can always be used as states. This can be stated by setting parameter “enforceStates = true” in the “Advanced” menu of the desired joint. This flag sets the attribute stateSelect of the generalized coordinates of the corresponding joint to “StateSelect.always”. When setting this flag to true for joint j1 in the four bar mechanism, Dymola detects a non-linear algebraic loop of 40 equations and reduces this to a system of 5 coupled non-linear algebraic equations.

7.1 Planar Loops

In Figure 10 the model of a V6 engine is shown that has a simple combustion model. It is available as MultiBody.Examples.Loops.EngineV6. The Modelica schematic of one cylinder is given in the middle part of the figure. Connecting 6 instances of this cylinder appropriately together results in the engine schematic displayed at the upper part of the figure. In the lower part the animation of the engine is shown. Every cylinder consists essentially of 1 prismatic and 2 revolute joints that form a planar loop, since the axes of the two revolute joints are parallel to each other and the axis of the prismatic joint is orthogonal to the revolute joint axes. All 6 cylinders together form a coupled set of 6 loops that have together 1 degree of freedom.

All planar loops, and especially the engine, result in a DAE that does not have a unique solution. The reason is that, e.g., the cut forces in direction of the axes of the revolute joints cannot be uniquely computed. Any value fulfills the DAE equations.

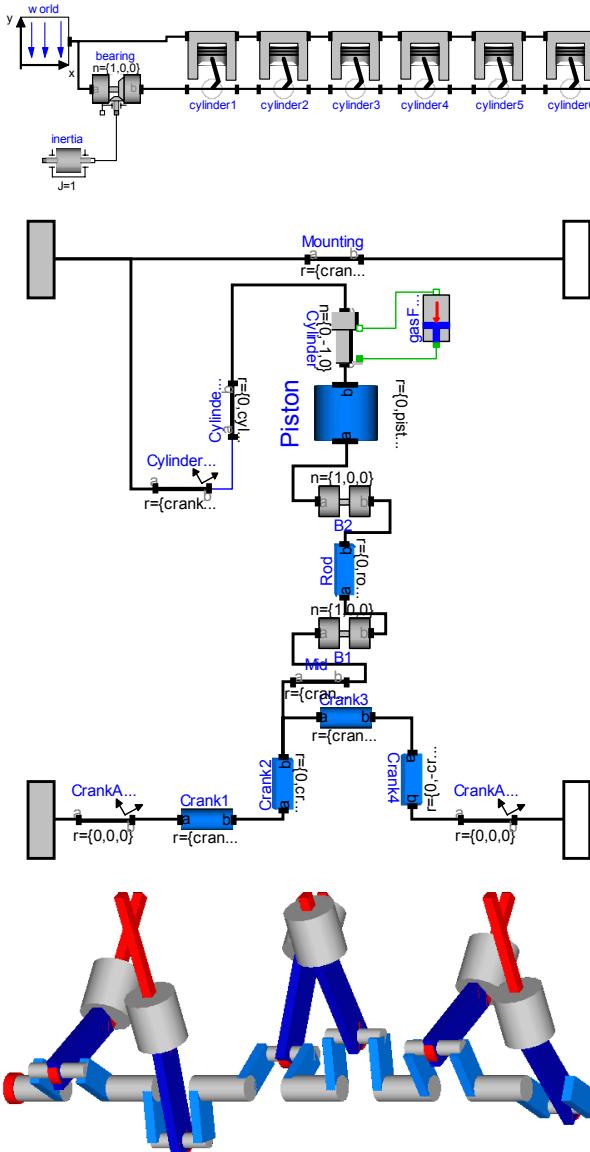


Figure 10. V6 engine with 6 planar loops and 1 dof

This is a structural property that is determined by the symbolic algorithms. Since they detect that the DAE is structurally singular, a further processing is not possible. Without additional information it is also impossible that the symbolic algorithms could be enhanced because if the axes of rotations of the revolute joints are only slightly changed such that they are no longer parallel to each other, the planar loop can no longer move and has 0 degrees of freedom. Algorithms based on pure structural information cannot distinguish these two cases.

The usual remedy is to remove superfluous constraints, e.g., along the axis of rotation of **one** revolute joint. Since this is not easy for an inexperienced modeler, the flag “*planarCutJoint*” is provided in the “Advanced” menu of a revolute joint that removes these constraints. This flag must be set to **true** for one revolute joint in every planar loop.

In the engine example, this flag is set in the revolute joint B2 in the cylinder model.

If a modeler is not aware of the problems with planar loops and models them without special consideration, Dymola displays an error message and points out that a planar loop may be the reason and suggests to use the “*planarCutJoint*” flag. This error message is due to an annotation in the Frame connector:

```
flow SI.Force f[3] annotation(
    unassignedMessage="...");
```

If no assignment can be found for some forces in a connector, the “*unassignedMessage*” is displayed. In most cases the reason for this is a planar loop or two joints that constrain the same motion. Both cases are discussed in the message.

Note that the non-linear algebraic equations occurring in planar loops can be solved analytically in most cases and therefore it is highly recommended to use the techniques discussed in the next two sections for such systems.

7.2 Analytic Loop Handling: User’s View

It is well known that the non-linear algebraic equations of most mechanical loops in technical devices can be solved analytically. It is, however, difficult to perform this fully automatically and therefore none of the commercial, general purpose multi-body programs, such as MSC ADAMS[1], LMS DADS[5], SIMPACK[21], have this feature. These programs solve loop structures with pure numerical methods. Multi-body programs that are designed for real-time simulation of the dynamics of specific vehicles, such as ve-DYNA[23], usually contain manual implementations of a particular multi-body system (the vehicle) where the occurring loops are either analytically solved, if this is possible, or are treated by table look-up where the tables are constructed in a pre-processing phase. Without these features the required real-time capability would be difficult to achieve.

In a series of papers and dissertations, especially [10][24][11][15], Prof. Hiller and his group in Duisburg have developed systematic methods to handle mechanical loops analytically. The “characteristic pair of joints” method [10][24] basically cuts a loop at two joints and uses geometric invariants to reduce the number of algebraic equations, often down to one equation that can be solved analytically. Also several multi-body codes have been developed that are based on this method, e.g., MOBILE [12]. Besides the very desired feature to solve non-linear algebraic equations analytically, i.e., efficiently and in a robust way, there are several drawbacks: It is

difficult to apply this method automatically. Even if this would be possible in a good way, there is always the problem that it cannot be guaranteed that the statically selected states lead to no singularity during simulation. Therefore, the “characteristic pair of joints” method is usually manually applied which requires know-how and experience.

In the MultiBody library the “characteristic pair of joints” method is supported in a restricted form such that it can be applied also by non-specialists. The idea is to provide joint aggregations in package MultiBody.Joints.Assemblies as one object that either have 6 degrees of freedom or 3 degrees of freedom (for usage in planar loops).

As an example, a variant of the four bar mechanism from Figure 9 is given in Figure 11. In the upper part of the figure, the mechanism is modeled with standard joints. In the lower part, the two spherical joints and the prismatic joint are collected together in an assembly object called “jointSSP” that is defined in

`MultiBody.Joints.Assemblies.JointSSP`.

This joint aggregation has a frame at the left side of the left spherical joint (frame_a) and a frame at the right side of the prismatic joint (frame_b). JointSSP, as all other objects from the Joints.Assemblies package, has the property, that **the generalized**

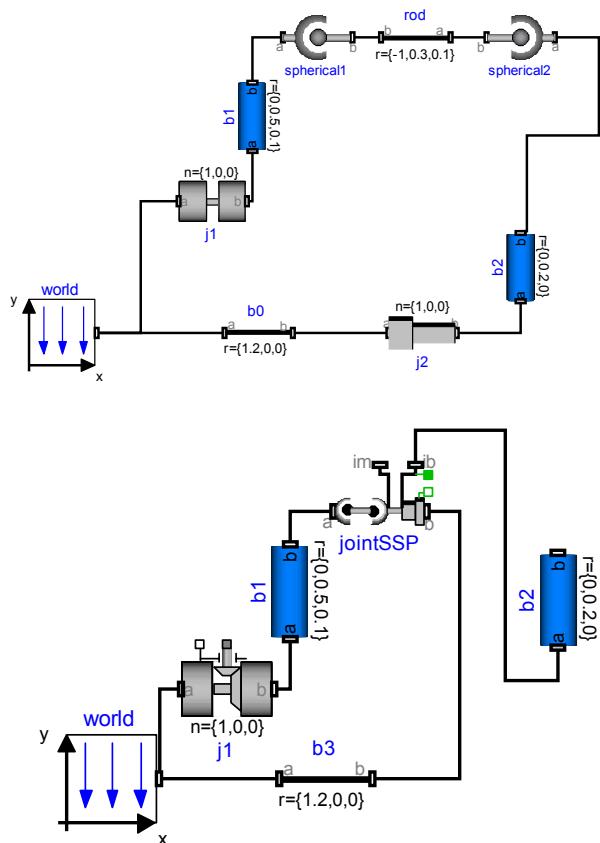


Figure 11. Analytic handling of four bar mechanism

coordinates, and all other frames defined in the assembly, can be calculated given the movement of frame_a and of frame_b. This is performed by analytically solving non-linear systems of equations (details are given in the next subsection). From a structural point of view, the equations in an assembly object are written in the form

$$\mathbf{q} = \mathbf{f}_1(\mathbf{r}^a, \mathbf{R}^a, \mathbf{r}^b, \mathbf{R}^b)$$

where $\mathbf{r}^a, \mathbf{R}^a, \mathbf{r}^b, \mathbf{R}^b$ are the variables defining the position and orientation of the frame_a and frame_b connector (see also Table 1) and \mathbf{q} are the generalized positional coordinates inside the assembly, e.g., the angle of a revolute joint. Given angle φ of revolute joint j1 from the four bar mechanism, frame_a and frame_b of the assembly object can be computed by a forward recursion

$$(\mathbf{r}^a, \mathbf{R}^a, \mathbf{r}^b, \mathbf{R}^b) = \mathbf{f}(\varphi)$$

Since this is a structural property, the symbolic algorithms can automatically select φ and its derivative as states and then all positional variables can be computed in a forwards sequence. It is now understandable that Dymola transforms the equations of the four bar mechanism to a recursive sequence of statements that has neither linear nor non-linear algebraic loops (remember, the previous “straightforward” solution had a nonlinear system of equations of order 5).

The aggregated joint objects consist of a combination of either a revolute or prismatic joint and of a rod that has either two spherical joints at its two ends or a spherical and a universal joint, respectively. For all combinations, analytic solutions can be determined. For planar loops, combinations of 1, 2 or 3 revolute joints with parallel axes and of 2 or 1 prismatic joint with axes that are orthogonal to the revolute joints can be treated analytically. The currently supported combinations are listed in Table 2. The missing combinations (such as JointSUP or Joint RPP) will be added in one of the next releases.

3-dimensional Loops:	
JointSSR	Spherical – Spherical – Revolute
JointSSP	Spherical – Spherical – Prismatic
JointUSR	Universal – Spherical – Revolute
JointUSP	Universal – Spherical – Prismatic
JointUPS	Universal – Prismatic – Spherical
Planar Loops:	
JointRRR	Revolute – Revolute – Revolute
JointRRP	Revolute – Revolute – Prismatic

Table 2. MultiBody.Joints.Assemblies aggregations

On first view this seems to be quite restrictive. However, mechanical devices are usually built up with rods connected by spherical joints on each end, and additionally with revolute and prismatic joints. Therefore, the combinations of Table 2 occur frequently. The universal joint is usually not present in actual devices but is used (a) if two JointXXX components can be connected such that a revolute and a universal joint together form a spherical joint, see Figure 12 and (b) if the orientation of the connecting rod between two spherical joints is needed, e.g., since a body shall be attached. In this case one of the spherical joints might be replaced by a universal joint. This approximation is fine as long as the mass and inertia of the rod is not significant.

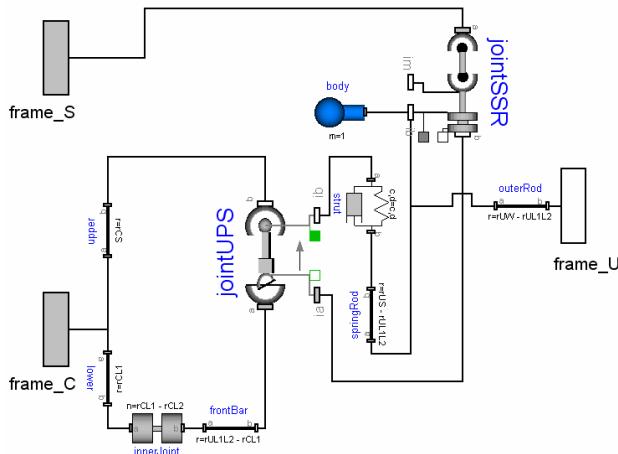


Figure 12. MacPherson with analytic loop handling

Let us discuss item (a) in more detail: The MacPherson suspension in Figure 12 is from the Modelica VehicleDynamics library [2]. It has three frame connectors. The lower left one (frame_C) is fixed in the vehicle chassis. The upper left one (frame_S) is driven by the steering mechanism, i.e., the movement of both frames are given. The frame connector on the right (frame_U) drives the wheel. The three frames are connected by a mechanism consisting essentially of two rods with spherical joints on both ends. These are built up by a jointUPS and a jointSSR assembly, see Figure 12. As can be seen, the universal joint from the jointUPS assembly is connected to the revolute joint of the jointSSR assembly. Therefore, we have 3 revolute joints connected together at one point and if the axes of rotations are chosen appropriately, this describes a spherical joint. In other words, the two connected assemblies define the desired two rods with spherical joints on each ends.

The movement of the chassis, frame_C, is computed somewhere else. When the generalized coordinates of revolute joint "innerJoint" (lower left part in figure) are used as states, then frame_a and frame_b of the jointUPS joint can be calculated.

After the non-linear loop with jointUPS is solved, all frames on this assembly are known, especially, the one connected to frame_b of the jointSSR assembly. Since frame_b of jointSSR is connected to frame_S which is computed from the steering mechanism, again the two required frame movements of the jointSSR assembly are calculated, meaning in turn that also all other frames on the jointSSR assembly can be computed, especially, the one connected to frame_U that drives the wheel. From this analysis it is clear that a tool is able to solve these coupled loops analytically.

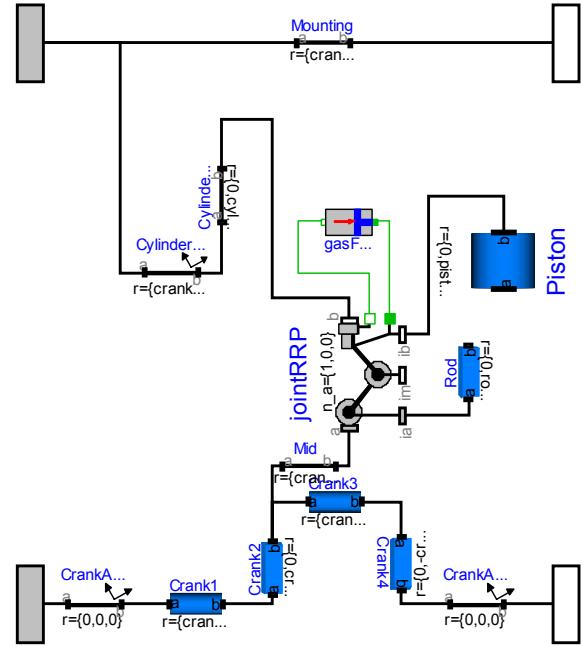


Figure 13. Cylinder of engine with analytic loop handling

Another example is the engine model from Figure 10. It is sufficient to rewrite the basic cylinder model by replacing the joints with a JointRRP object that has two revolute and one prismatic joint, see Figure 13. Since 6 cylinders are connected together, 6 coupled loops with 6 JointRRP objects are present. This model is available as MultiBody.Examples.Loops.EngineV6_analytic.

From Figure 10 it can be seen that the revolute joint of the crank shaft (left part of upper subfigure in Figure 10) might be selected as degree of freedom. Then the 4 connector frames of all cylinders can be computed. As a result the computations of the cylinders are decoupled from each other. Within one cylinder, see Figure 13, the position of frame_a and frame_b of the jointRRP assembly can be computed and therefore the generalized coordinates of the two revolute and the prismatic joint in the jointRRP object can be determined. From this analysis it is not surprising that Dymola is able to transform the DAE equations

into a sequential evaluation without any linear or non-linear loop. Compare this nice result with the model from Figure 10 that leads to a DAE with 6 algebraic loops and 5 non-linear equations per loop. Additionally, a linear system of equations of order 43 is present. The simulation time is about 5 times faster with the analytic loop handling.

7.3 Analytic Loop Handling: How it works

The basic technique for the analytic loop handling is explained at hand of the JointSSR (Spherical – Spherical – Revolute) assembly shown in Figure 14. It consists of two spherical joints connected by a rigid massless rod and a revolute joint connected by an additional massless rod to the spherical joint in the middle (optionally, a point mass can be present on the rod connecting the two spherical joints). At the upper part of Figure 14 the Modelica icon of the JointSSR object and in the lower part an animation view with some important position vectors is shown. The following derivation is a special case of the “characteristic pair of joints” method and is based on [24].

It is assumed that the positions and orientations of frame_a and of frame_b of the JointSSR object are calculated as a function of states. This means that the position vectors ${}^0\mathbf{r}_{sl}$, ${}^0\mathbf{r}_{rev}$ from the origin of the world frame to the origins of frame_a and of frame_b of the JointSSR object are known. Using the orientation objects of frame_a and of frame_b it is easy to compute position vector ${}^a\mathbf{r}_1$ that is directed from the origin of the revolute joint (= frame_b) to the origin of the first spherical joint (= frame_a) and is resolved in **frame_a** of the **revolute** joint (this frame is identical to frame_b of the JointSSR object). Position vector ${}^b\mathbf{r}_2$ is a parameter of the JointSSR object and is directed from the origin of the revolute joint to the origin of the second spherical joint and is resolved in **frame_b** of the

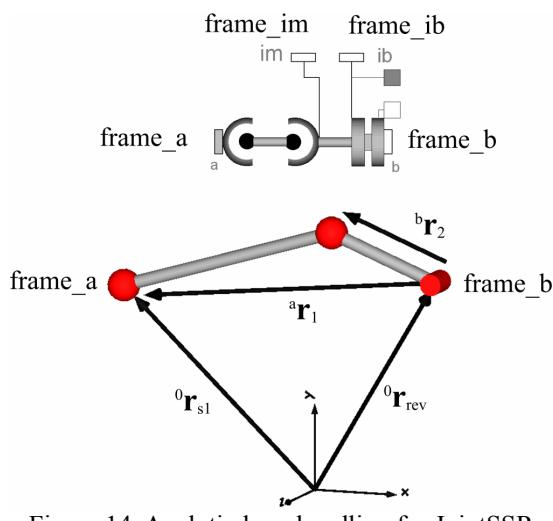


Figure 14. Analytic loop handling for JointSSR

revolute joint. The two spherical joints are connected together by a rod with a fixed length L which is a parameter of the JointSSR object. The length L can be also calculated by computing the vector from spherical joint 1 to spherical joint 2 with vectors ${}^a\mathbf{r}_1$, ${}^b\mathbf{r}_2$ and taking its length. The square of this length results in:

$$L^2 = ({}^b\mathbf{r}_2 - \mathbf{T}(\varphi) \cdot {}^a\mathbf{r}_1)^T \cdot ({}^b\mathbf{r}_2 - \mathbf{T}(\varphi) \cdot {}^a\mathbf{r}_1)$$

Since ${}^a\mathbf{r}_1$ and ${}^b\mathbf{r}_2$ are resolved in different frames, ${}^a\mathbf{r}_1$ has first to be transformed from frame_a to frame_b of the revolute joint using the relative transformation matrix \mathbf{T} between these two frames. This matrix is solely a function of the unknown rotation angle φ . In the equation above all variables are known (or are calculated somewhere else) with exception of φ . Therefore, we have one non-linear algebraic equation for one unknown, φ , and the goal is to solve this equation analytically. Multiplying out all terms and taking into account that $\mathbf{T}(\varphi)^T \cdot \mathbf{T}(\varphi)$ is the unit matrix, since transformation matrices are orthogonal, we arrive at

$$0 = {}^b\mathbf{r}_2^T \cdot {}^b\mathbf{r}_2 + {}^a\mathbf{r}_1^T \cdot {}^a\mathbf{r}_1 - L^2 - 2 \cdot {}^b\mathbf{r}_2 \cdot \mathbf{T}(\varphi) \cdot {}^a\mathbf{r}_1$$

The relative transformation matrix \mathbf{T} can be mathematically described as, see, e.g., [18]:

$$\mathbf{T} = \mathbf{n} \cdot \mathbf{n}^T + (\mathbf{E} - \mathbf{n} \cdot \mathbf{n}^T) \cdot \cos(\varphi) - \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix} \cdot \sin(\varphi)$$

where \mathbf{E} is the identity matrix and \mathbf{n} is a unit vector in direction of the axis of rotation. \mathbf{n} has the same coordinates with respect to frame_a and to frame_b. Inserting this formula in the constraint equation and rearranging terms results in

$$0 = A \cdot \cos(\varphi) + B \cdot \sin(\varphi) + C$$

with

$$A = -2 \cdot ({}^b\mathbf{r}_2^T \cdot {}^a\mathbf{r}_1 - (\mathbf{n}^T \cdot {}^b\mathbf{r}_2) \cdot (\mathbf{n}^T \cdot {}^a\mathbf{r}_1))$$

$$B = 2 \cdot {}^b\mathbf{r}_2 \cdot \mathbf{n} \times {}^a\mathbf{r}_1$$

$$C = {}^a\mathbf{r}_1^T \cdot {}^a\mathbf{r}_1 + {}^b\mathbf{r}_2^T \cdot {}^b\mathbf{r}_2 - L^2 - 2 \cdot (\mathbf{n}^T \cdot {}^b\mathbf{r}_2) \cdot (\mathbf{n}^T \cdot {}^a\mathbf{r}_1)$$

Note, that the coefficients A, B, C are computed from known quantities. This non-linear equation has two solutions in the range: $-180^\circ \leq \varphi \leq 180^\circ$:

$$\varphi_{1/2} = \text{atan2}(-B \cdot C - k \cdot A \cdot \sqrt{A^2 + B^2 - C^2}, -A \cdot C + k \cdot B \cdot \sqrt{A^2 + B^2 - C^2})$$

$$k = \pm 1$$

In the JointSSR object a guess value φ_{guess} is defined as a parameter. From the two solutions the one is selected during initialization that is closest to φ_{guess} . This determines the value of the constant k at initial

time. During simulation, the value of k is kept constant. The term under the square root may become negative so that no (real) solution exists anymore. This is the case when the length of ${}^a\mathbf{r}_1$ becomes larger as the sum of the lengths of the two rods of the JointSSR object, see Figure 14. This case is checked with an assert statement and if it is no longer valid, the simulation is stopped and an appropriate error message is given in which this situation is explained.

Note, for the JointSSP (Spherical – Spherical – Prismatic) assembly, a similar derivation leads to a simple quadratic equation that has two solutions.

Once angle φ is determined with the above formulas, all other desired positional quantities of the JointSSR object can be computed in a straightforward way. By differentiating the equations twice also the first and second derivative of the angle can be determined. The differentiation is automatically performed by the tool. Finally, the (unchanged) equations of the revolute joint and of the other components in the JointSSR object are used to build up the DAE system. It turns out that this approach results in a linear system of equations where at least the second derivative of φ and the as yet unknown force in the rod connecting the two spherical joints is contained. The dimension of this loop is reduced or the loop is even completely eliminated in some cases by the following approach:

In the revolute joint there is an equation that states that the projection of the cut-torque τ of frame_b on the axis of rotation \mathbf{n} of the revolute joint is zero, see Table 1: $\mathbf{n}^T \cdot \tau = 0$. By a torque balance around the origin of frame_b of the JointSSR object, the cut-torque τ at frame_b can be expressed as a function of the cut-forces and cut-torques at the other frame connectors of the JointSSR object and the unknown force in the rod connecting the two spherical joints (assuming this rod is cut for the torque balance). Inserting these relationships in the equation $\mathbf{n}^T \cdot \tau = 0$, results in one linear equation in the unknown rod force from which the rod force can be computed analytically as function of the cut-forces and -torques of frame_im and frame_ib (see Figure 14).

8 Force Elements

Force elements exert forces and torques between two frames. The icon of the most general one available in the MultiBody library (model MultiBody.Forces.ForceAndTorque) is displayed in Figure 15

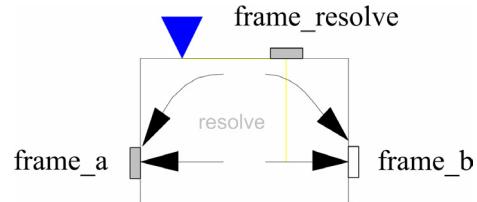
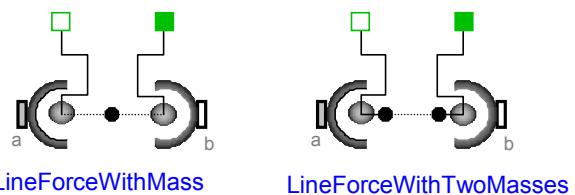


Figure 15. General force element

The 6 elements in the input signal vector are interpreted as the 3 coordinates of a force and the 3 coordinates of a torque acting at the component to which frame_b of the ForceAndTorque component is connected. The force and torque defined with the 6 elements of the input are assumed to be resolved in the frame to which connector frame_resolve is connected. If frame_resolve is not connected, it is assumed that the force and torque are resolved in frame_b. Additionally the force and torque act with “opposite sign” on frame_a (or more precisely, the force and torque on frame_a is computed by a force/torque balance between the two frames). Via sensor elements, any type of kinematical or force/torque information can be inquired. This can be used to compute the force and torque of a force element. Note, since the MultiBody library is purely equation based, also accelerations (e.g., from an acceleration sensor), and cut-forces and cut-torques (e.g., the normal force of a Coulomb friction element) can be utilized to compute the force and torque of a ForceAndTorque element.

8.1 Line Force Elements With Mass

More often, line force elements are needed, that exert a force on the line between the origins of two frames. The two basic line force elements of the MultiBody library are displayed in Figure 16.



LineForceWithMass LineForceWithTwoMasses

Figure 16. Line force elements that may have mass

The force acting between the origins of frame_a and of frame_b (on the line between these two points) is defined via the two 1-dimensional flange connectors at the top part of the icons (the two green filled and non filled squares). Here, models of the Modelica.Mechanics.Translational library can be connected. An example is given in Figure 17 where a 1-dimensional translational spring is connected between the 1D flange connectors.

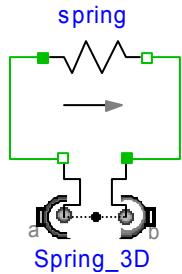


Figure 17. Line force with 1D spring

This approach has several advantages: (1) the distance between frame_a and frame_b is reported in the 1D flange connectors and can therefore be directly utilized in the force law without having to use a sensor object to inquire kinematical information. (2) For more complicated force laws, e.g., a hydraulic cylinder that is driven by a hydraulic circuit, it is advisable to first test the whole force law separately with 1-dim. elements and additional libraries such as a hydraulic or an electrical library. When this works, the force object is just connected to the 3-dimensional line force element of Figure 16.

In multi-body programs the assumption is usually made that force elements are massless. In reality this is not always justified since, e.g., a spring or a hydraulic cylinder has mass that might be significant in some applications. For example, the counter balance systems of large robots have usually a mass that is 5 – 10 % of the mass of the moving parts. By just examining the reaction force to the ground, it is clear that it is not possible to neglect this mass.

For these practical requirements, the line force elements provided in the MultiBody library have optionally one or two point masses on the line from the origin of frame_a to the origin of frame_b. The usage of a point mass is usually sufficient and has the advantage that not much data is required from the user (additionally data: mass of the point mass and its location) and that it can be handled very efficiently with only a small overhead in the computation compared to a force element without a point mass.

In element “LineForceWithMass” the point mass is located at a fixed relative distance between the two frame origins. Default is “in the middle”. This is useful, e.g., for a spring. In element “LineForceWithTwoMasses” two point masses are present that are located at an absolute distance with respect to frame_a and to frame_b, respectively. For example, point mass 1 might be located 0.5 m away from the origin of frame_a on the line to frame_b. This is useful, e.g., for a hydraulic cylinder.

8.2 Direct Coupling of Force Elements

Nearly all multi-body programs have the restriction that two force elements cannot be directly connected together. When this is desired, the user has to introduce a body with a small mass between the force elements leading usually leading to an unnecessary stiff model. Since the Modelica MultiBody library is purely equation based, there are **no such restrictions** and it is possible to connect 3-dimensional force elements directly together, such as a series connection of the “ForceAndTorque” element from Figure 15. This usually leads to nonlinear systems of equations.

It is also possible to connect line force elements directly together as demonstrated in Figure 18. This example is available from MultiBody.Examples.Elementary.ThreeSprings. In the upper part of this figure the Modelica schematic is shown consisting of three springs that are connected together at one point. The other ends of the springs are connected to the environment and to a body moving freely in space. In the lower part of the figure the animation of this system is shown.

Without special action difficulties would occur, since in every “line force element” there is an equation stating that the cut-torques at both ends of the line force element (= frame_a.t and frame_b.t) are zero. If three line force elements are connected together as in Figure 18, there is additionally the zero sum equation of flow variables stating that the sum of the cut-torques of the connected springs is zero. This is one equation too much, since all torques in this equation are already set to zero in the spring elements. On the other hand, the orientation

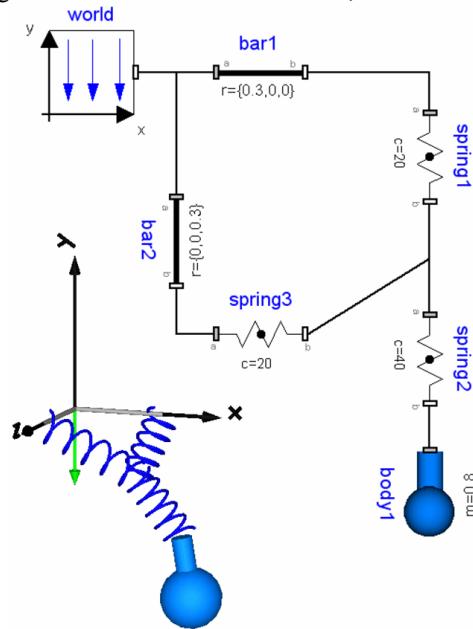


Figure 18. Springs connected directly together

object in the frame connector is not defined because a line force element does not compute it, which means that the orientation object in the connection point of the three springs is not defined. Therefore, the resulting DAE of Figure 18 would be structurally singular and has both overdetermined and underdetermined sets of equations.

It is possible to automatically fix this problem. One line force element that is directly connected at one point to other line force elements has to define that the orientation object in the frame connector defines a null rotation and on the other hand has to remove the equation that states that the cut-torque is zero. This is defined in the following way with Modelica:

```
model LineForceWithMass
  ...
  equation
    potentialRoot(frame_a.R, 100);
    potentialRoot(frame_b.R, 100);
    ...
    if isRoot(frame_a.R) then
      frame_a.R=Frames.nullRotation();
    else
      frame_a.t=zeros(3);
    end if;

    if isRoot(frame_b.R) then
      frame_b.R=Frames.nullRotation();
    else
      frame_b.t=zeros(3);
    end if;
  end LineForceWithMass;
```

A frame connector of a line force element is a potential root of a virtual connection graph (see section 5). The priority of this potential root is set to 100, as opposed to potential roots of bodies that have a priority of 0. This means that, whenever possible, a body is selected as a root. If this is not possible, a frame connector of a line force element is selected as root (meaning that only line force elements are connected together). Since exactly one frame of a connection point is selected as root, the corresponding line force element can provide the necessary equations as shown in the Modelica code fragment above.

9 Animation

The MultiBody library provides sub library “Visualizers” that contains models to visualize geometric parts, see Figure 19. All visualizer objects have a frame connector to connect the object to any other frame connector in a model. The properties of the visualizer object are described with respect to

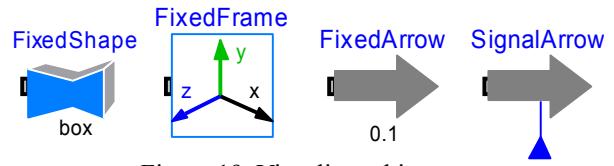
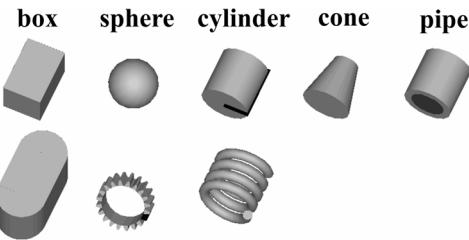


Figure 19. Visualizer objects

the frame to which the object is connected. All visualizer objects have a Boolean parameter “animation” with default “animation = **true**”. If “animation = **false**” is set, the animation of this object is switched off and all equations of this object are removed from the generated code. Additionally, in the World object there is a global flag “enableAnimation”. If this flag is set to **false**, the animation of all objects is removed (this is especially important for real-time simulation).

Visualizer components “FixedArrow” and “SignalArrow” display an arrow at a frame. “FixedFrame” displays a coordinate system with axes labels, see Figure 2. “FixedShape” displays either one of the geometric shapes from Figure 20 or it displays a 3D shape from a DXF or STL file. All models in the MultiBody library, such as a joint, a body, a force element or a sensor, have built-in



beam gearwheel spring

Figure 20. Geometric shapes visualized by “FixedShape”
animation properties that are based on the visualizer objects. Appropriate default values are available such that, without any additional action from the user, always an animation of the defined elements is displayed that can be further refined to get a nicer drawing. The main advantage of this approach is that a defined multi-body model can be quickly checked visually. This feature is implemented in the following way (which might be useful also for other applications):

```
...
protected
  outer MultiBody.World world;
  parameter Integer ndim =
    if world.enableAnimation and
      animation then 1 else 0;
  visualizers.Advanced.Shape
    shape [ndim] (
      each shapeType=shapeType,
      each color=color,
      ...
    )
```

Via an **outer** declaration the world object is accessed. The Visualizers.Advanced.Shape model is a shape without a frame connector that may have a fixed or dynamic shape using all the elements from Figure 20. An instance of this model is declared as an array with dimension “ndim”. This dimension is either zero or one, depending whether animation is enabled or not. A variable of array shape, such as “color” has the same value for all array indices and therefore it is defined as “**each** color = ...”. Modelica supports zero-sized component arrays and therefore the above definition just states that no object “shape” is present, when the dimension of the array is zero, i.e., when animation is disabled.

10 Summary and Outlook

It is expected that the new and free Modelica MultiBody library will be very helpful for the modeling of simple and complex 3-dimensional mechanical systems, especially for non-experts in the multi-body field, since the library is easy to use (in contrast to the previous ModelicaAdditions.-MultiBody library) and it is very powerful. Especially, several features are present to get real-time simulation performance. The MultiBody library is designed to work closely together with other Modelica libraries, in particular with the libraries:

- **Modelica.Mechanics.Translational** for 1-dim. translational line force elements.
- **Modelica.Mechanics.Rotational** for 1-dim. rotational elements to define drive trains driving, e.g., revolute joints. This library contains sophisticated elements such as bearing friction, torque dependent friction in gears, clutches, brakes.
- **PowerTrain** [20] which is an extension of the Rotational library dedicated to vehicle power trains and complicated planetary gears with losses. The Rotational, MultiBody and PowerTrain library are extended in the next version such that all 3D effects of 1-dim. drive trains attached to MultiBody models are taken into account in an efficient and user convenient way [22]. In particular support torques of drive train elements are calculated.
- **HyLib** [3][4] for the modeling of hydraulic systems. Hydraulic cylinders of HyLib can be directly attached to the 1D flanges of MultiBody line force elements.
- **VehicleDynamics** [2] for the modeling of the dynamics of vehicles providing a large set of components and also complete vehicles in

different levels of model details. The free VehicleDynamics library is currently based on the ModelicalAdditions.MultiBody library. It will soon be converted to the new MultiBody library.

- Import filters from **AutoDesk Mechanical desktop** [5] and from **SolidWorks** [9] to Modelica are available for the ModelicaAdditions.MultiBody library. It is planned to convert them soon to the new MultiBody library, see <http://www.mathcore.com>.

We plan to further continue the development of the MultiBody library in different directions. Since the field of possible improvements is large, e.g., modeling of elastic bodies, modeling of contact, interfaces to finite element and CAD programs, aero-elastic couplings of wings, etc., we are interested in cooperations. Please, feel free to contact the authors if you plan to use the MultiBody library as a basis for enhancements, especially if you provide your work also in the public domain.

Acknowledgements

Developments in the EU Project RealSim "Real-time simulation for design of multi-physics systems", in the years 2000-2002 under contract IST-1999-11979, have influenced the design of this library, e.g., the close integration of animation in all objects.

The idea to provide a general line force element with 1D-translational connectors has been taken from the VehicleDynamics library [2]. In the ModelicaAdditions.MultiBody library a user had to inherit from a “LineForce” superclass and always implement the force law with Modelica equations. The usage of the 1D flange connectors is more user friendly.

Appendix: Algorithm to Transform Overdetermined DAEs

In this appendix the algorithm is sketched to transform an overdetermined DAE to a standard DAE where the number of equations and unknowns are identical.

In Table 3, the set of Modelica built-in operators introduced in section 5 are formally defined. These operators are utilized to describe the relationships of the overdetermined types or records in the connector instances of a model: **Every** instance of an *overdetermined type or record* in an *overdetermined connector* is a **node** in a *virtual connection graph* that is used to determine when the standard equation “ $\mathbf{R}_1 = \mathbf{R}_2$ ” or when the equation “ $\mathbf{0} = \text{equalityConstraint}(\mathbf{R}_1, \mathbf{R}_2)$ ” has to be used for the generation of `connect(...)` equations. The **branches** of the virtual connection graph are implicitly defined by “`connect(...)`” and explicitly by “`Connections.branch(...)`” statements, see Table 1.

For example, a revolute joint has two connectors `frame_a` and `frame_b`. In this model, there is an algebraic relationship between the orientation objects of these two frames: $\text{frame_b.R} = f(\text{frame_a.R}, \varphi)$, where φ is the relative rotation angle. A definition of the form

`Connections.branch`

`(frame_a.R, frame_b.R);`

has to be present in this joint model in order to state that the overdetermined variables `frame_a.R` and `frame_b.R` are algebraically coupled.

Additionally, corresponding nodes of the virtual connection graph have to be defined as **roots** or as **potential roots** with functions “`root(...)`” and “`potentialRoot(...)`”, respectively, see Table 3. For example, connector `frame_a` in the World model has to be defined as “`Connections.root(frame_a.R)`” because all elements of `frame_a.R` are explicitly given in the World model (`frame_a.R = nullRotation()`). A “potential root” is, for example, a body object, since if the body is freely flying in space, body coordinates may be used as states from which the orientation object can be computed. It is a “potential root”, because body states should for efficiency reasons only be selected as states, if no other possibility exists.

Note, that `branch(...)`, `root(...)`, `potentialRoot(...)` do not generate equations. They only define nodes and branches in the virtual connection graph for analysis purposes to be discussed now.

Before `connect(...)` equations are generated, the virtual connection graph is transformed into a **set of spanning trees** by removing breakable branches (connections) from the graph. This is performed in

<code>connect(A,B);</code>	Defines breakable branches from the overdetermined type or record instances in connector instance A to the corresponding overdetermined type or record instances in connector instance B for a virtual connection graph.
<code>branch(A.R,B.R);</code>	Defines a non-breakable branch from the overdetermined type or record instance R in connector instance A to the corresponding overdetermined type or record instance R in connector instance B for a virtual connection graph. This function can be used at all places where a <code>connect(..)</code> statement is allowed. [This definition shall be used, if in a model with connectors A and B the overdetermined records A.R and B.R are algebraically coupled in the mode].
<code>root(A.R);</code>	The overdetermined type or record instance R in connector instance A is a (definite) root node in a virtual connection graph. [This definition shall be used if in a model with connector A the overdetermined record A.R is (consistently) assigned, e.g., from a parameter expressions]
<code>potentialRoot(A.R);</code> <code>potentialRoot(A.R, priority = prior);</code>	The overdetermined type or record instance R in connector instance A is a potential root node in a virtual connection graph with priority “prior” ($\text{prior} \geq 0$). If no second argument is provided, the priority is zero. “prior” shall be a parameter expression of type Integer. In a virtual connection subgraph without a <code>Connections.root</code> definition, one of the potential roots with the lowest priority number is selected as root [This definition is, e.g., used in a body, see <i>Parts.Bodys in Table 2</i>].
<code>b = isRoot(A.R);</code>	Returns true, if the overdetermined type or record instance R in connector instance A is selected as a root in the virtual connection graph.

Table 3. Operators “`Connections.XXX`” (e.g. `Connections.branch`) to define the set of overdetermined equations

the following way:

1. Every root node defined via the “Connections.root(...)” statement is a definite root of one spanning tree.
2. The virtual connection graph may consist of sets of subgraphs that are not connected together. Every subgraph in this set shall have at least one root node or one potential root node. If a graph of this set does not contain any root node, then **one potential root** node in this subgraph with the lowest priority number is selected to be the root of the subgraph. The selection can be inquired in a class with function Connections.isRoot(...), see Table 1.
3. If there are n selected roots in a subgraph, then breakable branches have to be removed such that the result shall be a set of n spanning trees with the selected root nodes as roots.

After this analysis, the connect(...) equations for overdetermined variables are generated in the following way:

1. For every breakable branch in one of the spanning trees, i.e., connect(A,B) statements, the usual “equality” connect equations are generated, “A.R = B.R”.
2. For every breakable branch **not** in any of the spanning trees, the equations “ $\mathbf{0} = \text{R.equalityConstraint}(A.R, B.R)$ ” are generated instead of “A.R = B.R”.

An example for a virtual connection graph is given in Figure 21. This example contains two independent subgraphs that are analyzed separately. The left subgraph has two (definite) roots. Four breakable branches, i.e., connect(...) statements have to be removed to arrive at two spanning trees. For every removed connect(...) statement the equalityConstraint(...) function is used to generate the connection equation. In the right subgraph of Figure 21 no definite root is present. Therefore, the potential root with the lowest priority has to be selected as root. If there are several roots with the same lowest priority, one of them is selected arbitrarily. Starting from the selected root, only one branch has to be removed to also arrive at a spanning tree in this subgraph.

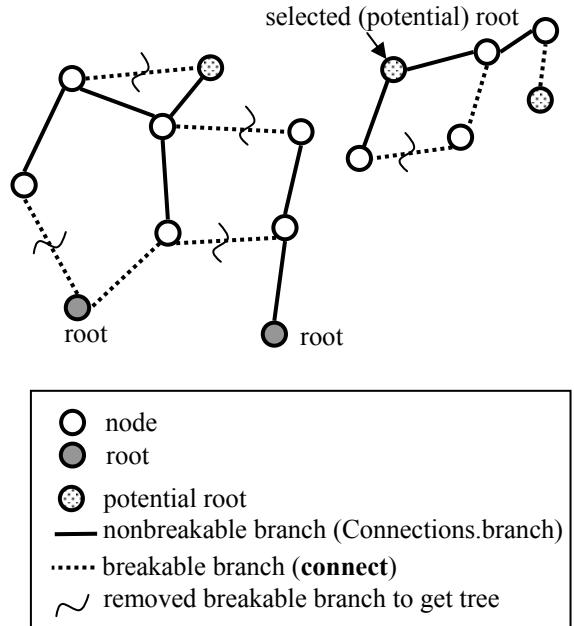


Figure 21. Example for virtual connection graph

Bibliography

- [1] ADAMS: **MSC ADAMS** at http://www.mscsoftware.com/products/quick_prod.cfm
- [2] Andreasson: **VehicleDynamics library**. Proceedings of the 3rd Int. Modelica Conference, Modelica'2003. <http://www.Modelica.org>
- [3] Beater P. (2003): **HyLib version 2.1**. <http://www.HyLib.com>
- [4] Beater P., and Otter M. (2003): **Multi-Domain Simulation: Mechanics and Hydraulics of an Excavator**. Proceedings of the 3rd Int. Modelica Conference, Modelica'2003. <http://www.Modelica.org>
- [5] Bunus B., Engelson V., and Fritzson P. (2000): **Mechanical Models Translation, Simulation and Visualization in Modelica**. Proc. of Modelica 2000 workshop, Lund, 2000. <http://www.modelica.org/-workshop2000/proceedings/Bunus.pdf>
- [6] DADS: **LMS DADS** at <http://www.lmsintl.com/>
- [7] Dynasim (2003): **Dymola Users Guide, Version 5.1**, <http://www.dynasim.se>.
- [8] Elmqvist, H. (1978): **A Structured Model Language for Large Continous Systems**. PhD-Thesis, Lund Institute of Technology, Lund, Sweden.
- [9] Engelson V. (2000): **Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing**. Linköping Studies in Science and Technology. Dissertation No 627. Department of Computer and Information Science, Linköping University (chapter 5).
- [10] Hiller M., and Woernle C. (1987): **A Systematic Approach for Solving the Inverse Kinematic Problem of Robot Manipulators**. Proceedings 7th World Congress Th. Mach. Mech., Sevilla.
- [11] Kecskemethy A. (1993): **Objektorientierte Modellierung der Dynamik von Mehrkörpersystemen mit Hilfe von Übertragungselementen**. Dissertation, VDI Fortschritt-Berichte, Reihe 20, Nr. 88.
- [12] Kecskemethy A. (1993): **Mobile - An Object-Oriented Tool-Set for the Efficient Modeling of Mechatronic Systems**. Proc. of the Second Conference on Mechatronics and Robotics, pp. 447-462, Duisburg/Moers, Sept. 27.-29. MOBILE homepage: <http://www.mechanik.tu-graz.ac.at/mobile>
- [13] Mattsson S.E., and Söderlind G. (1993): **Index reduction in differential-algebraic equations using dummy derivatives**. SIAM Journal of Scientific and Statistical Computing, Vol. 14, pp. 677-692.
- [14] Mattsson S.E., Olsson H., and Elmqvist H. (2000): **Dynamic Selection of States in Dymola**. Modelica Workshop 2000 Proceedings, pp. 61-67,
- [15] Möller M. (1992): **Ein Verfahren zur automatischen Analyse der Kinematik mehrschleifiger räumlicher Mechanismen**. Dissertation, Institut A für Mechanik der Universität Stuttgart.
- [16] Nikravesh, P.E (1988): **Computer-Aided Analysis of Mechanical Systems**. Prentice Hall.
- [17] Otter M., Elmqvist H., and Cellier F. (1996): **Modeling of MultiBody Systems with the Object-Oriented Modeling Language Dymola**. Nonlinear Dynamics, Vol. 9, pp. 91-112.
- [18] Roberson R.E., and Schwertassek R (1988): **Dynamics of Multibody Systems**. Springer Verlag.
- [19] Pantelides C. (1988): **The Consistent Initialization of Differential-Algebraic Systems**. SIAM Journal of Scientific and Statistical Computing, pp. 213-231.
- [20] PowerTrain (2002): **PowerTrain Library 1.0 – Tutorial**. DLR, www.dynasim.se/www/PowerTrainTutorial.pdf
- [21] SIMPACK: <http://www.simpack.de/>
- [22] Schweiger C., and Otter M. (2003): **Modelling 3D Mechanical Effects of 1D Powertrains**. Proceedings of the 3rd Int. Modelica Conference, Modelica'2003. <http://www.Modelica.org>
- [23] TESIS ve-DYNA: <http://www.thesis.de/en>
- [24] Woernle C. (1988): **Ein systematisches Verfahren zur Aufstellung der geometrischen Schließbedingungen in kinematischen Schleifen mit Anwendung bei der Rückwärtstransformation für Industrieroboter**. Fortschritt-Berichte VDI, Reihe 18, Nr. 59, Düsseldorf: VDI-Verlag , ISBN 3-18-145918-6.

Multi-Domain Simulation: Mechanics and Hydraulics of an Excavator

Peter Beater¹, and Martin Otter²

¹University of Applied Sciences Soest, Germany, pb@beater.de

²DLR, Oberpfaffenhofen, Germany, Martin.Otter@dlr.de

Abstract

It is demonstrated how to model and simulate an excavator with Modelica and Dymola by using Modelica libraries for multi-body and for hydraulic systems. The hydraulic system is controlled by a “load sensing” controller. Usually, models containing 3-dimensional mechanical and hydraulic components are difficult to simulate. At hand of the excavator it is shown that Modelica is well suited for such kinds of system simulations.

1. Introduction

The design of a new product requires a number of decisions in the initial phase that severely affect the success of the finished machine. Today, digital simulation is therefore used in early stages to look at different concepts. The view of this paper is that a new excavator is to be designed and several candidates of hydraulic control systems have to be evaluated.

Systems that consist of 3-dimensional mechanical and of hydraulic components – like excavators – are difficult to simulate. Usually, two different simulation environments have to be coupled. This is often inconvenient, leads to unnecessary numerical problems and has fragile interfaces. In this article it is demonstrated at hand of the model of an excavator that Modelica is well suited for these types of systems.

The 3-dimensional components of the excavator are modeled with the new, free Modelica MultiBody library (Otter et al. 2003). This allows especially to use an analytic solution of the kinematic loop at the bucket and to take the masses of the hydraulic cylinders, i.e., the “force elements”, directly into account. The hydraulic part is modeled in a detailed way, utilizing pump, valves and cylinders from HyLib, a hydraulics library for Modelica. For the control part a generic “load sensing” control system is used, modeled by a set of simple equations. This approach gives the required results and keeps the time needed for analyzing the problem on a reasonable level.

2. Modeling Choices

There are several approaches when simulating a system. Depending on the task it may be necessary to build a very precise model, containing every detail of the system and needing a lot of information, e.g., model parameters. This kind of models is expensive to build up but on the other hand very useful if parameters of a well defined system have to be modified. A typical example is the optimization of parameters of a counterbalance valve in an excavator (Kraft 1996).

The other kind of model is needed for a first study of a system. In this case some properties of the pump, cylinders and loads are specified. Required is information about the performance of that system, e.g., the speed of the pistons or the necessary input power at the pump shaft, to make a decision whether this design can be used in principle for the task at hand. This model has therefore to be “cheap”, i.e., it must be possible to build it in a short time without detailed knowledge of particular components.

The authors intended to build up a model of the second type, run it and have first results with a minimum amount of time spent. To achieve this goal the modeling language Modelica (Modelica 2002), the Modelica simulation environment Dymola (Dymola 2003), the new Modelica library for 3-dimensional mechanical systems “MultiBody” (Otter et al. 2003) and the Modelica library of hydraulic components HyLib (Beater 2000) was used. The model consists of the 3-dimensional mechanical construction of the excavator, a detailed description of the power hydraulics and a generic “load sensing” controller. This model will be available as a demo in the next version of HyLib.

3. Construction of Excavators

In Figure 1 a schematic drawing of a typical excavator under consideration is shown. It consists of a chain track and the hydraulic propel drive which is used to manoeuvre the machine but usually not during a work cycle. On top of that is a carriage

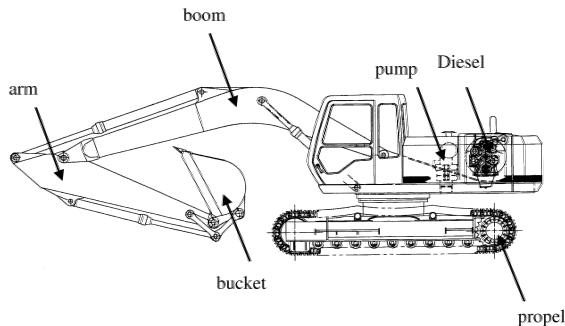


Figure 1 Schematic drawing of excavator where the operator is sitting. It can rotate around a vertical axis with respect to the chain track. It also holds the Diesel engine, the hydraulic pumps and control system. Furthermore, there is a boom, an arm and at the end a bucket which is attached via a planar kinematic loop to the arm. Boom, arm and bucket can be rotated by the appropriate cylinders.

Figure 2 shows that the required pressures in the cylinders depend on the position. For the “stretched” situation the pressure in the boom cylinder is 60 % higher than in the retracted position. Not only the position but also the movements have to be taken into account. Figure 3 shows a situation where the arm hangs down. If the carriage does not rotate there is a pulling force required in the cylinder. When rotating – excavators can typically rotate with up to 12 revolutions per minute – the force in the arm cylinder changes its sign and now a pushing force is needed. This change is very significant because now the “active” chamber of the cylinder switches and that must be taken into account by the control system. Both figures demonstrate that a simulation model must take into account the couplings between the four degrees of freedom this excavator has. A simpler model that uses a constant load for each cylinder and the swivel drive leads to erroneous results (Jansson et al. 1998).

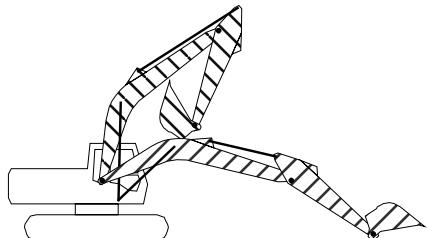


Figure 2 Different working situations

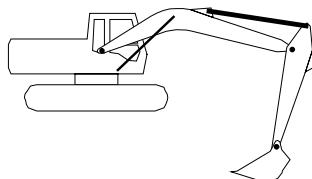


Figure 3 Effect of centrifugal forces

4. Load Sensing System

Excavators have typically one Diesel engine, two hydraulic motors and three cylinders. There exist different hydraulic circuits to provide the consumers with the required hydraulic energy. A typical design is a Load Sensing circuit that is energy efficient and user friendly. The idea is to have a flow rate control system for the pump such that it delivers exactly the needed flow rate. As a sensor the pressure drop across an orifice is used. The reference value is the resistance of the orifice. A schematic drawing is shown in figure 4, a good introduction to that topic is given in (anon. 1992).

The pump control valve maintains a pressure at the pump port that is typically 15 bar higher than the pressure in the LS line (= Load Sensing line). If the directional valve is closed the pump has therefore a stand-by pressure of 15 bar. If it is open the pump delivers a flow rate that leads to a pressure drop of 15 bar across that directional valve. Note: The directional valve is *not* used to throttle the pump flow but as a flow meter (pressure drop that is fed back) and as a reference (resistance). The circuit is energy efficient because the pump delivers only the needed flow rate, the throttling losses are small compared to other circuits.

If more than one cylinder is used the circuit becomes more complicated, see figure 5. E.g. if the boom requires a pressure of 100 bar and the bucket a pressure of 300 bar the pump pressure must be above 300 bar which would cause an unwanted

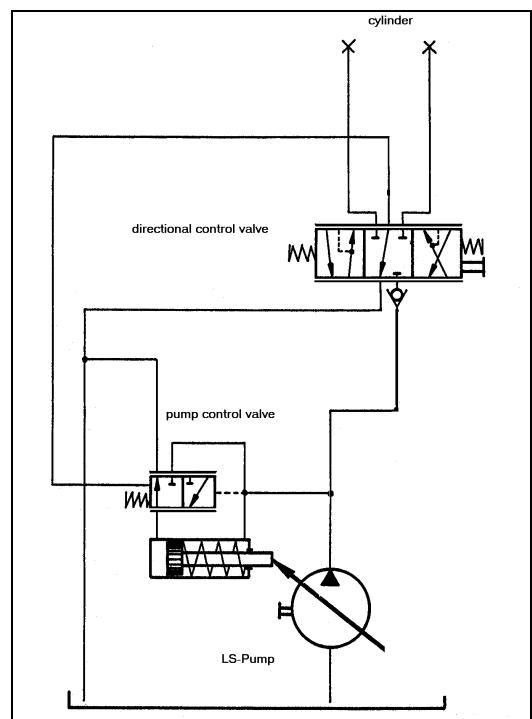


Figure 4 Schematics of a simple LS system (Zähe)

movement of the boom cylinder. Therefore compensators are used that throttle the oil flow and thus achieve a pressure drop of 15 bar across the particular directional valve. These compensators can be installed upstream or downstream of the directional valves. An additional valve reduces the nominal pressure differential if the maximum pump flow rate or the maximum pressure is reached (see e.g. Nikolaus 1994).

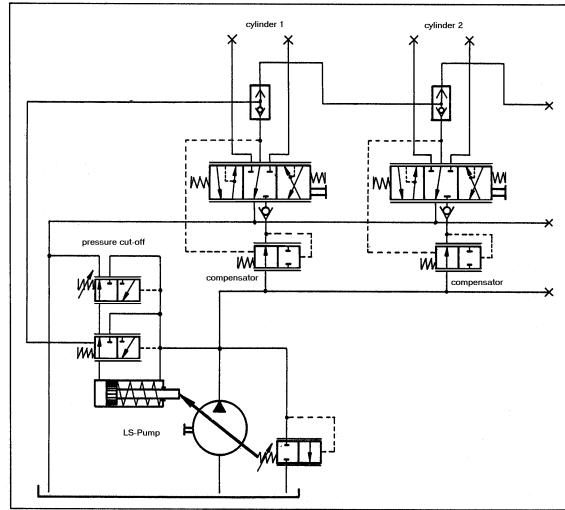


Figure 5 Schematic drawing of a LS system

5. Model of Mechanical Part

In Figure 6, a Modelica schematic of the mechanical part is shown. The chain track is not modeled, i.e., it is assumed that the chain track does not move. Components “rev1”, ..., “rev4” are the 4 revolute joints to move the parts relative to each other. The icons with the long black line are “virtual” rods that are used to mark specific points on a part, especially the mounting points of the hydraulic cylinders. The light blue spheres (b_2 , b_3 , b_4 , b_5) are bodies that have mass and an inertia tensor and are used to model the corresponding properties of the excavator parts.

The three components “cyl1f”, “cyl2f”, and “cyl3f” are line force components that describe a force interaction along a line between two attachment points. The small green squares at these components represent 1-dimensional translational connectors from the Modelica.Mechanics.Translational library. They are used to define the 1-dimensional force law acting between the two attachment points. Here, the hydraulic cylinders described in the next section are directly attached. The small two spheres in the icons of the “cyl1f”, “cyl2f”, “cyl3f” components indicate that optionally two point masses are taken into account that are attached at defined distances from the attachment

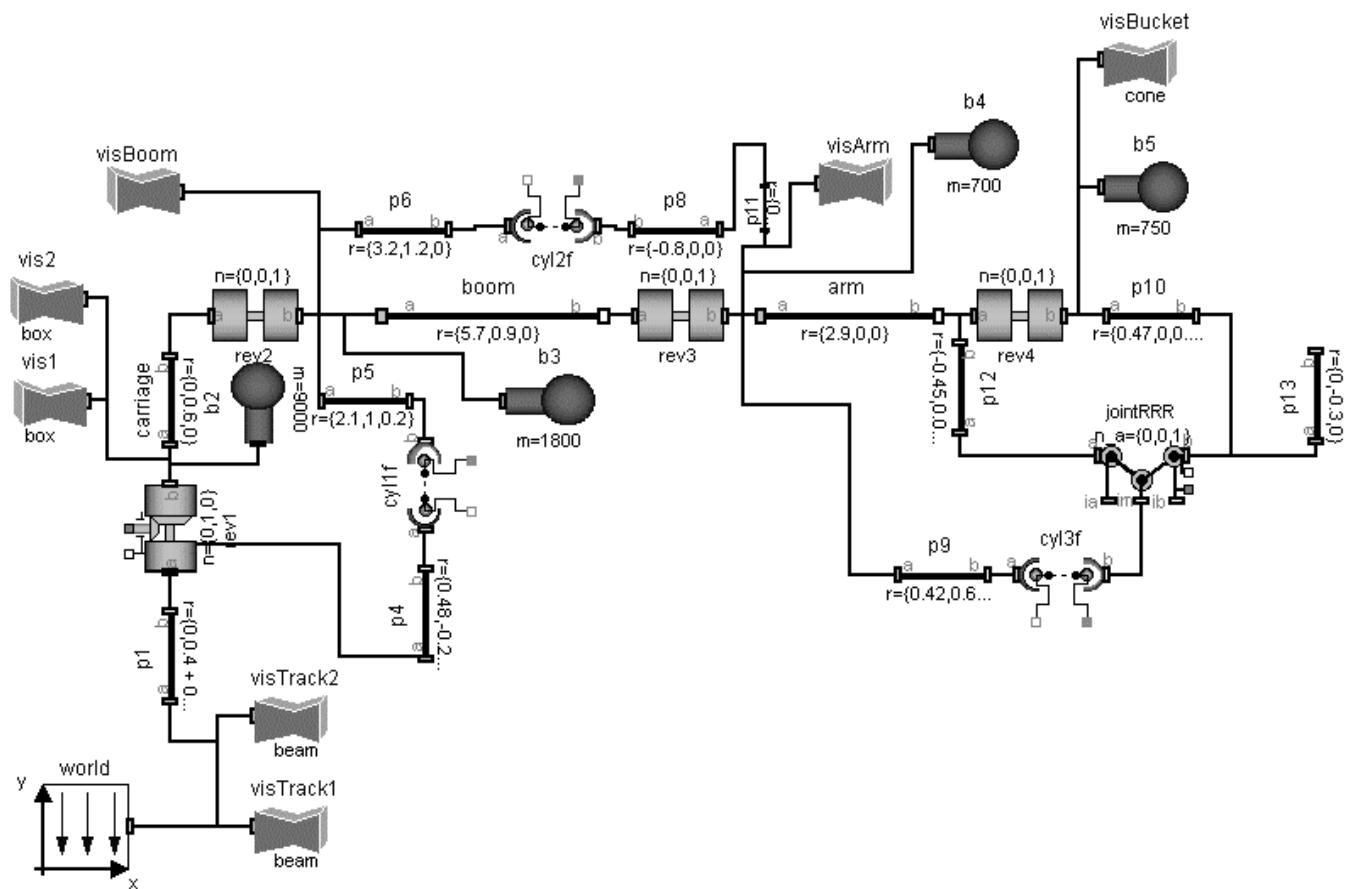


Figure 6 Modelica schematic of mechanical part of excavator

points along the connecting line. This allows to easily model the essential mass properties (mass and center of mass) of the hydraulic cylinders with only a very small computational overhead.

The jointRRR component (see right part of Figure 6) is an assembly element consisting of 3 revolute joints that form together a planar loop when connected to the arm. A picture of this part of an excavator, a zoom in the corresponding Modelica schematic and the animation view is shown in Figure 7. When moving revolute joint “rev4” (= the large red cylinder in the lower part of Figure 7; the small red cylinders characterize the 3 revolute joints of the jointRRR assembly component) the position and orientation of the attachment points of the “left” and “right” revolute joints of the jointRRR component are known. There is a non-linear algebraic loop in the jointRRR component to compute the angles of its three revolute joints given the movement of these attachment points. This non-linear system of equations is solved analytically in the jointRRR object, i.e., in a robust and efficient way. For details see (Otter et. al. 2003).

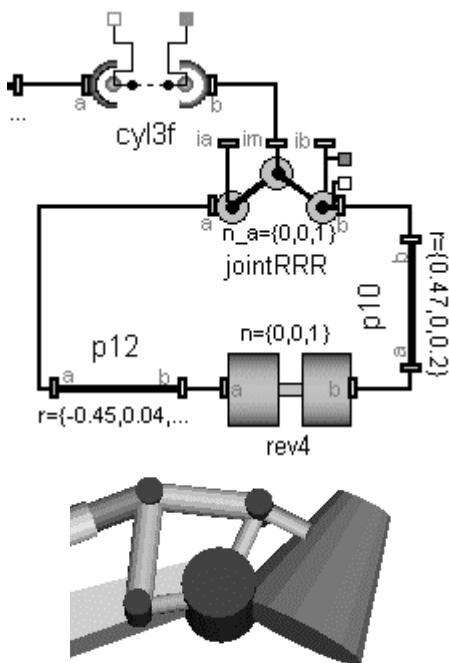


Figure 7 Foto, schematic and animation of jointRRR

In a first step, the mechanical part of the excavator is simulated without the hydraulic system to test this part separately. This is performed by attaching translational springs with appropriate spring constants instead of the hydraulic cylinders. After the animation looks fine and the forces and torques in the joints have the expected size, the springs are replaced by the hydraulic system described in the next sections.

All components of the new MultiBody library have “built-in” animation definitions, i.e., animation properties are mostly deduced by default from the given definition of the multi-body system. For example, a rod connecting two revolute joints is by default visualized as cylinder where the diameter d is a fraction of the cylinder length L ($d = L/40$) which is in turn given by the distance of the two revolute joints. A revolute joint is by default visualized by a red cylinder directed along the axis of rotation of the joint. The default animation (with only a few minor adaptations) of the excavator is shown in Figure 8.

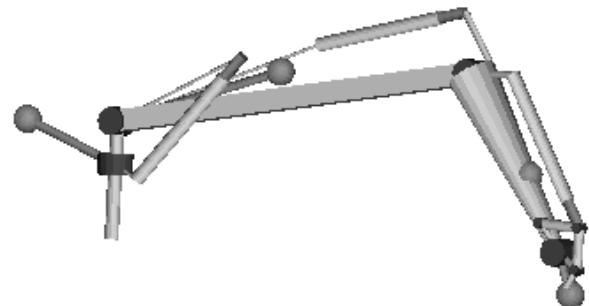


Figure 8 Default animation of excavator

The light blue spheres characterize the center of mass of bodies. The line force elements that visualize the hydraulic cylinders are defined by two cylinders (yellow and grey color) that are moving in each other. As can be seen, the default animation is useful to get, without extra work from the user side, a rough picture of the model that allows to check the most important properties visually, e.g., whether the center of masses or attachment points are at the expected places.

For every component the default animation can be switched off via a Boolean flag. Removing appropriate default animations, such as the “center-of-mass spheres”, and adding some components that have pure visual information (all visXXX components in the schematic of Figure 6) gives quickly a nicer animation, as is demonstrated in Figure 9. Also CAD data could be utilized for the animation, but this was not available for the examination of this excavator.



Figure 9 Animation of excavator (start/end position)

6. The Hydraulics Library *HyLib*

The (commercial) Modelica library *HyLib* (Beater 2000, *HyLib* 2003) is used to model the pump, metering orifice, load compensator and cylinder of the hydraulic circuit. All these components are standard components for hydraulic circuits and can be obtained from many manufacturers. Models of all of them are contained in *HyLib*. These mathematical models include both standard textbook models (e.g. Dransfield 1981, Merrit 1967, Viersma 1980) and the most advanced published models that take the behavior of real components into account (Schulz 1979, Will 1968). An example is the general pump model where the output flow is reduced if pressure at the inlet port falls below atmospheric pressure. Numerical properties were also considered when selecting a model (Beater 1999). One point worth mentioning is the fact that all models can be viewed at source code level and are documented by approx. 100 references from easily available literature.

After opening the library, the main window is displayed (Figure 10). A double click on the “pumps” icon opens the selection for all components that are needed to originate or end an oil flow (Figure 11). For the problem at hand, a hydraulic flow source with internal leakage and externally commanded flow rate is used. Similarly the needed models for the valves, cylinders and other components are chosen.

All components are modeled hierarchically. Starting with a definition of a connector – a port where the oil enters or leaves the component –

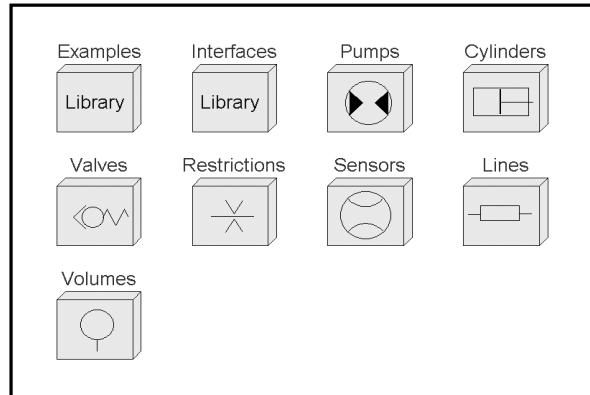


Figure 10 Overview of hydraulics library *HyLib*

a template for components with two ports is written. This can be inherited for ideal models, e.g., a laminar resistance or a pressure relief valve. While it usually makes sense to use textual input for these basic models most of the main library models were programmed graphically, i.e., composed from basic library models using the graphical user interface. Figure 12 gives an example of graphical programming. All mentioned components were chosen from the library and then graphically connected.

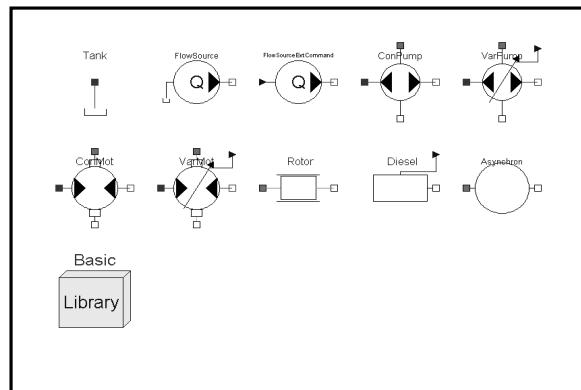


Figure 11 Pump models in *HyLib*

7. Library Components in Hydraulics Circuit

The composition diagram in Figure 12 shows the graphically composed hydraulics part of the excavator model. The sub models are chosen from the appropriate libraries, connected and the parameters input. Note that the cylinders and the motor from *HyLib* can be simply connected to the also shown components of the *MultiBody* library. The input signals, i.e., the reference signals of the driver of the excavator, are given by tables, specifying the diameter of the metering orifice, i.e. the reference value for the flow rate. From the mechanical part

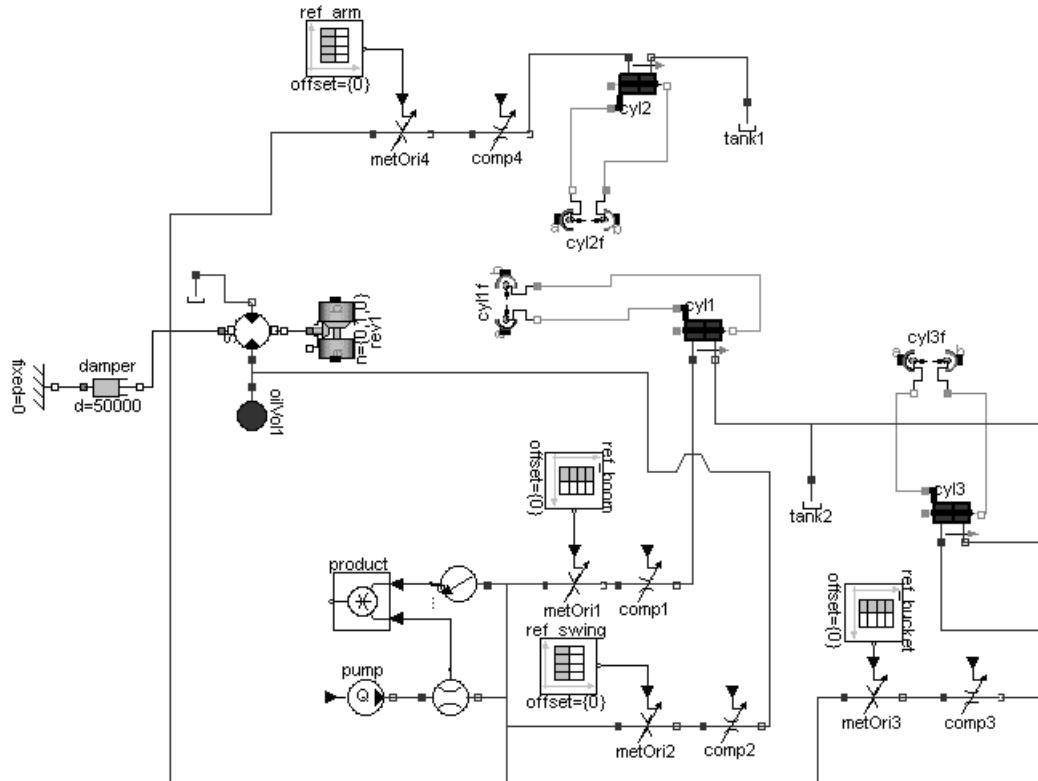


Figure 12 Modelica schematic of hydraulic part of excavator

of the excavator only the components are shown in Figure 12 that are directly coupled with hydraulic elements, such as line force elements to which the hydraulic cylinders are attached.

8. Model of LS Control

For this study the following approach is chosen: Model the mechanics of the excavator, the cylinders and to a certain extent the pump and metering valves in detail because only the parameters of the components will be changed, the general structure is fixed. This means that the diameter of the bucket cylinder may be changed but there will be exactly one cylinder working as shown in Figure 1. That is different for the rest of the hydraulic system. In this paper a Load Sensing system, or LS system for short, using one pump is shown but there are other concepts that have to be evaluated during an initial design phase. For instance the use of two pumps, or a separate pump for the swing.

The hydraulic control system can be set up using meshed control loops. As there is (almost) no way to implement phase shifting behavior in purely hydraulic control systems the following generic LS system uses only proportional controllers.

A detailed model based on actual components would be much bigger and is usually not available at the begin of an initial design phase. It

could be built with the components from the hydraulics library but would require a considerable amount of time that is usually not available at the beginning of a project.

In Tables 1 and 2, the implementation of the LS control in form of equations is shown. Usually, it is recommended for Modelica models to either use graphical model decomposition *or* to define the model by equations, but not to mix both descrip-

Table 1 Modelica code for definition for constants, parameters and variables for LS control system

```
// Definition of variables,
// parameters and constants
import SI = Modelica.SIunits;
SI.Pressure delta_p1;
SI.Pressure delta_p2;
SI.Pressure pump_ls;
SI.Pressure pump_ls1;
SI.Pressure pump_ls2;
SI.Pressure dp_ref(start = 15e5,
fixed = true);
Boolean pump_q_max;
Boolean pump_p_max(start = false,
fixed = true);

parameter Real k_LS = 1e-5;
parameter SI.Pressure p_max = 415e5
```

Table 2 Modelica code for LS Controller, see also Fig. 4 and 5

```

function conductance "Determine conductance of compensators"
  input SI.Pressure dp;
  output Real G;
algorithm
  G := min(1e-8, max(1e-13, 1e-8 - dp*5e-14));
end conductance

equation // Set of equations to model the LS controller
// define pressure differential across the metering orifices
// for load compensator and documentation purposes
delta_p1 = if ref_boom.y[1] <= 0 then pump.port_B.p
           else metOri1.port_A.p - metOri1.port_B.p;
delta_p2 = if ref_swing.y[1] <= 0 then 0.0
           else metOri2.port_A.p - metOri2.port_B.p;
delta_p3 = if ref_bucket.y[1] <= 0 then pump.port_B.p
           else metOri3.port_A.p - metOri3.port_B.p;
delta_p4 = if ref_arm.y[1] <= 0 then pump.port_B.p
           else metOri4.port_A.p - metOri4.port_B.p;

// calculate load pressure for pump controller
pump_ls1 = if ref_boom.y[1] <= 0 then pump.port_B.p
           else metOri1.port_A.p - compl.port_B.p;
pump_ls2 = if ref_swing.y[1] <= 0 then pump.port_B.p
           else metOri2.port_A.p - comp2.port_B.p;
pump_ls3 = if ref_bucket.y[1] <= 0 then pump.port_B.p
           else metOri3.port_A.p - comp3.port_B.p;
pump_ls4 = if ref_arm.y[1] <= 0 then pump.port_B.p
           else metOri4.port_A.p - comp4.port_B.p;
pump_ls = min([pump_ls1, pump_ls2, pump_ls3, pump_ls4]);

// define Boolean state for pump controller
pump_q_max = k_LS*(15e5 - pump_ls) > 8e-3;

// set Boolean state if max. pump pressure is reached (with hysteresis)
pump_p_max = pump.port_B.p > p_max or
             pre(pump_p_max) and pump.port_B.p > 0.95*p_max;

// calculate command signal for pump
pump.inPort.signal[1] = if pump_p_max then
                         min(7.5e-3, 7.5e-3 + k_LS*1e-2*(p_max - pump.port_B.p))
                     else if pump_q_max then 7.5e-3 else (k_LS*(15e5 - pump_ls));

// modify reference signal if maximum pump flow rate is exceeded
dp_ref = if pump_q_max and not pump_p_max then pump_ls else 15e5;

// calculate conductances of pressure compensators
comp1.inPort.signal[1] = conductance(delta_p1 - dp_ref);
comp2.inPort.signal[1] = conductance(delta_p2 - dp_ref);
comp3.inPort.signal[1] = conductance(delta_p3 - dp_ref);
comp4.inPort.signal[1] = conductance(delta_p4 - dp_ref);

```

tion forms on the same model level. For the LS system this is different because it has 17 input signals and 5 output signals. One might built one block with 17 inputs and 5 outputs and connect them to the hydraulic circuit. However, in this case it seems more understandable to provide the equations directly on the same level as the hydraulic

circuit above and access the input and output signals directly. For example, "metOri1.port_A.p" used in table 2 is the measured pressure at port_A of the metering orifice metOri1. The calculated values of the LS controller, e.g., the pump flow rate "pump.inPort.signal[1] = ..." is the signal at the

filled blue rectangle of the “pump” component, see Figure 12).

The strong point of Modelica is that a seamless integration of the 3-dimensional mechanical library, the hydraulics library and the non standard, and therefore in no library available, model of the control system is easily done. The library components can be graphically connected in the object diagram and the text based model can access all needed variables.

9. Some Simulation Results

The complete model was built using the Modelica modeling and simulation environment Dymola (Dymola 2003), translated, compiled and simulated for 5 s. The simulation time was 17 s using the DASSL integrator with a relative tolerance of 10^{-6} on a 1.8 GHz notebook, i.e., about 3.4 times slower as real-time. The animation feature in Dymola makes it possible to view the movements in an almost realistic way which helps to explain the results also to non-experts, see Figure 9.

Figure 13 gives the reference signals for the three cylinders and the swing, the pump flow rate and pressure. From $t = 1.1$ s until 1.7 s and from $t = 3.6$ s until 4.0 s the pump delivers the maximum flow rate. From $t = 3.1$ s until 3.6 s the maximum allowed pressure is reached.

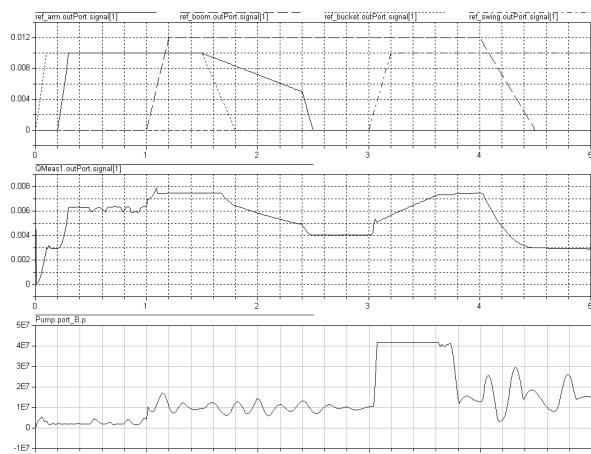


Figure 13 Reference, pump flow rate and pressure

Figure 14 gives the position of the boom and the bucket cylinders and the swing angle. It can be seen that there is no significant change in the piston movement if another movement starts or ends. The control system reduces the couplings between the consumers which are very severe for simple throttling control.

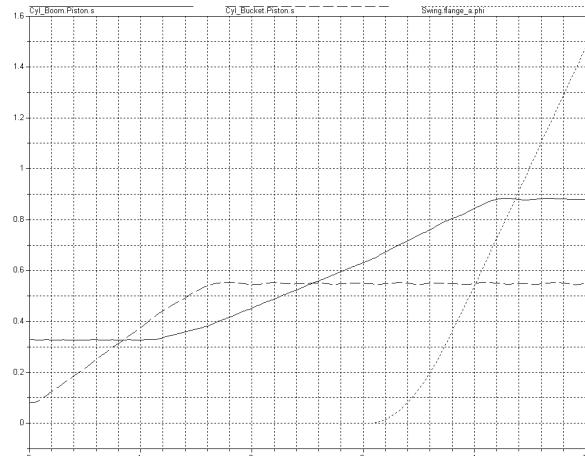


Figure 14 Boom and bucket piston position and swing angle

Figure 15 shows the operation of the bucket cylinder. The top figure shows the reference trajectory, i.e. the opening of the directional valve. The middle figure shows the conductance of the compensators. With the exception of two spikes it is open from $t = 0$ s until $t = 1$ s. This means that in that interval the pump pressure is commanded by that bucket cylinder. After $t = 1$ s the boom cylinder requires a considerably higher pressure and the bucket compensator therefore increases the resistance (smaller conductance). The bottom figure shows that the flow rate control works fine. Even though there is a severe disturbance (high pump pressure after $t = 1$ s due to the boom) the commanded flow rate is fed with a small error to the bucket cylinder.

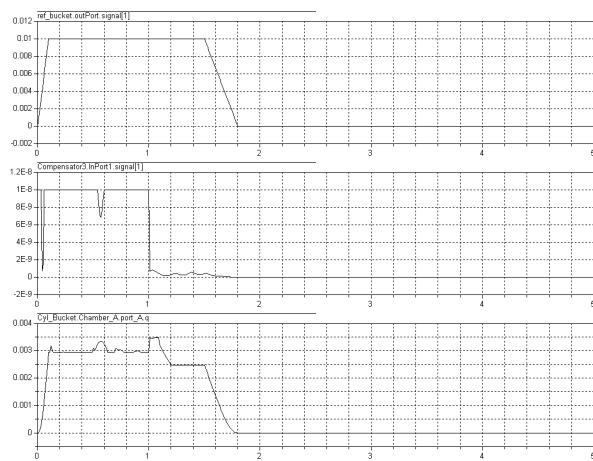


Figure 15 Operation of bucket cylinder

10. Conclusion

For the evaluation of different hydraulic circuits a dynamic model of an excavator was built. It consists of a detailed model of the 3 dimensional mechanics of the carriage, including boom, arm and bucket and the standard hydraulic components like pump or cylinder. The control system was not modeled on a component basis but the system was described by a set of nonlinear equations.

The system was modeled using the Modelica MultiBody library, the hydraulics library Hylib and a set of application specific equations. With the tool Dymola the system could be build and tested in a short time and it was possible to calculate the required trajectories for evaluation of the control system.

The animation feature in Dymola makes it possible to view the movements in an almost realistic way which helps to explain the results also to non experts.

Bibliography

- anon (1992): *Load Sensing Systems Principle of Operation*. Eaton Corporation. Hydraulics Division. Eden Prairie, MN, USA. (also under <http://www.hydraulics.eaton.com/products/pdfs/03-206.pdf>)
- Beater, P. (1999): *Entwurf hydraulischer Maschinen – Modellbildung, Stabilitätsanalyse und Simulation hydrostatischer Antriebe und Steuerungen*. Springer Verlag.
- Beater, P. (2000): *Modeling and Digital Simulation of Hydraulic Systems in Design and Engineering Education using Modelica and HyLib*. pp. 33 - 40. Modelica Workshop 2000, October 23 – 24, 2000, Lund, Sweden.
- Dransfield, P. (1981): *Hydraulic Control Systems - Design and Analysis of Their Dynamics*. Springer, Berlin, Heidelberg New York.
- Dymola (2003): *Dymola version 5.1*. <http://www.Dynasim.se>
- Elmqvist, H. (1978): *A Structured Model Language for Large Continuous Systems*. PhD-Thesis Lund Institute of Technology, Lund, Sweden.
- Jansson, A., Yahiaoui, M., and Richards, C. (1998): *Running Combined Multibody Hydraulic System Simulations within ADAMS*. International ADAMS User Conference
- HyLib (2003): *HyLib version 2.1*. <http://www.HyLib.com>
- Kraft, W. (1996): *Experimentelle und analytische Untersuchungen hydrostatischer Fahrantriebe am Beispiel eines Radbaggers*. PhD-Thesis RWTH Aachen, Germany.
- Merrit, H. E. (1967): *Hydraulic Control Systems*. New York: John Wiley & Sons.
- Modelica (2002): *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification. Version 2.0*. <http://www.Modelica.org>
- Nikolaus, H. W. (1994): *Load Sensing - lastunabhängige Dosierung von Verbraucherströmen*. Ölhydraulik und Pneumatik, o+p, 38, pp. 196- 201.
- Otter, M., Elmqvist, H., and Mattsson, S.E. (2003): *The New Modelica MultiBody Library*. Modelica'2003, 3rd International Modelica Conference, Linköping, Nov. 2-3.
- Schulz, R. (1979): *Berechnung des dynamischen Verhaltens hydraulischer Antriebe großer Leistung für Umformmaschinen*. PhD-Thesis. RWTH Aachen.
- Viersma, T. J. (1980): *Analysis, Synthesis and Design of Hydraulic Servosystems and Pipelines*. Amsterdam: Elsevier Scientific Publishing Company.
- Zähe, B., Gawlikowski, R., and Linden, D (1993): *Load Sensing als Mehrgrößenregelung*. o+p 37, pp. 638 - 644

Session 8B

Thermodynamic Systems – III

Modelling and Simulation of Heat Exchangers in Modelica with Finite Element Methods

Francesco Casella and Francesco Schiavo*

Politecnico Di Milano

Dipartimento Di Elettronica Ed Informazione (DEI)

Via Ponzio 34/5, 20133 Milano, Italy

e-mail: {casella, schiavo}@elet.polimi.it

Abstract

The complete development of a Modelica model for 1-D heat exchangers is presented. The numerical method, termed *Finite Element Method*, is briefly reviewed and its application to heat exchangers partial differential equations is presented. Implementation issues are tackled as well, and the component developed is validated through simulation within the framework of *ThermoPower*, a recently released Modelica library for thermal power plants modelling, simulation and control. The component is included into such library which is publicly available through the Web [1].

1 Introduction

The process of heat exchange between two fluids that are at different temperatures and separated by a solid wall occurs in many engineering applications. The device used to implement this exchange is termed a *heat exchanger* (HE), and specific applications may be found in space heating and air-conditioning, power generation, waste heat recovery, and chemical processing [2].

In this paper it is presented a Modelica model of the fluid side of heat exchangers, developed using a numerical method known as *finite element method*; complete models of HEs are then obtained by suitably assembling such models with metal wall and heat exchange models.

Such model is included in the power generation Modelica library *ThermoPower* [1].

The goal of this research is twofold. First, to show how Modelica can be used effectively in the modelling of physical phenomena described directly by *Partial Differential Equations* (PDEs); this aim is

achieved through the application of a specific numerical method, namely the *Finite Element Method*, which can approximate a PDE with a set of *Ordinary Differential Equations* (ODEs). Second, to amply the library of models for thermal power generation plants which has been developed here at Politecnico di Milano, offering to potential users a broader choice for the complexity and accuracy with which they would like to model some specific physical phenomena; this second aim is achieved exploiting the Modelica features for object-oriented modelling and the standardized model interfaces which have been defined within the library *ThermoPower* [1].

The paper is organized as follows: Section 2 recalls the basic physical laws for HEs; Section 3 is a brief introduction to the numerical methods used, while Section 4 shows how such methods can be used to develop models for HEs; Section 5 deals with the Modelica implementation of the model and Section 6 shows some simulation results; Finally, conclusions and perspectives on future work are given in Section 7.

2 First Principle Model

Consider a compressible fluid flowing through a pipe-shaped volume (V) with rigid boundary area and exchanging thermal energy through such boundary (figure 1).

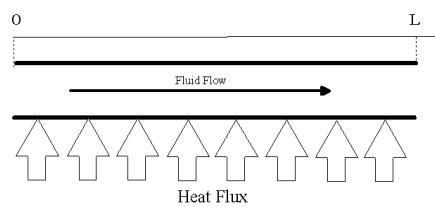


Figure 1: Heat Exchanger Scheme

*Corresponding author

Assume that

- the longitudinal dimension (x) is far more relevant than the other two;
- the volume V is “sufficiently” regular (that is V is such that the fluid motion along x is not interrupted);
- there are no phase-changes along the pipe (that is the fluid is always either single-phase or two-phase);
- the Reynolds number (Re) is such that turbulent fluid flow is assured along all the pipe, which in turn guarantees almost uniform velocity and thermodynamic state of the fluid across the radial direction;

When water or steam is assumed as the working fluid, the last hypothesis does not hold at very low flowrates; however, in practical system simulations, the plant never operates in such conditions for a long time.

It is then possible to define all the thermodynamic intensive variables with respect only to longitudinal abscissa (x) and time (t). Within this framework, the conservation equations for mass, dynamic momentum (neglecting the kinetic term) and energy (neglecting the diffusion term) can be formulated as follows:

$$A \frac{\partial p}{\partial t} + \frac{\partial w}{\partial x} = 0 \quad (1)$$

$$\frac{\partial w}{\partial t} + A \frac{\partial p}{\partial x} + \rho g A \frac{\partial z}{\partial x} + \frac{C_f}{2\rho A^2} \omega w |w| = 0 \quad (2)$$

$$\frac{\partial h}{\partial t} + w \frac{v \partial h}{A \partial x} = v \frac{dp}{dt} + v \frac{\omega}{A} \phi_{ext}, \quad (3)$$

where A is the pipe cross-section, ρ the fluid density, w the mass flow-rate, p the fluid pressure, g the gravity acceleration, z the pipe height, C_f the Fanning friction coefficient, ω the wet perimeter, h the fluid specific enthalpy, v the fluid specific volume, ϕ_{ext} the heat flux entering the pipe across the lateral surface.

3 Finite Element Methods For Time-Dependant Advection Equation

Consider the following first-order linear *partial differential equation* (PDE):

$$\begin{cases} \frac{\partial u}{\partial t} + \beta \cdot \nabla u + \sigma u = f & \text{in } \Omega \times (0, T) \\ u = g & \text{on } \partial\Omega^{in} \times (0, T) \\ u = u_0 & \text{on } \Omega \text{ for } t = 0 \end{cases} \quad (4)$$

where Ω denotes a bounded domain ($x \in \Omega$) in \mathbb{R}^m with boundary $\partial\Omega$, $T > 0$ is a prescribed time value ($t \in (0, T)$), $u(x, t)$ is the unknown (for example a temperature field), $f(x, t)$ is given function, $\beta(x, t)$ is a given velocity field, $\sigma(x, t)$ an adsorption coefficient, ∇ is the gradient operator; $u_0 = u_0(x)$ is the assigned initial datum and $g(x, t)$ is the assigned *Dirichlet* boundary condition defined on the inflow boundary $\partial\Omega^{in} = \{x \in \partial\Omega | \beta(x, t) \cdot \vec{n}(x) < 0\}$ (\vec{n} is the unit outward normal vector on $\partial\Omega$).

The equation (4) is called *time-dependant advection equation* [3] and it can represent the energy equation (3) for heat exchangers.

In the following, for the sake of simplicity, the equation (4) will have the form $\frac{\partial u}{\partial t} + Lu = f$, where L is the proper differential operator.

The approximated solution of the PDE (4) can be obtained through several numerical methods; on the other side, only methods that allow to transform a PDE into a set of ordinary differential equations (ODEs) or differential-algebraic equations (DAEs) with respect to time are suitable to use within the Modelica framework. Within this paper the focus is on a numerical method termed *Finite Element Method* (FEM) [3],[4]. Other interesting methods for the approximation of PDE (4) are the *Finite Difference Method* (FDM) and *Finite Volume Method* (FVM) [3], [5]. The advantage of using FEM instead of FVM or FDM is that it can provide more accurate solution or, in specific cases, the exact nodal values for the unknown [3].

The FEM is based on the discretization of the solution region into elementary elements. The unknown variable u is expressed in terms of assumed *approximating* or *interpolation* functions within each element. The interpolation functions are local, i.e. functions defined over smaller sub-domains, where these sub-domains extend over a few elements, and are zero everywhere else. The local interpolation functions are ordinarily very simple functions, such as low-degree polynomials. The interpolation functions are defined in terms of the values of the variable at specified points called *nodes*. Nodes usually lie on the element boundaries where adjacent elements are considered to be connected. In addition to boundary nodes, an element may also have a few interior nodes. The nodal values u_i of the variable and the interpolation functions for the elements completely define the behavior of the variable within the elements. For the finite element representation of a problem, the nodal values of the variable become the new unknowns. Once these unknowns are found, the interpolation functions define the variable

throughout the assemblage of elements. Clearly, the nature of the solution and the degree of approximation depend not only on the size and number of the elements used, but also on the interpolation functions selected [3].

3.1 The Method of Weighted Residual

The *Method of Weighted Residual* (MWR) is a mathematical technique for obtaining finite element equations from linear and non-linear PDEs. Referring to (4), the problem solved by the MWR is to find the nodal values of an approximated solution ($u_h(x, t)$) so as to make an error (called residual)

$$R_h(x, t) = \frac{\partial u_h(x, t)}{\partial t} + L u_h(x, t) - f(x, t) \quad (5)$$

small over the entire solution domain Ω , i.e.

$$\int_{\Omega} R_h v_h d\Omega \approx 0, \quad \forall v_h \in V_h, \quad (6)$$

where $v_h(x)$ are linearly independent *weighting functions* (as many as the nodal points) belonging to an appropriate finite dimensional space V_h . The *Petrov-Galerkin* methods used in the HE model development belong to this family.

3.2 Finite Element Basis Function and Space

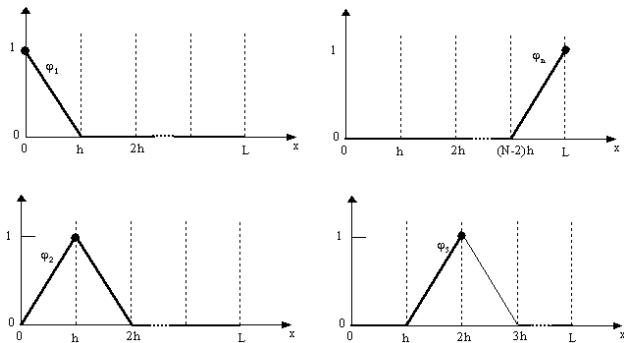


Figure 2: The “triangular” basis functions $\phi_j(x)$

The solution domain Ω is decomposed into elements K of mesh size h_K . The *finite element space* X_h^k is the finite dimension space of continuous piecewise-polynomial functions of degree k defined within each element K . The basic idea of the FEM is therefore to approximate the infinite dimensional solution, belonging to a infinite dimension space X , with a finite dimensional one, belonging to X_h^k (whose size will be

called N). In figure 2 the piece-wise linear ($k = 1$) interpolating functions are depicted. The space of *interpolating functions* will be called hereafter W_h and its interpolating functions $\phi_i(x)$. Then the *approximated solution* $u_h(x, t)$ of u is expressed as

$$\begin{aligned} u_h(x, t) &= \sum_{i=1}^N u_i(t) \phi_i(x) \quad \text{for } t > 0 \\ u_{0,h}(x) &= \sum_{i=1}^N u_{0,i} \phi_i(x) \quad \text{for } t = 0 \end{aligned} \quad (7)$$

3.3 Petrov-Galerkin Methods

In the following, for the sake of simplicity, the inner functional product notation $(u, v) = \int_{\Omega} uv dx$ is used. In addition the way boundary conditions are enforced into the approximated equation is not included, since it is presented separately later.

By expanding (6) and properly choosing the weighting function space, the *Petrov-Galerkin* (PG) approximation of the PDE problem (4) consists in finding $u_h \in W_h$ such that

$$\frac{d}{dt}(u_h, v_h) + (Lu_h, v_h) = (f, v_h) \quad \forall v_h \in V_h \quad (8)$$

with $W_h \neq V_h$ but $\dim(W_h) = \dim(V_h) = N, \forall h > 0$. Equation (8) has to be satisfied for any $v_h \in V_h$, that is it has to be satisfied for all the functions of any basis of the space V_h itself; the basis functions of V_h will be denoted as $\{\psi_i | i = 1 \dots N\}$. The functional space V_h is termed the space of *test* or *weighting* functions. Then being $\{\phi_j | j = 1 \dots N\}$ a basis for the space W_h , and substituting (7) into (8), it can be obtained a set of N ODEs for the unknown vector $U(t)$:

$$\mathbf{M} \frac{dU(t)}{dt} + \mathbf{A} U(t) = \mathbf{F}(t), \quad U(0) = U_0, \quad (9)$$

Where $U(t) = [u_i(t)]$, $F(t) = [(f, \psi_i)]$, $U_0 = [u_{0,i}]$, $A_{ij} = (L\phi_j, \psi_i)$, $M_{ij} = (\phi_j, \psi_i)$, for $i, j = 1 \dots N$. The matrix \mathbf{M} and \mathbf{A} are called the *mass* and *stiffness* matrix respectively.

General assumptions guarantee the existence and uniqueness of a solution [3].

The (standard) *Galerkin* method is a particular case of the PG one, where the test functions space (V_h) is chosen to be the same as the approximating functions space (W_h), therefore $M_{ij} = (\phi_j, \phi_i)$, $A_{ij} = (L\phi_j, \phi_i)$, $F_i(t) = (f, \phi_i)$.

The application of the standard Galerkin method to advection dominated problems (as the one considered) could lead to solutions with oscillatory behavior due

to numerical instabilities. To overcome such a problem it is possible to use a *stabilized Petrov-Galerkin method*.

The basic idea of stabilization methods is to relate the functional space V_h to W_h through a differential operator L_h somewhat related to the differential operator L , that is $V_h = \{w_h + L_h w_h | w_h \in W_h\}$. The equation (8) thus becomes

$$\begin{aligned} \frac{d}{dt}(u_h, w_h + L_h w_h) + (Lu_h, w_h + L_h w_h) &= \\ &= (f, w_h + L_h w_h) \quad \forall w_h \in W_h \end{aligned} \quad (10)$$

3.4 Treatment of Boundary Conditions

The boundary conditions (BCs) can be imposed in two different ways:

1. *Strong formulation (sf)*: the boundary conditions are enforced directly in the definition of the space W_h of the admissible solutions, while the *test functions* vanish on the boundary. The boundary conditions are satisfied at all nodes lying on $\partial\Omega^{in}$.
2. *Weak formulation (wf)*: the boundary conditions are enforced indirectly in the unknown nodal values of the approximated equation. The boundary conditions is not imposed exactly at all nodes of $\partial\Omega^{in}$, but a suitable linear combination between them and the residual of the PDE is enforced. Therefore the problem formulation becomes: for any $t \in [0, T]$ find $u_h \in W_h$ such that

$$\left\{ \begin{array}{l} \frac{d}{dt}(u_h, v_h) + (Lu_h, v_h) - \int_{\partial\Omega^{in}} \beta \cdot \vec{n} u_h v_h d\gamma \\ = (f, v_h) - \int_{\partial\Omega^{in}} \beta \cdot \vec{n} g_h v_h d\gamma \quad \forall v_h \in V_h \\ u_h(0) = u_{0,h} \end{array} \right. \quad (11)$$

It is important to note that the additional integral terms can be easily computed for the one-dimensional case since $\partial\Omega^{in}$ is a finite set of points (at most two: $x = 0$ and $x = L$).

The main differences of the two boundary condition formulations are:

- In the *wf* the nodal values on the boundary are unknown and therefore the number of finite element equations to be solved is higher than that obtained from the strong formulation.

- In the case of flow reversal (change of β sign in equation 4) the inflow boundary changes. In the *wf* the state variables (i.e. the nodal values) are always the same since the nodal values on the boundary are also problem unknowns. Instead, in the *sf*, the nodal values on the boundary are known, so that the state variables depend on the flow direction.

In the model developed the choice has been to adopt the *wf* since it can be accurate as *sf* while providing easier implementation in the case of flow reversal [5].

4 FEM Model for Heat Exchangers

In this section it will be shown how the numerical methods introduced can be applied to the balance equations so to transform them into a set of ODEs that can be used directly in Modelica models.

The spatial domain $([0, L])$ has been divided into a grid of uniformly spaced elements with size $l = L/(N-1)$, where $N (\geq 2)$ is the number of finite elements that are going to be used.

The interpolating functions have been chosen to be linear (figure 2); their analytical expression is

$$\begin{aligned} \varphi_1(x) &= \begin{cases} \frac{l-x}{l} & 0 < x \leq l \\ 0 & \text{otherwise} \end{cases} \\ \varphi_N(x) &= \begin{cases} \frac{x-(N-2)l}{l} & (N-2)l < x \leq L \\ 0 & \text{otherwise} \end{cases} \\ \varphi_i(x) &= \begin{cases} \frac{x-(i-2)l}{l} & (i-2)l < x \leq (i-1)l \\ \frac{il-x}{l} & (i-1)l < x \leq il \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (12)$$

In the following the notation $\bar{\varphi} = [\varphi_1 \cdots \varphi_N]^T$ will be used.

The *stabilized Petrov-Galerkin Method* termed *GALS (Galerkin/Least-Squares)*, which has been proven to be the most suitable one for the advection dominated case [6], has been used to obtain the test functions:

$$\psi_j(x) = \varphi_j(x) + \alpha \frac{l}{2} \frac{d\varphi_j(x)}{dx}, \quad j = 1 \dots N \quad (13)$$

where α is a stabilization coefficient ($0 \leq \alpha \leq 1$); for $\alpha = 0$ the standard (i.e. non stabilized) method can be obtained.

The following hypothesis have been taken into account in the finite element formulation:

- h linear on each element
- T linear with h
- v linear with h
- ϕ_{ext} linear on each element
- w uniform along the HE
- p uniform along the HE
- p, h, w are the state variables

That means h, T, v, ϕ_{ext} can be expressed as

$$\begin{aligned} h(x, t) &= \sum_{i=1}^N h_i(t) \varphi_i(x) = \bar{h}(t)^T \bar{\varphi}(x), \quad \bar{h} = [h_1 \cdots h_N]^T \\ T(x, t) &= \sum_{i=1}^N T_i(t) \varphi_i(x) = \bar{T}(t)^T \bar{\varphi}(x), \quad \bar{T} = [T_1 \cdots T_N]^T \\ v(x, t) &= \sum_{i=1}^N v_i(t) \varphi_i(x) = \bar{v}(t)^T \bar{\varphi}(x), \quad \bar{v} = [v_1 \cdots v_N]^T \\ \phi_{ext}(x, t) &= \sum_{i=1}^N \phi_i(t) \varphi_i(x) = \bar{\phi}(t)^T \bar{\varphi}(x), \quad \bar{\phi} = [\phi_1 \cdots \phi_N]^T \end{aligned} \quad (14)$$

The considered hypotheses do not affect the generality of the model, at least if there aren't any phase changes along the HE.

In the balance equations both the fluid density specific volume are involved, and their relation is well known to be $\rho = 1/v$; since v has been assumed to be linear with h (which is linear on each element), it should result $\rho = (\sum_{i=1}^N v_i \varphi_i)^{-1}$, that is ρ is not linear with h . As a matter of fact, for the sake of simplicity, it has been assumed that also ρ can be expressed as

$$\begin{aligned} \rho(x, t) &= \sum_{i=1}^N \rho_i(t) \varphi_i(x) = \bar{\rho}(t)^T \bar{\varphi}(x), \quad \bar{\rho} = [\rho_1 \cdots \rho_N]^T \\ \text{with } \rho_i &= (v_i)^{-1} \quad \forall i = 1 \cdots N \end{aligned} \quad (15)$$

It can be shown that the error introduced by this approximation (computed as $\int_0^h (v^{-1} - \rho) dx$) is $O(h)$.

Among the balance equations, the mass and dynamic momentum ones describe the fast pressure and flow rate dynamics, while the energy one describes the slower dynamics of heat transport with the fluid velocity; the most relevant phenomenon, for power generation plant modelling, is the latter one, so that the equation (3) has been discretized with a fine approximation through FEMs, while equations (1)-(2) have been treated with a coarser approximation.

4.1 Energy Balance Equation

Consider the energy balance equation for the HE:

$$\frac{\partial h}{\partial t} + w \frac{v \partial h}{A \partial x} = v \frac{dp}{dt} + v \frac{\omega}{A} \phi_{ext} \quad (16)$$

with reference to the advection equation (4) used in the finite element formulation, it results $\beta = w \frac{v}{A}$ and $\sigma = 0$, while the term f is simply the right hand side of the equation.

The application of a PG method, with weakly imposed boundary conditions, leads to a set of N ODEs:

$$\begin{aligned} &\int_0^L \left(\sum_{i=1}^N \dot{h}_i \varphi_i \right) \psi_j dx + \int_0^L \left(\frac{w}{A} \sum_{i=1}^N v_i \varphi_i \sum_{i=1}^N h_i \frac{d\varphi_i}{dx} \right) \psi_j dx \\ &\psi_j dx + \int_{\partial\Omega^{in}} \left(\frac{w}{A} \sum_{i=1}^N v_i \varphi_i \sum_{i=1}^N h_i \varphi_i \right) \psi_j dx = \\ &\int_0^L \sum_{i=1}^N v_i \varphi_i \left(\dot{p} + \frac{\omega}{A} \sum_{i=1}^N \phi_i \varphi_i \right) \psi_j dx + \\ &+ \int_{\partial\Omega^{in}} \left(\frac{w}{A} \sum_{i=1}^N v_i \varphi_i h_{in} \right) \psi_j dx, \quad \forall \psi_j \in V_h \end{aligned} \quad (17)$$

where h_{in} is the fluid specific enthalpy at the inflow boundary. Such set of ODEs can be easier represented with the following differential matrix equation:

$$M \dot{h} + \frac{w}{A} B \bar{h} + \frac{w}{A} C \bar{h} = \dot{p} M \bar{v} + \frac{\omega}{A} Y \bar{\phi} + \frac{w}{A} K \bar{v}, \quad (18)$$

where M, B, C, Y, K are defined as

$$\begin{aligned} M_{ji} &= \int_0^L \varphi_i \psi_j dx \\ B_{ji} &= \int_0^L \left(\sum_{k=1}^N v_k \varphi_k \right) \frac{d\varphi_i}{dx} \psi_j dx \\ C_{ji} &= \int_{\partial\Omega^{in}} \left(\sum_{k=1}^N v_k \varphi_k \right) \varphi_i \psi_j dx \\ Y_{ji} &= \int_0^L \left(\sum_{k=1}^N v_k \varphi_k \right) \varphi_i \psi_j dx \\ K_{ji} &= \int_{\partial\Omega^{in}} h_{in} \varphi_i \psi_j dx \end{aligned} \quad (19)$$

The detailed expressions for the matrices M, B and Y are reported in appendix A, while the matrices C and K (which express the BCs) will be analyzed thoroughly in the next section.

4.2 Mass Balance Equation

Consider the mass balance equation for the HE:

$$A \frac{\partial \rho}{\partial t} + \frac{\partial w}{\partial x} = 0 \quad (20)$$

Since pressure (p) and specific enthalpy (h) have been chosen as the thermodynamic state variables, it results

$$\frac{\partial \rho}{\partial t} = \frac{\partial \rho}{\partial h} \frac{\partial h}{\partial t} + \frac{\partial \rho}{\partial p} \frac{\partial p}{\partial t} \quad (21)$$

Substituting in such equation the expression reported in (14) for h and ρ , it follows

$$\frac{\partial \rho}{\partial t} = \bar{\rho}_h (\bar{\varphi} \bar{\varphi}^T) \dot{h} + \dot{p} \bar{\rho}_p \bar{\varphi} \quad (22)$$

where $\bar{\rho}_h = [\frac{\partial \rho_1}{\partial h}|_{h_1,p} \dots \frac{\partial \rho_N}{\partial h}|_{h_N,p}]$ and $\bar{\rho}_p = [\frac{\partial \rho_1}{\partial p}|_{h_1,p} \dots \frac{\partial \rho_N}{\partial p}|_{h_N,p}]$

Then, integrating the mass balance equation along the spatial domain, it results

$$\int_0^L \frac{\partial \rho}{\partial t} dx = -\frac{1}{A} \int_0^L \frac{\partial w}{\partial x} dx, \quad (23)$$

leading to the ODE

$$\bar{\rho}_h^T E \dot{h} + \dot{p} \bar{\rho}_p^T D = \frac{1}{A} (w_0 - w_L), \quad (24)$$

where w_0 and w_L are the fluid mass flow-rate at abscissa 0 and L respectively; E and D are a matrix and a vector (details can be found in appendix A):

$$E_{ji} = \int_0^L \varphi_i \varphi_j dx, \quad D_i = \int_0^L \varphi_i dx \quad (25)$$

4.3 Dynamic Momentum Equation

Consider the dynamic momentum balance equation for the HE:

$$\frac{\partial w}{\partial t} + A \frac{\partial p}{\partial x} + \rho g A \frac{dz}{dx} + v \frac{C_f \omega}{2A^2} w|w| = 0 \quad (26)$$

Substituting the expression reported in (14) for ρ and v and integrating along the spatial domain, the following expressions result (dz/dx is assumed as a constant parameter):

$$\begin{aligned} & \int_0^L \frac{\partial w}{\partial t} dx + \int_0^L A \frac{\partial p}{\partial x} dx + \int_0^L g A \frac{dz}{dx} \sum_{i=1}^N \rho_i \varphi_i dx + \\ & + \int_0^L \frac{C_f \omega}{2A^2} w|w| \sum_{i=1}^N v_i \varphi_i dx = 0, \end{aligned} \quad (27)$$

leading to the ODE

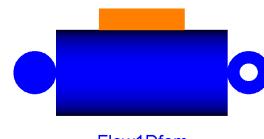
$$\begin{aligned} L \dot{w} + A (p_L - p_0) + g A \frac{dz}{dx} \bar{\rho}^T D + \\ + \frac{C_f \omega}{2A^2} w|w| \bar{v}^T D = 0, \end{aligned} \quad (28)$$

Assuming the Reynolds number is sufficiently high, C_f is approximately constant; for medium-range values of Re , it can be computed with Colebrook's equation. When dealing with water/steam flow in industrial plants, the transition and laminar regimes correspond to very low pressure drops, which need not be computed with high accuracy; therefore, a minimum value of $Re = 2100$ is assumed. Last, but not least, a small linear friction term is added to enhance numerical stability at low or zero flowrate; the parameter w_0 should be much smaller than the nominal flowrate, so that the added term is negligible during normal operation. Thus equation 28 becomes

$$\begin{aligned} L \dot{w} + A (p_L - p_0) + g A \frac{dz}{dx} \bar{\rho}^T D + \\ + \frac{C_f \omega}{2A^2} w(|w| + w_0) \bar{v}^T D = 0. \end{aligned} \quad (29)$$

5 Modelica Implementation

The developed model has been implemented in a component called `Flow1Dfem` (figure 3) which is part of the library *ThermoPower* [1].



Flow1Dfem

Figure 3: The Modelica Model

For the present model, it has been assumed that the fluid inside the HE is a water/steam mixture. The medium models used for water and steam are provided by the free "ThermoFluid" library [7].

The component is perfectly interchangeable with the actual *ThermoPower* component for 1-D HEs, since it has the same connectors: two flanges for water/steam flow and a terminal for heat flux. Here the definition of such interfaces is reported, for further details see [1]:

```
connector WaterFlangeA
  Pressure p;
  flow MassFlowRate w;
  input SpecificEnthalpy hBA;
```

```

output SpecificEnthalpy hAB;
end WaterFlangeA;

connector WaterFlangeB
  Pressure      p;
  flow MassFlowRate w;
  input SpecificEnthalpy hAB;
  output SpecificEnthalpy hBA;
end WaterFlangeB;

connector DHT
  parameter Integer N;
  Temperature T[N];
  flow HeatFlux phi[N];
end DHT;

```

In the code h_{AB} and h_{BA} are the fluid specific enthalpies in case its direction is from an A-type flange to a B-type one and viceversa. Such connectors support flow reversal. In the model there is a connector `infl` of type `WaterFlangeA` (corresponding to $x = 0$) and a connector `outfl` of type `WaterFlangeB` corresponding to $x = L$.

The model offers many customization possibilities through parameters: the HE geometry can be fully specified (length, diameter, height); the dynamic momentum term $\partial w / \partial t$ can be switched off to avoid fast pressure oscillations; the C_f coefficient can be either constant or computed by the Colebrook equation; the compressibility effect deriving from the discretization of equation (1) can be associated to either the upstream or downstream pressure; the numerical stabilization coefficient α can be chosen in the interval $[0, 1]$.

It should be noted that the matrices M , B , Y , E and the vector D are completely defined once the parameter α has been chosen; thus they can be computed once for all before the simulation starts by efficient Modelica compilers. The definition of such matrices is made thought some loops, as showed below:

```

M[1, 1] = 1/3 - 1*alfa/4;
M[N, N] = 1/3 + 1*alfa/4;
M[1, 2] = 1/6 - 1*alfa/4;
M[N, (N - 1)] = 1/6 + 1*alfa/4;
if N > 2 then
  for i in 2:N - 1 loop
    M[i, i - 1] = 1/6 + 1*alfa/4;
    M[i, i] = 2*1/3;
    M[i, i + 1] = 1/6 - 1*alfa/4;
    M[1, i + 1] = 0;
    M[N, i - 1] = 0;
    for j in 1:(i - 2) loop
      M[i, j] = 0;
    end for;
    for j in (i + 2):N loop
      M[i, j] = 0;
    end for;
  end for;
end if;

```

It can be noticed that many of the matrices entries are zeros, so it could appear that the use of a matrix

notation for the balance equations could increase the computational burden; nevertheless, it has been discovered (by direct inspection of the generated C code) that efficient compilers can simplify the set of ODEs obtained expanding the differential matrix equations in the Modelica code, removing the terms corresponding to the zero entries in the matrices.

5.1 Boundary Conditions and Flow Reversal

One of the most relevant features of the model is the capability to handle not only flow reversal in the HE, but also the most “unusual” transients for what concerns flow, that is the model is able to handle also transient where the fluid is entering or exiting from both the extremities (which are operating conditions which can be experienced when suddenly decreasing or increasing the heat-flux).

The matrices C and K , enforcing the boundary conditions into equation (18), depend on the inflow boundary $\partial\Omega^{in}$. It can be noted that, in the 1-D case, the inflow boundary can be constituted at most by the points $x = 0$ and $x = L$, depending on the fluid mass-flow rate direction in that specific direction.

Suppose, for example, that the inflow boundary is just $x = 0$ (that means `infl.w > 0` and `outfl.w < 0`). Considering the analytical expression for C and K and for the interpolating and weighting function, it results

$$\begin{aligned} C_{ij} &= \int_{x=0} \left(\sum_{k=1}^N v_k \varphi_k \right) \varphi_i \psi_j dx = \\ &= \begin{cases} (1 - \frac{\alpha}{2})v_1 & \text{if } i = j = 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

The same happens if the inflow boundary is $x = L$: the only non-zero entries for the matrices C and K can be $(1, 1)$ and (N, N) . The code for such entries is obtained through simple conditional equations:

```

C[1, 1] = if (infl.w >= 0) then
            (1 - alfa/2)*v[1, 1] else 0;
C[N, N] = if (outfl.w >= 0) then
            (1 + alfa/2)*v[N, 1] else 0;

K[1, 1] = if (infl.w >= 0) then
            (1 - alfa/2)*infl.hBA else 0;
K[N, N] = if (outfl.w >= 0) then
            (1 + alfa/2)*outfl.hAB else 0;

```

6 Simulations

The component has been tested with other models from the library *ThermoPower* using *Dymola* simulation environment [8]; specific configurations have

been set up in order to investigate the model behaviour with respect to the single balance equations and to their interactions in the most common layouts found in power plants. Many simulations have been carried out but, for the sake of brevity, only the most significative ones are reported here; all the test set ups are included in the library and are available on-line [1].

In all the reported simulations, the HE has a length of 10 m and radius 1 cm . All the simulations use $N = 20$ nodes.

The first simulation reported is aimed at testing the energy balance equation; the experimental layout is depicted in figure 4: the HE (*hex*) is connected with a mass flow rate source, an external source of heat flow, a valve (which accounts for head losses) and a sink with fixed pressure.

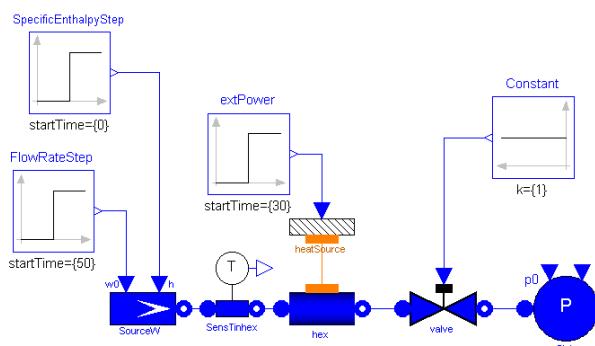


Figure 4: First Experiment Layout

The fluid involved in the experiment is liquid water at temperatures between 297 K and 322 K , the pressure inside the HE during the experiment varies from 1.6^5 Pa to 2^5 Pa and the mass flow rate is comprised in the interval $0.2 - 0.3\text{ Kg/s}$.

At the initial time instant there is a step variation from 10^5 J/m^3 to $1.42 \cdot 10^5\text{ J/m}^3$ of the specific enthalpy for the fluid of the flow rate source; at time 30 s there is a step variation of the energy flux entering the HE from 0 to $1.25 \cdot 10^4\text{ W/m}^2$; at time 50 s there is a step variation in the source mass flow rate from 0.3 Kg/s to 0.2 Kg/s .

The temperature of the fluid at the end of the HE is reported in figure 5. The exact solution (assuming ρ constant) for the underlying PDE would lead to a temperature step variation at time $t = 10\text{ s}$ and ramp variations at time $t = 30\text{ s}$ and $t = 50\text{ s}$; the simulation results show good accordance with such behavior.

The second experiment is aimed at testing the mass balance equation; the experimental layout, similar to the first one, is depicted in figure 6.

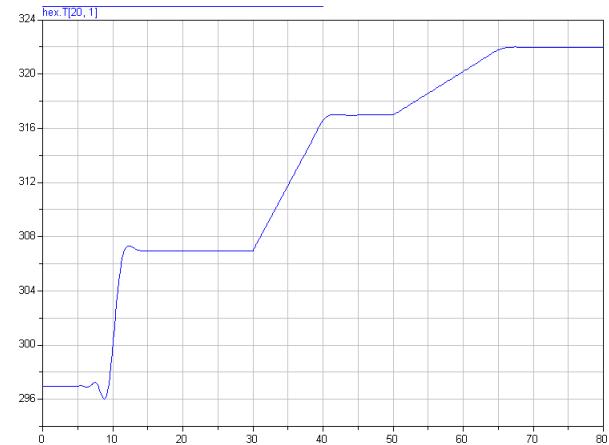


Figure 5: HE Outlet Temperature

The fluid involved in this experiment is superheated vapor with temperature and pressure at about 536 K and 10^5 Pa respectively; the mass flow rate flowing through the HE is about 10^{-2} Kg/s .

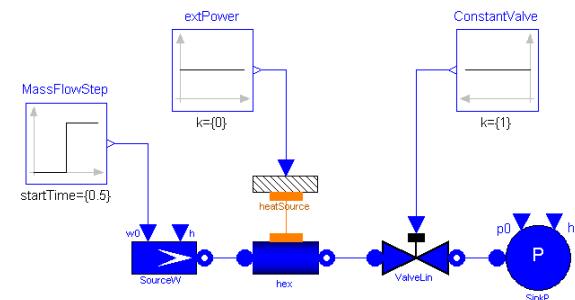


Figure 6: Second Experiment Layout

At time 0.5 s there is a 10% step increment of the mass flow rate; the consequent HE pressure transient is depicted in figure 7.

The solution of the equations for such experimental setup, assuming uniform gas properties and ideal gas content, would lead to a first order transient whose time constant is in good accordance with the simulation results.

The last test reported here involves a two side HE (*hexA* and *hexB*) in counterflow configuration (figure 8). The two fluid sides are separated by a metal wall 1 mm thick.

The operating fluid is liquid water with temperature in the range $296\text{ K} - 321\text{ K}$ and pressure about $3 \cdot 10^5\text{ Pa}$. The experiment setup is such that the mass flow rates for the two HE sides have the same value (0.31 Kg/s) with residence time 9.9 s .

At time 50 s there is a step variation from 10^5 J/m^3 to

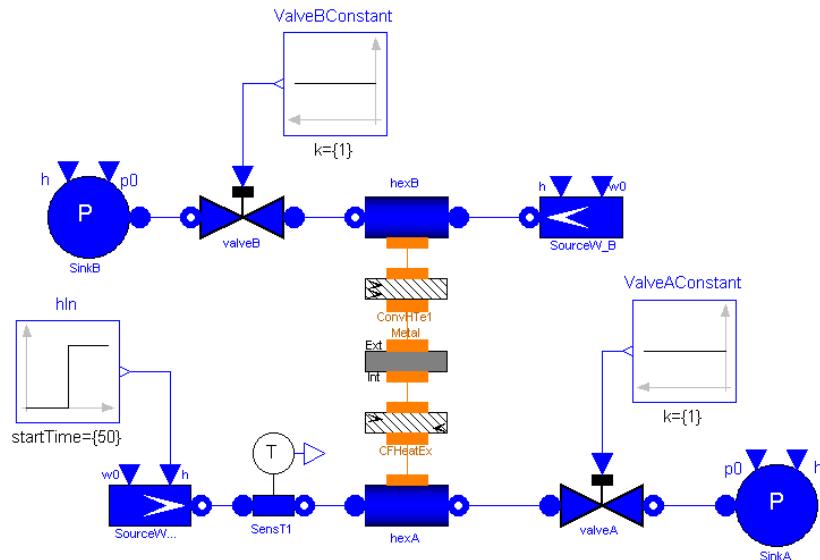


Figure 8: HE Counterflow Configuration

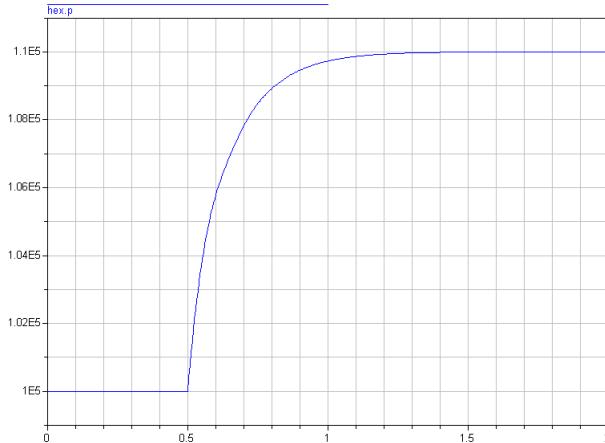
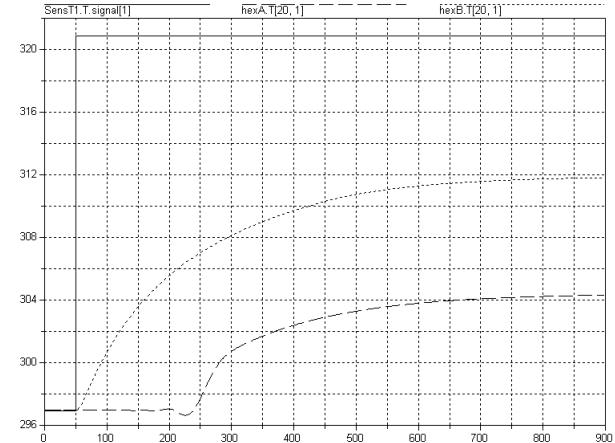


Figure 7: HE Pressure

$2 \cdot 10^5 \text{ J/m}^3$ of the specific enthalpy for the fluid of the flow rate source for *hexA*.

In figure 9 are reported the inlet temperature for *hexA* (continuous line), the outlet temperature for *hexB* (dotted line) and the outlet temperature for *hexA* (dashed line).

It should be pointed out that the last experiment has been conceived also to test the numerical robustness for the model: the results have shown that the coupling of two FEM-based components (*hexA* and *hexB*) does not affect the numerical stability, even for large values of the heat exchange coefficient. Further tests with different stabilization coefficients, not reported for the sake of brevity but available on-line, have confirmed the absence of numerical instabilities.

Figure 9: HE Temperatures: *hexA* inlet (continuous), *hexB* outlet (dotted) and *hexA* outlet (dashed)

7 Conclusions and Work in Progress

A Modelica FEM-based model for heat exchangers has been presented. The model has been implemented into a specific component (*Flow1DFem*) which is included in the *ThermoPower* library, developed for thermal power plants modelling, simulation and control. The component, whose internal implementation is completely shielded from the connectors, has been validated through simulations for specific plants configurations.

The possibility to effectively use Modelica to model physical systems that are originally described by PDEs has been shown in the specific case of the advection

equation.

Current work is headed essentially in two directions:

- the further improvement of the developed model with particular emphasis on extensions to handle also phase changes along the spatial domain;
- the development of Modelica models for other systems described by PDEs, such as flexible robot links.

References

- [1] F. Casella and A. Leva. Modelica open library for power plant simulation: Design and experimental validation. In *Modelica Conference*, Linköping, Sweden, 2003. www.elet.polimi.it/upload/casella/thermopower/.
- [2] F. P. Incropera and D. P. DeWitt. *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons, 1985.
- [3] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations*. Springer Verlag, 1997.
- [4] C. Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press, 1987.
- [5] M. L. Aime. *Engineering Methods And Tools For Modeling And Simulation Of Power Generation Plants*. PhD thesis, Politecnico Di Milano, 1999.
- [6] C. Baiocchi and F. Brezzi. Stabilization of unstable numerical methods. In *Problemi Attuali dell'Analisi e della Fisica Matematica*, Taormina, Italy, 1992.
- [7] H. Tummescheit and J. Eborn. Design of a thermo-hydraulic model library in Modelica. In *ESM (European Simulation Multiconference)*, Manchester, UK, 1998.
- [8] Dymola 5.1b. Dynasim AB. Lund, Sweden.

A Matrices Expression

$$M = l \begin{bmatrix} \frac{1}{3} - \frac{\alpha}{4} & \frac{1}{6} - \frac{\alpha}{4} & & \\ \frac{1}{6} + \frac{\alpha}{4} & \frac{2}{3} & \ddots & \\ & \ddots & \ddots & \frac{1}{6} + \frac{\alpha}{4} \\ & & \frac{1}{6} + \frac{\alpha}{4} & \frac{1}{3} + \frac{\alpha}{4} \end{bmatrix}$$

$$B = l \begin{bmatrix} B_{11} & B_{i,i+1} & & \\ B_{i,i-1} & B_{ii} & \ddots & \\ & \ddots & \ddots & B_{i,i+1} \\ & & B_{i,i-1} & B_{NN} \end{bmatrix}$$

$$\begin{aligned} B_{11} &= v_1 \left(-\frac{1}{3} + \frac{\alpha}{4} \right) + v_2 \left(-\frac{1}{6} + \frac{\alpha}{4} \right) \\ B_{i,i+1} &= v_i \left(\frac{1}{3} - \frac{\alpha}{4} \right) + v_{i+1} \left(\frac{1}{6} - \frac{\alpha}{4} \right) \\ B_{i,i-1} &= v_i \left(-\frac{1}{3} - \frac{\alpha}{4} \right) + v_{i-1} \left(-\frac{1}{6} - \frac{\alpha}{4} \right) \\ B_{i,i} &= v_{i-1} \left(\frac{1}{6} + \frac{\alpha}{4} \right) + v_i \frac{\alpha}{2} + v_{i+1} \left(-\frac{1}{6} + \frac{\alpha}{4} \right) \\ B_{NN} &= v_{N-1} \left(\frac{1}{6} + \frac{\alpha}{4} \right) + v_N \left(\frac{1}{3} + \frac{\alpha}{4} \right) \end{aligned}$$

$$Y = l \begin{bmatrix} Y_{11} & Y_{i,i+1} & & \\ Y_{i,i-1} & Y_{ii} & \ddots & \\ & \ddots & \ddots & Y_{i,i+1} \\ & & Y_{i,i-1} & Y_{NN} \end{bmatrix}$$

$$\begin{aligned} Y_{11} &= v_1 \left(\frac{1}{4} - \frac{\alpha}{6} \right) + v_2 \left(\frac{1}{12} - \frac{\alpha}{12} \right) \\ Y_{i,i+1} &= v_i \left(\frac{1}{12} - \frac{\alpha}{12} \right) + v_{i+1} \left(\frac{1}{12} - \frac{\alpha}{6} \right) \\ Y_{i,i-1} &= v_i \left(\frac{1}{12} + \frac{\alpha}{12} \right) + v_{i-1} \left(\frac{1}{12} + \frac{\alpha}{6} \right) \\ Y_{i,i} &= v_{i-1} \left(\frac{1}{12} + \frac{\alpha}{12} \right) + v_i \frac{1}{2} + v_{i+1} \left(\frac{1}{12} - \frac{\alpha}{12} \right) \\ Y_{NN} &= v_{N-1} \left(\frac{1}{12} + \frac{\alpha}{12} \right) + v_N \left(\frac{1}{4} + \frac{\alpha}{6} \right) \end{aligned}$$

$$E = l \begin{bmatrix} \frac{1}{3} & \frac{1}{6} & & \\ \frac{1}{6} & \frac{2}{3} & \ddots & \\ & \ddots & \ddots & \frac{1}{6} \\ & & \frac{1}{6} & \frac{1}{3} \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & & & \\ \frac{1}{2} & 1 & \cdots & 1 & \frac{1}{2} \end{bmatrix}^T$$

Process simulation in industrial projects

Magnus Holmgren

Solvina

magnus.holmgren@solvina.se

Gruvgatan 37, 421 30 V. Frölunda, Sweden

<http://www.solvina.se/>

Introduction

Solvina has used Dymola/Modelica since the company started in 1997. During that time we have performed a large number of simulation projects for different customers. Many of our customers are industrial production units and this paper will discuss some experiences of simulating for production units compared to simulation for development or research purposes.

As an example of such a project a steam net simulator that Solvina delivered to Iggesund paperboard will be described.

1 Simulation at Solvina

Solvina AB is a company located in Gothenburg, Sweden. Solvina works with modelling and simulator development. Most of our customers are nuclear, process or power industry but Solvina also work for other customers. Solvina has used Dymola/Modelica since 1997 and it has become our main modelling tool. For our customers we have developed two Modelica libraries:

- SteamNet library
- Pulp&Paper library

The SteamNet library is an extension of the ThermoFluid library and has been used in several projects including the Iggesund simulation project described here. The Pulp&Paper library contains models for both wet end and dry end paper simulation. It has been used to model the entire board machine at AssiDomän - Frövi paperboard.

2 Simulating in industrial projects

This paper describes experiences from working in industrial projects. With an industrial project, a project for production industry with little or none simulation experience is meant. Simulation work

in such projects is often part of a larger installation or redesign project. Working in such projects makes extra demands on the simulation studies:

- Clear goal
Specify in advance exactly what studies the simulator should be used for.
- Convince the organization
Make the customers organization believe in and use the results from the simulations.
- Limited time.
The simulation result has to be finished in time for factory start up.

The first point is also an advantage. It simplifies the development of the simulator when it is known exactly what it should be used for.

Another thing to consider is that in this type of projects the simulations have to be directly profitable. It has to be clear that the simulator earns money!

3 The project at Iggesund Paperboard

The simulation project Solvina made for Iggesund Paperboard is an excellent example of an industrial project. Iggesund Paperboard was installing a new control system for their steam distribution system.

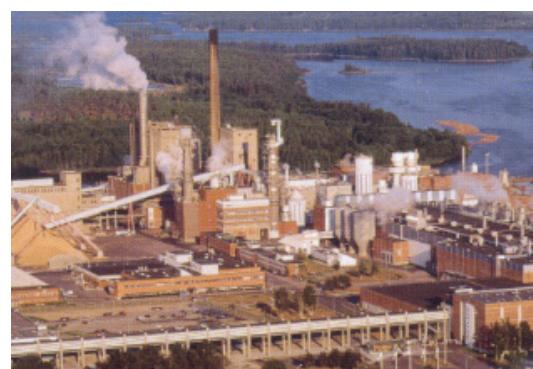


Figure 1 Iggesund Paperboard

The steam system is essential for the paperboard production and therefore Solvina was engaged to

assure the function of the new control system. Our tasks was:

- Verify the control system design
- Tune the control system for several operating conditions
- Train the operators in the new control system functionality before start up.
(Operators were used to non-computer based regulators)

The purpose of a steam system is to deliver steam of the right pressure to steam consumers in the process. To generate steam, boilers are used. In a large steam system the boilers generate steam at a high pressure, which is reduced to lower pressures through turbines and thereby generating electric power.

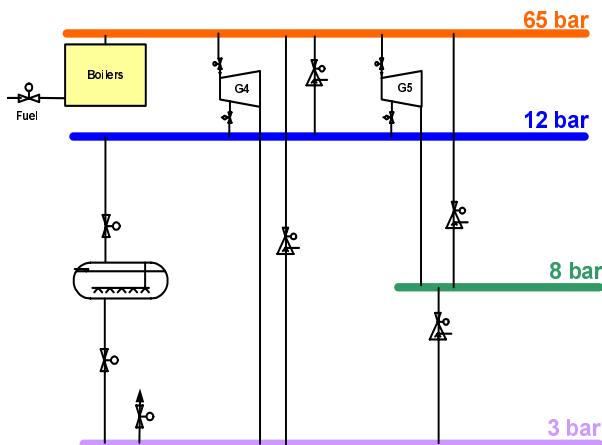


Figure 2 Iggesund steam net layout

The steam net at Iggesund paperboard has four boilers (two recovery boilers, one bark boiler and one oil boiler). The boilers generate high-pressure steam at 65 bar that is reduced through two turbines to three consumer steam nets (3,8 and 12 bar), which supply steam to the process. As a complement to the turbines, valves can reduce steam directly between the different steam nets.

It is important to keep constant pressures in the steam nets because varying pressure affects the quality of the board produced since it gives varying drying conditions in the steam dryer.

The need for steam varies with different consumers turned on and off. An accumulator is installed in the system and can be loaded or unloaded with steam depending on if the need of steam exceeds the production or not. For long term operation the boilers steam production has to be controlled to match the steam need. Boilers however are rather slow to adjust their production so the accumulator has an important role. There is also the possibility

to let out steam to the atmosphere (air blow) but then the energy the steam contains is lost.

4 Modelica modelling

Modelling usually starts from a model library. In this case the model was built using the SteamNet library.

Modelling requires a lot of specific data for the process. Good sources for data are drawings and documentation. In process industry logged production data is also often available, which we can use to model already running components.

However it is never possible to retrieve all data needed, but good engineering guesses often function well if checked particularly during validation.

The advantages with using Modelica in this kind of modelling projects instead of traditional flow sheeting programs are several.

- Better dynamical solvers available.
- Models can be modified easily.
- Flow sheeting programs only have standard components.
- Control systems can be modelled accurately.
- The models can be multi domain.

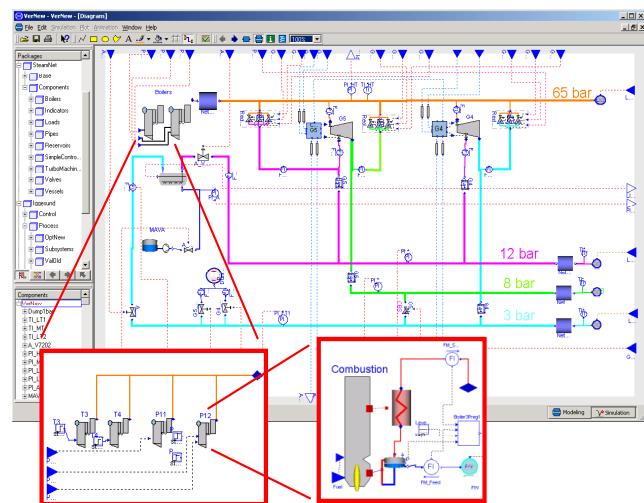


Figure 3 The Iggesund Dymola model

In the Iggesund case, besides the process model, an accurate model of the control system was made including logic for operator control and regulator initiation etc. It would show very important to have an exact model of the control system since many of the problems found in the new control system design was when changing between different control modes etc.

Modelling is often a relatively little part of a Simulation project. In the Iggesund project that totally was about one man-year only 10-15% of the time were modelling. Data collection was about 20% and validation about 30% of the time. The rest of the time was used for simulations with the model.

5 Model description

The model in the Iggesund project was started from the steam net library, which is based on the ThermoFluid library.

From the ThermoFluid library the Medium model for water and the efficient control volume are used. Flows are pressure driven. Static flow conditions are assumed.

The models focus on the dynamics of the process and properties that have importance for the control of the system are more accurately modelled. The valves are modelled with actuators and the flow characteristics of the valves were measured on place.

The boilers were modelled with a transfer function from fuel to heat. The water part of the boiler is modelled with a drum, a convection part and a superheater. The reason for modelling the water part of the boiler accurately was that the evaporation from the drum has an important effect on the dynamics of the high-pressure steam net.

The turbines are modelled as stodola turbines. The turbine control system with limitations for the flow conditions through the turbine was modelled. No account was taken to the inertia of the turbine since the turbine control system always are the limiting factor because it is designed not to allow any flow conditions that can bring instability to the turbine.

6 Validating models

The validation of a model is maybe the most important part. To write models is often a simple and relatively straightforward task. To prove their accuracy and make people believe in the results is often harder.

First the models are validated component by component against operating data or maybe even specially made tests. Next step is validating the model as a system.

To be able to communicate the model with operators and other customer personnel an operator interface is important. This gives all personnel

something to gather around. Many interesting discussions often take place when engineers, operators and management are gathered around a tool that allows them to test ideas and discuss them.



Figure 4 Iggesund simulator with charts

When everyone is convinced that the process model describes the real process system in a satisfying way the models can be used for simulations.

7 Simulations

How to use the models is of course entirely up to what your task is. In the Iggesund project the task was to check and tune the new control system. First the design of the control system was checked. The control system contained about 40 PID regulators that could operate in several modes depending on situation. Directly a few regulators that were misplaced during design were found. They couldn't function in the control configuration due to system effects even if it appeared logical.

After the design was done the tuning of the controllers started. First a preliminary tuning loop-by-loop was made according to schoolbook methods. After that the entire system was retuned so that it performed well during all different operating conditions. Examples of different operating conditions are high or low production or if one turbine is shut down. In every operating condition the control system should handle a number of disturbances such as board machine shut down, turbine failure etc.

Batch simulations were used to check and tune the control system.

8 Trimming the model

A problem with modelling large Modelica models using libraries is that the models often become stiff. A stiff model is no problem using the good solvers of Dymola although it can be annoying having to wait. However if the model should be used for operator training or in hardware in the loop applications real time performance are needed, preferably using a fixed step solver.

To solve this problem the states with short time constants have to be found and removed. This can be done by linearizing the model and calculating the eigenvalues. The largest eigenvalue will set the limit of the step size for which the model is stable. The linear model is calculated at one time point and the eigenvalues will change for another time point. The calculations will therefore only show the fast states in exactly that time point but can be seen as a hint which states that make the model stiff.

Often it is found that one or a few states have eigenvalues much larger than the rest of the model. It is then often possible to remove or remodel those states. It is seldom time constants bellow a second are interesting in process applications but shorter time constants are often introduced by mistake. For example by introducing a small control volume.

The Iggesund model could be trimmed to run about 10 times faster than real time with a fixed step solver. That was about 100 times faster than the original model just by eliminating fast states. Results of the same magnitude have been achieved with several other process models.

9 Operator training

Solvina has learned not only to deliver correct results, but also to ensure that the customer uses them. Simulator based operator training is one excellent way to ensure that.

For example in the Iggesund Paperboard project the operators must know what incidents the control system is tuned to handle and when they should interact.

Operator training with a simulator not just ensures that the system is correctly used but also gives the operator the ability to test and train incidents and thereby maybe eliminate operator mistakes in the real process. This can be worth as much as the entire simulation project.

Another positive effect with an operator interface is that it gives everyone access to the simulator. It can be used for teaching new operators the system functionality and it can be used for teaching engineers regulator tuning etc.



Figure 5 Operator training (Johanna from Solvina)

Solvina design operator interfaces with our LabVIEW based tool. A screen dump from the real control system is used as background and new figures and buttons are added with drag and drop and coupled to corresponding Dymola values and parameters.

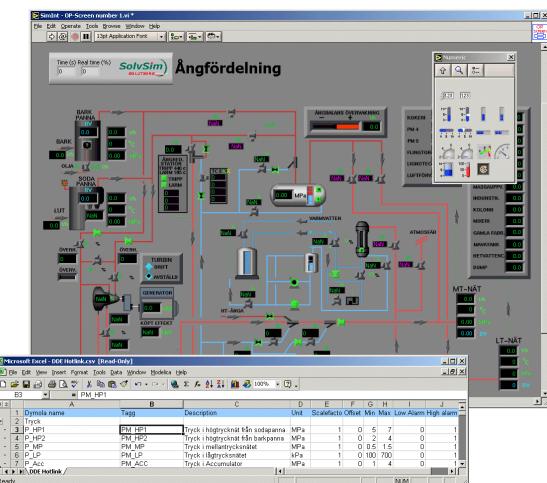


Figure 6 Development of operator interface in LabVIEW with SolvSim – OP-interface tools.

Some of the features incorporated in the operator interface tools are:

- Several operator screens
- Floating dialogs for regulators etc.
- Charts with history
- Stop and save state. Restart from saved state.

It is important that the operator interface is intuitive. It must also have the capabilities needed to investigate the process, that is charts and possibilities to stop/restart.

10 Project Results

The Iggesund Paperboard project was an extremely successful project. The direct results were:

- Perfect start up.
Everything functioned in automatic mode.
- Not one stop caused by the steam system since.
- Air blow of steam down from 4000 tons/month to 150 tons/month saving 2.5 million liters of oil per year.



Figure 7 Article in "Svensk Papperstidning" about the project "Zero problems with simulation"

11 Next step: Modelica with real control system

In all of Solvinas projects so far the control system has been modelled in Dymola. Although Dymola is suited for this kind of modelling it would be better to use the real control system code. Several advantages can be identified:

- Saving the cost of modelling advanced control systems.
- Debugging the real control system code before start up.
- Even better operator training using the real user interface and operator stations.
- Easier to maintain the simulator if only one version of the control code exists.

A simulator with a steam net system controlled by a Siemens Simatic control system has recently been developed at Solvina. A Modelica model is used, exactly as in previous simulators. The control system model was replaced with an external C function communicating the control signals with an external application.

The external application communicates with the Siemens control system. The control system in this case was run in a software emulated PLC. This makes it possible to run the entire application in one computer and it gives the ability to simulate the same hardware set-up as in the real process.

The reasons to have an extra application between Dymola and the Siemens system are several:

- It can handle start and stop of simulations.
- It can handle time synchronization.
- It can answer simple signals from the control system not simulated in Modelica such as power OK signals etc.
- It can make simple scenarios for fault cases not covered by the model such as fire scenarios etc.

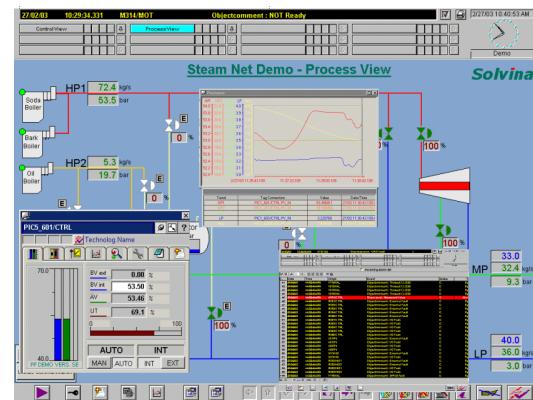


Figure 8 Siemens control system controlling a steam net simulator

12 Conclusions

It is both profiting and stimulating to work with simulations in industrial projects. Working with industrial customers make extra demands on us as having very strict timetables but it also gives us direct feedback from real processes and our results are often directly measurable in money.

An important point when modelling is to have a clear goal for what the simulator should be used for. Far too many simulation projects become long time-consuming projects that finally end almost unused because no clear goal was set in the beginning.

Finally it has to be shown that the simulations are profitable. This however is often clear when a clear goal for the simulations have been set together with a fixed timetable to reach them.

/ Magnus Holmgren, Solvina AB.

Gas Turbine Applications using ThermoFluid

Andreas Idebrant

andreas.idebrant@mathcore.com

MathCore Engineering AB

Teknikringen 1B, SE-583 30 Linköping, Sweden

<http://www.mathcore.com>

Lennart Näs

lennart.nas@power.alstom.com

Alstom Industrial Turbines AB

SE-612 82 Finspång, Sweden

<http://www.power.alstom.com>

Abstract

In a project between MathCore Engineering and Alstom POWER Sweden in Finspång Sweden, a Modelica model of a complete 43 MW gas turbine has been made. The main purpose of this model is to study transients under different working conditions. The model can be used to optimize start-up sequence, simulate load rejections, verify design, test different fuels etc.

A new library called GasTurbine containing components specialized for gas turbine modeling has been developed based on the existing public available ThermoFluid[1, 2] library.

1 Introduction

In this paper the modeling issues, using the ThermoFluid library, of a large industrial gas turbine are addressed. The gas turbine is the 43 MW GTX 100 from Alstom POWER in Sweden. This type of gas turbine is used for producing power to an external or internal electrical grid. The main fuel is natural gas or diesel oil. Testing of such big gas turbines in a separate test rig or at each specific site is costly and time consuming. Transient tests might also lead to performance degradation. A detailed dynamic model of a gas turbine could simulate and hereby prevent possible problems before they occur in real life.

The ThermoFluid library contains the framework for building thermodynamic applications such as a gas turbine in Modelica. ThermoFluid has also been used in previous projects to build gas turbines[3, 4]. Combined with the Modelica standard library it is possible to connect to other domains such as electrical grid nets, an electrical motor, control systems, etc.

Unfortunately the ThermoFluid library is complex to use even for an experienced user, familiar with Modelica. It does not contain the blocks needed to build a complete gas turbine. Therefore an

application library called GasTurbine has been made that is more easy to use and contains ready to use components especially designed for gas turbine applications. The current library contains about 100 components.

There were mainly two objectives with this project. The first objective was to make an existing model of a reference model made in a static simulation tool called IPSEpro[5]. This tool is a suitable tool for thermodynamic processes in general and it has in Finspång been added a library for gas turbine components. Complete static models of the Finspång gas turbine fleet are frequently used and tuned to correspond to real engine behavior. This kind of static tool is used to e.g. predict power output of a gas turbine at given conditions. The input data could be fuel type, air temperature, ambient pressure, component performance etc. The target for the model in Modelica was to have the steady state points identical to the result from the static model in IPSEpro. This was done step by step by verifying the calculation model and the gas routines for each component in the GasTurbine library.

The second objective was to make a simulation of a load rejection where the outlet power to a simulated electrical grid is disconnected instantly and a controller makes sure that the increasing rotational speed will be limited. The controller and the fuel gas system implemented in the Modelica model are built up identical as for the “real” engine.

2 The Gas Turbine

Figure 1 shows a cross-section diagram of the gas turbine GTX100. It is a middle range machine with maximum sustained output of 43MW. The main parts are the compressor, combustor, and the turbine. A simplified diagram of such a gas turbine is shown in Figure 2.

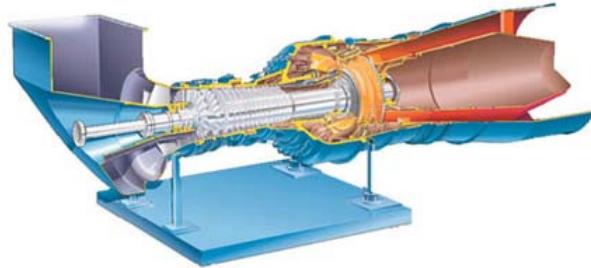


Figure 1: A cross-section diagram of the gas turbine GTX100 developed by Alstom POWER in Finspång Sweden.

The cycle starts with ambient air flowing into the compressor. The compressor increases pressure and temperature of the air. At the next stage, pressurized air and fuel are mixed in the combustion chamber and burnt with constant pressure. The resulting hot exhaust gas is expanded in the turbine stage and is released to the environment. The produced power is converted to electrical power in a generator connected to the outgoing shaft.

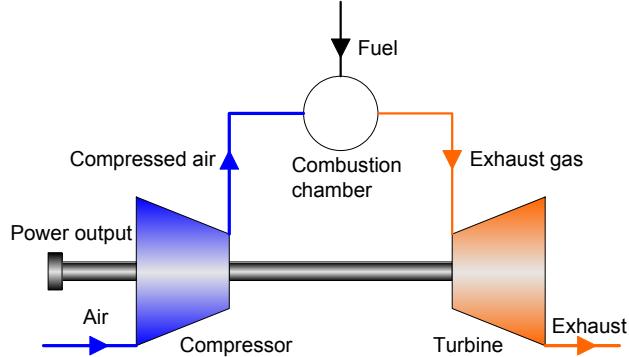


Figure 2: A schematic picture of a gas turbine.

The complete model of a gas turbine is of course more advanced. The real gas turbine consists of a quite advanced cooling system with bleed outputs from the compressor and a detailed fuel system. A controller adjusts the amount of fuel and controls the guide veins in the compressor. The different sub systems are explained in the next sections.

3 Controller

The controller used in this model is the same used in the “real” gas turbine. It is not complete and works only when the model is working at sustained conditions. For start up purposes there is an additional controller, which has not been implemented. The controller block can be seen in Figure 3. On the left side there are parameter inputs for ambient conditions. On the lower side there are inputs for power produced to the grid (P_{el}), rotational frequency (f), pressure after the

compressor (p_3), and temperature after the turbine (t_7). The outputs are the pilot fuel valve opening (x_{gp}), main fuel valve opening (x_{gm}), and the guide vein opening (IGV).

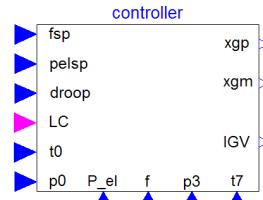


Figure 3: The controller block for GTX 100.

To be able to simulate load drops a simple model of a virtual grid was implemented. The basic idea is to use a clutch model to detach the mechanical flange from the simulated grid at a predefined instant. A simple PI controller attached to a variable damper is used to gradually increase the power output of the generator, see Figure 4.

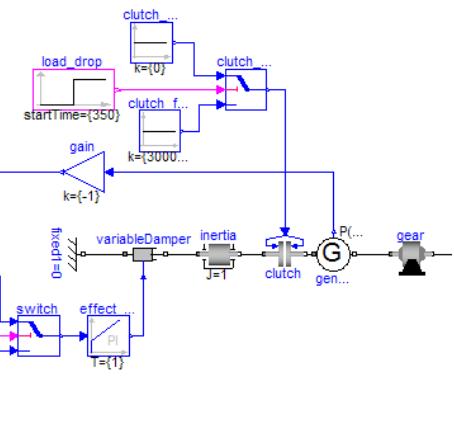


Figure 4: Controller part for simulating a grid net and load drop/rejection.

4 Fuel system

The real fuel system of a gas turbine is quite complex with a lot of pipes and valves with different physical properties. A simplified model has been made, which will be sufficient for these types of simulations. The structure of the simplified fuel system can be seen in Figure 5. The pilot and main valve is connected to the main controller in the total model.

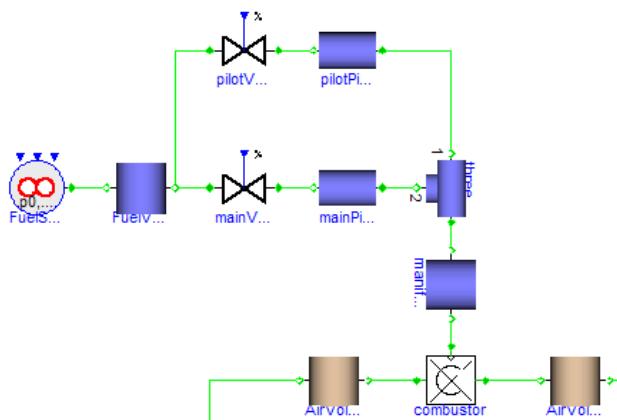
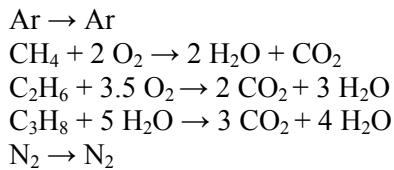


Figure 5: Picture of the simplified fuel system.

For combustion natural gas has been used with the substances C_2H_6 , C_3H_8 , CH_4 , CO_2 , and N_2 . The air that flows into the combustion chamber consists of the following substances: Ar, CO_2 , H_2O , N_2 , and O_2 . During combustion of the gas mix the following reactions occur (Ar and N_2 does not react):



These reactions can be transformed to expressions for mass flow in Modelica syntax according to below:

```
m_out[Ar] = m_air[Ar];
m_out[CO2] = m_air[CO2] +
m_fuel[CO2] +
(wCO2/wCH4)*m_fuel[CH2] +
2*(wCO2/wC2H6)*m_fuel[C2H6] +
3*(wH2O/wC3H8)*m_fuel[C3H8];
m_out[H2O] = m_air[H2O] +
2*(wH2O/wCH4)*m_fuel[CH4] +
3*(wH2O/wC2H6)*m_fuel[C2H6] +
4*(wH2O/wC3H8)*m_fuel[C3H8];
m_out[N2] = m_air[N2] + m_fuel[N2];
m_out[O2] = m_air[O2] -
(wO2/CH4)*m_fuel[CH4] -
3.5*(wO2/wC2H6)*m_fuel[C2H6] -
5*(wO2/wC3H8)*m_fuel[C3H8];
```

The notations $w\text{CH}_4$, $w\text{C}_2\text{H}_6$, $w\text{C}_3\text{H}_8$, $w\text{CO}_2$, $w\text{H}_2\text{O}$, and $w\text{O}_2$ denote the molecular weights for CH_4 , C_2H_6 , C_3H_8 , CO_2 , H_2O , and O_2 respectively. The mass flows $m_{\text{out}}[\text{Ar}]$, $m_{\text{out}}[\text{CO}_2]$,

$m_{\text{out}}[\text{H}_2\text{O}]$, $m_{\text{out}}[\text{N}_2]$, and $m_{\text{out}}[\text{O}_2]$ are the outgoing mass flows for Ar, CO_2 , H_2O , N_2 , and O_2 respectively. Similarly, the notation m_{air} and m_{fuel} denote the mass flows for air and fuel.

5 Cooling system

The cooling system consists of pipes (mixers and splitters) and volumes. Cooling increases the efficiency of the gas turbine. Air from the compressor and its bleed outputs is used to cool the exhausts from the combustion chamber. There are also some small flows directly to the ambient air. The volumes and valves are taken from the ThermoFluid library and have been modified to use the correct medium model. Since the exhaust gas and the air has the same composition in this case (but different mass fractions) the same medium model is used for air and exhaust gas. Figure 6 shows a typical implementation of a mixer that mixes gases with the same composition.

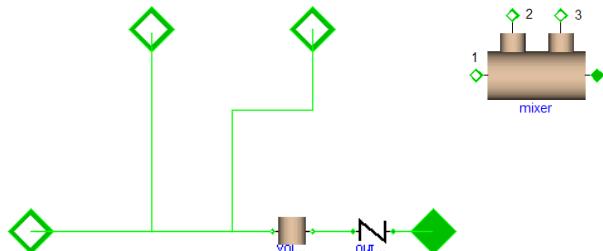


Figure 6: A mixer with three inputs and one output.

The splitting fracture of gases is controlled by adjusting the nominal mass flow rate at a given nominal pressure drop level. The diagram picture of a splitter is shown in Figure 7.

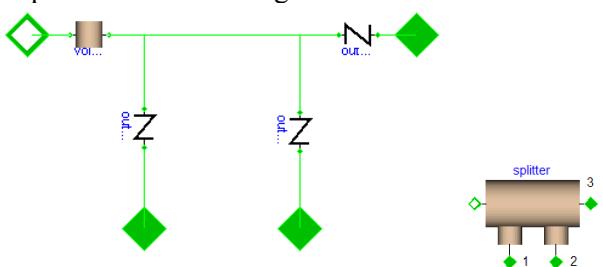


Figure 7: A splitter with one input and three outputs.

The complete cooling system can be seen in Figure 8.

6 The Modelica model

The complete Modelica model of the gas turbine is shown in Figure 8. The model consists of a controller, simulated grid, fuel system, cooling system, and the basic gas turbine part.

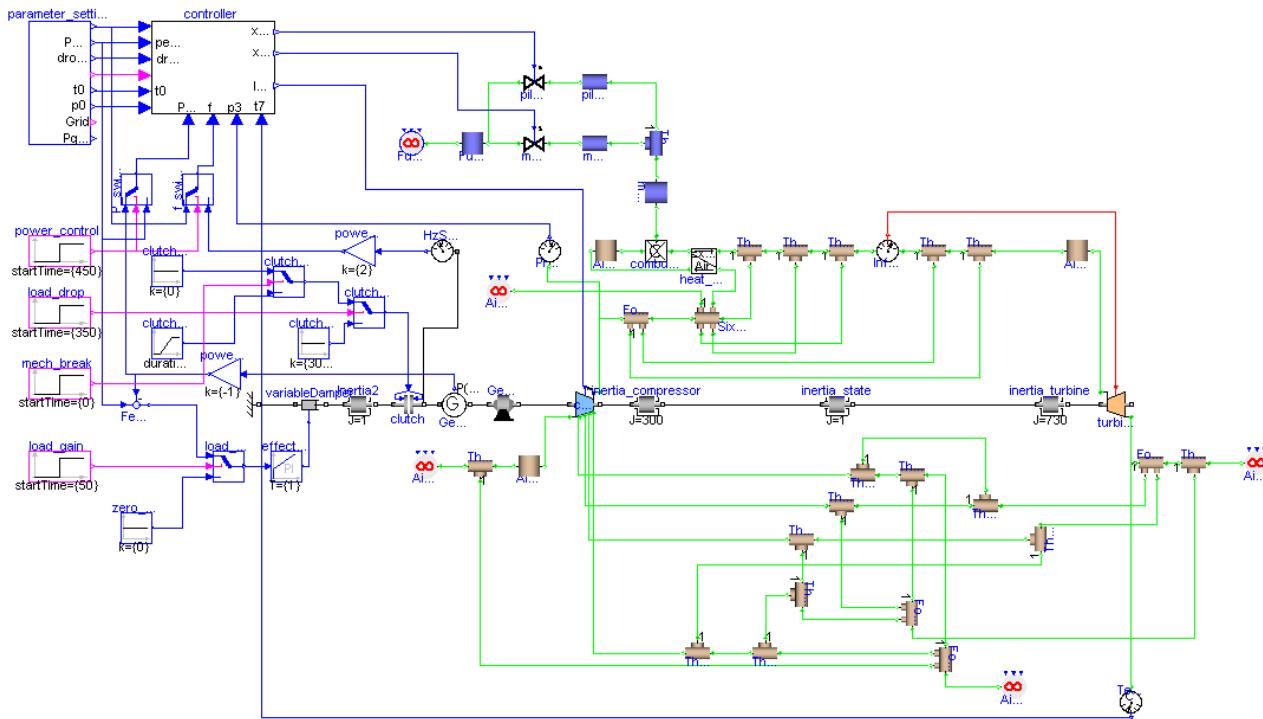


Figure 8: The complete Modelica model of a GTX100 gas turbine with controller, fuel system, cooling system and simulated grid.

For the thermodynamic parts of the model the connector for static momentum balance from the ThermoFluid library is used:

```
connector BaseFlow
  parameter Integer nspecies(min=1);
  parameter String MediumType =
"unspecified";
  SIunits.MassFraction mass_x[nspecies];
  SIunits.Pressure p;
  SIunits.SpecificEnthalpy h;
  flow SIunits.MassFlowRate
mdot_x[nspecies];
  flow SIunits.Power q_conv;
  SIunits.Density d;
  SIunits.Temp_K T;
  SIunits.RatioOfSpecificHeatCapacities
kappa;
  SIunits.SpecificEntropy s;
end BaseFlow;
```

This means that no dynamic momentum terms are taken into account in this model. This choice was initially made to reduce the computational burden. The complete model in Figure 8 has 240 continuous time states and 2644 nontrivial equations.

Due to the detailed cooling system, the Modelica model is hard to initialize. It is of great importance to choose the initial starting parameters carefully to avoid a stiff system. To get rid of the sometimes long initialization times, the model was once simulated past the initial stiff part. Then the simulation was stopped and the current state was

saved to a file, e.g. dsfinal.txt. In the next run the model states were initialized with the previously saved file and the simulation started much faster.

7 The GasTurbine Library

The Modelica library ThermoFluid was used as a toolbox for creating the components needed for the project. Currently, ThermoFluid does not contain ready made components needed for gas turbine modeling. Therefore an application library called GasTurbine (Figure 9) was created with specialized, ready to use components for gas turbines and especially for the GTX100. The current library consists of about 100 components for gas turbine applications.

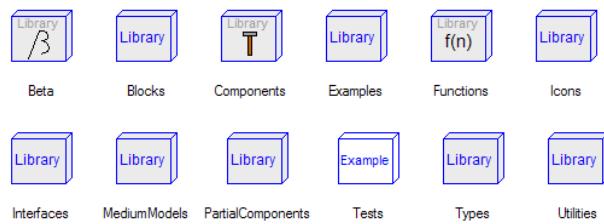


Figure 9: Overview of the GasTurbine library.

The detail of the models varies from simple compressors without maps to more advanced with inlet guide veins, maps, and bleed outputs.

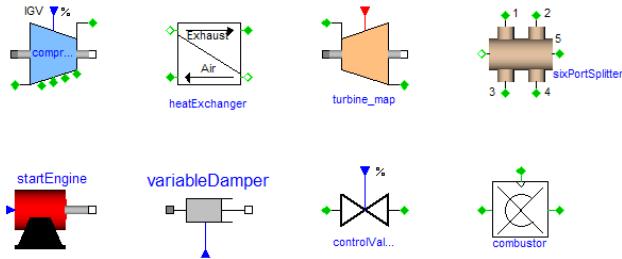


Figure 10: Sample components from the GasTurbine library.

The design of the library was developed with the end user in mind and all components can be connected directly without the need to redeclare or modify any part of the model.

8 Verification

A model can only be trusted if you can verify that the results produced from the model are valid. In this case data from a similar static model of GTX100 in the program IPSEpro was used. This means that only steady state values can be verified by this comparison but similarity in steady state points indicates that at least agreement in the handling of physical properties and calculation models. In the future real data from live experiments will be used to verify also the conditions during transients. When studying transients it is important that the dynamic parts in the model are accurate, e.g. the volumes and inertias. These dynamic parts do not make any significant impact on the static results.

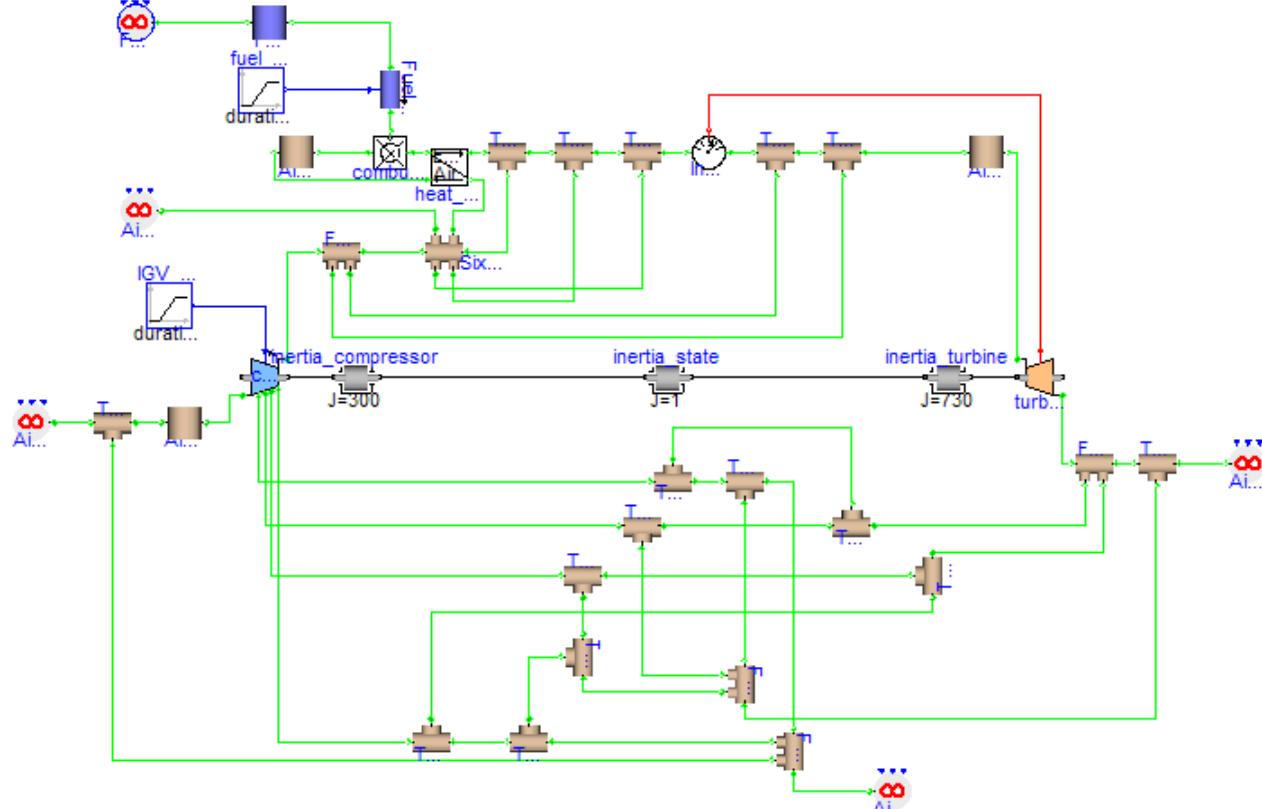


Figure 11: The model used for verification purposes, i.e. with IPSEpro.

The medium models used in the two models are not entirely identical. In IPSEpro the Janaf tables are used and in ThermoFluid the NASA tables. The small differences can however be neglected. The comparison between the two models has been

conducted with identical air and fuel composition. Below is an example table with results from the combustion chamber. Note that the pressure and mass flow are set to parameters (or input values) in IPSEpro to get the same steady state level:

Inlet combustor			
Variable	ThermoFluid	IPSEpro	Error [%]
Pressure [bar]*	13,4	13,4	0
Temperature [°C]	439,2	435,9	-0,75
Mass flow [kg/s]*	74,11	74,11	0
Entropy [J/K]	7097	7089	-0,12
Kappa	1,360	1,360	0,046
R	289,0	289,0	-0,0015
Outlet combustor			
Variable	ThermoFluid	IPSEpro	Error [%]
Pressure [bar]	13,31	13,31	0,023
Temperature [°C]	949,3	947,7	-0,17
Mass flow [kg/s]	75,11	75,11	-0,00067
Entropy [J/K]	7824	7847	0,30
Kappa	1,308	1,309	0,069
R	291,6	291,6	-0,0013

Table 1: Verification results ThermoFluid versus IPSEpro. *Input values (parameters) in IPSEpro.

Table 1 shows that there are minor differences between the two tools, which is not fully satisfactory in the work with having the two models being a reflection of one other, but the accuracy is considered as sufficient to rely on the Modelica model from a dynamic point of view.

9 Load Rejection

One important thing to test for new gas turbines is its capability to handle full load rejections/drops without tripping. In many cases it is considered as most vital to have the gas turbine back on the grid, producing full power in as short time as possible. A load drop is what happens when a power failure occurs. It could be due to lightning strikes, mechanical failure etc. When the gas turbine is running at full working power, e.g. 43MW, one want to make sure that the gas turbine does not reach trip speed when a power output to the grid is suddenly lost. The immediate result is that the rotational velocity increases to a certain level that could be close to trip level.

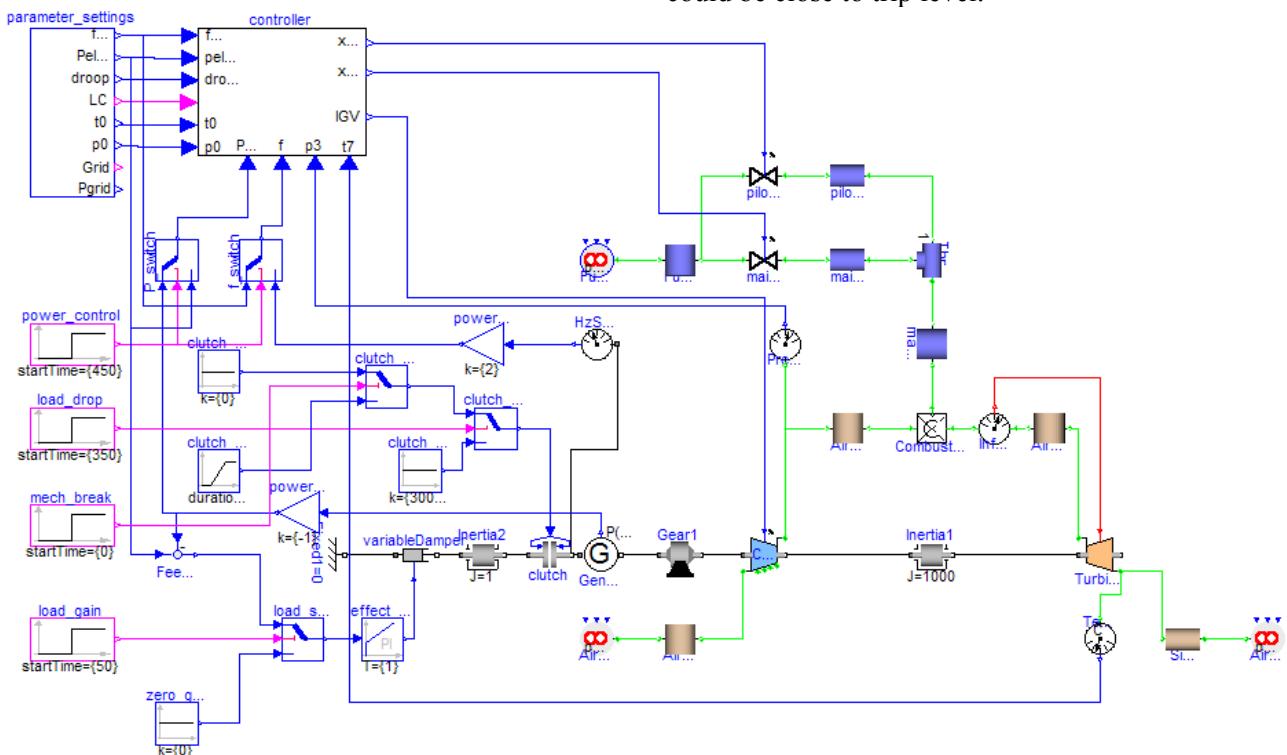


Figure 12: The Modelica model used for load drop/rejection simulations.

To avoid this a controller is designed to as fast as possible detect this failure and reduce the fuel input to the combustor and then lower the speed. The model used for this testing is shown in Figure 12. It is also important to have enough fuel so that the rotational speed remains at the working speed, e.g.

6600rpm, and make sure that the combustor still operates. The trick is to shut down the main fuel valve and open the pilot fuel valve to ensure that the combustor is not shut down. Figure 13 shows a whole operating cycle in an assumed “weak” local

grid (island mode), from ignition, loading, fuel valve switching and a full load drop at the end.

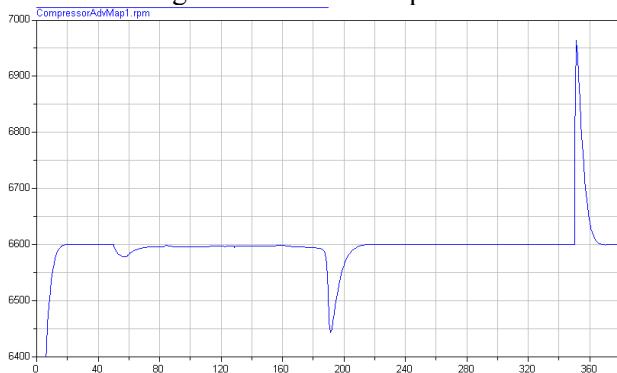


Figure 13: The rotational speed of the shaft during a load rejection test.

The first 50 seconds the gas turbine spins up to its working speed at 6600rpm. Then the generator starts to produce power and a sudden drop in rotational speed can then be seen until the controller brings the speed back to normal. At around 190 seconds the power has increased to about 37MW and the pilot fuel is shut down and is replaced with more main fuel, see Figure 14.

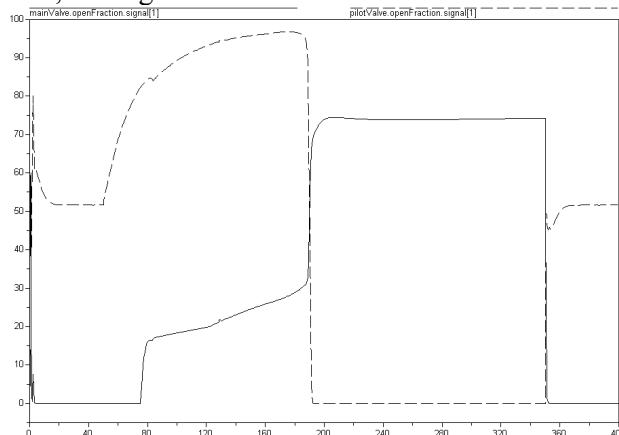


Figure 14: Percentage of valve opening during a load drop simulation. Pilot valve opening is dashed.

It can here be seen as a decrease in speed due to that the valve switching sequence is not tuned in the model and hereby the fuel flow is not constant during this process. When the full power has been reached at about 350 seconds a power failure occurs, i.e. the generator effect is decoupled, see Figure 15, and the rotational speed increases dramatically. Instantaneously, the controller reacts and shuts down the main fuel and starts feeding the combustor with pilot fuel and the rotational speed decreases to normal 6600rpm.

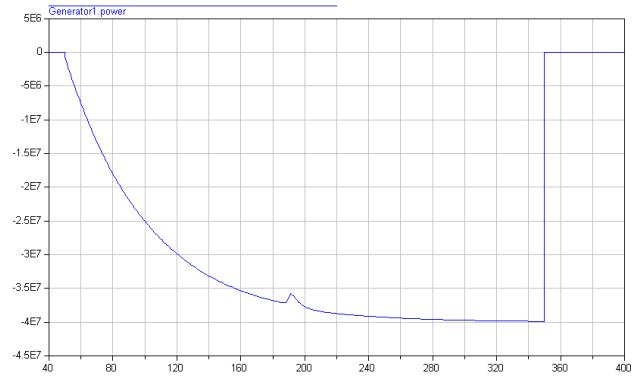


Figure 15: Generated power to the grid.

The requirement for this load rejection test is that the rotational speed must not exceed 10% of the nominal value. In this case the rotational speed must not exceed $6600 \times 1.1 = 7260$ rpm. The test shows that the maximum speed is about 6970 rpm, which is acceptable. This load drop experiment has been conducted with real gas turbines and a value around 7000 rpm has been a normal value for the GTX100.

10 Conclusions

In this article the building of a gas turbine model in Modelica has been described. A new library called GasTurbine has been developed to modify existing models in ThermoFluid to gas turbine applications. The experience is that the ThermoFluid library is somewhat hard to work with for a non-Modelica library developer and the GasTurbine library is more suited for the end user.

The gas turbine model has been verified with a static tool called IPSEpro with acceptable results from a dynamic point of view. A load rejection test has been performed with results similar to a real gas turbine.

At the end Modelica has been proven to be a suitable tool for building gas turbines in an object oriented way. The simulations are quite fast considering the amount of computations in the large model. Finding suitable initial values can be hard but new language constructs in Modelica have been introduced to help the end user to initialize models in a more convenient way.

11 References

- [1] H.Tummescheit, J. Eborn, F.J. Wagner: "Development of a Modelica Base Library for Modeling of Thermo-Hydraulic Systems". Modelica Workshop 2000 Proceedings, pp. 41-51.
- [2] H.Tummescheit: "Design and Implementation of Object-Oriented Model Libraries using Modelica". PhD thesis. Department of Automatic Control Lund Institute of Technology Lund, August 2002.
- [3] Pérez Gómez A.A: "Modelling of a Gas Turbine with ModelicaTM". Master thesis. Department of Automatic Control Lund Institute of Technology Lund, May 2001.
- [4] S.Haugwitz: "Modeling of Microturbine Systems". Master thesis. Department of Automatic Control Lund Institute of Technology Lund, May 2002.
- [5] IPSEpro is developed by SimTech Simulation Technology (<http://www.simtechnology.com>)

Session 9A

Mechatronic Systems – III

Multidomain Systems: Pneumatic, Electronic and Mechanical Subsystems of a Pneumatic Drive Modelled with Modelica

Peter Beater

Beater@fh-swf.de

Department of Mechanical Engineering – Automation
University of Applied Sciences Südwestfalen, Campus Soest
Lübecker Ring 2, D 59494 Soest, Germany

Christoph Clauß

Christoph.Clauss@eas.iis.fhg.de

Fraunhofer-Institut für Integrierte Schaltungen, Außenstelle Entwurfsautomatisierung
Zeunerstraße 38, D 01069 Dresden, Germany

Abstract

The simulation of pneumatic or electronic systems has been state of the art for a long time. For both of these domains there exist highly specialized simulation programs which can be regarded as a kind of industrial standards. However, often problems arise if different domains of technology occur within one system and very detailed models are needed.

As an example a pneumatic drive is presented that is used for teaching mechanical engineering students in Soest. It consists of pneumatic, mechanical, and electronic components. Each component is modelled very detailed using the Modelica language (Modelica, 2002). Without coupling of simulators the complete simulation model can be investigated by *one* tool.

1 Introduction

The engineer of today is used to powerful simulation tools. Within the last forty years these tools mutated from simple solvers of differential equations to computer-aided design software for technical systems. Tools like HSPICE in electronics, ADAMS in mechanics, or HOPSAN in hydraulics are highly specified to meet the needs of the discipline. These tools “know” the domain-intern peculiarities. Often the models and the simulation algorithms are closely related. Therefore, these tools are very advantageous in simulation, modelling, and postprocessing.

However, often problems arise if technical systems cover more than one established discipline, e. g. in automotive systems or in microsystems engineering. The two fundamental ways out are coupling of simulators, and compact modeling for one simulator.

From the very beginning the Modelica language has been designed for covering several technical disciplines. Complex systems can be modelled with one language to get one model. The further processing within the tool, e. g. the Dymola simulator, results in one mathematical model, typically a system of differential algebraic equations, which is solved by one simulation engine. The challenge of the Modelica approach is to show that its efficiency is not much worse than the efficiency of domain specific tools. To offer evidence of this is surely a long process (Clauss and Beater, 2002). In this paper the multidomain example of an electronically controlled pneumatic drive is presented. It demonstrates that the unified multidiscipline simulation tool Modelica/Dymola meets the challenge quite well.

At first the physical device is presented with emphasising the pneumatic and electronic parts. The Modelica model is shortly described, and simulation results are discussed. It is shown that numerical problems could be solved, and the performance can be accepted.

2 The Pneumatic Drive

Fig. 1 shows the pneumatic drive. It is a typical construction when a part has to be moved for several decimetres, e. g. in material handling. The required forces determine the diameter of the cylinder which is connected to the electrically operated directional control valve. At the ends of the cylinder magnetic switches are installed that signal the end of stroke to the electronic controller. For the controller standard CMOS ICs are used. The “programming” is done by connecting the logical blocks (AND, OR, RS). The task is to begin a repeated extending and retracting of the piston after the start button has been pushed and to stop in the extended position after the stop button has been pushed.

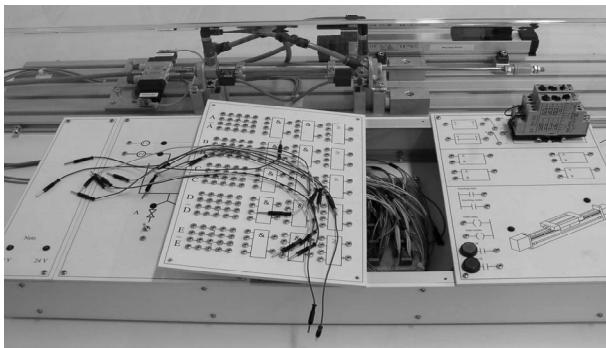


Figure 1 Pneumatic drive as laboratory set-up in Soest

3 The Pneumatic and Mechanic Parts

After preliminary trials using the analogue computer in the fifties the digital simulation of fluid power systems, i. e. hydraulic or pneumatic systems, became important in the eighties. Graphical user interfaces were added in the nineties. Using Modelica and its libraries it is easy to model pneumatic or mechanic systems. The user doesn't need to know all the details of component modeling. If nevertheless details are essential the source code of the models is available. Using models from the Modelica libraries the pneumatic drive according to Fig. 1 could be modelled.

Basically, three physical laws are needed to describe the movement of the piston. The first one is Newton's second law that models the movement of the mass (piston) as a function of the applied forces. It is described in the block SlidingMass of the Modelica library Mechanics.Translational. The forces can be calculated from the pressures in the cylinder chambers, which are described by:

$$m \cdot \dot{T} + T \cdot \dot{m} = - \frac{1}{c_v} \cdot |p_{\text{Chamber}}| \cdot \dot{x}(t) \cdot A_K \quad (1)$$

$$- \frac{1}{c_v} \cdot \alpha \cdot A_W \cdot (T - T_A) + \kappa \cdot T_{\text{surround}} \cdot \dot{m}$$

with:
 m gas mass [kg]
 T temperature in the chamber [K]
 c_v specific heat capacity [J/(kg · K)]
 p_{chamber} cylinder pressure [Pa]
 A_W heat transfer area [m^2]
 A_K piston area [m^2]
 κ ratio of specific heat capacities
 T_{surround} temp. of the environment [K]
 α coefficient of heat transfer [$\text{W}/(\text{m}^2 \cdot \text{K})$]

These equations are modelled in the library model PneuLib.Chamber. Two Chamber models, the SlidingMass for the piston and a Rod for the housing describe a simple cylinder. A complex model of a double sided cylinder with pneumatic stroke cushioning is shown in Fig. 2.

The mass flow rate to or from the cylinder chambers depends on the pressure upstream and downstream of the valve, p_1 and p_2 , and the electrical command signal for the valve. For the opened valve the mass flow rate can be described by the equation of flow through a nozzle:

$$\begin{aligned} \dot{m} &= p_1 \cdot C \cdot \rho_0 \cdot \sqrt{\frac{T_0}{T_1}} \cdot \sqrt{1 - \left(\frac{p_2 - b}{1 - b} \right)^2} && \text{for } \frac{p_2}{p_1} > b \\ \dot{m} &= p_1 \cdot C \cdot \rho_0 \cdot \sqrt{\frac{T_0}{T_1}} && \text{for } \frac{p_2}{p_1} \leq b \end{aligned} \quad (2)$$

with:
 \dot{m} mass flow rate [kg/s]
 p_1 upstream pressure [Pa]
 C sonic conductance [$\text{m}^3/\text{s} / \text{Pa}$]
 ρ_0 standard density of air [kg/m^3]
 T_0 standard air temperature [K]
 T_1 air temperature upstream [K]
 p_2 downstream pressure [Pa]
 b critical pressure ratio [1]

This equation is standardized in ISO 6358. Necessary is also a state equation for air, where the ideal gas law is used:

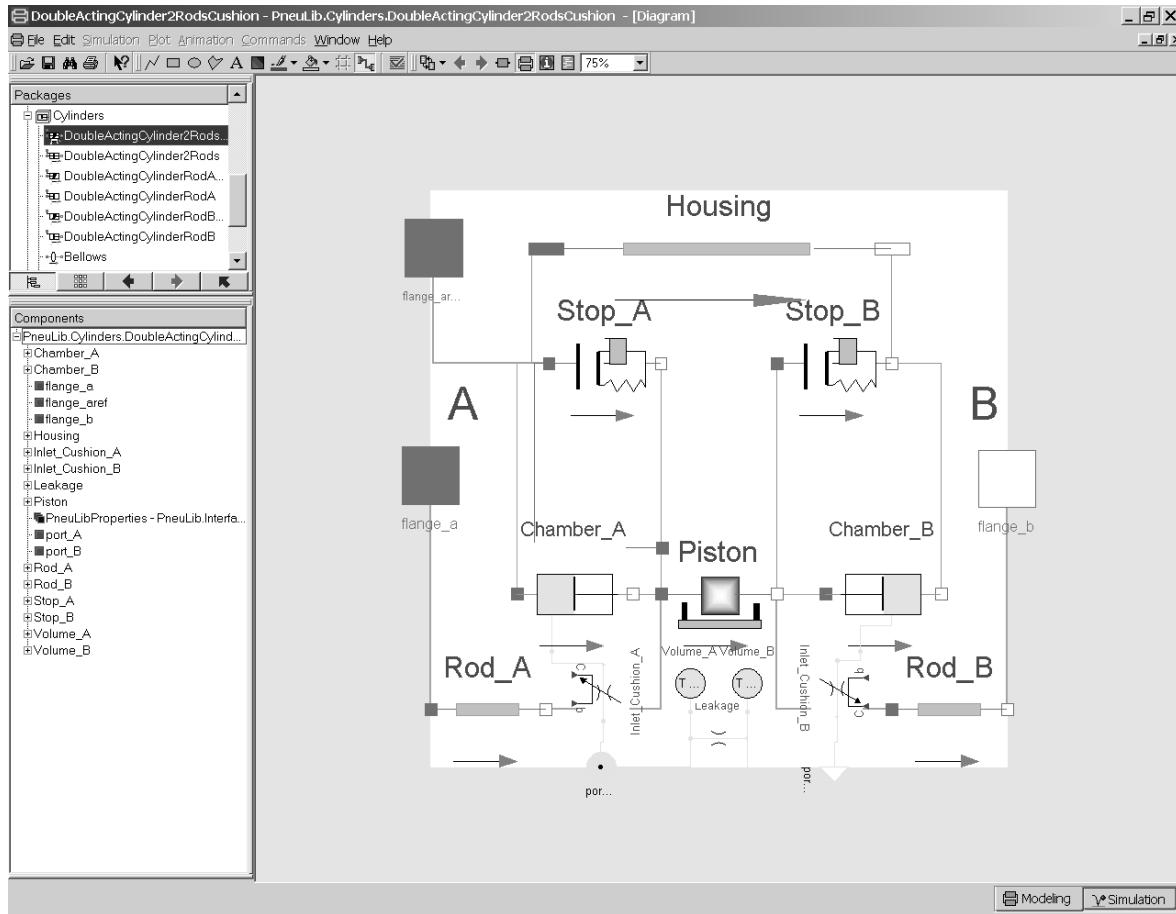


Figure 2 Object diagram of a double-sided two-ended cylinder with stroke cushioning screenshot from Dymola

$$p \cdot V = m \cdot R \cdot T \quad (3)$$

with: p pressure [Pa]

V volume [m^3]

m air mass [kg]

R specific gas constant [$J/(kg \cdot K)$]

T temperature [K]

For typical operating conditions of pneumatic drives, maximum pressure of about 1 MPa and a temperature around 20 °C, the differences between air and the modelled ideal gas are negligible. Additional equations are needed that describe e. g. the dynamic behaviour of the directional control valve or the stroke cushioning of the cylinder.

To couple component models of the library PneuLib the through variable mass flow rate \dot{m} and the across variable pressure is used. To have ports for the inflow and outflow of air two connectors are defined:



Figure 3 Connectors for port 1 (pressure supply) and port 2 (work)

An example of a simple component is a nozzle that is based on Eq. 2. While this equation is very well suited for measurement purposes it leads to problems when used in the digital simulation of pneumatic systems because the „gain“, i. e. the quotient $\frac{\dot{m}}{dp}$, goes to infinity as the pressure drop, $dp = p_1 - p_2$, goes to zero. This effect is known from models for incompressible hydraulic oil that use the simple „square root“ dependency

$$q \sim \sqrt{\Delta p} \quad (4)$$

and has led to the development of more accurate models (Beater 1999). In the pneumatics library the nozzle model according to Eq. 2 is used because it is a generally accepted standard but extended for the region of small pressure differences by a linear relationship between mass flow rate and pressure differential. This is based on the fact that then the turbulent flow becomes laminar and therefore a linear relationship exists between pressure differential and flow rate. This is also an example that simple "textbook" models are not suited for real engineering tasks but have to be extended to avoid numerical problems during integration. Figures 4 and 5 show the icon and the structure of the code. Used is the superclass TwoPortComp that defines all parts that are needed for components with two ports but no mass storage.



Figure 4 Icon of nozzle model

```
model NozzleNoStates
  "Nozzle model according to ISO 6358."
  extends PneuLib.Interfaces.TwoPortComp;
  parameter SonicConductance C=1e-7
    "sonic conductance";
  parameter CriticalPressureRatio b=b_default
    "critical pressure ratio";
  Real pressure_ratio(start=1.0);
  ...
  equation
    pressure_ratio = port_2.p/port_1.p;
    m_dot = ... ;
end NozzleNoStates;
```

Figure 5 Code of nozzle model

As in the hydraulics library HyLib (HyLib, 2002) there are also components that have lumped volumes directly connected at all pneumatic ports, see e. g. the object diagram of the cylinder in Fig. 2. This modeling concept allows pneumatic components to be connected in an arbitrary way, e. g. in series or in parallel, just by drawing connection lines, no special components for splits or mergers are required.

The advanced features of Modelica 2.1, e. g. the initial equation section, are used to make the initialization of states user friendly. In hydraulics atmospheric pressure is used as reference pressure. Therefore a system at rest has pressure states of zero. In pneumatics the gas mass in a volume is

used which is non-zero at atmospheric pressure. Therefore a number of calculations is needed to compute the gas mass in all lumped volumes which may include the calculation of the geometric volumes, e. g. of cylinders, before. As engineers usually think of pressure and not gas mass in a volume the input parameters for the initial conditions of the library models are pressures and the gas masses calculated by the library models.

The pneumatics library provides basic model classes for the modeling of cylinders - both standard cylinders with constant piston area and bellows which have a stroke dependent piston area - and motors, valves and restrictions, lumped volumes, lines and sensors. In total there are more than 80 models. Among them there are three models of pneumatic lines. Two describe the resistance by algebraic equations while the third one approximates the partial differential equations from the physical model by a set of ordinary differential equations. Laboratory experiments show an excellent correlation between measurement and simulation for the pressure drop and a good description of the dynamic behaviour, i. e. the frequency response.

For standard applications these classes cover all needed components. If, however, specially designed components are used these can be easily modelled by modifying library components. All relevant effects are available as submodels.

4 The Electronic Part

The control which is necessary for the pneumatic and mechanical parts can be modeled using Boolean algebra with the signal values 'true' and 'false' (Figure 6). A more detailed description is possible if multi-valued logic is used, e. g. with values for 'unknown', 'uninitialized'. Usually VHDL (Lehmann and Wunder, 1994) or Verilog-HDL (Palnitkar, 1996) are behavioral languages for digital logic for which powerful simulators exist, e. g. ModelSim (Modeltech, 2002). The VHDL language was used to verify the control unit design.

The control unit gets the input signals ON and OFF from outside to start and stop the machine. Further input signals are B1 (Br) for reaching the left

(right) stop. The output signals both for moving to the left (X_l) and to the right (X_r) are stored within RS flipflops. If an output signal switches to false, the inverse flipflop output allows the other output to be switched to true. These changes are caused by both the B_l and B_r signals. To connect the control unit with the pneumatic part a suitable signal conversion is necessary which is done by converter models.

For the unified modeling with Modelica the control unit is described at two levels, the Boolean level and the electrical level. For both of the levels a special small library 'Boole' and 'CMOS' has been developed.

'CMOS'-Library

Basing on the Modelica Standard Library CMOS transistors were combined to form the logic gates on the electronic level. The 'CMOS'-library contains the basic logic models Nand, Nor, Not, And, and a flipflop model RSFF. The following Modelica text shows the Nand gate model as an example:

```
model Nand "NAND"
import MEA = Modelica.Electrical.Analog;
MEA.Sources.RampVoltage VDD;
MEA.Semiconductors.PMOS TP1, TP2;
MEA.Semiconductors.NMOS TN1, TN2;
MEA.Basic.Capacitor C4, C7;
MEA.Basic.Ground Gnd;
MEA.Basic.Ground Gnd1, Gnd2, Gnd3;
MEA.Interfaces.Pin inp1, inp2, out;
equation
connect(inp1, TN1.G); connect(TN2.G, inp2);
connect(TP2.G, TN2.G); connect(VDD.p, TP2.D);
connect(VDD.p, TP2.B); connect(Gnd1.p, VDD.n);
connect(TP1.D, VDD.p); connect(TP1.B, VDD.p);
connect(C4.n, Gnd2.p); connect(TP1.S, C4.p);
connect(C4.p, out); connect(TN1.D, TP1.S);
connect(TN1.S, TN2.D); connect(C7.n, Gnd3.p);
connect(TN2.D, C7.p); connect(TN2.B, Gnd3.p);
connect(TN2.S, Gnd3.p); connect(TN1.B, Gnd.p);
connect(TP2.S, TP1.S); connect(TN1.G, TP1.G);
end Nand;
```

The MOS transistor models are used to be able to observe the electrical behavior in a great detail. Otherwise the number of variables becomes rather high. In practice this accurate level is not often necessary.

'Boole'-Library

The basic logic gates and the flipflop as well were modeled using the Boolean signals 'true' and 'false' of Modelica (two-valued logic) according to (Tiller, 2001). Delay times are neglected. Only the

flipflop needs a very small delay to avoid loops without delay. The following Modelica text shows the Nand gate of the 'Boole' library:

```
model Nand
import D = Boole.Interfaces;
extends D.DISO_wide;
D.LogicValueType out_immed(start=false);
equation
out_immed = not (in1 and in2);
out = pre(out_immed);
end Nand;
```

Due to the simplicity of 'Boole' the number of variables of the control unit model is much less than of the model based on 'CMOS'.

The 'Boole' library is a very preliminary stage of the digital electronic library which is under development to become a part of the Modelica Standard Library. The digital electronic library follows essentially the IEEE 1164 standard (VHDL IEEE-Package).

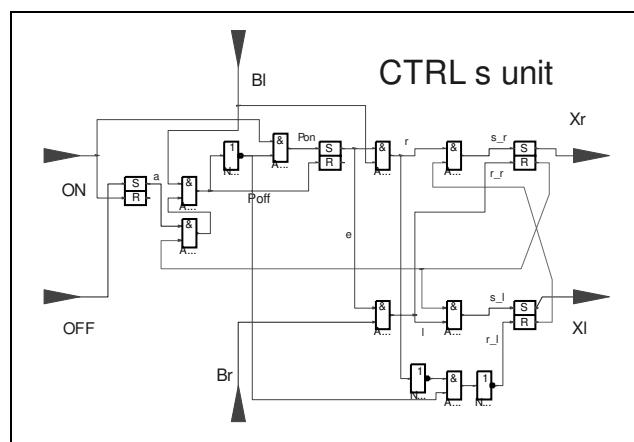


Figure 6 Control unit scheme

5 Results

With Dymola version 5 (Dymola, 2003) the model of the drive was composed graphically, analysed, translated into executable code, and simulated. The simulations started at the quiescent state (all voltages are zero, the pneumatic pressures are equal to the environment pressure) at time zero and finished after 2 seconds. In the following figures the behaviour of some variables is shown.

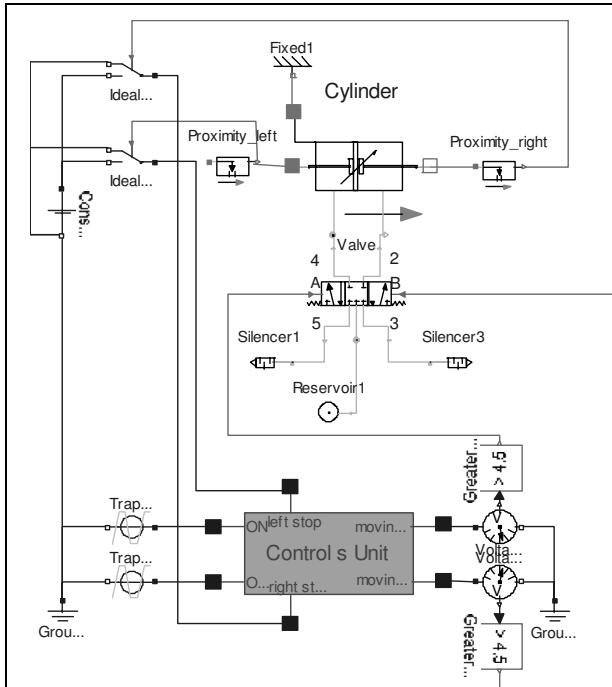


Figure 7 Object diagram of controlled drive

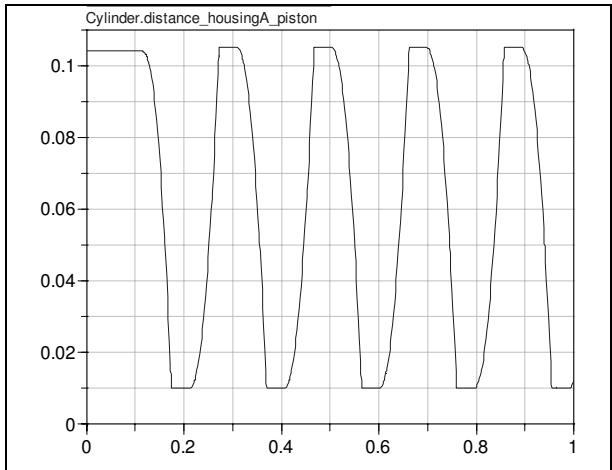


Figure 8 Piston position as function of time

To get a better feeling of the model the detailed subsystems "Pneumatic" and "CMOS" were replaced by much simpler models, "Integrator" and "Boole". They had the same input-output signals, e. g. an input to drive to the right, i. e. increase the state of an integrator linearly with time. Using the simpler models the complexity of the model and the required CPU time can be considerably reduced.

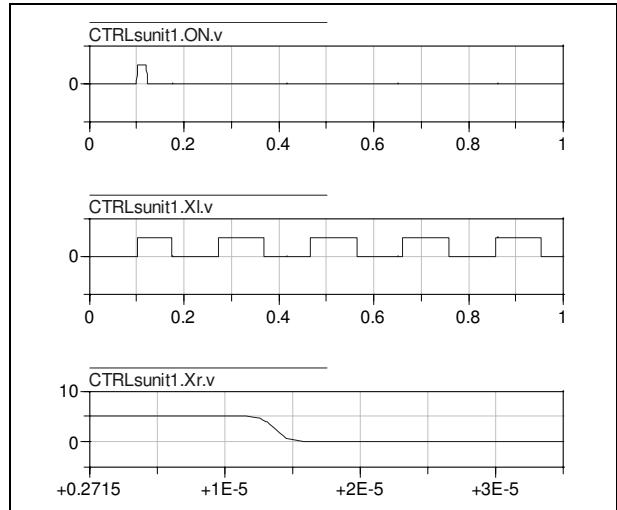


Figure 9 Start signal, command to go left, command to go right (zoomed)

Table 1 shows the simulation times and also that the multidomain model needs more computing time than the added times for Pneumatic/Boole and Integrator/CMOS. The "additional burden for multi domain" depends very much on the chosen tolerance for the DASSL integrator. In the best case, TOL = 1e-5, the complete model needs less than double the time than the model Integrator/Boole (Table 3). This effect has also been observed with a previous multidomain system (Clauß and Beater 2002).

Table 2 gives the eigenvalues of the complete system which can be uniquely associated with the pneumatic or electronic subsystem, respectively. The pneumatic system adds 14 states but the additional eigenvalues lie almost within those of the CMOS model.

6 Conclusion

A rather complicated multidomain example could be modeled and simulated in an easy way without simulator coupling. Depending on the task each of the two main subsystems was modeled more or less detailed. As a consequence the CPU times varied considerably but even for the most detailed model the "burden of multidomain" was acceptable.

However, to get more insight in the multidomain simulation with regard to both modeling and numerical aspects much more complex examples are desirable.

Table 1 Comparison of model parameters and simulation times (Dymola 5.1, Windows 2000, 2.6 GHz)

Typ	Equations	States	CPU time
Pneumatic-Boole	262	14	0.984 s
Pneumatic-CMOS	1027	57	78.3 s
Integrator-Boole	176	1	0.031 s
Integrator-CMOS	903	44	15.5s

Table 2 Eigenvalues of the system CMOS/Pneumatics

CMOS	Pneumatic
-3.1474e+006	-1.0000e+005 two times -4.8898e+004 -1.3571e+004 two times -4.3576e+002 -3.0000e+002 -2.5353e+001 -8.4914e+000
-2.9867e-001	seven times
-2.5719e-001	eight times
-1.8650e-001	
-1.4450e-001	eleven times
-9.2833e-002	seven times
-7.3814e-002	eight times
	-1.4388e-012 0 two times 1.2342e-013

Table 3 CPU times as function of tolerance

TOL	Pneumatic CMOS	Integrator CMOS
1e-4	78.3	15.5
1e-5	39.0	20.4
1e-6	49.4	26.0
1e-7	61.0	31.7
1e-8	74.3	38.9
1e-10	104	54.6

References

- Beater, P. (1999) Entwurf hydraulischer Maschinen - Modellbildung, Stabilitätsanalyse und Simulation hydrostatischer Antriebe und Steuerungen. Berlin, Heidelberg, New York, Springer-Verlag
- Clauss, C., Beater, P. (2002) Multidomain Systems: Electronic, Hydraulic, and Mechanical Subsystems of an Universal Testing Machine Modeled with Modelica. Proceedings of the 2nd International Modelica Conference, DLR München. pp 25 - 30
- Clauß, Chr., Leitner, Th., Schneider, A., Schwarz, P. (2000) Modelling of electronic circuits with Modelica. Lund, Modelica Workshop 2000, pp 3-11
- Dymola. 2003. www.Dynasim.se
- Herpy, M. (1976) Analog integrierte Schaltungen. Akadémiai Kiadó. Budapest
- HyLib (2002) Library of hydraulic components. www.HyLib.com
- Johnson, B., Quarles, T., Newton, A.R., Pederson, D.O., Sangiovanni-Vincentelli, A. (1991) SPICE3 Version 3e, User's Manual., Univ. of California, Berkeley, Ca., USA
- Lehmann, G.; Wunder, B.; Selz, M. (1994) Schaltungsdesign mit VHDL. Francis Verlag.
- Modelica (2002) Modelica Language Specification 2.0. www.Modelica.org
- Modeltech (2002) ModelSim Reference Manual
- Palnitkar, S. (1996) Verilog HDL: A Guide to Digital Design and Synthesis. SunSoft Press, Prentice Hall
- PneuLib (2002) Library of Pneumatic Components. www.Pneulib.com
- Tietze, U., Schenk, Ch. (1980) Halbleiter-Schaltungselektronik. Berlin, Heidelberg, New York, Springer-Verlag
- Tiller, M. (2001) Introduction to Physical Modeling With Modelica. Kluwer Academic Publishers.
- VHDL IEEE Package:
http://tech-www.informatik.uni-hamburg.de/vhdl/packages/ieee_1164

Object-Oriented Inverse Modelling of Multi-Domain Aircraft Equipment Systems with Modelica

Johann Bals*

Gerhard Hofer†

Andreas Pfeiffer‡

Christian Schallert§

German Aerospace Center (DLR)
 Institute of Robotics and Mechatronics
 Oberpfaffenhofen, 82234 Wessling, Germany
<http://www.robotic.dlr.de/control/>

Abstract

This paper describes the use of Modelica for investigating the multi-physical power behaviour of aircraft equipment systems within the 5th European Community (EC) programme "Power Optimised Aircraft" (POA) [1]. It gives an overview of the object-oriented structuring of an aircraft systems library which is currently being developed for the physical modelling of conventional and future "more electric" aircraft systems. An inverse modelling approach is presented, which allows to analyse the non-propulsive power behaviour as a result of given load profiles for the electrical, mechanical, hydraulic and pneumatic equipment systems. In addition the paper describes the definition of assessment criteria, to evaluate and quantify the energy consumption of the aircraft equipment systems. The criteria, their implementation in Modelica and the results from an example are presented.

Keywords: *object-orientation, aircraft systems, multi-domain modelling, inverse modelling, system assessment, more electric aircraft*

1 Introduction

Multi-physical modelling is gaining a more and more important role within areas such as robotics, the automotive or aircraft industry. Particularly with respect to the complexity of aircraft systems, such as air conditioning, electric power generation, avionics, flight controls, in-flight entertainment etc., the method of multi-

physical modelling allows to simulate all aircraft systems, which use different forms of power, in one integrated model. Different physical domains have to be considered in the simulation of complex aircraft systems. An example is presented in figure 1, which shows a diagram of the conventional power generation, distribution and use on a civil aircraft.

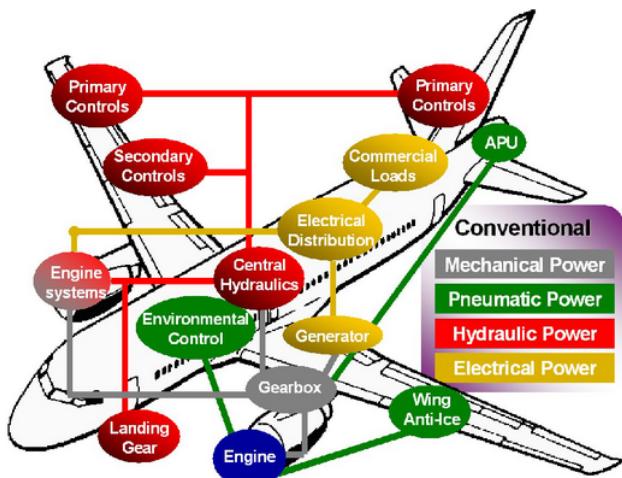


Figure 1: *Diagram of the conventional power distribution in a civil aircraft [2]*

Fuel is being converted into power by the engines of the aircraft. Most of this power is expended as propulsive power in order to move the aircraft. The remainder is converted into four forms of non-propulsive power, known as electrics, mechanics, hydraulics and pneumatics, which are necessary to operate the aircraft systems. On a conventional aircraft, a relatively large amount of the non-propulsive power extracted from the engines is lost, due to inefficient power conversion, transmission and consumption by the aircraft systems.

The European Aircraft Industry has identified the potential for improving the competitiveness of their

*Johann.Bals@dlr.de

†Gerhard.Hofer@dlr.de

‡Andreas.Pfeiffer@dlr.de

§Christian.Schallert@dlr.de

products by advancing the development of more power efficient aircraft systems. A reduction in operation cost of the next generation – power optimised – aircraft is projected to be achieved by a reduction of the system power demands, leading to savings in fuel consumption. To promote the development of new technology and more power efficient aircraft systems, the EC has founded the POA project [1], involving European aircraft, equipment and engine manufacturers. Two of the goals established for the POA project are the following: a reduction of the non-propulsive power consumption and a reduction of the fuel consumption.

Within the POA project, the aircraft manufacturer defines the top-level system requirements and a set of so called "feasible" system architectures. The engine and equipment manufacturers are responsible for developing advanced technology system components, such as generators, air conditioning packs and flight control actuators. Equipment hardware is being delivered to the so called Aircraft Systems Validation Rig (ASVR). By equipment testing on the ASVR, their performance is going to be validated while being operated simultaneously and connected to an aircraft-like electrical power supply. Whereas testing on the ASVR can represent just a cutout of a feasible systems architecture, the so called Virtual Iron Bird (VIB) offers the capability to analyse the entire aircraft architecture including all systems. Also, the VIB has the flexibility to investigate all sensible combinations of feasible system architectures. On the VIB, the aircraft systems are going to be represented by simulation models. The VIB uses component models, that are being delivered by the equipment manufacturers, to compose an integrated aircraft systems model. The models delivered to the VIB will be validated by stand-alone hardware testing done by the equipment manufacturers and by coupled hardware testing done on the ASVR. Using the validated component models, the VIB simulations can predict and compare the power consumption and behaviour of the various "feasible" system architectures. The simulation of the systems power consumption and dynamic behaviour is one of the VIB's contributions to the overall scope of the POA project. In addition, all the different system architectures are going to be optimised in a later step.

2 Object-Oriented Modelling Environment

The terms of reference within the current EC programme "POA" comprise the development of a structured simulation environment enabling to assess the various aircraft system architectures. By means of "Modelica", this simulation environment is being realised as a "Modelica Library", whose structure is presented in figure 2.

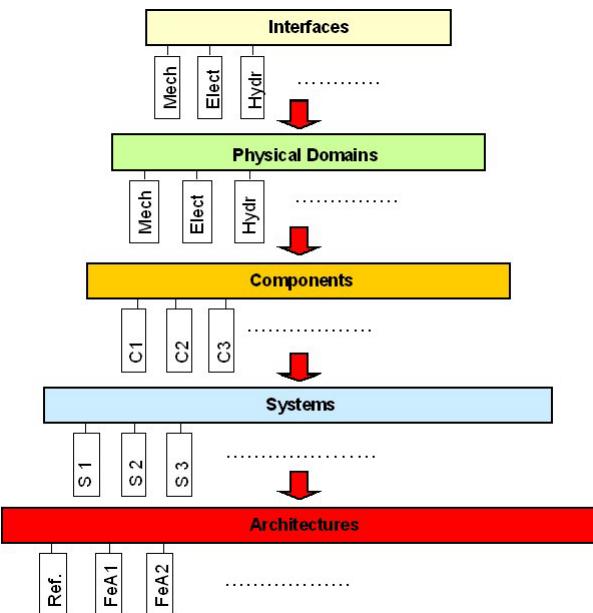


Figure 2: *Diagram of the hierarchically structured library*

Basically, the library consists of 5 levels, all of which being connected in a hierarchical manner. The sub-library, named "Interfaces", is the starting point of the entire library. It comprises several model connectors and is arranged according to the different domains, known as electrics, mechanics, hydraulics and pneumatics. The next higher level, that builds up on the sub-library "Interfaces" is called "Physical Domains", and enables the generation of basic domain specific models. Within this hierarchically structured library, the two previously mentioned levels are used to model the components of aircraft systems, as well as to generate the aircraft systems themselves. All simulation models showing the aircraft systems or their components have laid down interface definitions, which for example enable the exchange of component models with distinctive features on a specific system level. On the uppermost level of the entire library different "fea-

sible architectures” can be generated and thus assessed according to the criteria of power consumption. However, this is dependent on the number and the diversity of the numerous aircraft system architectures. The structured and object-oriented organisation of the entire library enables the automatic combination of the system models towards different architecture models. Figure 3 shows an example of an aircraft model containing an electrical power generation system (EPGS) on system level. The EPGS has several components, one of them is the shown electrical generator.

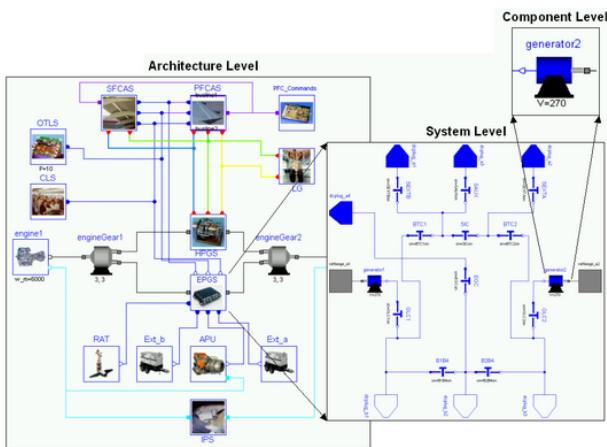


Figure 3: Modelica diagram of a hierarchical aircraft model

3 Inverse Modelling Approach

For the VIB aircraft system simulations an inverse rather than a direct modelling approach is used. An inverse model can be interpreted such that the meaning of the input and output functions is exchanged. The unknown variables of a direct model are treated as the known input functions of the inverse model, and the known variables of the direct model are treated as the unknown output functions of the inverse model.

Both modelling approaches are discussed in the following using a simple example with an electrical power source (engine and generator) and a control surface driven by an electromechanical actuator. For a given control surface load profile (torque and angular position) the basic VIB simulation task within the framework of the EC project POA is to compute the electrical power and the resulting change in fuel consumption.

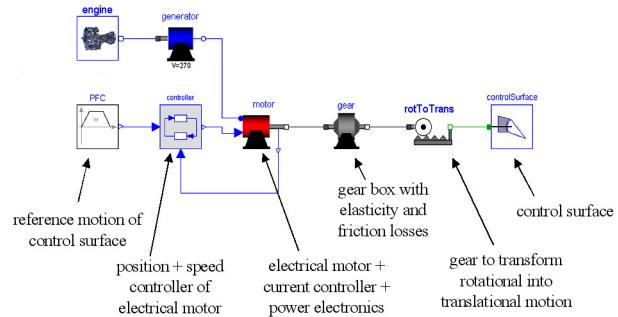


Figure 4: Diagram of a direct electro-mechanical actuator model

Figure 4 shows the direct model for the above example. The generator, driven by the engine, supplies the motor of the electro-mechanical actuator with electrical DC power. The voltage level of the generator is determined by means of the generator control unit, which is not shown in the figure 4. The motor is steered by a motor control unit and changes, via a gearbox, the position of the control surface according to the demanded values. In this example, the motor control unit commands by means of the demanded position, the necessary motor current to move the control surface under the predefined load.

For the comparison between the direct and the inverse modelling approach, only the part of the electromechanical actuator and the control surface model in figure 4 is considered. The simulation model of the electrical power source (engine and generator) is still the same for both modelling approaches. The generator model and the engine model are used in these two applications, to calculate the necessary electrical power and the resulting change in the fuel consumption.

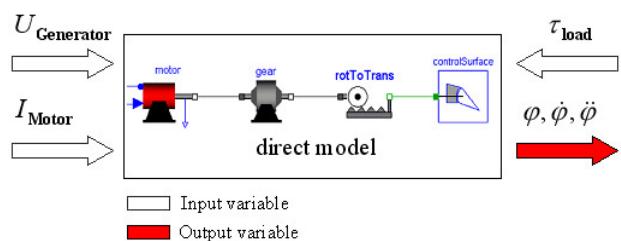


Figure 5: Diagram of the inputs and outputs variables of the direct electro-mechanical actuator model, shown as a black box

Focusing on the electromechanical actuator (motor and gearbox) and control surface model (figure 4), the input variables for the direct simulation are the motor current I_{Motor} (derived from the demanded and ac-

tual position by means of the motor controller), the generator voltage $U_{Generator}$ (impressed at the actuator motor) and the acting load τ_{Load} at the control surface (see figure 5). The unknown variable in this case is the real motion $\varphi, \dot{\varphi}, \ddot{\varphi}$ of the control surface, which will be calculated according to the given load profile. On the basis of this direct actuator model, the necessary electrical power can be computed by means of the actual actuator motor current I_{Motor} and its corresponding actuator motor voltage U_{Motor} . The actuator motor voltage U_{Motor} is an internal model variable and therefore not shown in the diagram of the inputs and outputs variables of the direct electro-mechanical actuator model in figure 5. By means of the two actuator motor variables, the necessary electrical power on the generator voltage level $U_{Generator}$ and the change in fuel consumption can be finally calculated with the generator and engine models.

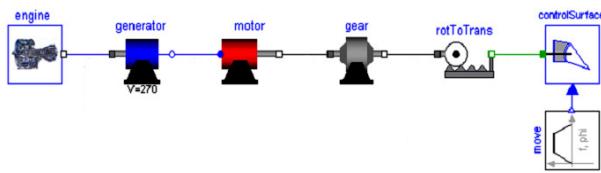


Figure 6: Diagram of an inverse electro-mechanical actuator model

Figure 6 presents an inverse model in contrary to the direct model shown in figure 4. Based on the inverse modelling definition, the meaning of input and output of the direct model is exchanged. For the inverse electromechanical actuator and surface model, the input variables are the predefined motion φ and load τ_{Load} found at the control surface and the generator voltage $U_{Generator}$, impressed at the actuator motor. The output variable (unknown variable) for the inverse model is the motor current I_{Motor} . Comparing the direct actuator model (figure 5) and the inverse actuator model (figure 7), the meaning of inverse and direct interpretation is well visible. The resulting necessary power of the generator and engine can be calculated in the same manner as for the direct model.

In Dymola, the DAE (differential-algebraic equation system) corresponding to the inverse model is being handled with the same methods like the DAE of any other (direct) model. The methods applied by Dymola are the Pantelides algorithm and the dummy derivative method. Since the Pantelides algorithm will differentiate equations, the known input functions may also be differentiated, which leads to the well known effect that the derivatives of the input functions must

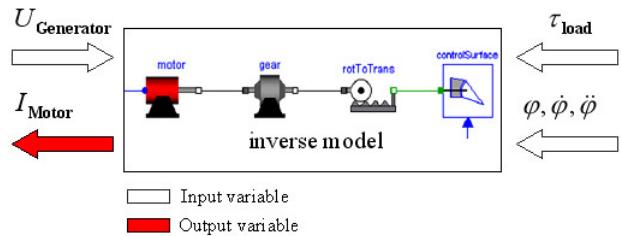


Figure 7: Diagram of the inputs and outputs variables of the inverse electro-mechanical actuator model, shown as a black box

exist up to a certain order [3].

In the present example in figure 7, it is imperative that the input signal φ is at least twice continuously differentiable to compute the required signal derivations $\dot{\varphi}, \ddot{\varphi}$ within the simulation models. To ensure that the model input signal is differentiable, the measured signal is treated by filter or spline-interpolation in this case.

Due to the fact that in Modelica the models are described in an object-oriented and physical manner, an inverse model is almost identical to the corresponding direct model. As the only significant difference, the inverse model does not require any representation of the controller structure that exists in the real system or component, whereas the direct model generally comprises the controller structure for calculation of the motor current I_{Motor} as a function of actual and demanded motor position. Due to the unavoidable control error and physical effects in the drivetrain (elasticity, friction) the actual control surface position is different from the predefined control surface position φ . This error induces errors in the resulting power consumption, which depend on the controller accuracy and the drivetrain effects.

In contrast to the direct model the inverse model matches per definition exactly the predefined load profile (τ_{Load}, φ) and therefore correctly describes the power consumption. A further advantage of the inverse modelling approach is the lower model complexity due to the absence of possibly complicated and proprietary controllers from partner companies.

For the above mentioned reasons an inverse modelling approach is used as a general concept for all of the electrical, hydraulic, mechanical and pneumatic power consumers. For each of the consumers, predefined load profiles during a typical flight profile are available to drive a multi-domain inverse model for simultaneous computation of the mechanical and pneumatic power take-off from the engines.

4 Power criteria

Among others, the goals of the POA project are the evaluation and optimisation of the power demands in future aircraft architectures. To measure and assess the quality of an architecture some criteria are needed which quantify the energy consumption, the peak power, the weight, etc. Predefined flight profiles (movement of surfaces, landing gear, state of the air-conditioning system) yield the power characteristics of the different physical domains such as hydraulics, electrics, mechanics and pneumatics from the architecture simulations. In the following the definitions of the criteria, which are related to the dynamic simulations, their implementation in Modelica and the results from an example are presented.

To evaluate the overall energy consumption during a flight profile, it is suitable to define the average power

$$P_{\text{Average}} := \frac{1}{t_e - t_0} \int_{t_0}^{t_e} P(t) dt$$

with the current power $P(t)$ at the time t , the start time t_0 and the terminal time t_e . P_{Average} describes, which integral averaged power is required for the operated manoeuvre in the timeframe $[t_0, t_e]$.

Beside the demand of average power there is also an interest on peak power which is relevant to the design of the aircraft components and systems. In a first step it is natural to define the peak power as

$$\max_{t \in [t_0, t_e]} P(t).$$

However arbitrary short peaks can unmeanly increase the value of the peak power, because only peaks holding a certain minimum duration T are of interest for evaluation. One approach for computing such a peak power could be sampling in combination with an algorithm for minimum power computation within a moving interval of length T . But this solution can be numerically very sensitive in respect of changes of initial values, parameters and the sampling time.

In order to achieve an appropriate solution, it can be helpful to define the peak power

$$P_{\text{Peak}} := \max_{t \in [t_0 + T, t_e]} P_{\text{Filtered}}(t)$$

for a fixed $T \in (0, t_e - t_0]$. P_{Filtered} denotes a filtered power characteristic determined from the original power P . The "continuously moving average" filter computes for every time point t the integral average

of the power P over a moving time window with the length T :

$$P_{\text{Filtered}}(t) := \frac{1}{T} \int_{t-T}^t P(\tau) d\tau \quad (t \in [t_0 + T, t_e]).$$

Choosing $T = t_e - t_0$ yields as a special case the average power, and the equation $P_{\text{Average}} = P_{\text{Peak}}$ holds. In this sense the peak power can be considered as a generalisation of the average power.

For implementation of the power criteria it is advantageous to define the energy function $E(t) := \int_{t_0}^t P(\tau) d\tau$. The differential equation $\text{der}(E) = P$; with the initial equation $E = 0$; determines the energy E in an unique way. Accordingly the criteria can be rewritten in terms of energy as

$$P_{\text{Average}} = \frac{E(t_e)}{t_e - t_0} \quad \text{and} \quad P_{\text{Filtered}}(t) = \frac{E(t) - E(t - T)}{T}.$$

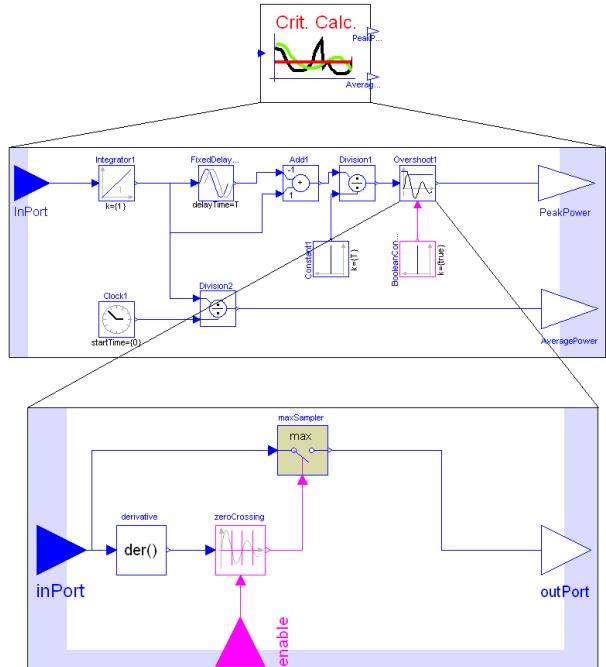


Figure 8: Modelica model for the criteria average and peak power

In figure 8 a Modelica model for the criteria is shown in the block "Crit. Cal.". The necessary time delayed evaluation $E(t - T)$ and its derivative $P(t - T)$ are realised in the block "FixedDelaywithDerivative". It remains to find the maximum of $P_{\text{Filtered}}(t)$. The general problem is to compute

$$\max_{t \in [t_0, t_e]} |u(t)|$$

for a time depending variable u . The Modelica solution with indicator functions is implemented in the block "Overshoot1" in figure 8.

```

block ZeroCrossing "Trigger zero crossing of input signal"
  extends Modelica.Blocks.Interfaces.BooleanBlockIcon;
  parameter Boolean includeInitialEvent=false
    "= true, if start time shall be included as event";
  parameter Boolean includeTerminalEvent=false
    "= true, if terminal time shall be included as event";
  end;
  Modelica.Blocks.Interfaces.InPort inPort(final n=1)
    "Input signal"
  end;
  Modelica.Blocks.Interfaces.BooleanOutPort outPort(final n=1,
    signal(fixed=true)) "Output signal" b;
  Modelica.Blocks.Interfaces.BooleanInPort enable(final n=1)
    "enables the block" b;
protected
  Boolean enabled(fixed=true) = enable.signal[1];
  Boolean disabled(fixed=false) = not enable.signal[1];
  Boolean u_pos(fixed=false) = enabled and inPort.signal[1] > 0;
equation
  outPort.signal[1] = (change(u_pos) and not initial()
    and not edge(enabled) and not edge(disabled))
  or (initial() and includeInitialEvent)
  or (terminal() and includeTerminalEvent);
end ZeroCrossing;

```

Figure 9: Modelica model for zeroCrossing

To determine the maximum of $|u|$ the block "zeroCrossing" in figure 8 creates a state event in the case that the derivative \dot{u} changes its sign (see figure 9 for the Modelica source code). The appearance of the state events is passed on as a boolean to "maxSampler" (= "triggeredMax" from ModelicaAdditions library). There the respective values of $|u|$ are compared and the greatest one is selected as u_{max} . In addition the values $|u(t_0)|$ and $|u(t_e)|$ can be selected for possible candidates of maximal values of $|u|$ by setting the parameter includeInitialEvent, includeTerminalEvent in block "zeroCrossing" in figure 9.

Due to the fact that all the time points t with $\dot{u}(t) = 0$ are defined by state events, these points and the respective values of u are computed very accurately by root finding.

Possible problems, like described for the sampling method, should be avoided by the introduced approach with filtering and determining the exact maximum of $P_{Filtered}$. It is remarkable on the shown definition and implementation of the criterion peak power, that maxima are computed with the help of derivatives, but no derivative of the power P is needed.

To demonstrate the criteria the example from chapter 3 is considered once again. Only the motor and the two gears are combined to one model "ElectricActuator" (see figure 10). The evaluation of the criteria are exemplified by the mechanical power at the engine shaft. Therefore, in figure 10 the additional model "Criteria" is inserted between "Engine" and "DCGenerator". In this model the mechanical power is mea-

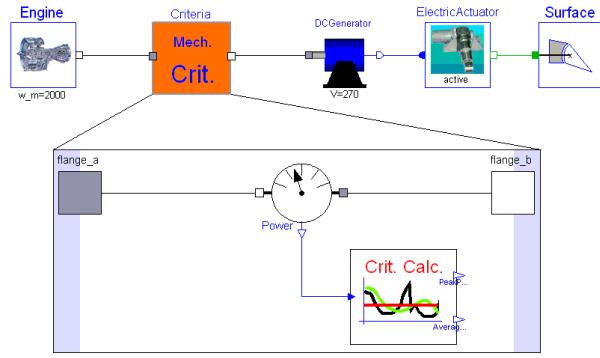


Figure 10: Model example for criteria evaluation

sured by a rotational power sensor and transferred to the criteria calculation block (see figure 8 for details) as an input signal.

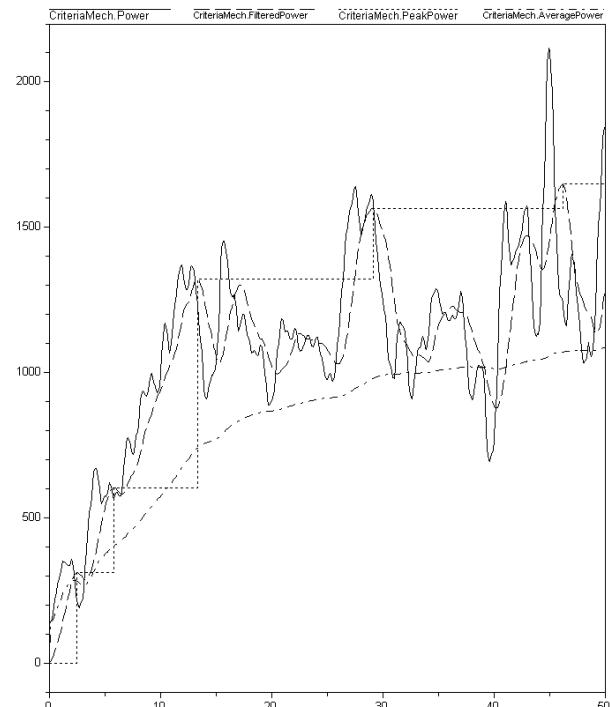


Figure 11: Simulation results of the above example in figure 10

For the overall simulation measured data for load torque and moving angles from a flight profile are loaded inside the model "Surface". The resulting power characteristics at the engine shaft are shown for 50 s in figure 11 with $T = 2$ s. Beside the both criteria – average power and peak power – the power P and the filtered power $P_{Filtered}$ are plotted as well. Please notice, that intermediate values of peak power do in general not correspond to the peak power up to the intermediate time, but only for $t = t_e$.

5 Conclusion

Within the framework of the European project "Power Optimised Aircraft" (POA), the "Virtual Iron Bird" (VIB) serves as an analysis and simulation tool to predict the behaviour and non-propulsive power demands caused by the systems installed on a large civil aircraft.

The VIB is set up as a hierarchically structured Modelica library, containing five different levels. To build up this modelling library, tailored and validated component models are being used, which are provided by the equipment manufacturers involved in the POA project.

Rather than a direct modelling approach, an inverse modelling approach is used for the aircraft system simulations on the VIB. The selected inverse approach has been described in this paper by an elementary modelling example.

In order to evaluate and later on to optimise the future aircraft architectures according to the POA project goals, certain assessment criteria are set up in Modelica for the VIB. The assessment criteria allow to quantify the different aircraft systems, which is discussed in this paper by an elementary example.

References

- [1] Power Optimised Aircraft, contract G4RD-CT-2001-00601 under the European Communities 5th framework Programme for Research – Promoting Competitive and Sustainable Growth – Key Action 4: 'New Perspectives in Aeronautics'. <http://www.poa-project.com>.
- [2] Faleiro, L.F.: Power Optimised Aircraft – The Future of Aircraft Systems. AIAA/ICAS International Air and Space Symposium and Exposition 2003: The next 100 year
- [3] M. Thümmel, M. Otter, J. Bals: Control of Robots with Elastic Joints based on Automatic Generation of Inverse Dynamics Models. IEEE/RSJ Conference on Intelligent Robots and Systems 2001

Wheel model library for use in vehicle dynamics studies

Mats Beckman and Johan Andreasson
 KTH Vehicle Dynamics, Sweden
 {mb,johan}@fkt.kth.se

Abstract

The implementation of a wheel model library is discussed. The modular structure and its benefits when configuring existing models and developing new ones is presented. The calculation of tyre-road properties is discussed, in particular the contact point estimation on uneven roads and detection of when the tyre loses contact with ground is explained. It is also shown how the implemented Magic Formula model for tyre force generation is validated and the influence of tyre dynamics on simulation time is examined.

1 Introduction

Working with vehicle dynamics modelling often requires a tyre model. A predefined library limits the effort and time needed to model a specific vehicle. This paper presents an extended and improved wheel library based on the library presented in [1].

The first attempt to solve a problem often uses a simple initial model and it is then favourable if the model can be enhanced with more complex features as more knowledge is gained. Thus, the user should be able to reconfigure the wheel models with a minimum of effort. As the complexity of the model increases, it is also desirable to be able to check sub parts separately. As a consequence, a modular structure is favourable and this is derived by identifying the tyre functions.

The models are intended to be used for vehicle simulations and will be included in future versions of the `VehicleDynamics` library [2].

2 Function identification

The function of a complete tyre can be divided into sub functions, each representing a specific tyre feature.

This makes it easier to replace sub functions and reuse code. Some sub functions that can be related to the wheel are identified and described below.

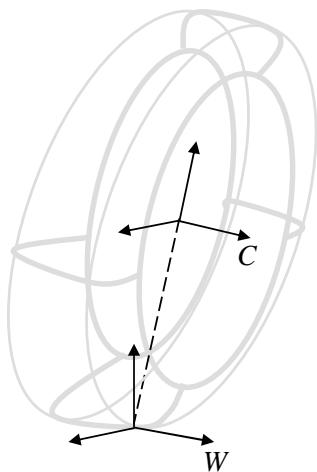
Interface An interface handles the communication between the vehicle and the wheel. To easily switch tyre model, a common interface between the vehicle and the wheel is defined. Interfaces should be able to connect to one dimensional (1D), 2D and 3D vehicle models.

Contact point The location where the tyre forces are assumed to act is of substantial interest. Finding this point, orienting it and calculating its speed are necessary in many tyre models.

Vertical dynamics Vertical dynamics is required to model the relation between the contact point and the wheel carrier. This can be modelled as linear spring-damper but it is also possible to model it more elaborately allowing e.g. the wheel to loose ground contact.

Tyre forces The tyre forces acts in the contact point in the road-plane. They are commonly identified as longitudinal force, lateral force, rolling resistance, aligning torque and overturning moment. All or some of these are normally relevant when studying a vehicle dynamics problem.

Roads Examining a vehicle's behaviour may include the use of different topological maps e.g. roads. These roads may be analytic like cross slopes, sine waves, bumps or non analytic like a measured real road, or any analytic road with a random noise component added. The road may also hold more information apart from the topology, this could include entities like friction values or road normal.

Figure 1: TYDEX frames W and C .

Graphics The visual behaviour of a model often gives the user valuable insights, it is therefore an aid if the wheel simulation can be easily visualised in a 2D or 3D environment, this includes both the visual appearance of the wheel as well some graphics representing the forces acting on it.

3 Definitions

Because of the tyre complexity, several reference frames are necessary to model its behaviour. To ensure that the model structure allows simple addition and reuse of components within new models, the modelling is based on DIN and TYDEX standards. According to the DIN representation the vehicle frame should be orientated so that x points forward, y to the left and z right up. The TYDEX definition of the carrier frame, C , and contact frame, W , is shown in figure 1. The carrier frame is fixed at the vehicle's suspension and the contact frame is located at the intersection of the carrier frame's z -axis and the road plane. For the representation of the graphics, a frame R is used to represent the rotation of the rim.

4 Implementation

In [1], each sub function of the wheel was implemented as a sub model that made it easy to reconfigure the wheel models by drag and drop functionality. The drawbacks were mainly that interfaces of the sub models required code repetition and that the structure made

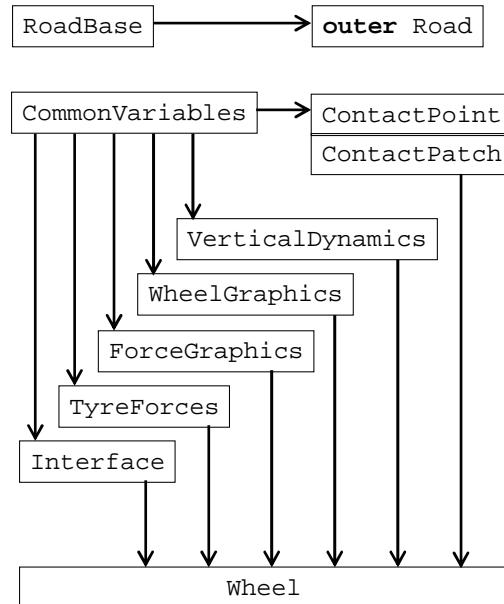


Figure 2: Model structure.

it easy to combine logically incompatible models.

To deal with this problem, the wheel model structure was redesigned with a common variable set, as well as a redesign of the drag and drop configuration to an architecture based on multiple extension. This definition requires more understanding of tyre to set up a new wheel model, thus limiting the risk of improper implementations. Still it is easy to reconfigure an already existing model.

The functionality implemented in the wheel library are found in Figure 2. A brief description of these follows below.

4.1 Common Variables

When extending multiple models, care must be taken so variable collisions are avoided. This is achieved by identifying the wheel common variables in one model and then letting every other model extend this variable set. Included are parameters and variables that describes the properties of the wheel, independent of what kind of implementation is used. Quantities like slip are thus not included since there are several different definitions. Some of the included variables are: 1) Parameters like mass and inertia as well as geometric properties such as spin axis vector and a boolean defining if the wheel is mounted towards left or right so that the model can consider wheel asymmetries. 2)

States of the wheel such as the frames C and W as well as wheel spin, camber angle and velocities in the tyre-road contact.

4.2 Interfaces

The interface defines the communication between the wheel and the vehicle with two connectors representing the states and the flow through frame C and frame R , respectively.

The frame C connectors are available for one-, two- and three-dimensional mechanics. The three-dimensional connector is either from the old or the new [3] MultiBody library. For the two-dimensional case there is the PlanarMultiBody library [1] and the one-dimensional connector is standard Translational. The two latter interfaces use parameters and external inputs to define un-used dimensions.

The frame R connector is normally a one-dimensional standard Rotational which is sufficient for most applications and also compatible with the PowerTrain library. In some cases, it is relevant to have a more detailed description of the power train and thus, frame R is also available with the three-dimensional connectors mentioned above.

4.3 Tyre Forces

The tyre forces are described in a separate model. The wheels library today, contains the Magic Formula model [4, 5], the Rill model [6] and the brush model [5].

4.3.1 Magic Formula model

The Magic formula was originally presented in [4], the idea is to represent the tyre force $f(s)$ characteristics with a trigonometric function

$$f(s) = D \sin(C \arctan(Bs - E(Bs - \arctan Bs))) \quad (1)$$

This has been improved successively and considers now aspects such as camber, vertical load and transient behaviour¹. The level of detail is controlled by user modes (UM), according to the specification in Table 1. The Magic Formula is a similarity approach, which means that it is based on the use of basic characteristics typically obtained from measurements. Through

steady state user modes	
UM0	only vertical spring
UM1	pure longitudinal slip
UM2	pure lateral slip
UM3	longitudinal and lateral slip (not combined)
UM4	combined slip forces, steady state
transient user modes	
UM11	pure longitudinal slip
UM12	pure lateral slip
UM13	longitudinal and lateral slip (not combined)
UM14	combined slip forces

Table 1: Specification of the Magic Formula user modes.

distortion, rescaling and multiplications, new relationships are obtained to describe off-nominal conditions. This classifies the Magic Formula as an semi-empiric model.

Magic Formula models the dynamic properties by calculating a dynamic slip in the longitudinal and lateral direction and then use the steady state force calculation.

4.3.2 Rill model

The Rill model calculates the slip in steady-state and calculates a corresponding tyre force with a curve fit using initial inclination $\partial f / \partial s(s=0)$, location and magnitude of max force $f_{max} = f(s_{max})$ and location and magnitude of force when the whole contact patch is sliding $f_{slide} = f(s_{slide})$ as parameters. The non-linear dependence of vertical load is handled by an interpolation between a set of the parameters for pre-defined load cases. This classifies the Rill model as semi-empiric.

Camber influence, roll resistance as well as overturning and aligning moment are then defined based on geometrical considerations. Unlike the Magic Formula model, the dynamic effects are modelled as a spring-damper filter applied after the steady state forces have been calculated.

¹The magic formula version implemented is 5.0.

4.3.3 Brush model

Unlike the Rill and Magic Formula models, which are semi-emperic, the brush model is analytical. The idea is to discretise the tyre with elastic bristles that touches the road plane and can deflect in a direction parallel to the road surface. Their compliance represent the elasticity of the combination of carcass, belt and actual tread elements of the real tyre. The effect of each bristle is added to a set of forces and torques acting on the tyre.

4.4 Contact point calculation

Using frame C and information about the road profile, the `ContactPoint` calculates location and orientation of frame W , indicated in Figure 1. Additionally, the distance between the frames and its time derivative as well as the camber angle are calculated.

The orientation of the frames are related as described by their unit vectors:

$$\begin{aligned} {}^W e_z &= n \\ {}^W e_x &= {}^C e_y \times {}^W e_z \\ {}^W e_y &= {}^W e_z \times {}^W e_x \end{aligned} \quad (2)$$

where n is the road normal. The actual location of frame W can be found by iteration as suggested in [6] which was implemented in Modelica in [7]. The idea is to start at the location of frame C , r_C and define a first approximation of the contact point, $r_{P1} = -R_0 {}^C e_z$ where R_0 is the undeformed wheel radius. The (x,y) -coordinates are then used to find the actual road altitude, z , giving $r_{P2} = (r_{P2}[1], r_{P2}[2], z)$. Due to camber, tyre deflection, and road uneveness, r_{P2} is normally not located along the line between r_C and r_{P1} , r_{P2} is then projected onto this line giving r_{P3} . However, if the road is uneven r_{P3} is no longer located at the road surface. Then r_{P3} is used as a new r_{P1} and the calculations can be iterated until the accuracy is sufficient.

However, the iteration may also diverge depending on the road surface, and the method has difficulties to cross sharp edges. Thus this method is not suitable when using e.g. meshed roads. Instead, the contact point is calculated using the deformation of the tyre from the previous time step. This allows the wheel to travel over unevenesses without causing numerical problems. Also a simple model that assumes a flat surface can be used to speed up simulations when appropriate.

4.5 Contact patch filtering

In reality, the contact between the road and tyre is spread over a patch about 1 dm^2 , depending on tyre dimensions, pressure, load and cambering. The tyre force models that are based on a contact point representation all require that the actual patch is similar to the test conditions when the tyre parameters were estimated. Different tyre pressure and load is normally tested in a test rig and can thus be handled by the tyre force model. However, when the road unevenes is significant within the tyre patch range, these have to be accounted for by some kind of filtering. In [5] a filter is suggested that lets a set of solid ellipses travel over the road profile and geometric calculations then give a resulting road plane that is used for the tyre force calculations.

This method is not implemented since it is believed to be very time consuming. Instead, a simpler filtering is implemented based on either a rectangle or a cross. Assuming that the contact patch can be represented by rectangle, then the resulting road plane is calculated as:

$$\begin{aligned} k &= \sum_{i,j} k_{i,j} \\ z &= \frac{1}{k} \sum_{i,j} k_{i,j} z_{i,j} \\ \frac{\partial z}{\partial x} &= \frac{1}{k} \sum_{i,j} k_{i,j} \frac{z_{i,j} - z}{\Delta x} \\ \frac{\partial z}{\partial y} &= \frac{1}{k} \sum_{i,j} k_{i,j} \frac{z_{i,j} - z}{\Delta y} \end{aligned} \quad (3)$$

where $k_{i,j}$ is a weight distribution. The number of evaluation are $i \cdot j - 1$. To speed up the calculation a cross shape can be used instead reducing the number of evaluations to $i + j - 2$.

4.6 Vertical dynamics

The vertical dynamics handles the load carrying task of the tyre belt. Typically this is modelled as a spring-damper with the exception that the tyre only generates vertical force when in contact with the ground. In the basic case, this is formulated as

```
contact = R < R_0;
```

In this case, there is contact as long as the distance between the wheel centre and the ground, R , is shorter

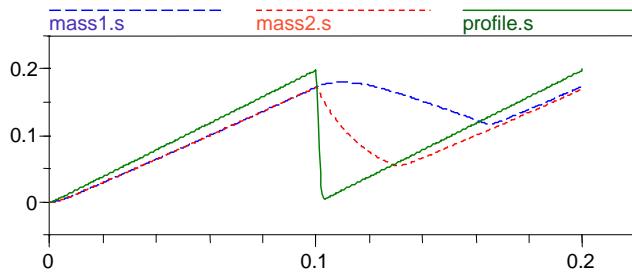


Figure 3: The simpler model (dash-dotted red) and the more advanced model (dashed blue) passing over a road profile (solid green).

than the undeformed tyre radius, R_0 . However this assumes that the vertical dynamics of the tyre belt is infinitely fast and leads to problem when the road surface is uneven.

When the tyre dynamics needs to be considered the following model is used

```
v = der(R);
v1 = -R1*c/d;
der(R1) = if (v < v1 and contact)
    then v else v1;
contact = R1 >= R;
```

Here, an additional state $R1$ is introduced to keep track on the actual deformation of the tyre. This state is limited so that once the tyre is compressed, it can cannot increase faster than the dynamics of the tyre. This needed when travelling over a road surface with a sudden quick altitude decrease, i.e. a pot hole. This is illustrated in Figure 3 where the two models are passing over a road profile². When passing the bump, the simple contact model fails to detect the loss of contact and forces the tyre downwards in an unnatural way trace 2. The more advanced model consider the fast change of the road plane which results in a better behaviour, trace 1.

4.7 Road

The wheel models need information about the altitude and the road surface condition at the tyre-road contact. These should however be independent of the wheel model and thus this road information is stored in a separate model together with graphical information. Since the road and the tyre exchange data, the standard

²The profile is exaggerated to make the difference appear clearly.

way in Modelica would be to have a road component which is connected to all wheels of a vehicle. This would however result in a close coupling of vehicle and road model. Instead the inner/outer Modelica language constructs are used, that only requires that the road model is defined at the top level of the vehicle model.

Information about the road condition is normally required once when generating tyre forces while the altitude may be called several times by both ContactPoint and ContactPatch. Thus it must be possible to call altitude and road condition separately and to deal with this, a basic road is defined as:

```
partial model RoadBase
    replaceable block Altitude = BaseXY;
    replaceable block Condition = BaseXY;
    parameter Integer nAltitudes=0;
    parameter Integer nConditions=0;
    Altitude altitude[nAltitudes];
    Condition condition[nConditions];
end RoadBase;
```

Here, `BaseXY` is a block taking two inputs Real x, y and returns Real z . This can be used for both altitude and road condition.

In the `CommonVariables`, a model `Road` is defined as:

```
outer model Road = RoadBase;
```

A model can then be instantiated whenever needed in the wheel model according to:

```
Road road(nAltitudes=n1, nConditions=n2);
```

giving a `road` containing vectors `altitude` and `condition` with $n1$ and $n2$ elements, respectively. In the top model, the desired road can be selected by setting:

```
inner model Road = DesiredRoad;
Road road;
```

In this case, the actual road model in the instances of the wheel is `DesiredRoad`, which easily can be swapped to any other road extending the `RoadBase` using the `redeclare` syntax.

4.8 Graphics

The `Wheels` library contains graphics that represents the wheel as well as forces generated in the contact point. The visualisation of the road is stored in the road model described above.

5 Validation

The Magic Formula implementation has been validated using the magic formula tyre already implemented in ADAMS/Car [8]. ADAMS/Car does not have a convenient configuration method to let the user decide tyre inputs like load, slip and camber, the easiest way to validate was to pick a full vehicle manoeuvre and export the slip, tyre load and camber angle for each wheel and then use these values as input to the Modelica model. Thus making sure that the same inputs generate the same output, Figure 4.

The chosen manoeuvre is a break in turn, which leads to both lateral and longitudinal slip. The braking force was set high enough to cause lock up, in Figure 4 this happens at $t = 3\text{s}$ when κ reaches -1.

The validation model is realised as an interface. The variables unknown to the tested sub model are provided by the test interface. These variables are set as parameters or external inputs and are controlled by the user.

6 Usage

The Wheels library allows the user to use wheel models already implemented, to configure these and to implement own models within the structure. All wheel models extends an interface model, thus allowing the use of the replaceable syntax along with `choicesAllMatching`, presenting all compatible wheel models to the user. This makes it easier to handle wheel model changes in a full vehicle model. New models can be made and the structure makes reuse of elements from models already implemented intuitive and code effective.

The use of replaceable models makes it possible to do most testing with two flexible models that can be configured with drop-down boxes as illustrated in Figure 5. In the first rig the wheel can be either free, constrained or affected by forces or torques. Also the test road and of course the tested wheel can also be exchanged in the same manner. The second rig is a mass mounted on the wheel via a spring-damper, representing the suspension and the distributed body weight. Since only one wheel is used, this is often referred to as the *quarter car model*. Except for the vertical motion, the rig can be controlled as previously mentioned.

Figure 6 shows an animation where a wheel is

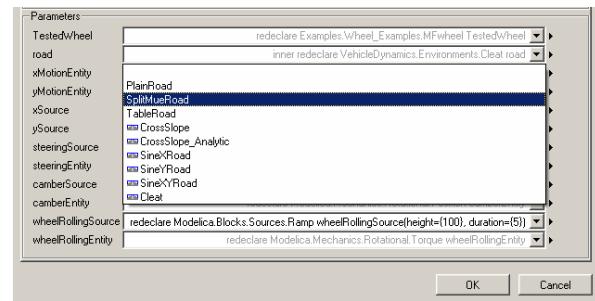


Figure 5: Parameter with drop down-boxes showing the available roads.

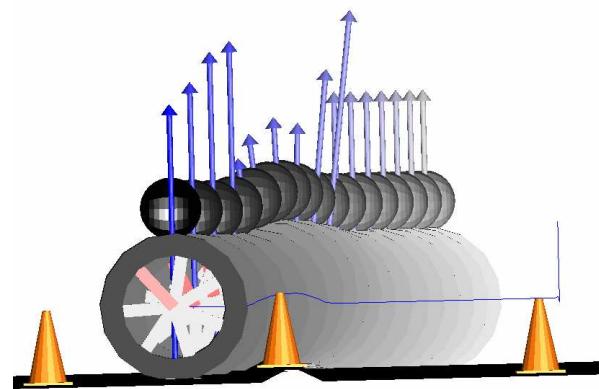


Figure 6: Animation view of tyre passing over a cleat.

used in a quarter car model. The model passes a cleat located by the centre cone, the normal force vector shows the tilt of the contact patch when the wheel ascends the cleat.

The CPU time required for different manoeuvres and different tyre configurations is measured to give an idea of the computational effort required. The manoeuvre simulated is a start from standstill with an applied torque on the drive shaft. At time $t = 4\text{s}$, when speed is gained, a ramp signal is applied turning the wheel around its vertical axis 4° in 0.1s . At time $t \approx 6.5\text{s}$ the wheel meets a slope that is reducing the wheel's speed until it stops at time $t \approx 7.2\text{s}$ and starts rolling back down, reaching the flat surface at time $t \approx 7.7\text{s}$.

The CPU time required for this manoeuvre at a 1.5GHz Pentium4 with 512Mb ram is measured and presented in Figure 7. The models compared are Magic Formula user modes 14 (MF UM14) and 4 (MF UM4) as well as a modified user mode 14 with a simpler transient slip model. The Rill model is also compared to these.

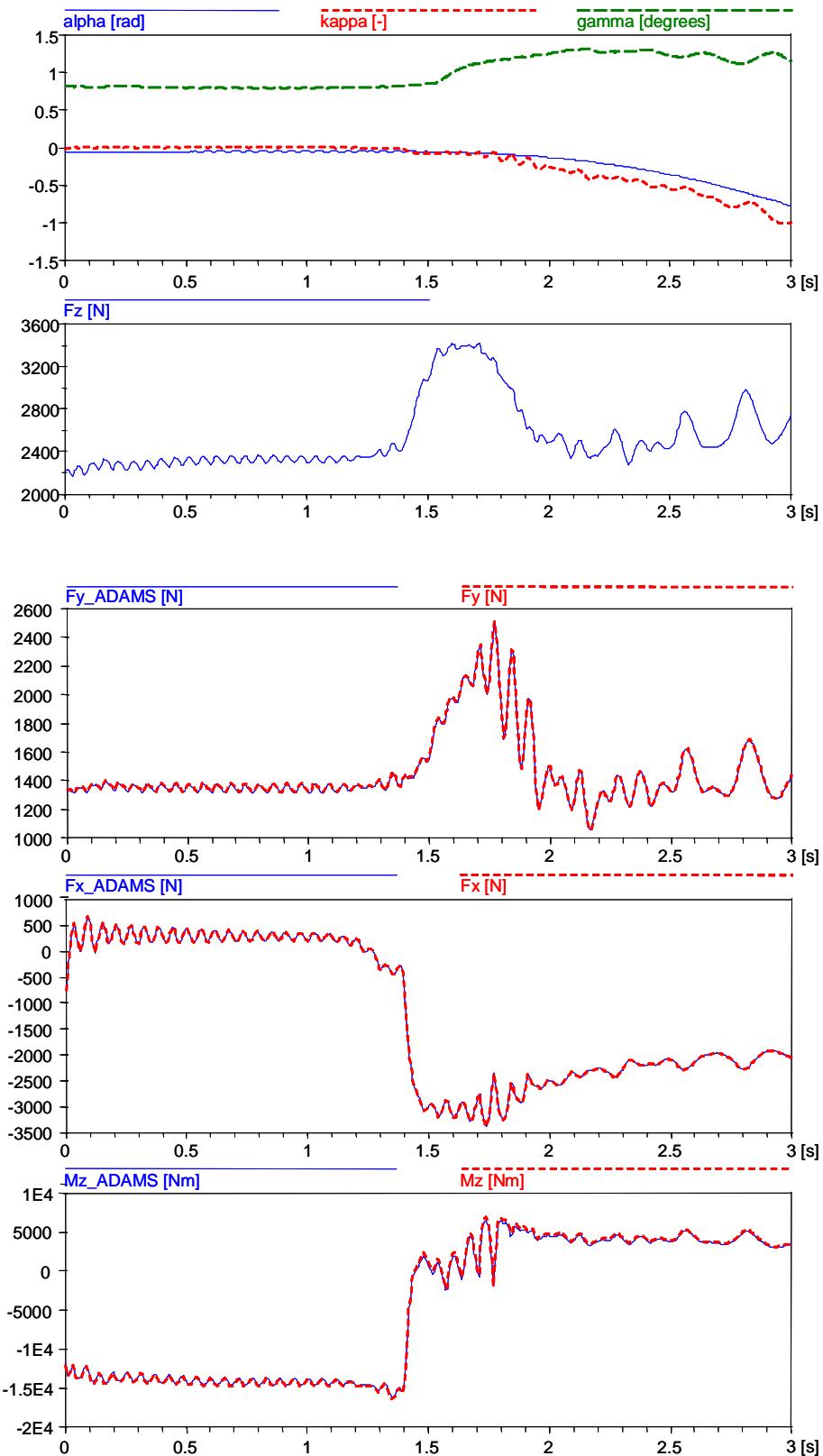


Figure 4: Validation plots comparing the result from the `wheels` library and the ADAMS output.

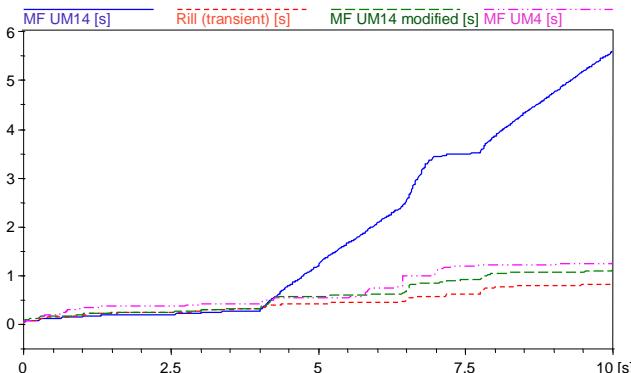


Figure 7: CPU time for different tyre models.

Figure 7 shows that UM14 requires a significantly more computational effort compared to UM4. The extra computing effort originates from the transient slip model implemented in UM14. The modified UM4 has a transient behavior modelled as a first order filter with coefficients only depending on wheel spin velocity. This model lacks physical characteristics that UM14 features such as relaxation length and load dependence. However the modified UM14 model is capable of starting from standstill which is not possible with a steady state model as the computational effort is on par with the original UM4. A disturbance to the UM14 models seems to oscillate for a longer period of time than other models like Rill. Besselink [5] has proposed a damping term that may address this issue.

7 Conclusions

The wheels contain ready-to-use tyre models as well as components that can be used to design own models. The modular structure makes it easy to reconfigure existing models and to reuse code when adding new functionality.

Compared to the previous version of the library, the models are further validated and new functionalities are implemented such as a better vertical behaviour and ability to handle uneven roads. Additionally, interfaces to the new MultiBody library are added.

8 Acknowledgements

This work has been funded by KTH Vehicle Dynamics and the Driving Dynamics project within the Swedish National Research Programme "The Green Vehicle/FCHEV". The authors would also like to thank Martin Otter and Hans Olsson for valuable support.

References

- [1] J. Andreasson and J. Jarlmark. Modularised Tyre Modelling in Modelica. In Peter Fritzson, editor, *Proceedings of the 2nd International Modelica Conference*, Oberpfaffenhofen, March 2002. The Modelica Association and Deutches Zentrum für Luft- und Raumfahrt.
- [2] J. Andreasson. Vehicledynamics library. In Peter Fritzson, editor, *Proceedings of the 3rd International Modelica Conference*, Linköping, November 2003. The Modelica Association and Linköping University.
- [3] M. Otter, H. Elmquist, and S.E. Mattson. The new Modelica MultiBody library. In Peter Fritzson, editor, *Proceedings of the 3rd International Modelica Conference*, Linköping, November 2003. The Modelica Association and Linköping University.
- [4] E. Bakker, H.B. Pacejka, and L. Lidner. A new tire model with application in vehicle dynamics studies. *SAE transactions, paper 890087*, pages 83–93, 1989.
- [5] H.B. Pacejka. *Tyre and vehicle dynamics*. Butterworth Heinemann, 2002.
- [6] G. Rill. *Simulation von Kraftfahrzeugen*. Vieweg, 1994.
- [7] J. Andreasson, A. Möller, and M. Otter. Modeling of a racing car with Modelicas MultiBody library. In Peter Fritzson, editor, *Proceedings of the Modelica'2000 Workshop*, Lund, October 2000. The Modelica Association and Lund University.
- [8] ADAMS, Mechanical Dynamics Inc. <http://www.adams.com/>.

MODELICA LIBRARY FOR SIMULATING ENERGY CONSUMPTION OF AUXILIARY UNITS IN HEAVY VEHICLES¹

Niklas Pettersson^{a,b} Karl Henrik Johansson^b

^aScania CV AB

^bDepartment of Signals, Sensors & System, Royal Institute of Technology

^aScania CV AB, 151 87 Södertälje, Sweden

^bOsquldasväg 10, 100 44 Stockholm, Sweden

niklas.pettersson@scania.se, kallej@s3.kth.se

Abstract

Models that can be used to analyse the fuel consumption of auxiliary units in heavy vehicles are presented. With the purpose of evaluating the influence from various drive concepts and control principals, a model library is developed in the modelling language Modelica. The library contains a mixture of models developed from physical principles and models fitted to collected data. Modelling of the cooling system is described in some detail. Simulation results are compared with measurement data from tests in a wind tunnel.

1 Introduction

This paper presents the work of developing vehicle models that can be used to evaluate alternative architectures for the drive of auxiliary units in heavy vehicles. With aid of the simulation models, the energy savings of new designs can be assessed, (Pettersson and Johansson, 2004). Here the ideas behind development and maintenance of a comprehensive model library are presented. The Modelica language is used to build models with a modular structure. Figure 1 shows composition of the model at the highest level. A more extensive version of the paper can be found in (Pettersson and Johansson, 2003).

In the simulations, the vehicle is set to drive a road with varying topology and speed limit that have been obtained from recordings of real roads. The vehicle is assumed to run on cruise control and with computer-controlled gear shifting (automated manual transmission). Algorithms from the production version of the control are incorporated in the simulation model. The vehicle model has been validated with respect to the energy consumption of

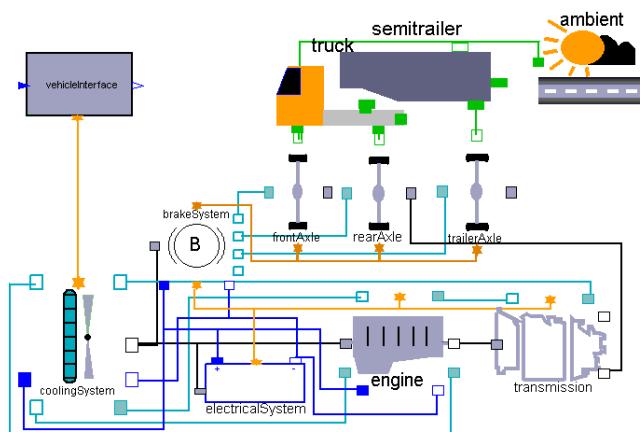


Fig 1. Modules of the simulation model.

the combustion engine and losses such as rolling resistance and air drag, (Sandberg, 2001). Influences from the, sub-systems, the cooling system, and the electrical network, were only included as a lumped effect on the net fuel consumption. This work refines the description of the auxiliary units. The paper describes the modelling of the cooling system in some detail. Sub-models are built from physical principles, resulting in grey-box models with parameters identified from various tests in a laboratory environment.

The sub-models are assembled into a model of the complete vehicle. Measurements collected from tests in a wind tunnel are used to tune the performance of the total model. Validation data is recorded from a dynamic driving cycle in the wind tunnel.

¹ This work is supported by Scania CV AB and Vinnova

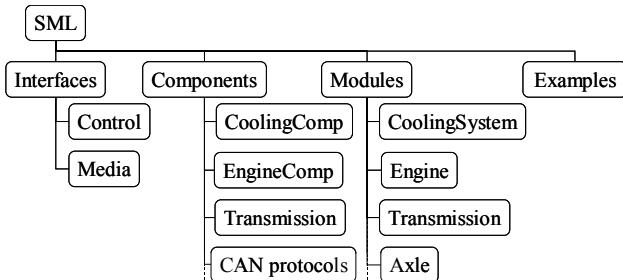


Fig. 2. Structure of the Scania Modelica Library.

2 Model library

The library is developed in Modelica, (Modelica Association, 2000). Modelica is well suited to describe behaviour of complex systems containing parts from different engineering disciplines, e.g., mechanics and electronics.

In contrast to the Modelica Standard Library, the library is not organised in different engineering disciplines. Instead it is organised after the parts of the truck. The library, named Scania Modelica Library, SML, consist of four main branches:

1. Interfaces
 2. Components
 3. Modules
 4. Examples

The principal structure of the library can be viewed in figure 2.

The Interface branch contains classes describing connections between model components. Although the library relies heavily on connector classes defined in the Modelica Standard Library, some unique connectors are defined. One example is the CAN connector, used to mimic the information flow between control units in the truck. Further, under the Interfaces sub-library Media, base classes for thermodynamic and hydraulic models are found. These base classes are mainly used in models of components in the cooling system. In the thermodynamic and hydraulic base classes many of the modelling ideas used are adopted from Modelica library ThermoFluid developed by Thummescheit, et al. (2000). However, here a somewhat simpler structure and less extensive description of media properties are used. In the Components branch models of all physical parts needed to build up the complete model of a truck are gathered. Modules, in the next branch, are a higher level of abstraction, and contain more compound models. The idea is to define a set of generic modules with well-defined

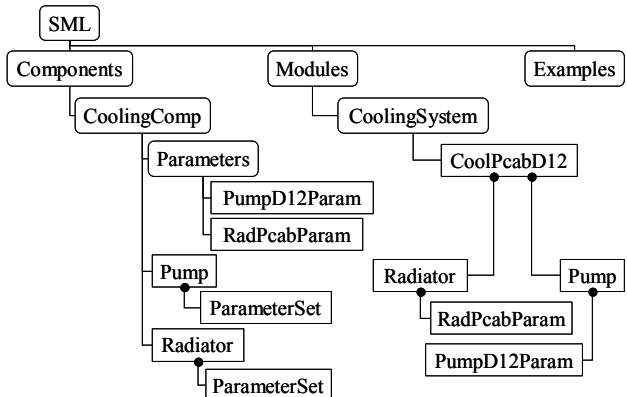


Fig. 3. Parameterisation of the model exemplified with the cooling module.

interfaces that can be used to for simulations with various purposes. In the last branch a number of working examples is built that can be used directly for simulations.

Figure 3 illustrate how the models are parameterised to obtain modules that correspond to physical modules. Each component contains a placeholder for a set of parameters of a defined structure. Parameter sets with values describing various versions of the components are gathered in special sub-libraries. When modules are put together, illustrated with the cooling module, the generic placeholders are replaced with the parameter set of the current versions of components. With this procedure, numerous variants of aggregated modules can be compiled from a small number of basic components and parameter sets.

3 Cooling system module

The cooling system is one of the modules of the vehicle model. Energy consumers in the cooling system are primarily the cooling fan and the water pump.

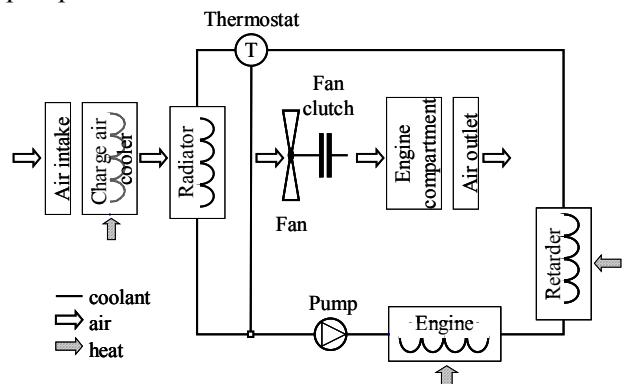


Fig. 4. Components in the cooling system module.

In heavy vehicles, these units normally are mechanically driven. The model corresponds to the current design of a Scania truck where the water pump is directly driven from the crankshaft while the cooling fan is connected to the shaft via a viscous clutch enabling a passive speed control. However, the basic structure allows for changing the model to describe other ways of driving and controlling these auxiliaries.

3.1 Cooling system components

The main parts of the cooling system are modelled, using the thermodynamic and hydraulic base classes. In figure 4 the structure of the cooling system is depicted. The model mainly consists of two adjoining flows of mass and energy: the flow of coolant fluid and the airflow.

The pump drives the flow of coolant fluid through the engine and the retarder. The retarder is a hydraulic brake mounted on the secondary side of the gearbox. When used to brake the vehicle, it produces heat that is emitted to the cooling system. The temperature of the coolant is controlled with the thermostat by splitting up the coolant flow into one part passing the radiator and one part flowing in a by-pass pipe. The air enters the cooling system in the air intake at the front of the truck cab and exits at the air outlet at the rear. The airflow is partly driven by the fan and partly by the pressure build up caused by the wind speed at the intake and outlet. The air is used to cool down both the turbo charged intake air to the engine, and the coolant fluid. The charge air cooler, or intercooler, and the radiator are connected in series so that the cooling air first passes the charge air cooler and then the radiator. Both charge air cooler and the radiator is cross directional heat exchangers, i.e., the hot and cool media streams are perpendicular to each other.

The models of the coolant and the air streams are built up with alternating control volumes and flow models. In the control volumes, mass and energy balances are defined, while in the flow models, relations between the pressure drop and the flow are determined. The control volumes describe the dynamic behaviour and are parameterised purely with geometrical quantities and properties of the contained media. The flow models describe pressure drops, heat transfer and consumed power based on empirical relations. No explicit identification of the parameters of the control volumes is needed, since they could be found in the technical specification of

the components: The parameters of the flow models, however, typically have to be estimated from experimental data.

3.2 Dynamics of the cooling system

For the control volumes it is possible to select which state representation that should be used. The transformation of state variables from the primary mass and energy balances to the selected states is dependent on the properties of the media inside the volume. The modelling of the control volumes is rather standard. Here it essentially follows the principles used in ThermoFluid (Tummescheit et al. 2000).

For the airflow, pressure, p , and temperature, T , are chosen as state variables. The transformed balance equations then become

$$\begin{aligned} m \frac{\partial u}{\partial p} \dot{p} + m \frac{\partial u}{\partial T} \dot{T} &= \dot{U} \\ V \frac{\partial \rho}{\partial p} \dot{p} + V \frac{\partial \rho}{\partial T} \dot{T} + \rho \dot{V} &= \dot{m} \end{aligned} \quad (1)$$

Here \dot{U} and \dot{m} denote the net flow of energy and mass into the control volume while m and V are the mass trapped in the volume and the size of the volume, respectively. Additionally, the air is regarded as an ideal gas yielding the following expressions for the density, ρ , and the partial derivatives in equation (1)

$$\begin{aligned} \rho &= \frac{pM}{TR} \\ \frac{\partial u}{\partial p} = 0, \quad \frac{\partial \rho}{\partial p} &= \frac{M}{TR} \\ \frac{\partial u}{\partial T} = c_v, \quad \frac{\partial \rho}{\partial T} &= -\frac{pM}{T^2 R} \end{aligned} \quad (2)$$

where M denotes the molar mass and c_v the specific heat capacity at constant volume, respectively, while R is the molar gas constant.

Similar expressions are used for the state derivatives of the coolant media, although only the temperature is chosen as state variable. The pressure of the coolant is determined purely from static hydraulic relationships.

3.3 Parameters of the flow models

For the airflow, pressure drops in the components along the flow path are modelled as an exponential friction loss

$$\Delta p = c | q | \dot{m}^e \quad (3)$$

The frictional pressure losses in the components coolant path is modelled with a second order polynomial

$$\Delta p = c_2 | q | q + c_1 q \quad (4)$$

The pressure rise in the pump and the fan depend on the flow through the components and the angular velocity of the shaft. In the model the following equations are used to describe the operation of the pump and the fan, respectively

$$\Delta p = R_1 |\omega| \omega + 2R_2 \omega q - R_3 |q| q \quad (5)$$

$$\Delta p = R_1 \rho |\omega| \omega + 2R_2 \omega \dot{m} - R_3 |q| \dot{m} \quad (6)$$

In equation (3)–(6), q and \dot{m} denotes volume flow rate and mass flow rate, respectively, while ω denotes the angular velocity of the pump or the fan.

The wind speed gives rise to a differential pressure at the air intake and outlet relative the ambient pressure. In the model, the pressure difference depends on the wind speed, v , the air density, ρ , and the non-dimensional coefficient CD according to

$$\Delta p = CD \frac{\rho}{2} v^2 \quad (7)$$

In order to find the parameter values of the sub-models, experimental data is collected from tests on individual components in a laboratory environment. Essentially parameters of equation (3)–(7) and other characteristics are identified for each component depicted in the overview of the cooling module in figure 4. Table 1 summarises which parameters that are identified and what data that are used.

Table 1 Summary of model components in the cooling module.

Component	Characteristic	Data source	Slack
Pump	- Pressure rise - Power consumption	Rig test Rig test	s
Engine	- Flow resistance - Heat capacitance - Heat emission to coolant - Heat emission from charge air	Rig test Data sheet Rig test Rig test	s
Retarder	- Flow resistance - Heat capacitance - Heat emission	Rig test Data sheet None	s
Thermostat	- Opening characteristic - Flow resistance - Dynamic response	Rig test Rig test Rig test	
Radiator	- Flow resistance coolant - Flow resistance air - Operating characteristics - Heat capacitance	Rig test Rig test Rig test Data sheet	
Air intake	- Pressure build-up	None	s
Charge air cooler	- Flow resistance	Rig test	s
Fan	- Pressure rise - Power consumption	Rig test Rig test	
Fan clutch	- Slip characteristics	Rig test	
Engine compartment	- Flow resistance	Rig test	
Air outlet	- Pressure build-up	None	s

Input from other parts of the total model is primarily heat losses that need to be cooled away. The engine emits heat to the cooling system both directly into the engine block, which is heated up by the combustion, and through the cooling of the charge air. The amount of heat depends on the current torque and speed of the engine.

In the model this is calculated from a look-up table. The table is obtained from measurements done in test cells. The heat emitted to the cooling system from the retarder is directly proportional to the braking power. In some sub-models, the parameters solely represent basic quantities such as mass or volume that are found from the data sheet of the corresponding component. The tests are performed in the laboratory under well-controlled conditions. As a result the obtained prediction errors are very small as can be seen by the example in figure 5, showing the pressure drops in the airflow path.

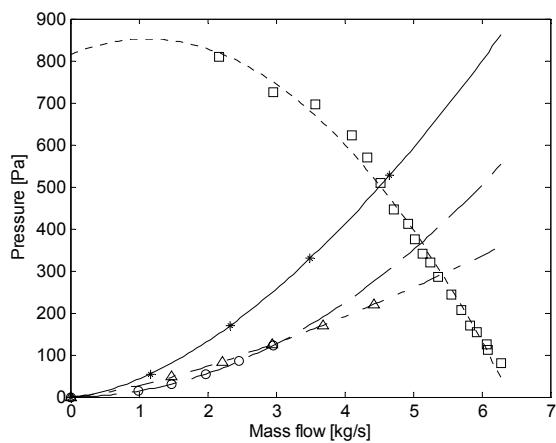


Fig. 5. Pressure drop as a function of airflow for the charge air cooler model (solid) compared with measurements (stars). Corresponding drops for radiator (dashed and triangles), and engine compartment (dash-dotted and circles). Pressure rise of the fan model (dotted) at 1400 rpm compared with measurements (squares).

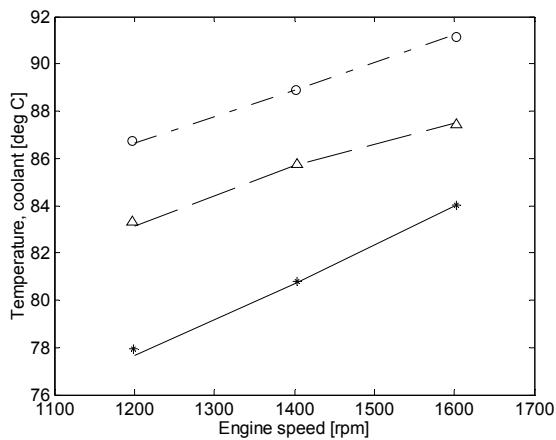


Fig. 6. Simulated temperature of the coolant in steady state at 80 km/h with full load and different speeds on the engine (solid) compared with measurements (stars). Corresponding at 60 km/h (dashed and triangles) and at 40 km/h (dash-dotted and circles).

4 Assembling the total model

The modelling errors in the sub-models are very small. However, when they are assembled to a full model, effects that are not handled in the sub-models may play an important role. It may be effects from the installation the truck cab such as the piping between the components. Non-linearities may amplify small errors in the sub-models when these are connected and new feedback paths are closed. It can be shown, using a simplified model of the cooling system, that the change of temperature of the coolant in steady state due to a small perturbation of the airflow is proportional to the squared inverse of the airflow. Thus, the simulated temperature will be very sensitive to modelling errors influencing the airflow. Further, for the pressure build-up due to the wind speed there exists no practicable experiment on a component level. Therefore, the result of the total model is verified through comparison with experimental data collected in a wind tunnel. In the wind tunnel, the vehicle is driven on a dynamometer with a defined load and speed of the engine. Fans are used to simulate the wind speed. Results from nine steady-state tests and two step-response tests are used to tune the model parameters. A number of the parameters in the sub-models are assigned as slack parameters that are adjusted to fit the behaviour of the total model to the measurements. In table 1 the choice of slack parameters is indicated in the last column. In figures 6 and 7 the cooling temperature obtained with the tuned model are compared with measurements.

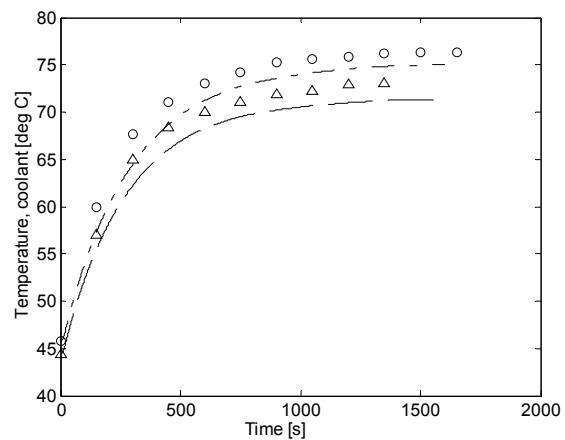


Fig. 7. Simulated response of the coolant temperature on a step in the engine load at 60 km/h with engine speed 1400 rpm (dashed), compared with measurements (triangles). Corresponding at 40 km/h (dash-dotted and circles).

Validation of the total model is performed. Data is recorded during a dynamic drive cycle in the wind tunnel, where the load and speed of the dynamometer is programmed to follow a cycle corresponding to a specified road. In figure 8 the simulation result is compared with measurements where the dynamometer follows the profile of a 57 km section of the road between the cities Koblenz and Trier in Germany. The validation shows that the model is capable to capture the main dynamics of the cooling system while it does not describe the small oscillations observed in the measurements. The oscillations around 80° C most likely have its origin in the complex dynamics of the thermostat. The model of the thermostat is a rather rough approximation and do not give raise to corresponding oscillations around the opening temperature. Despite the observed differences, the model should be sufficient to evaluate the energy consumption of the auxiliary units in the cooling system.

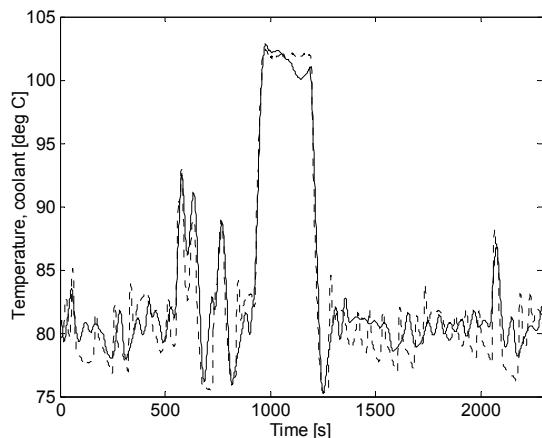


Fig. 8. Simulated coolant temperature (solid) during a dynamic driving cycle compared with measurements (dotted).

REFERENCES

- Modelica Association, (2002). ModelicaTM - A Unified Object-Oriented Language for Physical Systems Modeling. *Language Specification Ver2.0. <http://www.modelica.org/>*, 2002.
- Pettersson, N., Johansson K. H., (2003). Simulating energy consumption of auxiliary units in heavy vehicles. *Proceedings of 13th IFAC Symposium on System Identification*.
- Pettersson, N., Johansson K. H., (2004). Optimal control of the cooling system in heavy vehicles. *Proceedings of IFAC Symposium on Advances in Automotive Control*.
- Sandberg, T., (2001). *Heavy Truck Modeling for Fuel Consumption Simulations and Measurements*. Licentiate thesis, department of Electrical Engineering, Linköping University Sweden.
- Tummesheit, H., Eborn J. and Wagner FJ., (2000). Development of a Modelica Base Library for Modeling of ThermoHydraulic Systems. *Proceedings of Modelica Conference 2000*.

Session 9B

Tools - II

Approximation of black-box system models in Matlab with direct application in Modelica

Wim Lammens, Jos Vankan, Robert Maas and Johan Kos

National Aerospace Laboratory NLR
 P.O. Box 90502, 1006 BM Amsterdam, The Netherlands
 email: lammens@nlr.nl

Abstract

Modelica provides a hierarchical and object oriented approach for the modelling of dynamic systems. A system can relatively easily be composed by connecting a number of sub-system and component models together. The resulting integrated system model can be used in design and optimisation studies. The physical behaviour as defined in the sub-system and component models is a key determinant for the system behaviour. Therefore it is of importance that this physical behaviour is adequately modelled. It may however occur that some component models are, for various reasons, represented by no more than a data set of computational or experimental results of the component behaviour. If the precise physical basis for the behaviour of such components is not known or deliberately not taken into account, their behaviour can be considered as a "black-box" input-output relation. In such cases a black-box modelling approach is useful.

This paper describes a generic modelling approach based on approximation methods and applicable for black-box type models in Modelica. Various approximation techniques including polynomial methods, splines, neural networks and kriging models, are applied from a Matlab based graphical software tool with an automatic interface to Modelica. The complete process of model approximation and incorporation into Modelica system models is described and illustrated with a case study.

1. Introduction

Numerical simulation of physical processes and optimisation of design objectives are commonly used in system design. Modelica is a powerful object oriented modelling language for hierarchical definition of dynamic systems [1]. Modelica system

models usually consist of more than one "lower level" system models, resulting in a hierarchically integrated model that is suitable for system design studies. These lower level system models are divided into sub-system models and component models, where a component model is assumed to contain no lower level system models, and a sub-system model is assumed to consist of two or more component models or other sub-system models.

The physical behaviour of the sub-systems and components is a key determinant for the system behaviour. Therefore it is of importance that this physical behaviour is adequately modelled, both with respect to the component behaviour and with respect to system behaviour. Sometimes the model of the physical behaviour of a (sub-)system or component may be too complex to be simulated efficiently within the constraints of the integrated system model. Within such a system model one or a few component models of extreme complexity may exist among several relatively simple component models, resulting in an undesirable and unbalanced system model.

Alternatively, in collaborative development projects as for example the EU project POA (Power Optimised Aircraft) [2], sub-system or component information may be supplied from one to another company and therefore proprietary constraints may prevent the use of detailed models of the physical behaviour of sub-systems or components. It may occur that some component models are, for example, represented by no more than a table with measurement results of the component behaviour.

In such situations a modelling approach based on alternative system representations is useful. Generic representations based on approximate models can then be applied to sub-system or component behaviour. Different approximate modelling approaches can be distinguished. For example an

implementation in Modelica of an approach to system identification, where the focus is on estimation of coefficients of the differential-algebraic equations (DAEs) of the time dependent sub-system or component behaviour, is described in [3]. The approach described in the present paper focuses on approximation based on steady-state sub-system or component behaviour.

The steady-state sub-system or component behaviour is assumed to be available as representative data sets, without the detailed mathematical models. These data sets, which represent the underlying system behaviour, may for example arise from series of complex system simulations (e.g. [4]) or physical experiments. If validation or optimisation of the integrated system is considered and consequently large numbers of system evaluations are typically required, approximate representations of the behaviour of the complex sub-systems and components can provide good possibilities for efficient evaluation at low computational cost and with adequate accuracy in Modelica.

Different methodologies are available for efficient approximate representation of system behaviour that is given by data sets. Matlab [5] provides a number of standard functions and toolboxes for approximation and curve fitting, such as the Spline, Curve Fitting and Neural Networks toolboxes, and in addition some other more specific Matlab programs are available, for example with an implementation of the kriging method [11]. In this study a number of these methods are investigated and applied to system simulations in Modelica.

This paper describes and illustrates the development of approximate models in Matlab and their application in Modelica system models. It starts out from data sets representing (sub-)system or component behaviour, which are approximated using a Matlab based tool called MultiFit [6]. An approximate model that has been generated with this tool can be automatically translated into Modelica code. This code can then be incorporated in a Modelica systems model. This complete process is described and illustrated with an example application of a standard engine model.

2. Approximation methods

A large variety of methods and tools is available for approximating system behaviour that is given by

data sets. We limit this study to black-box type systems with one or more inputs and outputs. The most relevant approximation methods for such systems are considered and implemented in a generic Matlab based software tool. This tool, named MultiFit and developed by NLR, has been used in previous studies on approximate models of aircraft systems [4][6]. The tool provides a generic and intuitive graphical user interface (GUI) to approximation methods based on polynomial functions (in this case the approximation method is commonly referred to as response surface method [7]), splines [8], neural networks [9] and kriging models [10]. NLR has enhanced the MultiFit tool with the facility to automatically export approximate models from Matlab to Modelica code, see Figure 1.

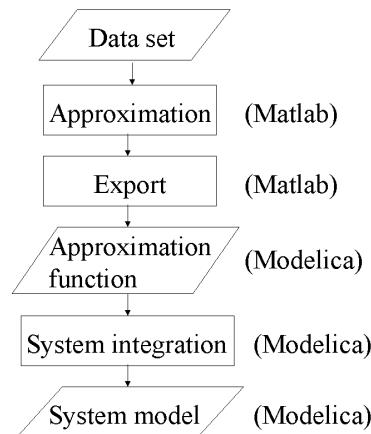


Figure 1 Process diagram of the creation and incorporation of an approximate model into a Modelica system model.

In the approximation tool MultiFit a set of efficient methodologies for approximate representation of data sets has been implemented. The tool makes use of a number of Matlab functions and toolboxes, such as the Neural Network toolbox [5], and other more specific Matlab programs like the DACE program [11] for the kriging method [10] for approximation. The output data that have to be approximated are identified by the variable y , are assumed to be scalar and depend on a vector x of n independent variables. The approximation is performed by fitting an approximate model, based on a specific approximation method, to a given data set $\{(x_i, y_i), i=1,2,\dots,m\}$, which is referred to as training data set. As such this training data set consists of the discrete samples y_i at input values x_i and represents the real system. Let the approximation be defined as $\hat{y} = \hat{f}(x)$, where $x \in \mathbb{R}^n$

and $y, \hat{y} \in \mathbb{R}$. The approximation error is expressed as $\varepsilon = |y - \hat{y}|$ and is measured in a set of x -data points, which is also referred to as validation data set. The quality of the approximation model depends on the achieved accuracy, expressed in terms of the error ε by for example the *root mean square error* (RMSE). A brief description of the approximation methods used in MultiFit is the following:

1. Polynomials: A polynomial function in x is fitted to the data set using a standard least-squares regression technique. The order of the polynomial can be varied between 2 and 6.
2. Splines: Cubic Splines, provided in MultiFit are piecewise smooth polynomial approximations to the data. Both interpolating and smoothing cubic splines as available from Matlab (csapi and csaps) can be called from MultiFit.
3. Artificial Neural Networks (ANNs): The ANN-type provided in MultiFit is a feed-forward ANN with one hidden layer. The number of hidden nodes is automatically determined within an interval supplied by the user, such that the approximation is optimised.
4. Kriging models: The kriging method is based on the formula $\hat{y} = \hat{f}(x) + \hat{\epsilon}(x)$, where $\hat{f}(x)$ is a polynomial regression function and a model of the deviation of the regression function $\hat{\epsilon}(x)$, which is stochastic with non-zero covariance [10]. In MultiFit kriging interpolation methods are used with a combination of a polynomial regression function of order zero, one or two, and an error model function based on a Gaussian, exponential or cubic spline correlation function, as in [11].

3. The MultiFit GUI

An example of the MultiFit GUI is given in Figure 2. As an illustration of the GUI functionality here a very basic example of a 4th order polynomial fit to a one-dimensional sinusoidal data set of 101 points is represented. The GUI directly presents to the user the data set (in a 2D or 3D plot) that is selected for the fit and the available approximation methods for which the user can make appropriate selections from dynamically generated pop-up menus.

An approximation method that is the best for one data set is not always suitable for another data set. Therefore MultiFit provides an automatic approximation assessment based on RMSE values that gives information about the quality of the fit of

the different methods (Figure 3). This way it is possible to select the optimal fit for a given data set.

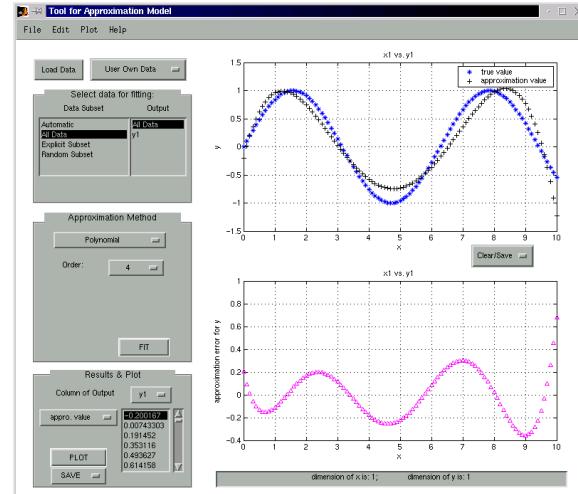


Figure 2 Example of the MultiFit GUI with a 4th order polynomial fit to a sinusoidal data set (upper graph in the GUI) and the approximation error (lower graph).

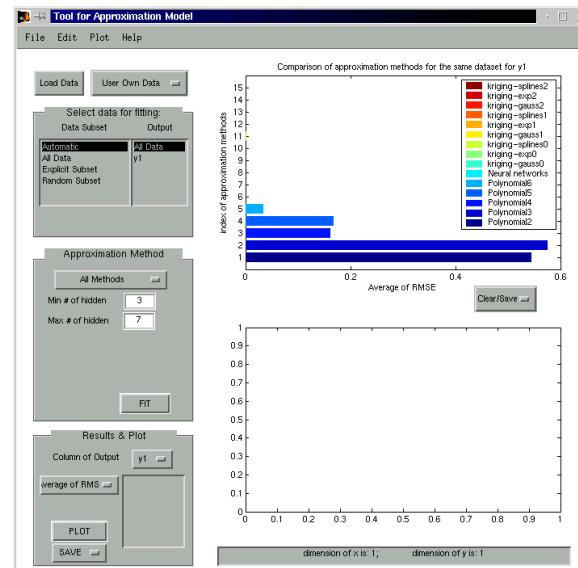


Figure 3 Assessment of fits on a data set using all available approximation methods.

4. Translation to Modelica

MultiFit takes advantage of existing MATLAB approximation functionality. Therefore the process of fitting takes place in MATLAB. A fit result is a function (the approximation function), which can be represented in Modelica format. MultiFit has a functionality that facilitates automatic translation of the resulting approximation function to Modelica

code, which can be easily called from the MultiFit file menu (Figure 4).

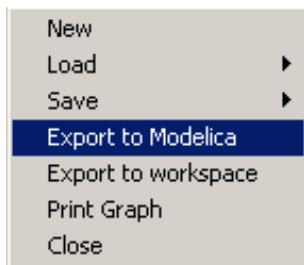


Figure 4 The MultiFit GUI offers easy automatic transfer of approximation models to Modelica, directly from the file menu.

The process is as follows: After an approximation method has been satisfactorily fitted to a data set with MultiFit, this approximation function can be translated to Modelica with the MultiFit export facility. The actual translation is achieved by literal translation of the Matlab expression of the approximation function to Modelica syntax. All the relations between inputs and outputs are explicitly translated with highly accurate export of all real valued parameters.

The exact interfaces of the approximate model in the Modelica environment are not known beforehand. Therefore a modular representation of the approximation function is required. The approximation function is written in a separate Modelica source file as a Modelica *function* (e.g. *sin_poly4* in Figure 5) with the approximation function expression included as a Modelica *algorithm*.

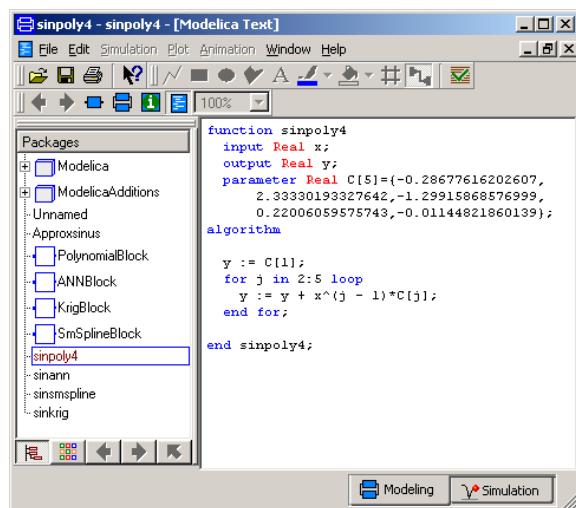


Figure 5 Example of the Modelica source code of a 4th order polynomial approximation of a 1-D sinus function as translated from Matlab to Modelica by MultiFit.

The Modelica approximation function can be included in a component model by inserting an equation of the type $y=<\text{functionname}>(x)$ (e.g. $y=\sin_poly4(x)$) in the Modelica code of the component model that should be approximated. In this way several approximation modules can be used to fit a sinus function.

To illustrate the use of the approximated sinus functions in Modelica, these approximate functions are evaluated in Dymola [12] and compared to the exact Modelica sinus function. In Figure 6 the overall Modelica model is shown, which simulates a sinusoidal signal as a function of time and calls four different approximation functions (4th order polynomial, smoothing spline, ANN and kriging with constant regression and Gaussian correlation) that were automatically generated from MultiFit using the sinusoidal data set of 101 data points as was shown in Figure 2.

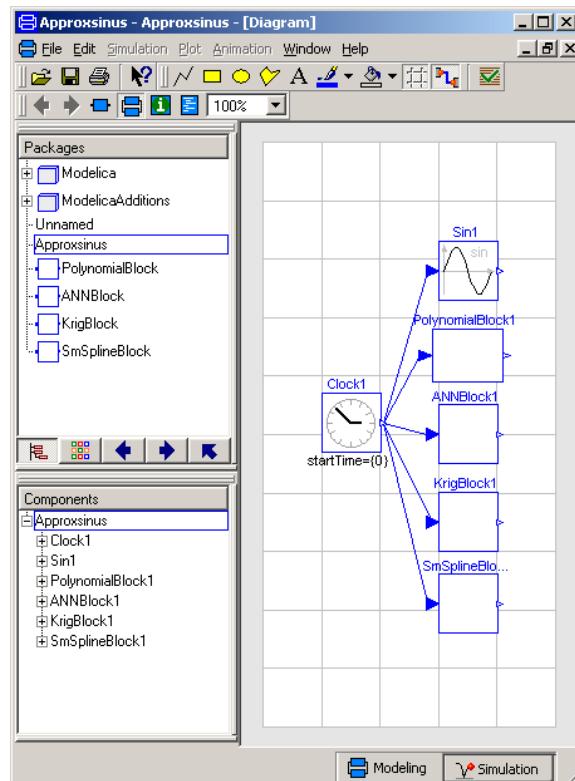


Figure 6 Modelica model with multiple modules to approximate a sinus function.

Figure 7 illustrates the fit results compared to the Modelica sinus signal and confirms the MultiFit information that the polynomial fit has a relatively large approximation error, compared to the other methods.

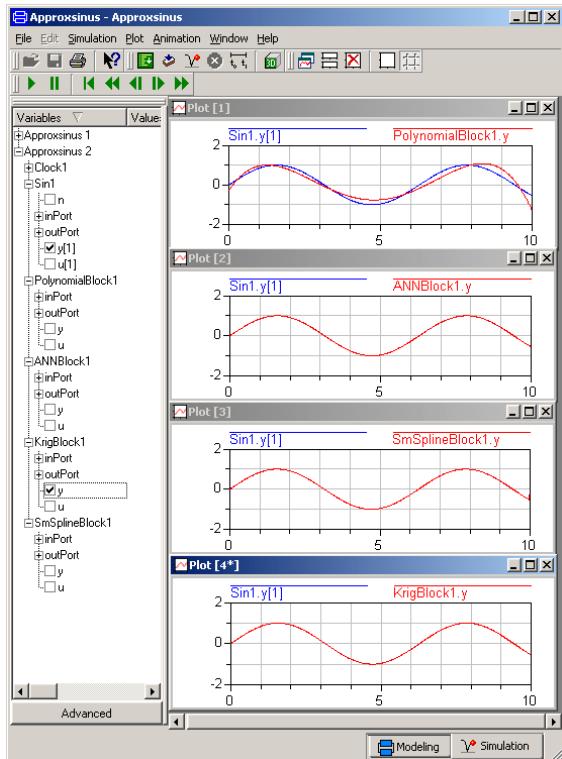


Figure 7 Dymosim simulation results with sinus approximations by (from top to bottom) 4th order polynomial, kriging (constant regression Gauss corr.), ANN, smoothing spline.

5. Verification of the translation to Modelica

To illustrate the translation process and demonstrate the validity of the result, a small verification study of the translation from Matlab to Modelica of all the MultiFit approximation methods is presented. In order to investigate the validity of the translation process not just in a theoretical case with a data set based on some analytical function, a more realistic case is considered where the data set is based on a numerical experiment of the behaviour of an aircraft air-conditioning system. This experiment is based on samples of the local temperature in the aircraft cabin as predicted by CFD simulations, as a function of different settings of the air-conditioning system in terms of inflow temperature and velocity [4], see Figure 8. This data set consists of 121 data points, each of which with a scalar output value (i.e., cabin temperature value).

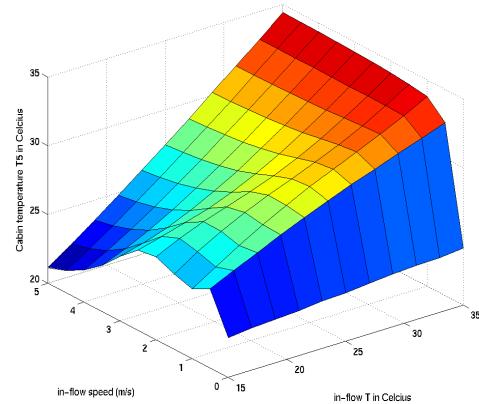


Figure 8 Example data set of local cabin temperature as a function of inflow temperature and velocity, obtained from aircraft cabin CFD simulations

This verification intends to assess the validity and accuracy of the translation process as follows: the approximation methods are first fitted to the data set with MultiFit. Then all the resulting approximation functions are translated to Modelica with the MultiFit export facility. The resulting Modelica code then contains the mathematical expressions of the approximation functions, which should be identical to the expressions in Matlab. It should be noted that in the translation process all real values (of parameters etc.) are exported in %.16e format in order to avoid any truncation errors.

The different approximation functions are stored in different Modelica *function* objects and are called from one single Modelica *model* object shown in Figure 9. Then the Modelica approximation functions are compiled in Dymola and the resulting *dymosim* executable is evaluated in a test set of input points (i.e. x-values). In this verification study this test set consists of the training data set that was used to create the approximation functions in MultiFit. Then the resulting output values of the approximation functions are transferred back to Matlab (again in %.16e format) and compared to the output values of the original approximation functions in Matlab. For each approximation function, the maximum value of the difference between Modelica and Matlab output arrays is considered and shown in Figure 10.

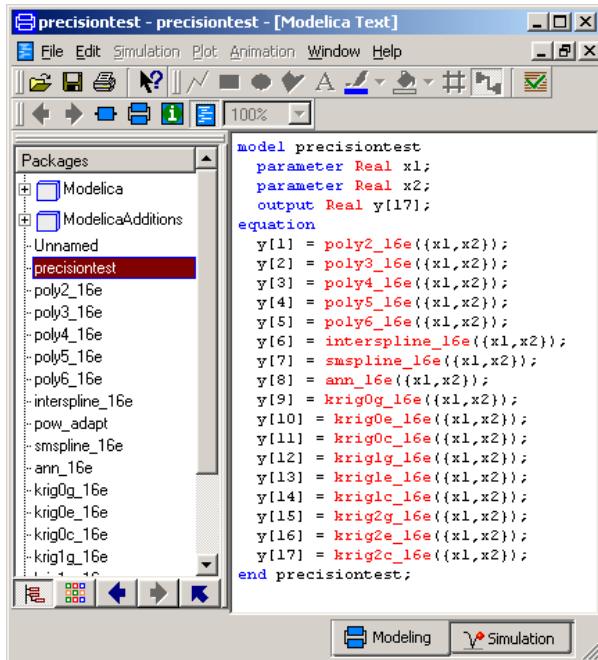


Figure 9 Example of the Modelica model that calls all the approximation functions that have been translated to Modelica.

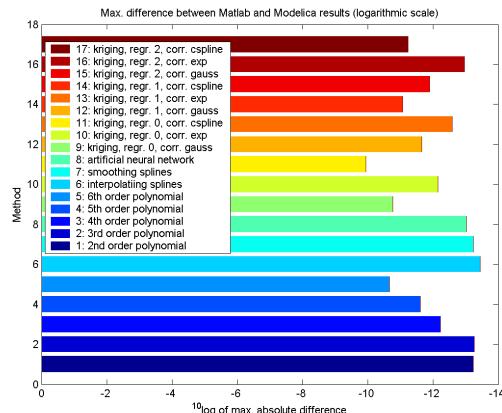


Figure 10 Histogram plot of the maximum difference values for each approximation method.

The differences shown in Figure 10 are, as expected, not far from machine precision ($\sim 10^{-16}$) for most of the fits. However, some of the fits, in particular the higher order polynomial and some of the kriging models, have much larger differences between the Matlab and Modelica computations (up to $\sim 10^{-10}$). For the polynomials this effect is due to the rather large x-values (temperatures of up to 35 °C) that exist in the considered data set and give very large values in the polynomial arithmetic (i.e., up to $35^6 \approx 1.8 \cdot 10^9$ in the 6th order polynomial) that amplify the possible difference in the exported parameter values. It should be noted that the polynomials have

been implemented in a straightforward way, without the re-scaling of variables.

For the kriging models the differences between the Matlab and Modelica computations are due to the very large values of some of the coefficients and arithmetic operations in the approximation functions in these cases. In particular in the kriging models with exponential and Gaussian correlation functions (fits 9, 11, 12, 14, 15 and 17) coefficients of order 10^3 and 10^4 occur.

The following can be concluded from this verification study. The accuracy of the translation is very high, as is shown for the realistic case in this section. Machine precision is not achieved for the maximum values of the differences between the fits in Matlab and Modelica but the discrepancies between the fits can be explained, as shown in the above example. Moreover, these differences are negligible when compared to the errors (e.g., RMSE) of the approximation functions (both in Matlab and in Modelica) in validation data points, which are in the orders of 10^{-1} to 10^{-3} .

6. Application example

As explained in the introduction of this paper, the reasons for applying approximate models of (sub-) system or component behaviour are various. In this section we will consider the case where a sub-system model is available in Matlab/Simulink and has to be integrated into a system model in Modelica because of the ease of multi-physical modelling and sub-system model integration in Modelica. However, the effort of translating the complete model of the physical behaviour as implemented in the Matlab/Simulink model into Modelica code is not intended and therefore an approximate modelling approach is applied.

Figure 11 shows a Matlab/Simulink [5] demonstration model of a combustion engine. Although this is a relatively simple representation of an engine, the emphasis is on the fact that the steady-state behaviour of this model can be integrated into a Modelica system model without completely translating the internal model logic by using the approximate modelling approach. Therefore the model will be treated as if it were a black-box. The inputs of the model are the fuel throttle angle and the torque load. From these values the model calculates the crank speed of the engine in radials per second. If constant values are inserted for

the torque load and the throttle the crank speed quickly converges to a steady-state value, see Figure 12.

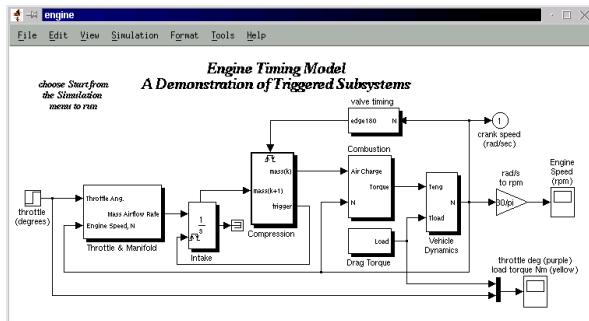


Figure 11 Simulink demo model of an engine [5].

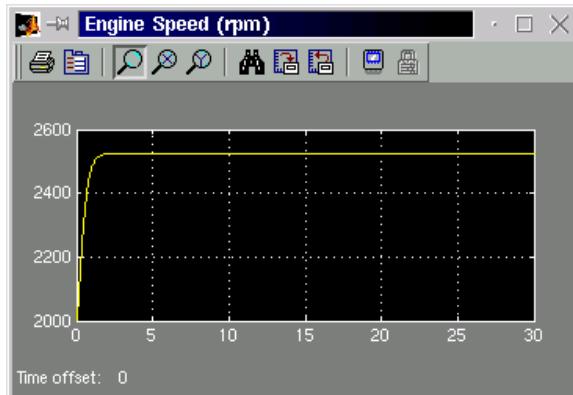


Figure 12 Simulation results for the calculated crank speed (converted to revolutes per minute) with constant torque (20 Nm) and throttle (10 degrees) values.

In order to create a data set of the behaviour of the engine, a design study has been performed where the throttle angle and the torque load have been varied from 0 to 20 degrees and from 0 to 50 Nm, with steps of 2 deg and 5 Nm, respectively. For each torque-throttle combination a simulation of 30 seconds has been performed. For some combinations, i.e. a relatively large torque load compared to the throttle angle, the crank speed drops far below zero and the simulation becomes unstable and the crank speed is not defined. Therefore the model is considered inadequate for these input combinations and these points are excluded from the training data set. The resulting training data set contains 94 valid points and is plotted in Figure 13.

All fit methods offered by MultiFit have been applied to this data set, except the spline methods, since this data set is "gridded" and therefore not appropriate for the applied spline implementation [5]. To determine the best fit method a separate

validation set has been generated beside the data set. The validation points are also plotted in Figure 13.

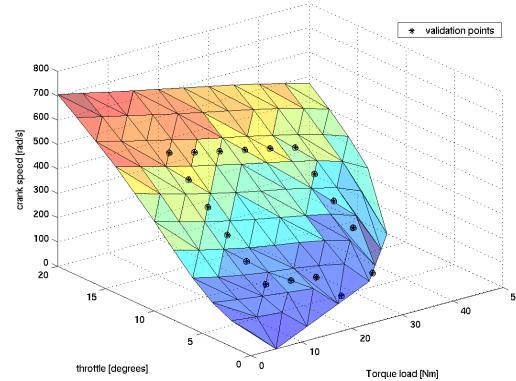


Figure 13 Data set generated from Simulink engine model, with validation data points

The RMSE of the approximate model in the validation points is used as the criterion to determine the optimal fit method for this data set. The results are plotted in Figure 14.

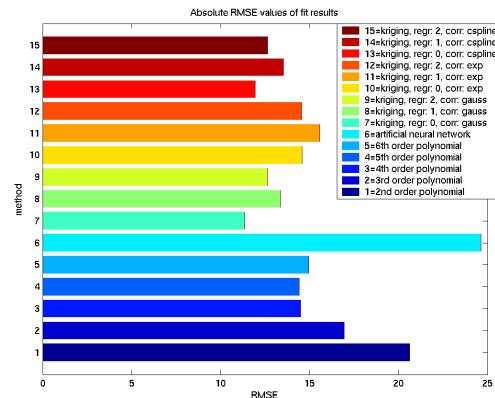


Figure 14 Absolute RMSE values based on a 18 point validation set of the engine model.

It was found that the kriging method with constant regression and Gauss correlation has the lowest RMSE (Figure 14) and gives the best approximation. Therefore, the corresponding approximation function has been translated to Modelica, see Figure 15.

In Modelica an engine model has been created that calls the approximate function to define the relation between the throttle, torque load and crank speed, see Figure 16. Note that the case in which the throttle-torque combination would exceed the domain of the original Simulink model is interpreted as an engine shut-down. In this case the crank speed is set to zero.

```

algorithm
  // scale x
  for i in 1:2 loop
    sx[i] := (x[i] - Ssc[1, i])/Ssc[2, i];
  end for;
  // make correlation vector
  for i in 1:94 loop
    corr := 0;
    for j in 1:2 loop
      corr := corr - theta[j]* (sx[j]-S[i, j])^2;
    end for;
    corr := exp(corr);
    r[i] := corr;
  end for;
  //rescale y
  sy := gamma*r;
  sy[1] := 1*beta[1] + sy[1];
  y := Ysc[1] + Ysc[2]*sy[1];

```

Figure 15 Kriging (constant regression, Gauss correlation) approximation model of Simulink engine in Modelica code

```

model Engine "engine model based on
approximation"
  Real w_rad;
  output Modelica.SIunits.Power
  P_generated=f lange.tau*w_rad
  "Power supplied to loads";
  Modelica.Blocks.Interfaces.InPort Throttle
  "Throttle angle"
  Modelica.Mechanics.Rotational.Interfaces.Flange_
  b flange
  equation
    if noEvent((Throttle.signal[1] < 6 and
    5*Throttle.signal[1] >= flange.tau)
    or (Throttle.signal[1] >= 6 and
    2.5*Throttle.signal[1] + 15 >= flange.tau)) then
      w_rad =
      engdata_kri0G({flange.tau,Throttle.signal[1]});
    else
      w_rad = 0;
    end if;
    der(f lange.phi) = w_rad;
end Engine;

```

Figure 16 Modelica implementation of the wrapping engine model

The engine model has a rotational mechanical connector (the engine shaft) on one side and a signal input (throttle angle) on the other side. Therefore the model can be connected to other Modelica components like signal generators and gear-boxes. The engine model has been integrated, together with some of these other components, into a Modelica system model, as shown in Figure 17.

In combination with prescribed torque load and throttle signal the engine model has been tested, where for the sake of simplicity a gear-box ratio equal to one was used, see Figure 18.

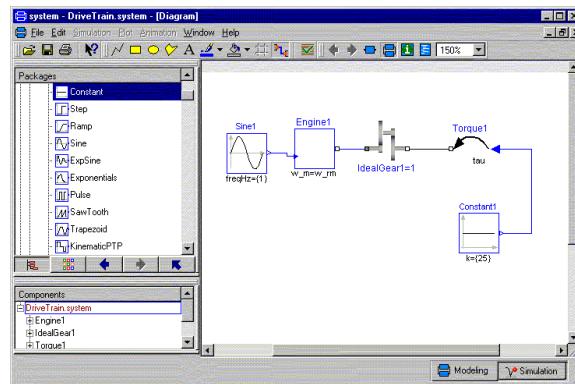


Figure 17 Engine in combination with torque load

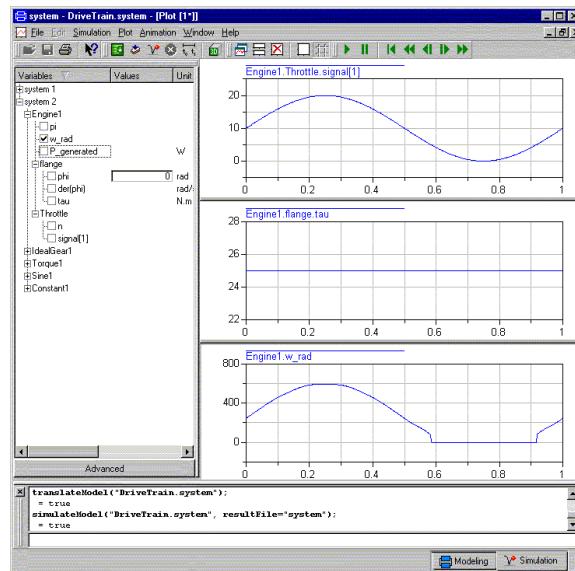


Figure 18 Dymola simulation results with a constant torque load and a varying throttle signal

Figure 18 shows that below a certain throttle value the engine crank speed will be zero if a constant torque is applied. Figure 19 also shows how both the torque and the throttle can be varied, e.g. when simulating a strongly simplified automatic gearbox based on the steady state engine behaviour.

Since the engine model has a mechanical interface it can be connected to other physics models e.g. a generator, hydraulic and pneumatic pumps etc. to simulate integrated multi-physics systems.

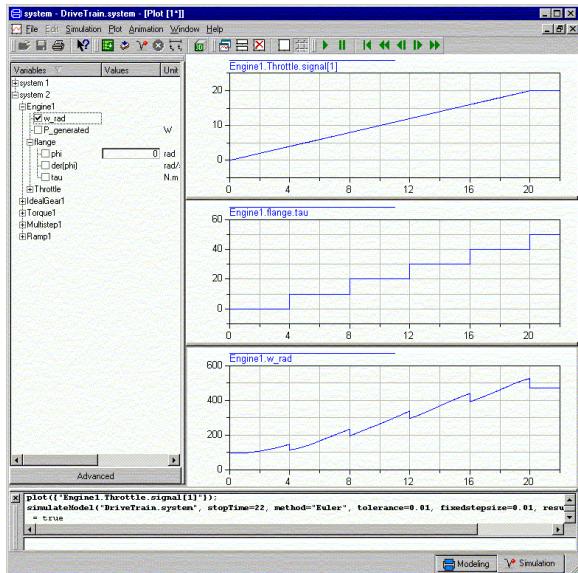


Figure 19 Dymola simulation results with a step-wise increasing torque load and a linear increasing throttle angle

7. Conclusions

An approximate system representation in Modelica has been presented, which is complementary to Modelica's multi-physics modelling paradigm. This complementary system representation is based on input-output data sets. This system representation is particularly useful if the physical behaviour of a considered (sub-)system is too complex, not known, computationally expensive, protected, or just not available.

The approximate system representation in Modelica can be generated with the Matlab based tool MultiFit that was developed at NLR. This tool provides easy and common access to several approximation methods, since each of these methods has its specific merits and there is no "globally best" method available.

It can be concluded that the combination of the NLR tool MultiFit with Modelica supports the full process from approximation of data sets to the integration with other multi-physics components for system optimisation. Specifically in the cases that models have restricted information or are computationally complex the approximation approach promises to be useful for integrated system design.

8. References

- [1] Modelica Association, <http://www.modelica.org>.
- [2] Power Optimised Aircraft, contract G4RD-CT-2001-00601 under the European Communities 5th framework Programme for Research - Competitive and Sustainable Growth - Key Action, New Perspectives in Aeronautics. <http://www.poa-project.com..>
- [3] Sjöberg J., Fyhr F., and Grönstedt T., Estimating parameters in physical models using MathModelica, 2nd International Modelica Conference 2002.
- [4] Vankan, W.J. and R. Maas: Approximate modelling and multi objective optimisation in aeronautic design, CMMSE 2002 Conference, Alicante, Spain, 2002.
- [5] The MathWorks: Developers of Matlab and Simulink, <http://www.mathworks.com>.
- [6] Vankan, W.J., J. Kos, W.F. Lammen: Approximation Models for Multi-Disciplinary System design - Application in a Design Study of Power Optimised Aircraft, Eurogen 2003 Conference, Barcelona, Spain, 2003.
- [7] Myers, R.H. and D.C. Montgomery: Response surface methodology: Process and product optimization using designed experiments, Wiley, New York, 1995.
- [8] de Boor, C.: A Practical Guide to Splines, Springer-Verlag, 1978.
- [9] Caudill, M.: Neural Networks Primer, San Francisco, CA: Miller Freeman Publications, 1989.
- [10] Simpson, T.W., T.M. Mauery, J.J. Korte and F. Mistree: Kriging models for global approximation in simulation based multi-disciplinary design optimization, AIAA Journal, 39(12), 2001.
- [11] Lophaven, S.N et al. DACE: A Matlab Kriging Toolbox, Technical University of Denmark, IMM-TR-2002-12, <http://www.imm.dtu.dk/~hbn/dace>.
- [12] Dynasim AB, <http://www.dymola.com>.

Acknowledgements: The presented work has partially been performed within the framework of the POA project, under contract G4RD-CT-2001-00601. Furthermore the support of Ms. C. Shen, of the University of Amsterdam, in the work on the approximation modelling, and of Dr. H. van der Ven, of NLR, and Prof. A. Veldman, of the University of Groningen, in the work on the cabin airflow CFD simulations, are gratefully acknowledged.

Parsing and Semantic Analysis of Modelica Code for Non-Simulation Applications

Michael Tiller

Powertrain Research Department, Ford Motor Company

ABSTRACT

While most discussions involving Modelica focus on its technical capabilities (*i.e.* object-oriented modeling, handling of DAEs, standard libraries, *etc.*), the benefits of having a formal specification of the language syntax and semantics for non-simulation applications are often overlooked. Unlike many proprietary modeling technologies, where the syntax and semantics of the models change according to the whims of the tool vendor, the syntax and semantics of Modelica models are clearly spelled out in the Modelica specification and considerable effort is made to maintain backward compatibility while adding new capabilities to the language. Not only does this allow vendors to develop simulation environments that independently support a common language, it also allows for the development of ancillary tools to support the model development process. Recognizing some of the best practices in software development, this paper discusses a set of utilities used to analyze existing Modelica models and provide feedback on the structure of the models. These analyses can highlight problematic or unused code, check that code is compliant with specific style guidelines or generate "intelligent" reports on differences between different versions of a model.

1 Motivation

For years, Ford Motor Company has been developing several proprietary Modelica libraries. While we have a talented team of developers and we meet on a regular basis to discuss the evolving structure of our model libraries, it is still difficult to contain the "entropy" that develops due to code fragments that are no longer actively maintained.

After many years focusing on development, it was necessary to take a step back and consider how to manage the growing complexity of our model libraries. Recognizing the common challenges between software development and model development, we have always tried to leverage the best practices from software engineering and incorporate them into our model development. For example, we use a version control system internally to manage releases of our model libraries and we have a web-based issue tracking system that we use to log bugs and feature enhancements. However, these capabilities were easy to leverage because of the availability of general-purpose, out-of-the-box tools (*e.g.* CVS).

Unfortunately, there are many code analyses that we would like to perform that are not supported by general-purpose software engineering tools because they require language specific information. Furthermore, existing Modelica tools focus mainly on simulation-oriented capabilities. As a result, we decided to implement our own utilities to assist us in maintaining our code base.

2 Syntax and Semantics

2.1 Introduction

This section will discuss the steps, tools and ideas involved in taking Modelica code as it appears in a file and creating a representation that captures the underlying "meaning" (*e.g.* type, baseclasses, scope) of the various structural entities.

It should be noted that the analysis capabilities described in this paper do not implement and/or check all the semantics defined in the Modelica specification. Instead, they assume that the code is legal Modelica code generated by a tool (*e.g.* Dymola) that conforms to the Modelica specification. Ideally, we hope that our semantic processing may eventually encompass all the semantics discussed in the Modelica specification but fortunately the analyses described in the paper do not require a complete implementation, only the capability to definitively resolve the types of entities during instantiation.

2.2 Tools

Before presenting additional details about the individual steps involved in processing Modelica code, it is useful to include some discussion of ANTLR [1], the tool used to automate the process of parsing Modelica code. The ANTLR toolset can generate software objects for performing

lexical analysis, grammar parsing and tree parsing (these tasks will be discussed in detail in the remainder of this section). ANTLR includes several useful features including:

- Java, C++ and C# as target languages
- Portable and readable generated code
- Automatic syntax tree construction.
- Active community
- Ongoing development

A surprisingly common question people ask is "Why was Modelica developed with its own unique grammar? Why not simply use XML to describe the format of Modelica files?" Indeed, the wealth of available eXtensible Markup Language [2] (XML) parsers and tools [3] would make the parsing of Modelica files almost trivial. Terrence Parr, author of ANTLR, has provides an excellent discussion of this question in his essay "Humans should not have to grok XML" [5]. The short answer is that XML only addresses the issue of syntax, not the meaning of the constructs themselves. Furthermore, XML is best applied to file formats that are automatically read and written by computers not humans. It is for these reasons that the vast majority of programming languages (*e.g.* Java, C++, Haskell, C#, Python, Perl and Tcl) choose to define their own unique syntax (that is intuitive to human readers and writers) while only a handful of languages like XSLT [4] employ XML syntax. Viewed in this way, the approach taken when developing Modelica is completely consistent with how programming languages, in general, are developed. That being said, a very compelling argument can be made for using XML to represent data structures needed by or resulting from semantic processing [6]. For example, one tool could be responsible for reading the Modelica code and generating an XML representation of the abstract syntax tree. Such a file could then be read by other tools and transformed into representations of instantiated models, hybrid differential-algebraic equations and pseudo-simulation code, *etc.* Such an approach would allow a clean partitioning of tasks and formal description of the various intermediate representations (*i.e.* using Document Type Definitions (DTDs) or XML Schemas).

2.3 Lexical Analysis

The first step in our process to uncover the meaning in Modelica code is to break the code into "tokens". Conceptually, tokens are the words that exist in Modelica (*i.e.* strings of characters

delimited by whitespace). It is very easy to identify the tokens in a given file, but it is also necessary during this step to classify these tokens. Some tokens are easily recognized as keywords (*e.g.* replaceable, parameter, final). Other categories of tokens include literals (*i.e.* integers, reals, strings and Boolean values), punctuation (*i.e.* semicolons, periods, parentheses, *etc.*) and so on. Section 2.1 of the Modelica specification discusses the categories of tokens involved and the patterns used to recognize them. Using ANTLR, our lexical specification for Modelica required 12 non-trivial rules to identify tokens.

2.4 Grammar Definition

Previously, lexical analysis was described as the process by which "words" are extracted from Modelica code. Extending this analogy, grammatical analysis is the process of constructing meaningful "sentences". These sentences can describe definitions of new Modelica types, declarations of components or variables in a class, equations, modifications and so on.

Just as with lexical analysis, the patterns used to describe the grammar of the Modelica language can be found in Section 2.2 of the Modelica specification. An important aspect of creating or processing a grammar definition is avoiding any potential ambiguity. When described using an LL(k) grammar (as required by ANTLR), it is necessary for the parser to look two tokens ahead in order to resolve any ambiguities.

Using ANTLR, our description of the Modelica language involved 35 tokens (and their associated regular expressions), 70 rules and 32 fundamental node types.

2.5 Syntax Trees

2.5.1 Tree Construction

While processing lexical tokens and matching them to grammatical rules, ANTLR includes features to automatically generate a syntax tree to represent the underlying structure of the file being parsed. During tree construction, the goal is to filter out tokens that are only of syntactic significance (*e.g.* semicolons, which only exist to explicitly terminate certain structures) and preserve information that is necessary to fully understand the intent of the code. ANTLR provides a shorthand notation for tree construction that is very convenient, but there are still a few common operations that lack a shorthand representation.

2.5.2 Data Structures

ANTLR builds trees out of nodes and then associating these nodes through child and sibling relationships. By default, ANTLR assumes these nodes are homogenous (*i.e.* they are all of the same type in the target language). This approach works well for "text-to-text transformation" applications (where specific patterns of nodes are simply transformed into other patterns of nodes without a lot of semantic information). However, if the nodes in the resulting tree are likely to have a wide range of different types of information and/or methods associated with them, it is possible to instruct ANTLR to use specific node types (in the target language) for specific structural entities in the tree. The result is a heterogeneous tree structure. As mentioned in Section 2.4, the resulting trees are composed of 32 fundamental node types.

One of the advantages of using heterogeneous node types is the ability to "promote" entities that would normally be tokens into member data associated with that node. For example, Modelica definitions **must** include the name of the class being defined. One approach would be to store this name token as a child node of the definition node in the constructed tree. However, since this is an element that is always present, you can save some complexity in the tree structure (and some lookup time during processing) by storing this information directly as just a string in the definition node itself (as opposed to a child node). We use heterogeneous trees and reserved the use of child and sibling nodes for those structures that are variable (*i.e.* elements whose presence is not known *a priori*).

2.5.3 Tree Walking

ANTLR includes support for creating tree walker objects. Such "tree grammars" are typically much simpler than the formal grammar because they do not include strictly syntactic elements like punctuation and keywords. While tree parsers can be quite useful, we have chosen to use a more programmatic approach for most of the analysis. Rather than walking the tree, most of our analyses involve searching the tree structure for specific elements and then performing operations on those elements. The one case where we currently employ a tree parser is as a validator for our generated tree. By constructing the tree grammar we expect as a result of tree construction, we can apply that tree parser to any tree available (either from directly parsing Modelica code or resulting from

programmatic manipulation of an existing tree structure) and identify any structures not described in the tree grammar. This is analogous to using a DTD or XML schema to validate an XML file.

2.6 Semantic Analysis

As mentioned previously, we assume that all code being parsed is syntactically and semantically legal. In this way, we can avoid implementing the complete semantics of the Modelica specification. Nevertheless, it is still necessary to implement many of the semantics in order to understand what is implied by the code. Without this knowledge, it would be impossible to perform the analyses described in Section 3.

The semantics in the Modelica specification [7] cover all aspects of the language necessary to translate a Modelica model into a system of hybrid, differential-algebraic equations (DAEs). Fortunately, for non-simulation applications only a handful of these semantics are required. Specifically, we have implemented a set of semantics that allows us to instantiate all the components in a model (even those affected by redeclarations). We have neglected all semantics associated with equations and algorithms. As a result, the main task required as part of this instantiation is name lookup as described in Section 3.1 of the Modelica specification.

2.7 Issues

While creating these tools, there were several issues that we uncovered both in both the Modelica specification and ANTLR that are worth mentioning.

2.7.1 Modelica

In Modelica, comments are lexically significant but not grammatically significant and this can make the preservation of comments while rewriting Modelica code a challenge. One way to address this situation would be to make comments grammatically significant. Given the availability of descriptive strings for documentation purposes in Modelica, comments are really only necessary for "commenting out" definitions, declarations, equations or algorithmic statements. As such, they could be inserted as elements in the grammatical rules for those entities. While this would constrain the situations where comments could be used, it would make their preservation much simpler.

In addition, there are some features described in the Modelica specification that have never been implemented. Examples of such features include

the `within` statement and the `analysisType()` function. If a feature goes unimplemented for several years, it is probably worth revisiting that feature to see whether it is truly necessary or desirable. Weeding rarely used or unnecessary features out of the language helps minimize the work associated with developing parsing and semantic analysis tools which, in turn, makes Modelica easier to adopt.

Finally, there are a handful of rules in the Modelica grammar that make the task of resolving ambiguities difficult. Specifically, the use of "initial" as both a keyword and a function name is problematic since the same string, 'initial', can fall into two different token categories (and this depends on where it appears grammatically). Another example of this kind of problematic "reuse" is the '`end`' string which can be used to close a long definition or appear as an element in an expression. Once again, this ambiguity presents a burden for the parser developer.

2.7.2 ANTLR

We chose to generate heterogeneous trees while processing the Modelica grammar. While ANTLR supports heterogeneous trees, using them with C++ as the target language presented many problems. For example, a bug in the garbage collecting mechanism of the AST base classes appears when using heterogeneous trees. In addition, even though ANTLR allows node types to be associated with specific tokens, this applies only during creation of the nodes. When they are referenced from within a rule, a cast is necessary. It is worth mentioning that C++ language support for heterogeneous node types in ANTLR are relatively new. All things considered, these are only minor annoyances and hopefully future versions of ANTLR will include improved support for heterogeneous AST construction.

3 Analyses

Most of the analyses described in this section require that models can be instantiated according to the instantiation process described in the Modelica Language Specification. As a result of this process, a syntax tree is generated to represent the structural elements of the instantiated model. It is then possible to conduct an analysis of the model by "walking the tree" looking for certain patterns and/or performing specialized calculations. This section discusses several specific types of analyses that are applicable to Modelica code.

3.1 Simple Metrics

The idea of "software metrics" has been around for many years [8]. We will begin our discussion with a few simple code metrics that can also be found in non-modeling contexts.

3.1.1 LOC

A common metric in software engineering is "lines of code" (LOC). While easy to measure, the metric itself is normally not that meaningful. For our purposes, we will count lines in each non-package definition and tally these lines for each package. Furthermore, we will define a "line" as any statement that ends in a semicolon. In other words, since line feeds and carriage returns are not grammatically significant, we will focus on the number of statements which is roughly equivalent to the number of lines.

3.1.2 Restricted Class Breakdown

Another statistic that is easy to collect but not very meaningful, is the breakdown of definitions by restricted class (RCB). This metric mainly serves how heavily utilized each restricted class type is within a given package hierarchy. This metric is similar to lines of code because it measures the "volume" of the code but does not accurately assess its complexity.

3.1.3 Inheritance Complexity

A more useful metric (and one that requires implementing instantiation semantics) is quantifying inheritance complexity. Inheritance complexity is a reflection of how confusing the use of inheritance would be to a user. While inheritance is useful for promoting reuse and avoiding the maintenance issues associated with redundant code, it can also make it difficult for users to understand the complete details of a model. Ideally, inheritance should be restricted to definitions that are:

- Used often – Definitions that developers are likely to be familiar with them.
- Necessary – To avoid base classes that introduce unnecessarily fine distinctions.
- Minimal – To keep the number of classes that developers must be familiar with to a minimum.
- Easily resolved – Modelica features such as replaceable types, dynamic scoping and lookup in enclosing scopes can make it hard for developers to easily figure out or remember what the base classes really are.

The inheritance complexity (IC) is computed as follows¹. First, it is assumed that a definition that does not extend from another definition has an IC value of 1. For each extends clause, various adjustments are made to this score. If the definition being extended is used by fewer than 10 definitions, the IC is incremented by 1. If the definition being extended is used by fewer than 5 definitions, the IC is incremented by an additional 1. If the definition being extended contained less than 3 declarations and less than 3 equations then the IC is again incremented by 1. The IC value for the definition being extended is then multiplied by a scaling factor and added to the IC for the current definition. If the type being extended is replaceable and locally defined, the scale factor is 2. If the type is replaceable but defined outside the scope of the current definition, the scale factor is 3. Finally, if the definition being extended is declared outer, the scale factor is 2.

3.2 Style Guidelines

Looking beyond simple metrics, another type of analysis is to check for conformance to style guidelines. Style guidelines are formulated to promote reusability and consistency of code and many of these style guidelines can be formulated in such a way that they can be automatically verified. Any definitions that contain non-conforming code can be identified in automatically generated reports.

At Ford, we have an extensive set of style guidelines. In this section, we will preset a few of these guidelines, discuss why these guidelines were adopted and explain how we automatically check for conformance.

3.2.1 Naming Conventions

According to our style guidelines, all Modelica definitions must begin with a capital letter while declarations must begin with a lower case letter unless they contain only a single letter in which case they should be capitalized. This rule was adopted because it makes it easy to recognize whether a fully qualified name corresponds to a type or an instance.

To check naming conventions, we visit each definition in memory and process the list of enclosed definitions and declarations looking for non-conforming names.

¹ This is just an initial algorithm to demonstrate how such a metric could be calculated. With time, a better algorithm could probably be developed.

3.2.2 Documentation

For a model library to be generally useful, it is important for model libraries to be well documented. Using the tools described in this paper, we are able to automatically review all definitions and declarations and check for the existence of documentation annotations. Furthermore, this analysis can check to see if descriptive strings have been associated with each definition and declaration so that generated GUI dialogs include additional useful information.

3.2.3 Mixing Equations and Components

The last guideline we will discuss is a restriction against representing behaviour both textually and graphically in the same model. To accomplish this, we must classify each declaration as either textual or graphical. For the purposes of this analysis, connector definitions that appear graphically are ignored. The point of this guideline is to avoid confusion that can develop when trying to grasp the behaviour of a model when aspects of that behaviour span both the text layer and the diagram layer.

As of Dymola 5.x [9], it has been possible to quickly assess this restriction visually by inspecting the Modelica source layer. By default, everything that appears in the diagram layer is filtered out. As such, if you see equations and graphical icons in the Modelica source, the definition you are viewing violates this rule. Nevertheless, visual inspection for entire model libraries is not practical and that is the motivation behind having a tool capable of automatically and exhaustively checking an entire library.

3.3 Coverage Analysis

3.3.1 Background

The most elaborate analysis possible with our tools is what we call "coverage analysis". For each of our model libraries (*i.e.* libraries composed of component, subsystem or system model), we try to maintain a companion test suite library. The goal of the test suite library is to include tests of every model in the model library.

These test suites are useful for several reasons. First, they provide us with a way to assess whether recent bug fixes and/or enhancements to our model library have not corrupted any of the models. In addition, we perform similar checks across tools or tool versions. Finally, we can analyze the test suite library identify any coverage gaps (*i.e.* any components that are not tested).

3.3.2 Analysis Algorithm

The first issue that must be addressed is which models to apply the analysis algorithm to. Stated another way, which models are the test cases? Some rather obvious criteria are:

- Any model in a test suite library.
- Any model that extends from certain base classes (e.g. `extends TestCase;`).
- Any model that does not contain connectors.

Of these, the last criteria is the most general and requires the least discipline on the part of the test suite developer. However, because of the time required to conduct the analysis and the large amount of potential data generated as a result, it may be desirable to use one of the more restrictive criteria. Regardless of the criteria chosen, the algorithm is the same.

The first step in the process is to instantiate each test case. Although the complete instantiation process is described in detail in the Modelica specification, the basic principle is to construct the component tree for each model (factoring in redeclarations, base classes, *etc.*). As a result of it, it should be possible to identify the type of every instantiated component. The set of instantiated types is recorded as each test case is instantiated.

When every test case has been instantiated, you are left with the set of all types that were instantiated by at least one test case. You can then iterate over the set of all type definitions in your model library and check to see if they are in the set of instantiated types. Any definition that was not instantiated represents either a gap in coverage by the test suite or a definition that should be deprecated.

Coverage analysis is a good way to make sure that your model library doesn't contain any unused or unnecessary definitions. It also provides feedback on whether a given test suite provides accurate coverage.

4 Results

4.1 Running the Analysis

Normally, the use of our models is scattered over a number of different packages. Obviously, we would like to have a complete test suite that exercises every single model we have. A more reasonable near-term goal would be that every model is used in one of the many packages (most of them application specific) that we have developed.

To support this possibility, the command line syntax of our tool requires the first argument to be the package being analyzed and all other arguments are assumed to be packages that may potentially use components in the first package. A typical command line invocation might look something like:

```
% Metrics Ford FordTestSuite AppLib1 ... AppLibN
```

4.2 Sample Library Results

To demonstrate the results that are generated from our tool, consider the sample package shown in Figure 1. The details of the models are not particularly meaningful for the purposes of evaluating the metrics for the code. Running our `Metrics` program tells us that the library includes 3 models, 1 type definition and 1 package. For a simple package like the one shown in Figure 1, this is obvious. These kinds of statistics are interesting for larger packages where counting definitions becomes impractical. While we will get to additional metrics in subsequent sections, for now let us focus on coverage analysis. Assume we use the package in Figure 2 as our set of regression tests for package in Figure 1. The results of the analysis are shown in Table 1.

```
package CompLib "Component Library"
model A "Simple model"
  Real x;
  annotation(
    Documentation(info="Simple model"));
equation
  der(x) = 2.3*time;
end A;
model B "Typical model"
  type GrowthRate = Real(min=0);
  Real x;
  parameter GrowthRate c=2.3;
equation
  if time<1.0 then
    der(x) = c*time/2;
  else
    der(x) = c*time;
  end if;
end B;
model C "Detailed model"
  Real x, y;
  parameter Real Alpha=0.1, Beta=2;
  parameter Real Gamma=4, Delta=0.4;
equation
  der(x) = Alpha*x*y-Beta*x;
  der(y) = Gamma*y-Delta*x*y;
end C;
end CompLib;
```

Figure 1: Sample Component Library

```

package CompTestSuite
  import CompLib.*;
  model System1
    A a1, a2;
  end System1;
  model System2
    A a;
    C c;
  end System2;
end CompTestSuite;

```

Figure 2: Sample Test Suite

Definition Name	Times Used	Is Documented
CompLib.A	3	Yes
CompLib.B	0	No
CompLib.B.GrowthRate	0	Not Applicable
CompLib.C	1	No

Table 1: Sample Coverage Analysis

4.3 Ford and Modelica Libraries

We thought it would be interesting to compare the metrics of our proprietary Ford powertrain library with the Modelica standard library. For the purposes of this analysis, only the examples in the Modelica standard library were used. The results from this analysis are shown in Figure 3. The X-axis in each plot lists a series of categories and the Y-axis indicates the percentage of definitions in each library that fall into that category.

The documentation and naming convention metrics cannot be applied to type definitions. That is why, for each of these metrics, two sets of results shown. One set includes the all possibilities while the other set only considers the cases where the metric can be applied meaningfully. This highlights the number of type definitions in the Modelica library (e.g. Modelica.SIunits).

Some interesting results found in Figure 3 are:

- Nearly all the models in both libraries are represented by either strictly textual or strictly graphical information.
- Over 70% of the Ford library isn't covered by a test case.
- The biggest difference between the libraries in the documentation. About 90% of the definitions that can be documented in the Ford library do not include documentation while this is true for less than 40% of the definitions that can be documented in the Modelica standard library.
- Naming convention compliance is surprisingly similar for the libraries.

5 Future Applications

The analyses described in this paper are just a few of the many non-simulation related tasks that can be automated with an appropriate library for parsing and processing Modelica code. Other potential applications could include command-line compilers, "lint" like analysis for undesirable construct, pretty-printing tools, ETAGS generators for Emacs, intelligent differencing tools and so on. Although unimplemented, these tasks further justify the utility of such capabilities. Rather than discuss each of these detail, we will present one example in some detail.

5.1 Obfuscation and Filtering

So far, none of the analyses that have been discussed involved rewriting Modelica code. However, for reasons related to protecting intellectual property, it is quite likely that developers of Modelica code may wish to somehow obfuscate or remove certain sensitive models. Note that even with tools capable of encrypting Modelica models, there may still be a need for obfuscation (e.g. exporting models to a Modelica tool or environment that doesn't support encryption).

The most extreme course of action would be to filter models out. Another more moderate approach would be to obfuscate models so that they functioned properly but were hard to understand. To filter models, it would only be necessary to remove their definitions from an existing tree structure before writing that tree structure back out as Modelica code.

Obfuscation is a bit more difficult to implement. The first step would be to identify which definitions needed to be obfuscated (e.g. using a special annotation) and then which elements of that definition were impacted (e.g. only protected elements). For the elements to be obfuscated, several actions are possible programmatically. First, you would almost certainly want to strip off any descriptive strings. Second, for real variables you would probably change their type to Real rather than something that hinted at their units. Finally, you could change the names of these elements so that their names did not hint at their meaning. This last requirement is very tricky because it would require changing any references to the previous name.

6 Conclusions

While the emphasis in most Modelica applications is on modeling, as Modelica becomes

used for "enterprise scale" activities it will be increasingly necessary to have tools capable of analyzing the quality of the underlying code. This paper highlights several practical analyses that are currently in use and several other potential analyses that could be facilitated by such tools.

7 Acknowledgments

Peter Aronsson and Peter Fritzson from PELAB at Linköping University provided me with the source code for their Open Source Modelica project. Although I did not use the code directly, I it was useful as a reference in developing the tools discussed in this paper.

Adrian Pop, also from PELAB, has done considerable work in understanding the role of XML in processing Modelica code. His work discusses the ideas about XML presented in Section 2.2 in greater detail.

Finally, I would also like to thank Hans Olsson at Dynasim AB for helping to explain, in implementation terms, the details described in the Modelica specification.

8 References

1. T. Parr, "ANTLR 2.7.2 Reference Manual", <http://www.antlr.org/doc/index.html>
2. "Extensible Markup Language (XML) 1.0, Second Edition", *World Wide Web Consortium*, <http://www.w3.org/TR/REC-xml>
3. L. M. Garshol, "XML tools by name", http://www.garshol.priv.no/download/xmltools/name_ix.html
4. "XSL Transformations Version 1.0", *World Wide Web Consortium*, <http://www.w3.org/TR/xslt>
5. T. Parr, "Humans should not have to grok XML", <http://www-106.ibm.com/developerworks/library/x-sbxml.html>
6. A. Pop, P. Fritzson, "ModelicaXML: A Modelica XML Representation with Applications", *Modelica'2003 Conference Proceedings*.
7. "Modelica Language Specification, Version 2.0", *Modelica Association*, 2002, http://www.modelica.org/documents/Modelica_Spec20.pdf
8. C. Jones, "Applied Software Measurement : Assuring Productivity and Quality," *McGraw Hill*, 1991.
9. "Dymola User's Manual, Version 5.0a", *Dynasim AB*, Sweden, 2002

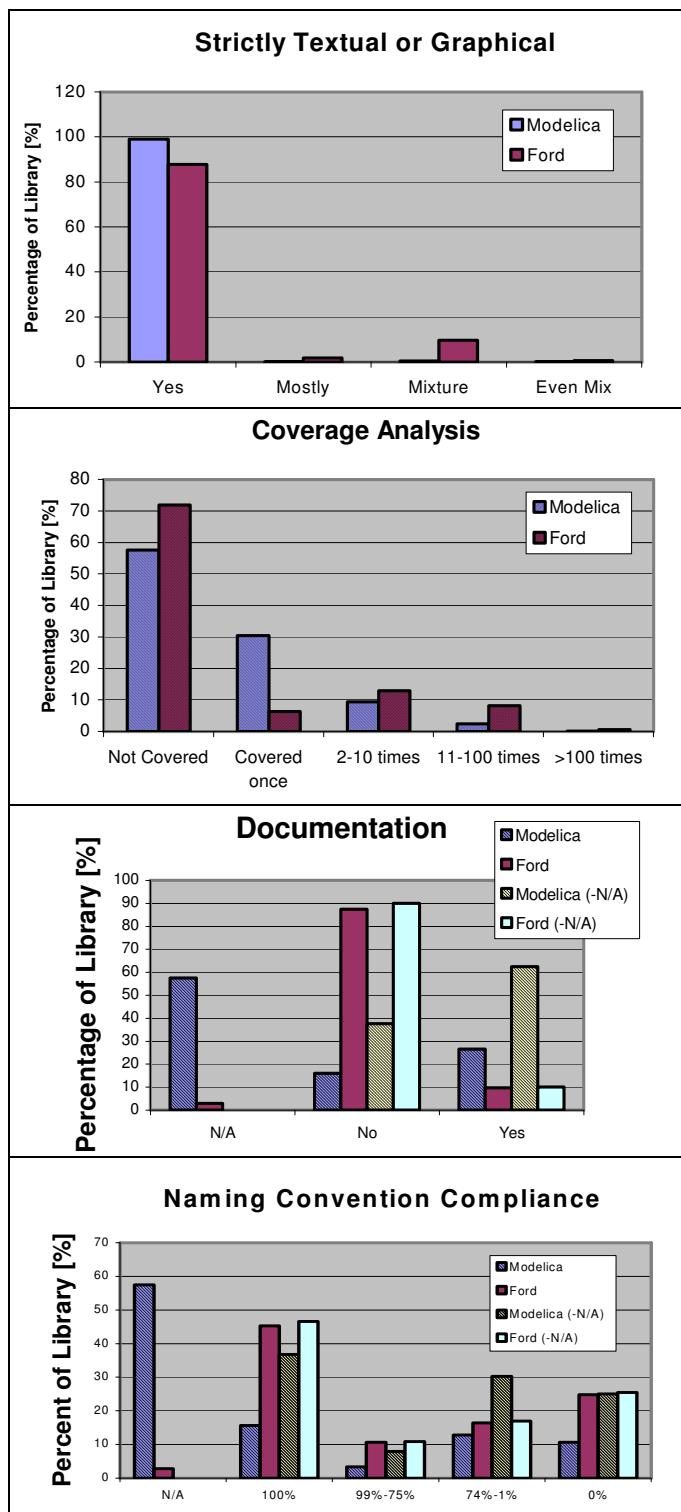


Figure 3: Comparing Ford and Modelica Libraries

ModelicaXML: A Modelica XML Representation with Applications

Adrian Pop

Peter Fritzson

PELAB, Programming Environment Laboratory,
 Department of Computer and Information Science,
 Linköping University, SE-58183, Linköping, Sweden
adrpo@ida.liu.se, <http://www.ida.liu.se/~adrpo>
petfr@ida.liu.se, <http://www.ida.liu.se/~petfr>

Abstract

This paper presents the Modelica XML representation with some applications. ModelicaXML provides an Extensible Markup Language (XML) alternative representation of Modelica source code. The language was designed as a standard format for storage, analysis and exchange of models. ModelicaXML represents the structure of the Modelica language as XML trees, similar to Abstract Syntax Trees (AST) generated by a compiler when parsing Modelica source code. The ModelicaXML (DTD/XML-Schema) grammar that validates ModelicaXML documents is introduced. We reflect on the software-engineering analyses one can perform over ModelicaXML documents using standard and general XML tools and techniques. Furthermore we investigate how can we use more powerful markup languages, like the Resource Description Framework (RDF) and the Web Ontology Language (OWL), to express some of the Modelica language semantics.

1 Introduction

The structure of a Modelica model can be derived from the source code representation, by using a Modelica compiler front-end (the lexical analyzer and the parser).

The compiler front-end takes the source code representation and transforms it to *abstract syntax trees* (AST), which are easier to handle by the rest of the compiler. As pointed out in [20], a clear disadvantage of this procedure is the need of embedding a compiler front-end in every tool that needs access to the structure of the program. Writing such a front-end for an evolving and advanced language like Modelica is not trivial, even with the support of automated tools like Flex/Bison or ANTLR [28].

To overcome these problems, a standard, easily used, structured representation is needed. ModelicaXML is such a representation that defines a structure similar to abstract syntax trees using the XML markup language.

This representation provides more functionality than a

typical C++ class library implementing an AST representation of Modelica:

- Declarative query languages for XML can be used to query the XML representation.
- The XML representation can be accessed via standard interfaces like Document Object Model (DOM) [3] from practically any programming language.

The usages of the ModelicaXML representation for Modelica models, combined with the power of general XML tools, will ease the implementation of tasks like:

- Analysis of Modelica programs (model checkers and validators).
- Pretty printing (un-parsing).
- Translation between Modelica and other modeling languages (interchange).
- Query and transformation of Modelica models.

Although ModelicaXML captures the structured representation of Modelica source code, the semantics of the Modelica language cannot be expressed without implementing specific XML-based tools. To address this issue we have investigated the benefits of using other markup languages like the Resource Description Framework (RDF) and the Web Ontology Language (OWL). These languages, developed in the Semantic Web Community [13], are used to express semantics of data in order to be automatically processed by machines. We believe that using such technology for Modelica models would enable several applications in the future:

- Models could be automatically translated between modeling tools.
- Models could become autonomous (active documents) if they are packaged together with the operational semantics from the compiler, and therefore, they could be simulated in a normal browser.
- Software information systems (SIS) could more easily be constructed for Modelica, facilitating model understanding and information finding.
- Model consistency could be checked using Description Logic (DL) [2].
- Certain models could be translated to and from the Unified Modeling Language (UML) [15].

The paper is structured as follows: Related work is presented in Section 2. Modelica, XML and the ModelicaXML Document Type Definition (DTD) are discussed in Section 3. In Section 4 we present the software-engineering tasks one can perform on the ModelicaXML representation using XML tools and technologies. Section 5 investigates the use of RDF and OWL for representing semantics of Modelica models. Conclusions, future research directions and summary of the work are presented in Section 6.

2 Related Work

In the field of general programming languages, JavaML [20] has been developed as structured representation of Java source code. JavaML emphasizes the power of such structured representation when leveraging XML tools. When it comes to domain specific modeling languages, there are several [21, 22, 27] approaches to specifying models in XML. These approaches deal with model transformation, exchange and management (regarding adaptation to already existing simulation tools) or with code generation from the intermediate XML representation to C++. Our interest focuses more on providing flexible and general software-engineering tooling support for the Modelica programmer. For this purpose the ModelicaXML is covering the full Modelica language [8, 23], including algorithm sections and expression operators. Furthermore, we consider more powerful markup languages for defining some of the Modelica static semantics and we discuss future use of such Semantic Web technologies.

3 Modelica XML Representation

Modelica [8, 23] is an object-oriented language used for modeling of large and heterogeneous physical systems. For modeling with Modelica, commercial software products such as MathModelica [7] or Dymola [4] have been developed. However, there are also open-source projects like the OpenModelica Project [24]. Our research is part of the OpenModelica Project and aims at providing a more flexible framework with the use of XML technologies.

In sub-section 3.1 we briefly introduce the concepts of XML and DTD and give an example of a Modelica model with its ModelicaXML representation.

3.1 The eXtensible Markup Language (XML)

The Extensible Markup Language (XML) [5] is a standard recommended by the World Wide Web Consortium (W3C). XML is a simple and flexible text format derived from Standardized Generalized Markup Language (SGML) [14]. The XML language is widely

used for information exchange over the Internet. The tools one can use for parsing, querying, transforming or validating XML documents have reached a mature state. Such tools exist both as open-source projects and commercial software products.

A small example of an XML document is shown below:

```
<?xml version="1.0"?>
<!DOCTYPE persons SYSTEM "persons.dtd">
<persons>
  <person job="programmer">
    <name>John Doe</name>
    <email>
      grigore@none.ro
    </email>
  </person>
  ...
  <person job="manager">
    <comment>Classified</comment>
  </person>
</persons>
```

An XML document is simply a text in which the information is marked up using tags. The tags are the names enclosed in angle brackets. For easy identification we show *elements* in **bold** face and *attribute* names in *italics* throughout the XML example. The information delimited by **<persons>** and **</persons>** tags is an XML element. As we can see, it can contain other elements called **<person>** that nests additional elements within itself.

The attributes are specified after the tag as an unordered name/value list of *name="value"* items. In our example, the attribute *job* with the value "programmer".

The first line states that this is an XML document. The second line express that an XML parser must validate the contents of the elements against the Document Type Definition (DTD) [18] file, here named "persons.dtd". The DTD provides constraints for the contents much like grammars used for programming languages.

There are two criteria to be met in order for an XML document to be valid. First, all the elements have to be properly nested and must have a start/end tag. Second, all the contents of all elements must obey their DTD grammar specifications.

We will define a DTD for the above example:

```
<!-- the person.dtd file -->
<!ENTITY % person-job-attribute
  "job(programmer|manager)
  #REQUIRED">
<!ELEMENT persons (person*)>
<!ELEMENT person
  ((name+, email*) | comment+)>
<!ATTLIST person
  project CDATA #IMPLIED
  &person-job-attribute;>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
```

The above DTD defines one entity, four elements, and

one attribute list containing two attributes. The entities are underlined, **bold** is used for elements, and attributes are specified in *italics*.

The entity (ENTITY) declaration defines person-job-attribute as a text value that can be used anywhere inside the DTD and the XML document. The XML parser will replace the entity with its defined text where it is used. The principal element (ELEMENT) declared in DTD is **persons** and has zero or more elements **person** nested inside. The special characters inside the element definitions are “*” meaning: zero or more, “|” meaning: selection – either left side or right side, “+” meaning: one or more.

The attribute (ATTLIST) list defines two attributes for the **person** element: *project* and *job*.

The *project* attribute can contain character data (CDATA) and is optional (#IMPLIED). The *job* attribute can only have one of the two values, either “programmer” or “manager”.

There is another XML document structure standard, called XML-Schema [18], which is similar to DTD but is encoded in XML. This standard reconstructs all the capabilities of the DTD and extends them with: namespaces, context sensitivity, the possibility to define several root elements in the same schema, integrity constraints, regular expressions, sub-typing, etc. Tools for transforming XML-Schema representations from/to a DTD representation are available. We use the DTD variant in this example only because it is clearer than the too verbose XML-Schema.

One can consult the World Wide Web Consortium website [5, 18] for more information regarding XML, DTD and XML-Schema.

3.2 ModelicaXML example

To introduce the Modelica XML representation, we give a Modelica example and show its corresponding representation as ModelicaXML.

Elements are in **bold**, attributes are in *italic* and entities are using underline throughout this section, except from Modelica keywords.

```
class SecondOrderSystem
  parameter Real a=1;
  Real x(start=0); Real xdot(start=0);
equation
  xdot=der(x); der(xdot)+a*der(x)+x=1;
end SecondOrderSystem;
```

For ease of presentation, a ModelicaXML document is split into several parts, each representing a more nested level. The ellipses from one level are detailed in the next level:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE program SYSTEM
  "ModelicaXML.dtd">
<program within"...">
  <definition
    ident"SecondOrderSystem"
    restriction"class">
```

```
  ...
  </definitionprogram>
```

The root element is a Modelica **program**. The child elements of **program** are a sequence of **definition** elements and an optional *within* attribute (see Figure 1, sub-section 3.3 for schemata).

```
<definition
  ident"SecondOrderSystem"
  restriction"class">
  <component...</componentequation...</equationdefinition

```

The **definition** element can have **import**, **extends**, **elements**, **equation**, or **algorithm** as sub-elements. In our case we only have **component** (i.e., variable) and **equation** sub-elements inside **definition** (see Figure 2, sub-section 3.3 for schemata).

```
<component
  ident"a" type"Real"
  variability"parameter"
  visibility"public">
  <modification_equalsreal_literal value"1"/>
  </modification_equals>
</componentcomponent
  ident"x"
  type"Real"
  visibility"public">
  <modification_argumentselement_modificationcomponent_reference ident"start"/>
        <modification_equalsreal_literal value"0"/>
        </modification_equals>
      </element_modification>
    </modification_arguments>
</component>
```

The first **component** (i.e., variable, see Figure 3, sub-section 3.3 for schemata) has the *variability* attribute set to “parameter” as in “parameter Real a=1;”. The second **component** declaration (i.e., variable) in the example represents the “Real x(*start*=0);” line from our Modelica class. All components have the *visibility* attribute set to “public”. The last **component** is similar to the second **component** and is not presented.

```
<equationequ_equalcomponent_reference ident"xdot"/>
    <callcomponent_reference ident"der"/>
        <function_argumentscomponent_reference ident"x"/>
        </function_arguments>
      </call>
    </equ_equal>
  </equation>
```

Equations are enclosed in the **equation** element (see Figure 4, sub-section 3.3 for schemata)

The equation section of the **SecondOrderSystem** model describes two equations. The first equation is quite straightforward. Equality is represented by an **equ_equal** element with two elements inside. The right-hand side is a function call (using the **call** element) to a derivative and the left hand side is a component reference represented with the element with the same name. The second equation below is more complex. It has function calls represented using the **call** element, binary operations (see Figure 6, sub-section 3.3 for schemata) such as **add**, **mul** for addition (+) and multiplication (*). The **component_reference** elements denote variable references. For the function calls, the arguments are specified using the element **function_arguments** that can contain expressions, named arguments or for indices.

```

<equation>
  <eq_equal>
    <add>
      <call>
        <component_reference ident="der"/>
        <function_arguments>
          <component_reference
            ident="xdot" />
        </function_arguments>
      </call>
    </add>
    <add>
      <component_reference ident="x"/>
      <mul>
        <component_reference ident="a"/>
        <call>
          <component_reference
            ident="der"/>
          <function_arguments>
            <component_reference
              ident="x" />
          </function_arguments>
        </call>
      </mul>
    </add>
    <integer_literal value="1"/>
  </eq_equal>
</equation>

```

ModelicaXML Schemata are explained in the next subsection.

3.3 ModelicaXML Schema (DTD/XML-Schema)

When designing the ModelicaXML representation we started from the Modelica grammar. We simplified the common cases to compact the XML representation without loss of information or structure. The Modelica DTD/XML-Schema has a rather close correspondence to the Modelica grammar with the following exceptions: attributes are used to make the XML representation more concise and the DTD/XML-Schema jumps over some non-terminals from the Modelica grammar to make the

XML representation more compact.

The OpenModelica Project [29] parser for Modelica source code, written in ANTLR [28], was changed to output the ModelicaXML representation. There are many components in the OpenModelica Project that use the ANTLR Modelica parser. Using our ModelicaXML language such tools can be decoupled from this parser. One clear advantage of this approach is that only one parser is maintained and future Modelica language extensions or modifications could be easily integrated.

For presentation purposes we translated our first DTD implementation to XML-Schema using XML Spy [19]. The purpose of this translation was to generate pictures from the XML-Schema. Also, another reason was to have schemata files in both formats for future use. Perhaps, the DTD variant will be discontinued in the future because the XML-Schema is more widely used now.

All elements from our schema have the optional attributes from the **location** entity (which are *sline*, *scolumn*, *eline* and *ecolumn*) and the **info** attribute, which can be used to store additional information. These **location** attributes are used to generate a mapping between key elements in our schema and the Modelica source code representation. In the following we present some of the important elements from the DTD/XML-Schema.

The content of our ModelicaXML root element, namely **program** is depicted in Figure 1. Inside the root element we can have none or several **definition** elements. The optional attribute **within** can be used inside a **program** element. The rounded corner boxes on the line connecting two elements can be sequence (like in Figure 1) or choice (like in the bottom part of Figure 2).

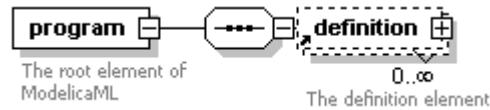


Figure 1: The **program** (root) element of the ModelicaXML Schema

The required attributes for **definition** are **ident** and **restriction** (which can have one of the “class”, “model”, “record”, “block”, “connector”, “type”, “package”, or “function” values). Optional attributes are **final**, **partial**, **encapsulated**, **replaceable**, **innerouter**, **visibility** (one of “public”, “private” values) and **string_comment**.

The **definition** element is detailed in Figure 2. Presented in the picture at the bottom are the **derived** element (that handles constructs of the type “class X = Y;”) and the **enumeration** element used to declare enumeration types. The upper part of Figure 2 shows the other allowed elements that can appear inside the **definition** element. All the elements in the upper part have the **visibility** attribute, taking one of the “public” or “private” values. The **visibility** attribute values are stating the “public” or “private”

part from the Modelica source code. We can see that the **definition** element is recursive, which allows the declaration of classes inside classes.

The **definition** element can contain **import**, **extends**, **external**, **equation**, **algorithm**, **annotation** and **component** elements. The latter can use **constrain** element for handling statements like "type X=Y extends Z;".

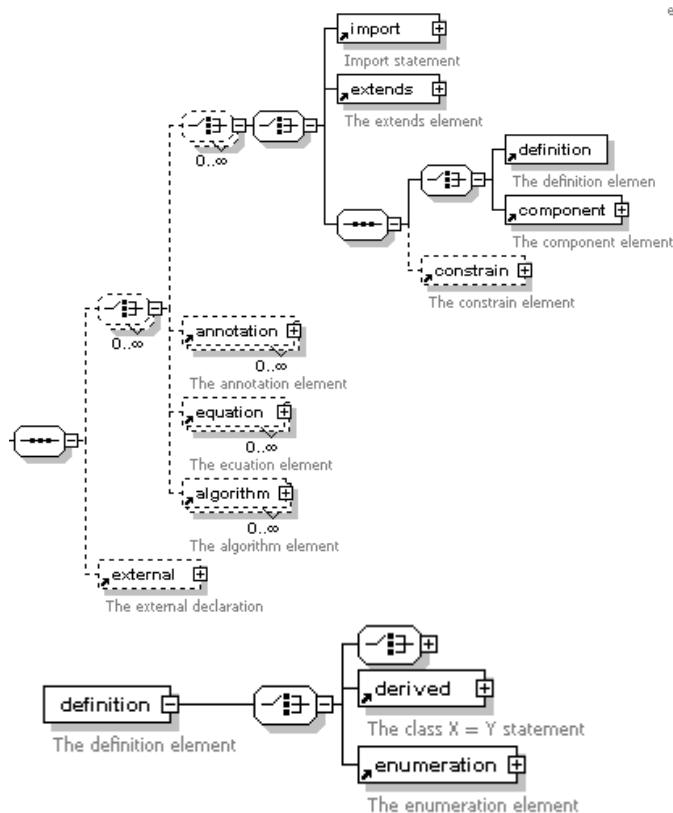


Figure 2: The **definition** element from the ModelicaXML Schema

Component elements, with schemata presented in Figure 3, have attributes representing the Modelica type prefix (*flow*, *variability* and *direction*), and type name (*type*).

The name of the component is stored in the *ident* attribute. These attributes are important because one can query the ModelicaXML representation for a specific component having desired *type* and *ident*. How XML query languages can be used is explained in section 4.

The **type_array_subscripts** element and the **array_subscripts** element are expressing the fact that Modelica array subscripts can be declared either at the type level or at the component level.

One can use the element **modification_arguments** to further modify the component. Comments for a **component** can be specified with the **comment** element. The elements **modification_equals** and **modification_assign** are used to modify the component; as sub-elements they can have Modelica expressions.

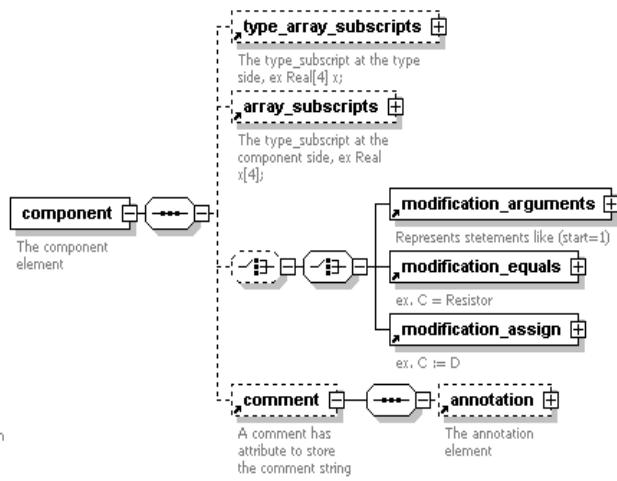


Figure 3: The **component** element from the ModelicaXML Schema

An **equation** element, presented in Figure 4, can have *initial* as an attribute to state if it represents a Modelica initial equation.

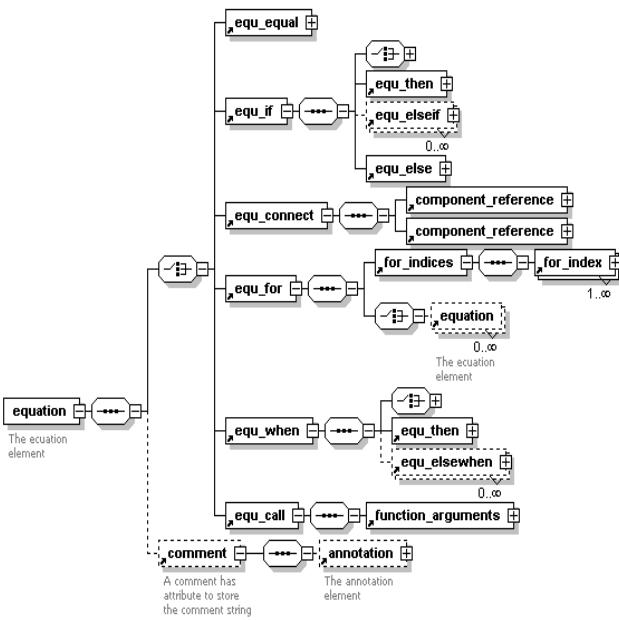


Figure 4: The **equation** element from the ModelicaXML Schema

The content and the structure of the **equation** element are closely following the definition from the Modelica Language Specification [8]. The **equ_connect** element takes component references as arguments here, instead of connect references, as in the version 2.0 of the Modelica Language Specification.

The collapsed parts from the **equ_if** and **equ_when** elements are the Modelica expressions, detailed in Figure 6. The Modelica expressions are present in the collapsed parts of the algorithm elements **alg_if** and **alg_when** and **alg_while**.

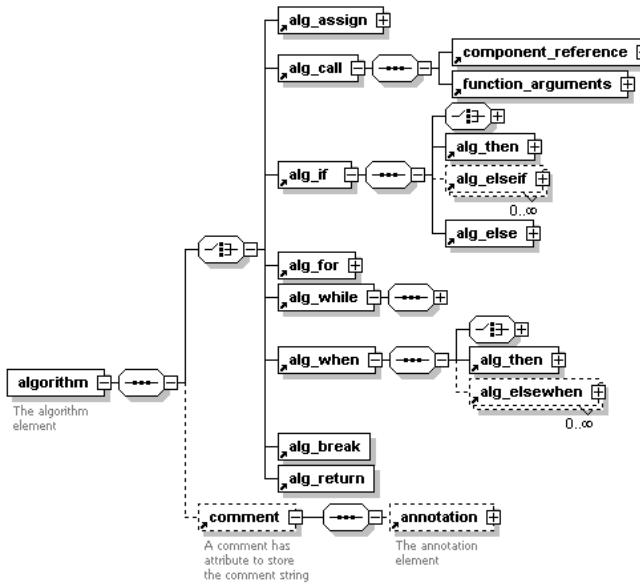


Figure 5: The **algorithm** element from the ModelicaXML Schema

The **algorithm** element is presented in Figure 5. We point out that the elements **alg_break** and **alg_return** are recently added statements of the algorithm section in the latest version (2.1) Modelica Language Specification.

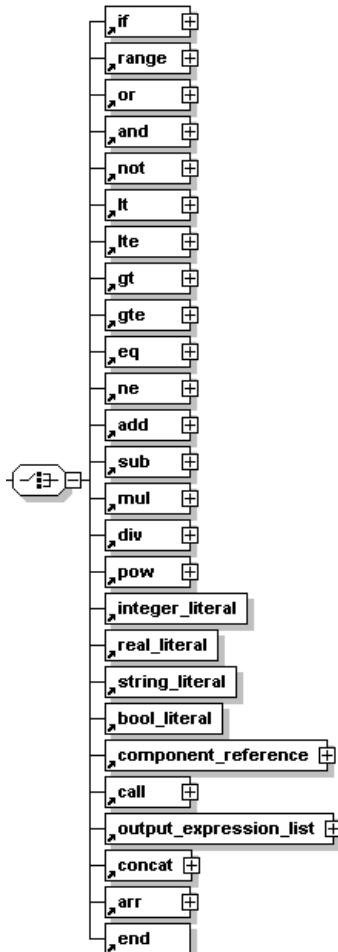


Figure 6: The expressions from ModelicaXML schema

The elements that can appear in ModelicaXML expressions can be found in Figure 6. These are binary operations, literals, component references, array constructions, array operators and logical operations.

The constructs from the ModelicaXML schemata not covered here, along with the full "modelicaXML.xsd" (the XML-Schema version) and "modelicaXML.dtd" (the DTD version), can be found at the OpenModelica Project website.

4 ModelicaXML and XML tools

This section introduces various XML tools and explains their usage in conjunction with ModelicaXML. In the following, in different sub-sections we cover: the stylesheet language for transformation (XSLT) [6], the query language for XML documents (XQuery) [17] and the Document Object Model (DOM) [3].

4.1 The Stylesheet Language for Transformation (XSLT)

XSL is a stylesheet language for XML. XSLT is the part of XSL that deals with transformation of XML documents.

Using XSLT one can implement pretty printers (un-parsers) that can transform ModelicaXML back into Modelica source code. Alternative transformations could transform ModelicaXML into other general, modeling or markup languages (HTML, XHTML, etc). Transformers that translate other modeling languages (provided that they have an XML representation) into ModelicaXML can also be implemented with XSLT. Using XSLT and ModelicaXML, implementation of HTML documentation generators, similar with what the commercial software Dymola provides, becomes trivial. We cannot provide the HTML documentation generator here because of space reasons, but it will be included in the OpenModelica Project.

We illustrate the usage of XSLT with an example that transforms Modelica code. For this example we assume that Modelica code was already translated to ModelicaXML. After the transformation, one can output the Modelica code from the changed ModelicaXML representation using our "modelica-xml2modelica.xslt" stylesheet from the OpenModelica Project.

Example of changing a component name, both in the declaration of the component and in the component references:

```

<xsl:stylesheet version="1.0 ...>
<!-- example of component rename -->
<xsl:param name="comp_old_name"/>
<xsl:param name="comp_new_name"/>
<!-- we echo everything that is not a component or a component reference -->
<xsl:template match="*|@*|text()">
    <xsl:copy>

```

```

<xsl:apply-templates
    select="*|@*|text()"/>
</xsl:copy>
</xsl:template>
<!-- we match the old component and we
output the new name --&gt;
&lt;xsl:template match="component"
    [@ident=$comp_old_name]"&gt;
    &lt;component ident="{{$comp_new_name}}"&gt;
        &lt;xsl:apply-templates/&gt;
    &lt;/component&gt;
<!-- we match the old component
reference and we output the new
component name --&gt;
&lt;/xsl:template&gt;
&lt;xsl:template match="component_reference"
    [@ident=$comp_old_name]"&gt;
    &lt;component_reference
        ident="{{$comp_new_name}}"&gt;
        &lt;xsl:apply-templates/&gt;
    &lt;/component_reference&gt;
&lt;/xsl:template&gt;
&lt;/xsl:stylesheet&gt;
</pre>

```

The XSLT engine is using templates that match on the XML tree structure. The matching is performed by the XPath expression appearing as the value of the *match* attribute. By using **xsl:apply-templates** element we instruct the XSLT engine to apply the rest of the templates on the sub-tree that we already matched. When this stylesheet is applied on our `SecondOrderSystem` example from section 3.2 with the parameters “`xdot`” and “`xdot_new`” it will change the component name and all the component references of `xdot` to `xdot_new`.

XSLT can distinguish between components with the same name defined in different classes by the use of XPath expressions. To rename such occurrences we first match the class in which is defined and then the actual component. This applies for both declarations and component references.

A search-and-replace tool could perform this transformation, but such a tool has no knowledge about the context and it will replace even the occurrences appearing inside comments.

4.2 The Query Language for XML (XQuery)

XQuery is a query language similar with what SQL is for relational databases. Using XQuery, one can easily retrieve information from XML documents. The XQuery and XSLT are overlapping in some features, and our example could be implemented in XSLT also.

We give a short example of a query over our “`SecondOrderSystem.xml`” example from section 3.2. In words, “find all parameter components with type Real and show the initialization value”:

```

<table border="1">
{
    for $b in
        (document("SecondOrderSystem.xml") /*/
            definition/component)

```

```

        where $b/@type = "Real" and
              $b/@variability="parameter"
        return <tr><td>
                { $b/* }
                { $b/modification_equals }
            </td></tr>
}
</table>

```

We executed this query in the Qexo [9] implementation of XQuery and the result in HTML is as follows:

```

<table border="1">
<tr><td>
    ident="a" type="Real"
    variability="parameter"
    visibility="public"
    <modification_equals>
        <real_literal value="1" />
    </modification_equals>
</td></tr>
</table>

```

As expected, the attributes and the set value of the element corresponding to “`parameter Real a=1;`” from our Modelica example was returned as the answer.

Using XQuery, any types of queries can be asked about the Modelica model. This opens-up the possibility of easily debugging very large models. User interfaces can be implemented to hide the query building from the user. Static type checking can also be implemented as a series of queries on the model, but is not trivial, because the class hierarchy is not explicitly defined in XML.

XQuery uses XPath as sub-language to select the part of tree that matches the XPath expression. In our XML representation one can match an entire component having a specified *ident* attribute. The XPath language can be used to handle scooping.

4.3 Document Object Model (DOM)

The Document Object Model (DOM) [3] is a standard interface that allows programs to access/update the content, structure and style of XML documents. DOM is similar with a general tree-management library.

There are open-source implementations for DOM APIs in Java, C, C++, Perl, Python and other programming languages.

Any Modelica tool written in various programming languages can use the DOM API to directly access/modify the ModelicaXML representation.

5 Towards an Ontology for the Modelica Language

This section investigates the possibility of using the markup languages Resource Description Framework (RDF) [11], RDF Vocabulary Description Language (RDFS) [10] and OWL [16] developed in the Semantic Web Community [13] for development of a Modelica ontology.

An ontology is a description (like a formal specification of a program) of both the objects in a certain domain and the relationships between them. In the context of the Semantic Web there is a layered approach for specifying increasingly richer semantics for the upper layers as in Figure 7.

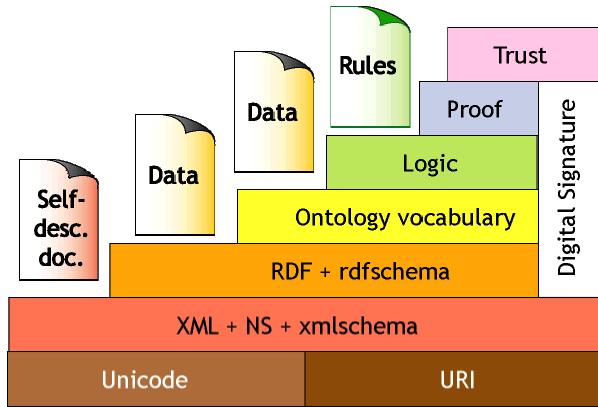


Figure 7: The Semantic Web Layers

At the bottom in top of Unicode and Uniform Resource Identifiers (URI) is XML, namespaces (NS) and XML-Schema. XML specifies a term list with no relations. On top of XML comes RDF to define a vocabulary and some relations. RDFS (RDF schema) defines a vocabulary for constructing RDF vocabularies.

The Ontology layer uses languages like OWL to define description logic relationships.

With ModelicaXML we are now at the XML level! Using RDF we can express graphs and we can model inheritance relationships and place queries over this relation. This can be achieved easily with a smart parser. Using OWL we can place restrictions over relations and concepts and we can reason with inference using Description Logics.

5.1 The Semantic Web Languages

This sub-section briefly introduces the Semantic Web Languages: Resource Description Framework (RDF/RDFS) and Web Ontology Language (OWL).

We illustrate the use of Semantic Web Languages by taking a Modelica model and its representation in OWL.

```
class Body "Generic body"
  Real mass;
  String name;
end Body;
class CelestialBody "Celestial body"
  extends Body;
  constant Real g = 6.672e-11;
  parameter Real radius;
end CelestialBody;

CelestialBody moon(name = "moon",
  mass = 7.382e22, radius = 1.738e6);

Body body_instance(name = "some body",
  mass = 7.382e22);
```

Our Modelica model has two classes (concepts) **Body** and **CelestialBody** the latter being a subclass of the former (by using "extends" statement).

The encoding in OWL is as follows:

```
<?xml version="1.0" ?>
<rdf:RDF>

  <!-- namespaces declaration -->
  xmlns=".../inheritance.owl#"
  xmlns:modelica=".../inheritance.owl#"
  xml:base=".../inheritance.owl">
<owl:Ontology rdf:about=
  ".../inheritance.owl" />

  <!-- define Body -->
<owl:Class rdf:ID="Body">
  <rdfs:label>Generic Body</rdfs:label>
</owl:Class>
  <!-- define mass -->
<owl:DatatypeProperty rdf:ID="mass">
  <rdfs:domain rdf:resource="#Body"/>
  <rdfs:range
    rdf:resource="XMLSchema#float"/>
</owl:DatatypeProperty>
  <!-- define name -->
<owl:DatatypeProperty rdf:ID="name">
  <rdfs:domain rdf:resource="#Body"/>
  <rdfs:range
    rdf:resource="XMLSchema#string"/>
</owl:DatatypeProperty>

  <!-- define CelestialBody -->
<owl:Class rdf:ID="CelestialBody">
  <rdfs:label>
    Celestial Body
  </rdfs:label>
  <rdfs:subClassOf
    rdf:resource="#Body" />
  <!-- cardinality restriction on the
    g constant: one and only one in
    CelestialBody -->
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
        rdf:resource="#g"/>
      <owl:cardinality rdf:datatype
        ="XMLSchema#nonNegativeInteger">
        1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
  <!-- define g -->
<owl:DatatypeProperty rdf:ID="g">
  <rdfs:domain
    rdf:resource="#CelestialBody"/>
  <rdfs:range
    rdf:resource=" XMLSchema#float"/>
</owl:DatatypeProperty>
  <!-- define radius -->
<owl:DatatypeProperty
  rdf:ID="radius">
  <rdfs:domain
    rdf:resource="#CelestialBody"/>
  <rdfs:range
    rdf:resource=" XMLSchema#float"/>
</owl:DatatypeProperty>
```

```

<!--
  instance declaration of CelestialBody
-->
<CelestialBody rdf:ID="moon">
  <name rdf:datatype="XMLSchema#string">
    moon
  </name>
  <mass rdf:datatype="XMLSchema#float">
    7.382e22
  </mass>
  <radius rdf:datatype="XMLSchema#float">
    1.738e6
  </radius>
  <g rdf:datatype="XMLSchema#float">
    6.672e-11
  </g>
  <g rdf:datatype="XMLSchema#float">
    intentional error
    (string is not float)
  </g>
</CelestialBody>

<!--
  instance declaration of Body
-->
<Body rdf:ID="body_instance">
  <name rdf:datatype="XMLSchema#string">
    some body
  </name>
  <mass rdf:datatype="XMLSchema#float">
    7.382e22
  </mass>
  <!--
    intentional error
    (Body does not have a radius)
  -->
  <radius rdf:datatype="XMLSchema#float">
    1.738e6
  </radius>
</Body>
</rdf:RDF>

```

In the OWL representation of the Modelica model we first define **Body** as being an **owl:Class** with “Generic body” as label. The attributes of **Body**, namely: **mass** and **name** are represented as **owl:DatatypeProperty**. The datatype is a binary relation having a **range** (type) and a **domain** (in our case the **Body** concept). As **range** we use the datatypes from XML-Schema, in our case, for **mass** we use “**float**” and for **name** we use “**string**”.

The class **CelestialBody** is defined as **owl:subClassOf** the **Body** class according to the “extends” statement from our Modelica model. As an OWL feature in the definition of **CelestialBody** we show a local cardinality restriction placed on the **g** relation. This means that in the instances of **CelestialBody**, the **g** component has to appear exactly once. The representation of **g** or **radius** components is similar to the representation of **mass** or **name**.

The **moon** instance of the **CelestialBody** class sets the values of the components. We intentionally added the **g** component twice and with a wrong type. We also declare an instance of the **Body** class that has a **radius** component (which is an error).

To verify the model, our file: “**inheritance.owl**” was fed into an OWL Validator [32].

The validator, as expected, reports the following errors:

- For the **g** component that has a string as value: “Range Type Mismatch. Use of this property implies that object is of type XMLSchema#float”.
- For the **radius** component in the **body_instance** declaration: “Domain Type Mismatch. Use of this property implies that subject is of type #CelestialBody. Subject is declared type [Body]”
- For the **moon** instance: “Cardinality Violation. Resource #moon violates the cardinality restriction on class #CelestialBody for property #g. Resource has 2 statements with this property. Maximum cardinality is 1”.

The OWL language has more constructs than our example has covered. One can consult the OWL website [16] for more details.

5.2 The roadmap to a Modelica representation using Semantic Web Languages

In the example above we have presented a small ontology that models our Modelica model, consisting of both classes and instances. With a clever parser, such ontologies could be generated from Modelica libraries and then used for composing Modelica models.

The roadmap to a Modelica representation in OWL has the following steps:

- Define an RDFS vocabulary for Modelica source code constructs. Such a vocabulary should include concepts like class, model, record, block, etc.
- Transform the Modelica libraries in their OWL representation using the above vocabulary.
- An OWL validator can then check the correctness of both the concepts and the instances of these concepts.

At the end of this roadmap we would have Modelica represented in OWL. The future benefits of such a representation were underlined in the Introduction section. Here, we briefly explain how they could be achieved.

The Autonomous Models

In the OpenModelica Project [24], the Modelica compiler is built from the formal specification (expressed in Natural Semantics [26]) of the Modelica Language. This specification can be compiled to executable form using the Relational Meta-Language (RML) tool [30, 31]. The rules from Natural Semantics could be translated to OWL or RuleML [12] and shipped together with the model. Using the rules from the model a normal browser could compile and simulate the Modelica model. We assume that the platform should have a C compiler.

The Software Information System (SIS)

Having the Modelica ontologies that model the source code one could use the approach detailed in [33] and build the domain model of the problem. Merging them together would result in a Software Information System.

Using such a Software Information System users can ask queries about the Modelica source code concepts (components, classes, etc) that are classified according to the domain model concepts of the problem.

Model consistency could be checked using Description Logic

Modelica models represented in OWL (Description Logics) can be fed into a reasoning tool like FaCT [25] for consistency checking.

Moreover, such support would be of great help to the Modelica library designers that could formally check relevant properties of the class hierarchies.

The checks one can do using Description Logics on the Modelica OWL representation are the following:

- Ensure that the classes and the class hierarchy are consistent (ensure that a class can have instances and is not over-constrained).
- Find the explicit relations between classes, regarding for example sub-typing or equivalence.

Translation of Models to/from Unified Modeling Language (UML)

The UML language has its XML representation called XMI [1]. Translation from Modelica models conforming to a Modelica ontology to XMI could be possible using XSLT.

6 Conclusion and future work

We have presented the ModelicaXML language and some applications of XML technologies. We have shown that there are some missing capabilities with such XML representation and we addressed some of them. We have presented a roadmap to an alternative representation of Modelica in OWL and the use of representation together with the Semantic Web technology.

As future work, we consider completing the ModelicaXML with the definition of all the intermediate steps representations from Modelica to flat Modelica and further to the code generation. This complete representation would allow various open-source tools to act at these formally defined levels, independent of each other. More information could be added in the future to such XML representation, like: model configuration, simulation parameters, etc.

Further insights in the direction of Semantic Web Languages and their use to express Modelica semantics is necessary. Compilation in both directions between OWL and the Relational Meta-Language (RML) is worth considering.

7 Acknowledgements

We would like to thank the anonymous reviewers for their valuable and insightful comments or suggestions.

8 References

1. CORBA, XML and XMI Resource Page, <http://www.omg.org/xml/>.
2. Description Logics Website. *Description Logics*, <http://dl.kr.org/>.
3. World Wide Web Consortium (W3C). *Document Object Model (DOM)*, <http://www.w3.org/DOM/>.
4. Dynasim. *Dymola*, <http://www.dynasim.se/>.
5. Word Wide Web Consortium (W3C). *Extensible Markup Language (XML)*, <http://www.w3.org/XML/>.
6. Word Wide Web Consortium (W3C). *The Extensible Stylesheet Language Family (XSL/XSLT/XPath/XSL-FO)*, <http://www.w3.org/Style/XSL>.
7. MathCore. *MathModelica*, <http://www.mathcore.se/>.
8. *Modelica: A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification version 2.1*, Modelica Association, 2003.
9. Qexo - The GNU Kawa implementation of XQuery, <http://www.gnu.org/software/qexo>
10. World Wide Web Consortium (W3C). *RDF Vocabulary Description Language (RDFS/RDF-Schema)*, <http://www.w3.org/TR/rdf-schema/>.
11. Word Wide Web Consortium (W3C). *Resource Description Framework (RDF)*, <http://www.w3c.org/RDF>.
12. The Rule Markup Initiative, <http://www.dFKI.uni-kl.de/ruleml/>.
13. Semantic Web Community Portal, <http://www.semanticweb.org/>.
14. World Wide Web Consortium (W3C). *Standard Generalized Markup Language (SGML)*, <http://www.w3.org/MarkUp/SGML>.
15. UML Website. *Unified Modeling Language (UML)*, <http://www.uml.org/>.
16. Word Wide Web Consortium (W3C). *Web Ontology Language (OWL)*, <http://www.w3.org/TR/2003/CR-owl-features-20030818/>.
17. Word Wide Web Consortium (W3C). *XML Query (XQuery)*, <http://www.w3.org/XML/Query>.
18. Word Wide Web Consortium (W3C). *XML Schema (XSchema)*, <http://www.w3.org/XML/Schemas>.
19. Altova. *XmlSpy*, <http://www.xmlspy.com/>.

20. Greg Badros. *JavaML: A Markup Language for Java Source Code*, in *Proceedings of The 9th International World Wide Web Conference*, May 15-19, 2000, Amsterdam, Nederlands.
21. Johansson Björn, Jonas Larsson, Magnus Sethson and Petter Krus. *An XML-Based Model Representation for model management, transformation and exchange*, in *ASME International Mechanical Engineering Congress*, November 17-20, 2002, New Orleans, USA.
22. Wolfgang Freiseisen, Robert Keber, Wilhelm Medetz, Petru Pau and Dietmar Stelzmueller. *Using Modelica for testing embedded systems*, in *Proceedings of The 2th International Modelica Conference*, March 18-19, 2002, Munich, Germany.
23. Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica*, Wiley-IEEE Press, 2003.
24. Peter Fritzson, Peter Aronsson, Peter Bunus, Vadim Engelson, Levon Saldamli, Henrik Johansson and Andreas Karstöm. *The Open Source Modelica Project*, in *Proceedings of The 2th International Modelica Conference*, March 18-19, 2002, Munich, Germany.
25. Ian Horrocks. *The FaCT System*,
<http://www.cs.man.ac.uk/~horrocks/FaCT/>.
26. Gilles Kahn. *Natural Semantics*, in *Programming of Future Generation Computers*, Fuchi K. and Niva M., Editors, 1988, Elsevier Science Publishers: North Holland. p. 237-258.
27. Jonas Larsson, Björn Johansson, Petter Krus and Magnus Sethson. *Modelith: A Framework Enabling Tool-Independent Modeling and Simulation*, in *European Simulation Symposium*, October 23-26, 2002, Dresden, Germany.
28. Terence Parr. *ANTLR Practical Computer Language Recognition and Translation*, <http://www.antlr.org/book/>.
29. Peter Aronsson Peter Fritzson, Peter Bunus, Vadim Engelson, Levon Saldamli, Henrik Johansson, Andreas Karstöm. *The Open Source Modelica Project*, in *Proceedings of The 2th International Modelica Conference*, March 18-19, 2002, Munich, Germany.
30. Mikael Pettersson. *Compiling Natural Semantics*, Lecture Notes in Computer Science (LNCS) 1549, Springer-Verlag, 1999.
31. Mikael Pettersson. *Compiling Natural Semantics*, Department of Computer and Information Science, Linköping University, Linköping, Dissertation No. 413, 1995.
32. Dave Rager. *OWL Validator*. 2003,
<http://owl.bbn.com/validator/#www>.
33. Christopher Welty. *An Integrated Representation for Software Development and Discovery*, 1996.

Meta Programming and Function Overloading in OpenModelica

Peter Aronsson, Peter Fritzson, Levon Saldamli, Peter Bunus and Kaj Nyström

{petar, petfr, levsa, petbu, kajn}@ida.liu.se

Programming Environments Laboratory (PELAB)

Department of Computer and Information Science Linköping University, Sweden

Abstract

The OpenModelica framework is an Open Source effort for building a complete compiler for Modelica started at the programming environments laboratory at Linköping university. It is written in a language called RML [10], Relational Meta Language, based on natural semantics. Natural semantics is a popular formalism for describing the semantics (i.e. the *meaning* of language constructs) for compilers. By using the RML language this formalism is combined with efficient compilation into optimized C code.

The OpenModelica framework is used to experiment with new language features and language design for the ongoing development of the Modelica language. The design of the Modelica language is performed in the Modelica Design Group (by the Modelica Association) - an open group of Modelica users, researchers, vendors, etc., where the the Modelica language is evolved through intensive discussions in threedays workshops, three or four times a year.

Recently, support for Meta-programming and function overloading (including an external interface to LAPACK) have been implemented in the OpenModelica compiler. This paper present the design and implementation of these language constructs in the OpenModelica framework and illustrates how to utilize this framework for research in e.g. language design, meta-programming and modeling and simulation methodology.

1 Introduction

The OpenModelica[6, 9] environment consist of a compiler that translates Modelica [3, 5] code into flat Modelica, which basically is the set of equations, algorithms and variables needed to simulate the compiled Modelica model. The environment also includes a shell, i.e. an interactive command and expression interpreter, similar to a Matlab prompt, where models

can be entered, computations can be performed and functions can be called. In this environment it is also possible to execute Modelica scripts, i.e. Modelica functions or expressions executed interactively or a set of algorithm statements defined in a text file.

An example of a session in the OpenModelica shell is given below:

```
> ./mosh.exe
Open Modelica 1.0
Copyright 2003, PELAB,
Linkoping University
>>> loadModel(Modelica)
true
>>> model A=Modelica.Electrical
.Analog.Interfaces.OnePort;
Ok
>>> translateModel(A)
record
flatClass = "fclass A
Real v;
Real i;
Real p.v;
Real p.i;
Real n.v;
Real n.i;
equation
v = p.v - n.v;
0.0 = p.i + n.i;
i = p.i;
end A";
exefile = ""
end record
>>>
```

The OpenModelica compiler has been developed at the programming environments laboratory (PELAB) at the department of Computer and Information science at Linköping University. It is used to conduct research on Modelica and tools for modeling and simulation. Current research activities at PELAB involve automatic parallelization [1], support for Partial Differential equations in Modelica [11] and debugging techniques for Modelica [2]. The OpenModelica framework is also used as a testbench for new language

constructs, discussed at the Modelica design meetings held by the Modelica Association [7] approximately four times per year. Many of the ideas presented in this paper have originated from these meetings and some have been elaborated and refined.

The rest of the paper is organized as follows. The next section present the design of Meta-programming in Modelica and how it is used in the OpenModelica compiler. This is followed by a section on operator overloading in Modelica. The paper ends with conclusions and future work.

2 Meta-programming

Meta-programming is to write programs having other programs (so called object-programs) as data [12]. A program can for instance take another program as input data, perform computations on the program by traversing its internal structure (the abstract syntax of the program) and return a modified program as output data.

Often, the object program language and the meta-program language are the same, like for instance in LISP or Java reflection. This is also the approach we have taken for Modelica. A language needs some way of representing the object-program as data. A simple and naive approach is to use strings for this. For example as follows:

```
String equationCode =
"equation v = L*der(i);"
```

However, with this naive approach there is no internal structure of the object. We cannot even guarantee syntactic correctness, e.g. that the program inside the string corresponds to a valid (from a grammatical point of view) piece of code. This is a major disadvantage, and therefore most Meta-programming languages do not use this approach.

Another solution is to encode the object-program using data types of the meta-language. This basically means that data types for the abstract syntax are defined in the language itself. This approach has the benefit of ensuring correct syntax of object-programs. It is used in for instance Java reflection where the class `java.lang.Class` is the *datatype* for a Java class. The class has methods to query a Java class for its methods, members, interfaces, etc.

Our current design uses built-in Modelica types to handle Modelica code, like for instance `TypeName` for a Modelica type name or `VariableName` for a Modelica variable name.

To create data values of the object program in the meta-program a *quoting* mechanism is needed. This

approach is used in several different programming languages, such as Tick-C [4], LISP, MetaML [13] and Mathematica [14]. A quote is used to distinguish the object-program from the meta-program. For instance LISP use the quote character as quotation mechanism.

```
'(plus 1 3)
```

Furthermore to allow insertion of values into quoted expressions an anti-quote mechanism is needed. An anti-quote will lift the following expression up to the meta-program level, and it will thus be *evaluated* and replaced by a piece of object code. For example in LISP the anti-quote is the comma character.

```
(let x '(plus z 3))
'(foo ,x 1)
```

will result in

```
'(foo (plus z 3) 1)
```

2.1 Design Requirements

The requirements for meta-programming support in Modelica are the following:

- **Ease of use** Meta-programming should be easy to learn and use. This means that e.g. the syntax should be similar to what a Modelica programmer is used to. It should be possible to write small pieces of code and insert them with a single command. For instance, adding an equation to an existing model should be a short one-line command.
- **Advanced** At the same time, it should be advanced enough to make it possible to perform the tasks needed by an advanced user who for instance wants to use meta-programming to write diagnosis applications, system identification, applications or other technically advanced tasks where a high level of automation is needed.
- **Backwards compatibility** The design of new language constructs and semantic behavior should be compatible with the current Modelica language [8]. This means that old Modelica code will work with these new extensions.

Considering these requirements, the proposed design is given in the next section.

2.2 Design for Meta-programming

For quoting Modelica code we propose the keyword `Code` together with parentheses and for anti-quoting we propose the keyword `Eval` also with parentheses.

A reasonable limitation for `Eval` is to only allow it in the same context as expressions, i.e. the `Eval` keyword including parenthesis can only be used where expressions can be used. This simplifies the parsing, without limiting the practical usage of the anti-quote mechanism.

The use of these two keywords are given in the following example:

```
myExpr := Code(der(x) + x);
```

Then the code expression

```
Code(equation x=Eval(myExpr))
```

will evaluate to

```
Code(equation x = der(x) + x;)
```

We also introduce built-in types for `Code` expressions. A type for any piece of Modelica code is in our proposal of the type `Code`. Then we have subtypes (subclasses) of this type for specific Modelica code pieces, like `VariableName` for code representing a variable name, `TypeName` for code representing a type name. Table 1 gives the type names for the proposed kinds of Modelica code that can be constructed using the `Code` quote.

Note that some of the cases are overlapping. For instance a variable reference (`component.reference`) is also an expression. In such cases, the most specialized type will be used. For instance, in this case the expression `Code(a.b.c)` will have the type `VariableName` which is a subtype of `TypeName` which is a subtype of `Expression`. This can be inconvenient in some cases when for instance we want to create a piece of code for a type name. To partly remedy this lack we introduce an optional extra "argument" to `Code` giving the type name of the `Code` piece. For example to create a type name we can write¹

```
Code(Modelica.SIunits, TypeName)
```

Also, to fulfill the ease-of-use requirement to a greater extent, and allow for easy use of type and variable names as arguments to functions, we also propose an *automatic quoting* mechanism. The rule is quite simple and solves our problem mentioned above:

- When the expected type (formal parameters and in operator arguments) of an expression is a subtype of `Code` (i.e. any of the types presented in Table 1), if the type of the argument expression is not a subtype of `Code`, the expression is automatically quoted, i.e. becomes a `Code` literal with the same type as the expected type.

¹This also helps in implementing a parser for `Code` constructs, since ambiguities can then be resolved by inspecting the second argument to `Code`.

For example, if we have a function `foo` taking a `TypeName` as an argument and we call it with

```
foo(a.b.c)
```

this will be automatically translated (by the automatic quoting mechanism) into

```
foo(Code(a.b.c, TypeName))
```

With these constructs at hand it is possible to start using Meta-programming by a set of built-in functions for updating Modelica code such as models and functions. Such functions are already partly available in the OpenModelica research compiler and will not be presented in further detail. Instead we will give an example on how to use Meta-programming and scripting functionality to achieve a parameter sweep on a Modelica model. The function is presented in Figure 2 and can be used as follows: We call the `paramSweep` function in the interactive environment and store the result in the variable `r`:

```
>> r:=paramSweep(test,R1.R,1:10);
>>
```

Then we call the function `typeOf` which returns a string representation of the type of a variable:

```
>> typeOf(res)
"SimulationResult[10]"
>>
```

which results in `SimulationResult[10]`, i.e. a vector of 10 elements with the element type being a record of information about a simulation execution.

2.3 Implementation in OpenModelica

In this section we will describe how the Meta-programming support has been implemented in OpenModelica. The support for the quoting mechanism `Code` and `Eval` is added to the internal representation (the abstract syntax tree or AST) using the following data types (in RML code):

```
datatype Code =
  TYPENAME of Path |
  VARIABLENAME of Component_ref |
  EQUATIONSECTION of bool *
    EquationItem list |
    ALGORITHMSECTION of bool
      * AlgorithmItem list |
    ELEMENT of Element |
    EXPRESSION of Exp |
    MODIFICATION of Modification
```

The datatype declarations in RML are similar to those in the Standard ML language. The vertical bar (pipe character) indicates alternatives, the capital letter words are names of node type constructors. For instance, a data object of type `Code` is:

Type	Non-terminal in grammar	Description
TypeName	name	The name of a type, e.g. Modelica.SIunits
VariableName	component_reference	The name of a variable, e.g. a[3].b.c
EquationSection	equation_clause	An equation section, e.g. equation x=y; z=1;
AlgorithmSection	algorithm_clause	An algorithm section, e.g. algorithm x:=sin(y);
Element	element	A class definition, components, import statements and extends clauses declared in a class.
Expression	expression	A Modelica expression, e.g. foo(1:3,a+1)+PI.
Modification	modification	A modification of a component declaration, extends clause, etc. for instance "=1.5" or "(R=10)".

Figure 1: Types for Code expressions.

```

function paramSweep
  "A function for performing a parameter sweep of a model"
  input TypeName modelName;
  input VariableName variable;
  input Real values[:];
  input Real startTime=0.;
  input Real stopTime=1.;
  output SimulationResult result[size(values,1)];
protected
  Boolean flag;
  SimulationObject simObj;
algorithm
  (flag,simObj) := translateClass(modelName);
  assert(flag,"Error translating class.");
  for i in values loop
    result[i]:=simulate(
      startTime=startTime,
      stopTime=stopTime,
      params = SimulateParams(
        {Code(Eval(variable) = Eval(values[i]))},{}))
  );
  // If variable is R1.R and values[i] is 5.6
  // then parameters is Code(R1.R=5.6)
  end for;
end paramSweep;

```

Figure 2: A parameter sweep function using Meta-programming.

```
TYPENAME(
  QUALIFIED( "A" ,
  IDENT( "B" )
)
)
```

which represents the AST for a typename A.B. The new AST type Code contain several different AST nodes, such as a Path node for representing type names, etc. The boolean values for equation and algorithm sections indicate whether the section has prefix initial, e.g. if it is an initial equation or initial algorithm. The Eval construct does not need additional AST types since it is limited to be used as an expression and can thus be expressed as a built-in operator (e.g. a function call in the grammar).

The semantic parts needed for Code and Eval are straightforward to implement. For instance resolving types for Code expressions can be done immediately by using the built-in types presented in Table 1. The semantic rules for Eval must ensure that the evaluated Modelica code has the correct type for insertion in the abstract syntax of its context, i.e. that the result from a eval expression is indeed an expression.

3 Function Overloading

What does it mean to have overloading of operators and functions in a language, and why do we need it? The main reason to have this mechanism in a programming language is economy of notation — overloading allows us to reuse well-known notation and names for more than one kind of data structure. This is convenient and gives more concise and readable code. The concept of overloading can be defined roughly as follows:

- A function or operator is *overloaded* if it has several definitions, each with a different type signature.

The concept of Modelica function type signature is the same as the Modelica class type of the function, and can be defined roughly as follows:

- A Modelica function type signature consists of the set of input formal parameter and result types together with the names of the formal parameters and results, in the order they are defined. To avoid certain lookup and type resolution difficulties, overloading is defined based on the input formal parameters only.

In fact, overloading already exists to a limited extent for certain operators in the standard Modelica 2.1. For example, the plus (+) operator for addition has several different definitions depending on the data type:

- $1 + 2$ – means integer addition of two integer constants giving an integer result, here 3.
- $1.0 + 2.0$ – means floating point number addition of two Real constants giving a floating-point number result, here 3.0.
- $"ab" + "2"$ – means string concatenation of two string constants giving a string result, here "ab2".
- $\{1,2\} + \{3,4\}$ – means integer vector addition of two integer constant vectors giving a vector result, here $\{4,6\}$.

Overloading of certain built-in functions also exists. For example, the size function is defined for arrays of different functionality and occurs in two variants: with one (e.g. `size(A)`) or two arguments (e.g. `size(A, 1)`). Scalar functions of one or more arguments are implicitly defined also for arrays. However, the above examples are just special cases. It is very desirable for the user to be able to define the standard operators as overloaded for user-defined data structures of choice, and to define different overloaded variants of functions with the same name.

To handle function overloading a new short class definition construct is defined, similar to the enumeration defintion. It introduces the new keyword `overload` and has the following grammar rule (added to the class specifier rule):

```
'=' overload '( name_list )'
```

where `name_list` is a list of type names. It can only be used to define functions, like for instance:

```
function solve =
  overload(denseSolve,
           sparseSolve,
           bandSolve);
```

The description of user-defined overloaded operators and functions in Modelica presented here is based on design proposals that have been discussed at several Modelica design meetings by the Modelica Association. The presentation here is roughly the outcome of those discussions, with a few small details added. This design has reached the stage of being approved for test implementation, but not yet made it into the Modelica language specification. Thus, there might be some changes in the final version.

<i>Operator</i>	<i>Operator Example</i>	<i>Function</i>	<i>Function Example</i>
+	a+b	plus	plus(a,b)
+	+a	unaryPlus	unaryPlus(a)
-	a-b	minus	minus(a,b)
-	-a	unaryMinus	unaryMinus(a)
*	a*b	times	times(a,b)
/	a/b	divide	divide(a,b)
^	a^b+	power	power(a,b)
=	a=b	equal	equal(a,b)
:=	a:=b	assign	assign(a,b)
<	a<b	less	less(a,b)
<=	a<=b	lessEqual	lessEqual(a,b)
==	a==b	equalEqual	equalEqual(a,b)
>=	a>=b	greaterEqual	greaterEqual(a,b)
>	a>b	greater	greater(a,b)
<>	a<>b	notEqual	notEqual(a,b)
[]	a[b,c,...]	index	index(a,{b,c,...})
[] :=	a[b,c,...] := v	indexedAssign	indexedAssign(a,{b,c,...},v)

Figure 3: Overloaded operators together with their associated built-in function names

3.1 Operator Overloading

The mechanism for overloading operators is only defined for the standard operators mentioned in Table 3. Adding arbitrary new operators is not possible. Each operator is associated with the name of a built-in function, as defined in Table 3. Note that equality = and assignment := are not expression operators since they are allowed only in equations and in assignment statements respectively. All binary expression operators are left associative. When an operator is applied to some arguments, e.g. a+b, this is interpreted as an application of the corresponding built-in function, e.g. plus(a,b). The usual lookup of the function definition of plus is performed. If a user-defined function plus with matching type signature is found, this function is used, otherwise the standard built-in operator/function +/plus implicitly available in the top-level scope is found if it is defined for the argument data types in question. For example, two addition functions named plus are defined within the same scope, where each definition can be distinguished by the nonequivalence of the second formal parameter types:

```
function plus
  input Real      x;
  input Real[2]   y;
  output Real     z;
  ...
end plus;
function plus
  input Real      x;
  input MyRecord  y;
```

```
  output Real      z;
  ...
end plus;
```

A user-defined record class DiagonalMatrix, shown in figure 4, defines the + (plus) and the [] (index) operators for diagonal matrices that are internally represented compactly as vectors using the DiagonalMatrix data type.

3.1.1 Lookup Rules

Lookup of function definitions (or operators represented by their corresponding built-in functions) will follow the usual Modelica lookup rules[8], with the following additions:

- Both the function name and the input formal parameter types of the called function are used during the lookup process to distinguish matching definitions. The matching criterion for lookup of functions is identity of function names and equivalence of input formal parameter types. In such a match, if the function names are identical and some argument types are not equivalent to corresponding formal parameter types, assignment conversion of argument types, e.g. from Integer to Real, is performed when applicable, and then equivalence of types is checked once more for failure or success of the match.

```

package DiagonalMatrices
  record DiagonalMatrix "Diagonal matrix stored compactly as a vector"
    Real v[:]      "Compact vector representation";
  end DiagonalMatrix;
  function plus "Addition of diagonal matrices"
    input DiagonalMatrix a;
    input DiagonalMatrix b;
    output DiagonalMatrix c;
  algorithm
    // Insert matrix size checks here
    c.v := a.v + b.v; // Use builtin array assignment
  end plus;
  function index "Indexing of a diagonal matrix"
    input DiagonalMatrix a;
    input Integer      b[2]; // Exactly two indices are allowed
    output Real        c;
  algorithm
    c := if (b[1] == b[2]) then v[b[1]] else 0;
  end index;
end DiagonalMatrices;

```

Figure 4: The DiagonalMatrix example, using operator overloading for addition.

- There is an implicit “import” of the packages containing the function argument type definitions, where the desired operator or function definition also might be found. If there is a package scope containing the first argument type definition, this scope is searched first during lookup. If this fails, the package scope containing the second argument type is searched, etc., until this procedure has been repeated for all arguments having a user-defined type. This is the Koenig lookup rule originally used for lookup of overloaded definitions in C++.

The second rule might sound strange, but makes the lookup more specific, and avoids the need to write many import statements specifically for importing function definitions. It is enough to refer to the argument type. For example:

```

Matrices.Symmetric.Matrix A4;
equation
  solve(A4,v2) + func2(5+5,v2) = 0;

```

Here the first argument to `solve` is the variable `A4`. Its type is `Symmetric.Matrix` defined within the package `Matrices.Symmetric`. Therefore the scope of this package is searched first during the lookup, and the function `solve` is found. However, `func2` is searched in the usual way since the type of `v2` is not defined within any package scope.

3.2 Implementation in OpenModelica

Since operator overloading already exist in Modelica today, the design and implementation of operator and function overloading can be performed at a low effort. The largest change is to introduce Koenig lookup mechanism. For this purpose we add the fully qualified type name to a type, giving a new definition of a type as a tuple

```
type Type = (TType * Absyn.path option)
```

Thus, a type now have the fully qualified class name of its definition, making it possible to search for function definitions from the scope where the type is defined, i.e. the Koenig C++ lookup rule.

The same class name can also be used for function types when deoverloading.

This is the major change needed to the type system. The rest of the implementation is concerned with adding the rules to the lookup mechanism and the actual deoverloading mechanism, when the overloaded names are looked up and replaced with the correct function name, according to the types of the input arguments of the function call.

The deoverloading of function calls is performed by traversing a list of function types until a match is found. The only addition needed in this case is to add the function type candidates through the koening lookup rule. For this purpose we add the relation²:

²A RML relation can be seen as a function call, taking arguments as input and producing outputs

```
relation get_koenig_function_types:
  (Env.Env,
   Absyn.Path,
   Absyn.Exp list,
   Absyn.NamedArg list)
  => Types.Type list
```

Its arguments are

- Env.Env The environment for lookup of types, classes,etc.
- Absyn.Path The function name, e.g. A.foo or solve.
- Absyn.Exp list A list of expressions for the positional arguments.
- Absyn.NamedArg list A list of named arguments (a pair of identifiers and expressions)

The result from this relation is a list of function types, to be added to the rest of the function type candidates for the deoverloading process. The relation checks each expression in order to find its type. If the type is user defined, it will look in the types scope to find potential function types.

Below follows a short example run in OpenModelica, using Complex numbers and operator overloading. First we present a short Complex number package:

```
encapsulated package ComplexNumbers
  record Complex
    Real re;
    Real im(start=0);
  end Complex;

  function foo = overload(
    complexFoo,realFoo);

  function complexFoo
    input Complex x;
    input Complex y;
    output Complex res(
      im=x.im + y.im,
      re=x.re + 2 * y.re);
  end complexFoo;

  function realFoo
    input Real x;
    input Complex y;
    output Complex res(
      im=y.im,
      re=x + y.re);
  end realFoo;

  function plus
    "Overloaded user-defined
     complex number addition"
    input Complex x;
    input Complex y;
    output Complex res(
      re = x.re + y.re,
```

```
      im = x.im + y.im);
  end plus;

  function times
    "Overloaded user-defined
     complex number multiplication"
    input Complex x;
    input Complex y;
    output Complex res(
      re = x.re * y.re
      - x.im * y.im,
      im = x.re * y.im
      + x.im * y.re);
  end times;

  function unaryMinus
    "Overloaded user-defined
     complex number unary minus"
    input Complex x;
    output Complex res(re = -x.re,
      im = x.im);
  end unaryMinus;
end ComplexNumbers;
```

The package also contain an overloaded function foo, for illustration of the overload operator. Then we define a test class that uses operator and function overloading:

```
model test
  import ComplexNumbers.Complex;
  import Vectors.Q4Position;
  Complex c1,c2,c3;
  Q4Position p1,p2,p3;
equation
  c1=c2+c3; // ComplexNumbers.plus
  c2=c1*c3; // ComplexNumbers.times
  c3=-c2; // ComplexNumbers.unaryMinus
  c2=foo(c1,c2);
  // ComplexNumbers.complexFoo
  c3=foo(1.0,c1*c3);
  // ComplexNumbers.realFoo
  p1=foo(p2,p3); // Vectors.foo
  p1=p2+p3; // Vectors.plus
end test;
```

We translate the model in the OpenModelica environment:

```
>>> translateClass(test)
record
flatClass = "fclass test
Real c1.re;
Real c1.im;
Real c2.re;
Real c2.im;
Real c3.re;
Real c3.im;
Real p1[1];
Real p1[2];
Real p1[3];
Real p1[4];
```

```

Real p2[1];
Real p2[2];
Real p2[3];
Real p2[4];
Real p3[1];
Real p3[2];
Real p3[3];
Real p3[4];
equation
  TMP0 = ComplexNumbers.plus(c2,c3);
  c1.re = TMP0.re;
  c1.im = TMP0.im;
  TMP1 = ComplexNumbers.times(c1,c3);
  c2.re = TMP1.re;
  c2.im = TMP1.im;
  TMP2 = ComplexNumbers.unaryMinus(c2);
  c3.re = TMP2.re;
  c3.im = TMP2.im;
  TMP3 = ComplexNumbers.complexFoo(c1,
    c2);
  c2.re = TMP3.re;
  c2.im = TMP3.im;
  TMP4 = ComplexNumbers.realFoo(1.0,
    ComplexNumbers.times(c1,c3));
  c3.re = TMP4.re;
  c3.im = TMP4.im;
  p1 = Vectors.foo(p2,p3);
  p1 = p2 + p3;
end test;
",
  exefile = ""
end record

```

4 Conclusions

In this paper we have presented two new areas of interest for the design of the Modelica modeling language, Meta-programming and function overloading. A design of these two language features have been presented and a test implementation has been made in the OpenModelica environment. The effort for implementing these two features have been low, especially for function overloading since most of the required mechanisms were already in place.

The OpenModelica research compiler for Modelica also has some rudimentary support for simulation of Modelica models. This makes it at the same time an interesting tool and/or for Modelica beginners, wanting to learn the language or use Modelica as a computational language, a free replacement of e.g. Matlab or Mathematica.

Function and operator overloading are two modern language mechanisms that make it easier for a user to write programs. Thus, these two new potential additions to the Modelica language will strengthen the Modelica language as a computational programming language, allowing users to write sophisticated numerical computation code, which also allow fast execution due to the Modelica type system. This aspect will also be aided by the Meta-programming

mechanisms, which will allow users to *program* models using scripts, to be used in e.g. design optimization, system diagnosis, and adapting models in a more flexible way for large and complex system modeling.

Future work on the compiler includes implementing full support for Modelica version 2.1. Also, better support for simulation of models must be added. There is also a great need of an updated Modelica test suite, to be able to test modelica compilers against the specification.

References

- [1] P. Aronsson. Licentiate thesis: *Automatic Parallelization of Simulation Code from Equation Based Simulation Languages*. Department of Computer and Information Science, Linköpings universitet, Sweden, 2002.
- [2] P. Bunus. Licentiate thesis: *Debugging and Structural Analysis of Declarative Equation-Based Languages*. Department of Computer and Information Science, Linköpings universitet, Sweden, 2002.
- [3] P. Fritzson, V. Engelson. Modelica - A Unified Object-Oriented Language for System Modeling and Simulation. In *Proceedings of the 12th European conference on Object-Oriented Programming, LNCS*. Springer Verlag, 1998.
- [4] Dawson R. Engler and Massimiliano Poletto. A 'c tutorial.
- [5] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation*. Wiley - IEEE Press, 2003. ISBN 0-471-471631.
- [6] P. Fritzson, P. Aronsson, P. Bunus, V. Engelson, L. Saldamli, H. Johansson, and A. Karström. The open source modelica project. In *Proceedings of the 2nd International Modelica Conference, Germany*, March 2002.
- [7] The modelica association. <http://www.modelica.org>.
- [8] The Modelica Association. *The Modelica Language Specification Version 2.1*, June 2003. <http://www.modelica.org>.
- [9] P. Aronsson, P. Fritzson, L. Saldamli, and P. Bunus. Incremental declaration handling in open source modelica. In *SIMS - 43rd Conference on Simulation and Modeling on September 26-27, Oulu, Finland*, 2002.
- [10] Mikael Pettersson. *Compiling Natural Semantics*. PhD thesis, Linköping Studies in Science and Technology, 1995.
- [11] L. Saldamli. Licentiate thesis: *PDEMModelica - Towards a High-Level Language for Modeling with Partial Differential Equations*. Department of Computer and Information Science, Linköpings universitet, Sweden, 2002.

- [12] Tim Sheard. Accomplishments and research challenges in meta-programming. *Lecture Notes in Computer Science*, 2196:2–??, 2001.
- [13] W. Taha and T. Sheard. Multi-stage programming with explicit annotations. In *Partial Evaluation and Semantics-Based Program Manipulation, Amsterdam, The Netherlands, June 1997*, pages 203–217. New York: ACM, 1997.
- [14] S. Wolfram. *The Mathematica Book, 5th Ed.* Wolfram Media, Inc, 2003.